Datentyp `list` in Python

Einführung

Der Datentyp `list` in Python ist eine geordnete Sammlung von Elementen, die unterschiedliche Datentypen enthalten kann, einschließlich anderer Listen. Listen sind veränderlich, was bedeutet, dass Elemente hinzugefügt, entfernt oder geändert werden können.

Listen erstellen

```
# Eine leere Liste
leere_liste = []

# Eine Liste mit gemischten Datentypen
gemischte_liste = [1, "Hallo", 3.14, [1, 2, 3]]
```

Zugriff auf Listenelemente

Der Zugriff auf Elemente erfolgt über den Index, der bei 0 beginnt.

```
zahlen = [1, 2, 3, 4, 5]
erstes_element = zahlen[0]  # 1
letztes_element = zahlen[-1]  # 5
```

Listenmanipulation

Elemente hinzufügen

- `append(element)`: Fügt ein Element am Ende der Liste hinzu.
- `insert(index, element)`: Fügt ein Element an einer spezifischen Position ein.

Elemente entfernen

- `remove(element)`: Entfernt das erste Vorkommen eines Elements.
- `pop(index)`: Entfernt und gibt ein Element an einer bestimmten Position zurück. Ohne Angabe eines Index wird das letzte Element entfernt.

Listen kombinieren

- `extend([elements])`: Fügt die Elemente einer anderen Liste hinzu.

Listenoperationen

Slicing

Teillisten können durch Slicing extrahiert werden.

```
zahlen = [1, 2, 3, 4, 5]
teil_der_zahlen = zahlen[1:3]  # [2, 3]
```

Iteration

Listen können mit einer `for`-Schleife durchlaufen werden.

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

Listenkomprehension

Listenkomprehensionen bieten eine kompakte Methode, um Listen zu erstellen.

```
quadrate = [x**2 for x in range(10)]
```

Die Funktionen `sort()` und `sorted()` in Python

Einführung

In Python gibt es zwei gängige Wege, Sammlungen zu sortieren: die `sort()`-Methode, die Listen in-place sortiert, und die `sorted()`-Funktion, die eine neue sortierte Liste aus jeder Iterierbaren erzeugt.

Die `sort()`-Methode

Die `sort()`-Methode wird direkt auf Listen angewendet und sortiert die Elemente der Liste in-place, d.h., die Liste wird direkt modifiziert.

Syntax

```
liste.sort(key=None, reverse=False)
```

Beispiel

```
zahlen = [3, 1, 4, 1, 5, 9, 2]
zahlen.sort()
print(zahlen)  # [1, 1, 2, 3, 4, 5, 9]
```

Die `sorted()`-Funktion

Die `sorted()`-Funktion kann auf jede Iterierbare angewendet werden und erzeugt eine neue sortierte Liste, ohne die ursprüngliche Datenstruktur zu ändern.

Syntax

```
sorted(iterierbar, key=None, reverse=False)
```

Beispiel

```
zahlen = (3, 1, 4, 1, 5, 9, 2)  # Ein Tupel
sortierte_liste = sorted(zahlen)
print(sortierte_liste)  # [1, 1, 2, 3, 4, 5, 9]
```

Parameter `key` und `reverse`

Sowohl `sort()` als auch `sorted()` akzeptieren die Parameter `key` und `reverse`, um das Sortierverhalten zu steuern.

- `key`: Ein Funktion, die angibt, nach welchem Kriterium die Elemente sortiert werden sollen.
- `reverse`: Ein Boolean, der angibt, ob die Liste in absteigender Reihenfolge sortiert werden soll (`True`) oder nicht (`False`, Standard).

Beispiel mit `key`

```
wörter = ["Banane", "Apfel", "Erdbeere", "Dattel"]
wörter.sort(key=len)
print(wörter)  # ['Apfel', 'Dattel', 'Banane', 'Erdbeere']
```

Beispiel mit `reverse`

```
zahlen = [3, 1, 4, 1, 5, 9, 2]
print(sorted(zahlen, reverse=True))  # [9, 5, 4, 3, 2, 1, 1]
```