LU02c - Immutable Values

Immutable values, oder unveränderliche Werte, sind Daten, die nach ihrer Erstellung nicht mehr verändert werden können. Sie stehen im Gegensatz zu den veränderbaren, oder „mutable“ Werten, die modifiziert werden können.

Der Unterschied zwischen mutable und immutable values

- **Mutable Values**: Diese Werte können verändert werden. Ein einfaches Beispiel wäre eine Liste in Python, die verändert werden kann, indem Elemente hinzugefügt oder entfernt werden.

```
# Beispiel für mutable values (Liste)
my_list = [1, 2, 3]
my_list.append(4) # my_list ist jetzt [1, 2, 3, 4]
```

- **Immutable Values**: Diese Werte können nicht verändert werden. Ein Beispiel wäre ein Tupel in Python, das nach seiner Erstellung nicht mehr verändert werden kann.

```
# Beispiel für immutable values (Tupel)
my_tuple = (1, 2, 3)
# my_tuple.append(4) würde einen Fehler erzeugen, da ein Tupel nicht
verändert werden kann
```

Bedeutung von Unveränderlichkeit in der funktionalen Programmierung

Unveränderlichkeit fördert Reine Funktionen

Durch die Verwendung von unveränderlichen Werten wird sichergestellt, dass eine Funktion keine Seiteneffekte hat. Da die Daten nicht verändert werden können, kann eine Funktion, die unveränderliche Werte verwendet, als „rein“ betrachtet werden.

Einfachere Parallelverarbeitung

Unveränderliche Werte erleichtern die Parallelverarbeitung, da keine Sperren oder Synchronisationsmechanismen benötigt werden, um den Zugriff auf Daten zwischen Threads zu steuern.

Vorhersagbarer Code

Da unveränderliche Werte nicht geändert werden können, wird der Code vorhersagbarer. Man muss nicht befürchten, dass ein anderer Teil des Codes die Daten unerwartet ändert.

Herausforderungen bei der Verwendung von Immutable Values

Die Unveränderlichkeit ist nicht immer einfach umzusetzen und kann in einigen Fällen zu Leistungseinbußen führen. Da unveränderliche Werte nicht verändert werden können, muss bei Bedarf ein neuer Wert erstellt werden, was zusätzlichen Speicherplatz erfordert.

```
# Beispiel: Änderung eines unveränderlichen Wertes erzeugt einen neuen Wert
original_tuple = (1, 2, 3)
new_tuple = original_tuple + (4,) # original_tuple bleibt unverändert,
new_tuple ist (1, 2, 3, 4)
```

Fazit

Die Unveränderlichkeit ist ein Schlüsselkonzept in der funktionalen Programmierung, das zur Entwicklung klarer und wartbarer Codes beiträgt. Durch die Festlegung, dass Werte nach ihrer Erstellung nicht verändert werden dürfen, werden potenzielle Fehlerquellen und Komplexitäten im Code vermieden. Insbesondere in Mehrthreading-Umgebungen bietet die Unveränderlichkeit Vorteile, da keine Sperren benötigt werden, um den Zugriff auf Daten zu kontrollieren. Allerdings erfordert die Implementierung der Unveränderlichkeit auch ein Umdenken und kann in bestimmten Fällen Herausforderungen bei der Leistung und dem Speicherbedarf mit sich bringen. Dennoch ist die Nutzung der Unveränderlichkeit, insbesondere in Verbindung mit reinen Funktionen, ein mächtiges Werkzeug in der funktionalen Programmierung, das dazu beiträgt, robuste und leicht verständliche Programme zu erstellen.

M323-LU02, M323-AF1