Iterables und Iteratoren in Python

Einführung

In Python sind Iterables und Iteratoren grundlegende Konzepte, die bei der Arbeit mit Schleifen und der Iteration über Datenstrukturen wie Listen, Tupeln und Dictionaries eine wichtige Rolle spielen. Diese Konzepte sind eng miteinander verbunden, aber sie haben unterschiedliche Funktionen und Merkmale.

Iterables

Ein Iterable ist ein Objekt, das die `__iter__()`-Methode implementiert. Diese Methode gibt einen Iterator zurück, der verwendet werden kann, um über die Elemente des Iterables zu iterieren.

Beispiele für Iterables

- Listen
- Tupel
- Dictionaries
- Sets
- Strings
- Generatoren

```
liste = [1, 2, 3, 4, 5]
for element in liste:
    print(element)
```

Iteratoren

Ein Iterator ist ein Objekt, das die `__next__()`-Methode implementiert. Diese Methode gibt das nächste Element im Iterable zurück, wenn es verfügbar ist, oder wirft eine `StopIteration`-Ausnahme, wenn das Iterable erschöpft ist.

Beispiele für Iteratoren

- `range()`-Objekte
- `map()`-Objekte
- `filter()`-Objekte
- Generatorausdrücke

```
iterator = iter(range(5))
print(next(iterator))  # Ausgabe: 0
print(next(iterator))  # Ausgabe: 1
print(next(iterator))  # Ausgabe: 2
print(next(iterator))  # Ausgabe: 3
print(next(iterator))  # Ausgabe: 4
```

Unterschied zwischen Iterables und Iteratoren

- Ein Iterable ist ein Objekt, das die `__iter__()`-Methode implementiert und einen Iterator zurückgeben kann.
- Ein Iterator ist ein Objekt, das die `__next__()`-Methode implementiert und verwendet werden kann, um über die Elemente eines Iterables zu iterieren.

Iteration in Python

In Python wird die Iteration über ein Iterable durch die `for`-Schleife oder die Verwendung der Funktionen `iter()` und `next()` durchgeführt.

Beispiel: Iteration über eine Liste

```
liste = [1, 2, 3, 4, 5]
for element in liste:
    print(element)
```