Zusammenfassung LB2

1. **Einzelner Rückgabewert:** Eine pure Funktion gibt nur einen Wert zurück.
2. **Argumentbasierte Berechnung:** Der Rückgabewert wird ausschliesslich auf Basis der übergebenen Argumente bestimmt.
3. **Keine Seiteneffekte:** Es erfolgt keine Modifikation existierender Werte.

HOF

Eine HOF (High Order Function) in Scala ist einfach eine Funktion, die eine andere Funktion als Argument annimmt und/oder eine Funktion zurückgibt.

Anonyme Funktionen

- Zentral in Scala: erlauben prägnanten und effizienten Code.
- Oft innerhalb anderer Funktionen definiert, ohne explizite `def`-Deklaration.

Map

Die `map`-Funktion ist ein fundamentales Konzept in der funktionalen Programmierung. Sie ermöglicht es, eine Operation auf jedes Element einer Liste (oder einer anderen Collection) anzuwenden und daraus eine neue Liste mit den bearbeiteten Elementen zu erstellen.

Die `map`-Funktion nimmt eine Funktion als Argument. Diese Funktion wird auf jedes Element der Liste angewendet. Das Ergebnis ist eine neue Liste, in der jedes Element das Ergebnis der Anwendung dieser Funktion ist.

Filter

Die `filter`-Funktion nimmt eine Prädikatfunktion (Entscheidungsfunktion) als Argument. Diese Funktion wird auf jedes Element der Liste angewendet. Nur die Elemente, für die das Prädikat (Entscheidung) `true` ergibt, werden in die neue Liste aufgenommen.

Foldleft

Die `foldLeft`-Funktion ist ein wichtiges Werkzeug in der funktionalen Programmierung. Sie wird verwendet, um eine Liste (oder eine andere Sammlung) von Elementen zu einem einzelnen Wert zu reduzieren, indem sie eine Akkumulatorfunktion sequenziell auf die Elemente der Liste anwendet.

Die `foldLeft`-Funktion beginnt mit einem Anfangswert und wendet eine Akkumulatorfunktion auf jedes Element der Liste an, wobei das Ergebnis jedes Schrittes als Eingabe für den nächsten Schritt verwendet wird. Das Endergebnis ist ein einziger Wert.

- **Höhere Abstraktionsstufen:** Erstellung komplexerer Funktionen durch Kombination einfacherer Funktionen.
- **Currying und Partial Application:** Schaffung von Funktionen, die einige Argumente vorab festlegen und den Rest bei späteren Aufrufen erwarten.
- **Funktionale Komposition:** Zusammensetzung mehrerer Funktionen zu einer neuen Funktion

- **Pipelines**: Funktionen werden durch Punkte aneinander gekettet (werden zur Pipeline). Die Daten fliessen dabei von Funktion zu Funktion. Jede Funktion in der Pipeline erhält den Return-Wert der vorhergehenden Funktion als Parameter.
- **IO**: Ein Datentyp, der die Kommunikation mit Input- und Output-Kanälen repräsentiert und die Möglichkeit bietet den Input bzw. Output verzögert (lazy evaluation) auszuwerten. Der Programmierer entscheidet im Code, wann die "impure"-Aktion effektiv ausgeführt werden soll... und bis dahin (bis zum Ausführen der "impure"-Aktion) ist gilt alles als Pure (also alles was den IO-Datentyp verwendet ohne die "impure"-Aktion auszuführen)
- **Streams/LazyList**: Streams sind ein fortlaufender Strom an Daten aus sauberen (pure) oder unsauberen (impure) Quellen und sind mit Rekursion realisiert. Streams haben eine Datenquelle (Producer) und eine Datensenke (Consumer) und dazwischen weitere Operatoren/Filter, die Kopien der Daten aus der Quelle verändern und weiter reichen bis zum eigentlichen Ziel (der Senke) - eine Pipeline zwischen Quelle und Senke. Weil der Streams-Datentyp allerdings den Nachteil hat, dass er das erste Element nicht lazy auswertet (Details) wurde dieser als deprecated markiert und wird zukünftig durch LazyList ersetzt.
- **Parallele Prozesse**: Parallele Prozesse sind vor allem dann wichtig, wenn mehrere CPU-Kerne für die Abarbeitung von Programmen verwendet werden können. Dadurch kann die Verarbeitungszeit insgesamt reduziert werden (weil die verwendete CPU-Rechenleistung erhöht wird).