LU02e - By Value und By Reference bei Dataclasses und Objekten in Python

Einführung

In Python werden Objekte, einschließlich Dataclasses, immer By Reference übergeben. Das bedeutet, dass bei der Übergabe eines Objekts an eine Funktion eine Referenz auf das Objekt übergeben wird und keine Kopie davon erstellt wird. Änderungen, die innerhalb der Funktion an den Attributen des Objekts vorgenommen werden, wirken sich direkt auf das ursprüngliche Objekt aus.

Dataclasses

Dataclasses wurden in Python 3.7 eingeführt und bieten eine bequeme Möglichkeit, Klassen zu definieren, die hauptsächlich Daten speichern. Sie reduzieren den Boilerplate-Code, der normalerweise mit der Definition von Klassen einhergeht, und bieten gleichzeitig eine klare und strukturierte Art der Datenverwaltung.

Hier ein einfaches Beispiel für eine Dataclass:

```
from dataclasses import dataclass

@dataclass
class Person:
    name: str
    age: int
```

Diese Dataclass Person hat zwei Attribute: name und age.

By Reference bei Dataclasses

Da Dataclasses Objekte in Python sind, werden sie By Reference übergeben. Das bedeutet, dass jede Veränderung der Attribute eines Dataclass-Objekts innerhalb einer Funktion direkt auf das Originalobjekt wirkt.

Beispiel: By Reference mit Dataclasses

```
from dataclasses import dataclass

@dataclass
class Person:
    name: str
    age: int
```

```
def birthday(person: Person):
    """
    Increments the age of the person by 1.
    """
    person.age += 1

if name == 'main':
    p = Person(name='Alice', age=30)
    print(f'Vor der Änderung: {p}') # Output: Vor der Änderung:
Person(name='Alice', age=30)
    birthday(p)
    print(f'Nach der Änderung: {p}') # Output: Nach der Änderung:
Person(name='Alice', age=31)
```

In diesem Beispiel wird die Instanz der Dataclass Person an die Funktion birthday übergeben. Da Objekte By `Reference` übergeben werden, wird das Attribut age der Originalinstanz p direkt in der Funktion verändert.

Mutable und Immutable Attribute in Dataclasses

Python-Dataclasses können sowohl mutable als auch immutable Attribute enthalten. Unabhängig davon, ob die Attribute mutable oder immutable sind, wird die Dataclass selbst immer By `Reference` übergeben.

Beispiel: Mutable Attribute in Dataclasses

```
from dataclasses import dataclass, field
from typing import List
@dataclass
class Student:
name: str
grades: List[int] = field(default_factory=list)

def add_grade(student: Student, grade: int):
"""
Adds a grade to the student's grade list.
"""
student.grades.append(grade)

if name == 'main':
s = Student(name='Bob')
print(f'Vor der Änderung: {s.grades}') # Output: Vor der Änderung: []
add_grade(s, 90)
print(f'Nach der Änderung: {s.grades}') # Output: Nach der Änderung: [90]
```

In diesem Beispiel wird die Liste grades (ein mutable Attribut) innerhalb der Funktion add_grade verändert. Da das Objekt Student by reference übergeben wird, wird die Original-Liste grades modifiziert.

Immutable Dataclasses

Wenn Sie möchten, dass eine Dataclass unveränderlich ist, können Sie sie durch Setzen des frozen-Parameters in @dataclass unveränderlich machen. In einer gefrorenen (frozen) Dataclass sind alle Attribute unveränderlich, und jede versuchte Änderung führt zu einem Fehler. Dies ähnelt dem Verhalten von By `Value`, da die ursprüngliche Instanz nicht verändert werden kann.

Beispiel: Immutable Dataclass

```
from dataclasses import dataclass
@dataclass(frozen=True)
class ImmutablePerson:
name: str
age: int
```

In diesem Fall ist die Dataclass ImmutablePerson unveränderlich, was bedeutet, dass jedes Attribut, sobald es gesetzt ist, nicht mehr geändert werden kann. Jeder Versuch, das Attribut age oder name zu ändern, führt zu einem Fehler.

Zusammenfassung

In Python werden Objekte, einschließlich Dataclasses, immer By `Reference` übergeben. Änderungen an den Attributen eines Objekts innerhalb einer Funktion wirken sich direkt auf das Originalobjekt aus. Um ungewollte Änderungen zu vermeiden, kann eine Dataclass als `frozen` deklariert werden, was sie unveränderlich macht und vor unbeabsichtigten Modifikationen schützt.

Tipp: Wenn Sie mutable Attribute in einer Dataclass verwenden, sollten Sie sorgfältig überlegen, ob die direkte Veränderung dieser Attribute innerhalb von Funktionen in Ihrem Programmdesign erwünscht ist. Verwenden Sie `frozen` Dataclasses, um unbeabsichtigte Änderungen zu verhindern.