LU04g - Die sorted-Funktion in Python

Die `sorted`-Funktion ist eine eingebaute Funktion in Python, die es ermöglicht, Elemente einer Liste oder eines anderen Iterables zu sortieren. Diese Funktion nimmt ein Iterable und optional eine Schlüsselfunktion sowie einen `reverse`-Parameter als Argumente und gibt eine neue sortierte Liste zurück.

Syntax

Die Syntax der `sorted`-Funktion ist wie folgt:

```
sorted_result = sorted(iterable, key=function, reverse=False)
```

- **iterable**: Das Iterable, das sortiert werden soll.
- **key**: Eine optionale Funktion, die angibt, nach welchem Kriterium sortiert werden soll.
- **reverse**: Ein optionaler Boolean-Wert, der angibt, ob die Liste in umgekehrter Reihenfolge sortiert werden soll (Standardwert ist `False`).

Beispiele

Listen

Ein einfaches Beispiel für die Verwendung der `sorted`-Funktion ist das Sortieren einer Liste von Zahlen:

```
numbers = [3, 1, 4, 1, 5, 9, 2, 6, 5]
sorted_numbers = sorted(numbers)
print(sorted_numbers)  # Output: [1, 1, 2, 3, 4, 5, 5, 6, 9]
```

Tuples

Auch Tupel können sortiert werden, und wiederum gibt `sorted` eine Liste zurück:

```
my_tuple = (3, 1, 2)
sorted_tuple = sorted(my_tuple)
print(sorted_tuple)  # Output: [1, 2, 3]
```

Listen, die Tupel enthalten, können ebenfalls sortiert werden. Hier sortieren wir eine Liste von Tupeln basierend auf dem zweiten Element jedes Tupels:

```
list_of_tuples = [(1, 'one'), (4, 'four'), (3, 'three'), (2, 'two')]
sorted_tuples = sorted(list_of_tuples, key=lambda x: x[1])
print(sorted_tuples)  # Output: [(4, 'four'), (1, 'one'), (3, 'three'), (2,
```

```
'two')]
```

Sets

Sets können ebenfalls sortiert werden, jedoch gibt `sorted` eine Liste zurück:

```
my_set = {3, 1, 2}
sorted_set = sorted(my_set)
print(sorted_set)  # Output: [1, 2, 3]
```

Listen, die Sets enthalten, können ebenfalls sortiert werden. In diesem Beispiel sortieren wir die Sets basierend auf ihrer Größe:

```
list_of_sets = [{1, 2}, {1, 2, 3, 4}, {1}, {1, 2, 3}]
sorted_sets = sorted(list_of_sets, key=lambda x: len(x))
print([len(s) for s in sorted_sets])  # Output: [1, 2, 3, 4]
```

Liste von Dictionaries

Eine häufige Anwendung für die key-Funktion ist die Sortierung einer Liste von Dictionaries. In diesem Fall verwenden wir eine Lambda-Funktion, um die Liste nach einem bestimmten Schlüssel in jedem Dictionary zu sortieren:

```
countries = [{'name': 'Germany', 'population': 83}, {'name': 'France',
'population': 67}, {'name': 'Italy', 'population': 60}]
sorted_countries = sorted(countries, key=lambda x: x['population'])
print(sorted_countries)
# Output: [{'name': 'Italy', 'population': 60}, {'name': 'France',
'population': 67}, {'name': 'Germany', 'population': 83}]
```

Lambda-Funktionen bieten eine kompakte Möglichkeit, das Sortierverhalten von `sorted` anzupassen, insbesondere wenn wir Listen von komplexen Objekten wie Dictionaries sortieren wollen.

Liste von Objekten

Es ist auch möglich, eine Liste von Objekten zu sortieren. Angenommen, wir haben eine Klasse `Person` mit den Attributen `vorname`, `name` und `jahrgang`:

```
class Person:
    def __init__(self, vorname, name, jahrgang):
        self.vorname = vorname
        self.name = name
        self.jahrgang = jahrgang

persons = [Person('Alice', 'Müller', 1985), Person('Bob', 'Schmidt', 1990),
Person('Charlie', 'Meier', 1980)]
```

```
sorted_persons = sorted(persons, key=lambda x: x.jahrgang)
print([p.jahrgang for p in sorted_persons])  # Output: [1980, 1985, 1990]
```

In diesem Beispiel verwenden wir eine Lambda-Funktion als `key`, um die Liste der Personen nach ihrem `jahrgang` zu sortieren.

Verständnis der Lambda-Funktion im key-Argument

Die `lambda`-Funktion im `key`-Argument der `sorted`-Funktion dient dazu, die Sortierreihenfolge der Elemente im Iterable anzupassen. Die `lambda`-Funktion wird auf jedes Element des Iterables angewendet, und der zurückgegebene Wert wird als Schlüssel für die Sortierung verwendet.

Hier sind einige Anwendungsbeispiele:

1. **Für Listen von Zahlen oder Strings**: In diesen Fällen ist die `lambda`-Funktion oft unnötig, da das Standardverhalten von `sorted` meist ausreicht.
2. **Für Listen von Tupeln**: Wenn Sie eine Liste von Tupeln sortieren und nur eines der Tupel-Elemente als Sortierkriterium verwenden wollen, können Sie die `lambda`-Funktion verwenden.

   ```
   sorted(list_of_tuples, key=lambda x: x[1])
   ```

   In diesem Beispiel wird das zweite Element jedes Tupels (`x[1]`) als Schlüssel für die Sortierung verwendet.

3. **Für Listen von Dictionaries**: Sie können eine `lambda`-Funktion verwenden, um die Liste nach einem bestimmten Dictionary-Schlüssel zu sortieren.

   ```
   sorted(list_of_dicts, key=lambda x: x['some_key'])
   ```

   Hier wird das Dictionary-Element mit dem Schlüssel `'some_key'` als Sortierschlüssel verwendet.

4. **Für Listen von benutzerdefinierten Objekten**: Wenn Sie eine Liste von Instanzen einer benutzerdefinierten Klasse sortieren, können Sie mit der `lambda`-Funktion nach einem bestimmten Attribut der Objekte sortieren.

   ```
   sorted(list_of_objects, key=lambda x: x.some_attribute)
   ```

   In diesem Fall wird das Attribut `some_attribute` jedes Objekts als Sortierschlüssel verwendet.

Die `lambda`-Funktion gibt für jedes Element im Iterable einen Wert zurück, und `sorted` verwendet diese Werte, um die Elemente in der sortierten Liste anzuordnen.

Mehrere Sortierschlüssel in Lambda-Funktionen

Es ist möglich, mehrere Sortierkriterien in einer Lambda-Funktion zu definieren. Dies ist nützlich, wenn der erste Sortierschlüssel für mehrere Elemente identisch ist und Sie eine weitere Sortierung

durchführen möchten.

Die sorted-Funktion in Python erlaubt dies durch die Verwendung von Tupeln als Rückgabewert der Lambda-Funktion. Der erste Wert im Tupel dient als primärer Sortierschlüssel, der zweite Wert als sekundärer Sortierschlüssel und so weiter.

Beispiel: Mehrere Sortierschlüssel für eine Liste von Tupeln

```
list_of_tuples = [(1, 'one', 3), (4, 'four', 2), (3, 'three', 1), (2, 'two', 3), (3, 'three', 2)]
sorted_tuples = sorted(list_of_tuples, key=lambda x: (x[2], x[1]))
print(sorted_tuples)
# Output: [(3, 'three', 1), (4, 'four', 2), (3, 'three', 2), (1, 'one', 3), (2, 'two', 3)]
```

In diesem Beispiel sortieren wir die Liste von Tupeln zuerst nach dem dritten Element (Index 2) jedes Tupels. Wenn zwei Tupel das gleiche dritte Element haben, wird als nächstes das zweite Element (Index 1) für die Sortierung verwendet.

Beispiel: Mehrere Sortierschlüssel für eine Liste von Dictionaries

```
students = [
    {'name': 'Alice', 'grade': 90, 'age': 20},
    {'name': 'Bob', 'grade': 90, 'age': 21},
    {'name': 'Charlie', 'grade': 85, 'age': 22},
    {'name': 'David', 'grade': 90, 'age': 19}
]
sorted_students = sorted(students, key=lambda x: (x['grade'], x['age']))
print(sorted_students)
# Output: [{'name': 'Charlie', 'grade': 85, 'age': 22}, {'name': 'David', 'grade': 90, 'age': 19}, {'name': 'Alice', 'grade': 90, 'age': 20}, {'name': 'Bob', 'grade': 90, 'age': 21}]
```

Hier sortieren wir eine Liste von Studenten-Dictionaries zuerst nach der Note (grade). Wenn zwei Studenten die gleiche Note haben, wird als nächstes das Alter (age) für die Sortierung verwendet.

Durch die Verwendung von Tupeln als Rückgabewerte in der Lambda-Funktion können wir eine sehr flexible und mehrstufige Sortierung erzielen.

M323-LU04