Kapitel 14.2: Sortierung – Glossar

Der Kleiner-Als Operator: **lt**(self, other)

An den doppelten Unterstrichen vor und nach dem Methodennamen erkennt man, dass es sich um eine vordefinierte Methode handelt. In diesem Fall ist es der Vergleichsoperator Kleiner-Als (lt = less than).

Hat man 2 Objekte einer Klasse, kann man die beiden Objekte mit < miteinander vergleichen, wenn man die **lt** Methode in der Klasse definiert hat. Der Kleiner-Operator liefert `True` zurück, wenn das aufrufende Objekt (self) echt kleiner ist als das Objekt, welches als Parameter übergeben wird.

```
obj1 = Klasse()
obj2 = Klasse()
if obj1 < obj2:
    print("Objekt1 ist kleiner als Objekt2")
```

Es gibt noch andere Vergleichsoperatoren, die auf diese Weise für Klassen definiert werden können:

`__le__` → kleiner gleich (less or equal) `__gt__` → größer als (greater than) `__ge__` → größer gleich (greater or equal) `__eq__` → gleich (equals) `__ne__` → ungleich (not equal)

Der Kleiner-Als-Operator ist aber besonders wichtig, weil er z. B. implizit zum Sortieren von Listen herangezogen wird.

Listen sortieren

list.sort()

sortiert eine Liste aufsteigend unter Verwendung des Kleiner-Als-Operators.

- Bei Strings, Integers und Floats ist der Operator automatisch definiert.
- Bei Klassen muss man die `__lt__` Methode selbst implementieren.
- Ist der Kleiner-Als-Operator in der Klasse nicht definiert, gibt es eine Fehlermeldung bei der Ausführung des Programms.

list.sort(reverse = True)

möchte man eine Liste absteigend (also andersrum, *reverse*) sortieren, dann gibt man der Methode einfach den Parameter `reverse = True` mit.

list.sort(key = lambda x: x.method())

Man kann auch andere Methoden der Klasse benutzen, um die Liste zu sortieren. Für jedes Element der Liste wird die Methode `method` aufgerufen, der man natürlich noch weitere Parameter mitgeben kann. Die Methode `method` liefert typischerweise einen Integer- oder Float-Wert zurück, sodass jedes Element der Liste mit einem Wert verbunden wird. Über diese Werte kann man die Elemente der Liste jetzt sortieren.

Natürlich kann auch hier mit `reverse=True` absteigend sortiert werden. `list.sort(key = lambda x: x.method(), reverse = True)`

list.sort(key = lambda x: function(x))

Statt einer Methode kann man auch eine Funktion (oder eine Methode einer anderen Klasse) aufrufen und `x` als Parameter übergeben. Auch die Funktion aufgerufene Funktion sollte einen Zahlenwert zurückgeben, der dann für die Sortierung verwendet wird.

random.shuffle(list)

Eine Liste kann man mischen, also in eine zufällige Reihenfolge bringen, indem man aus dem random-Modul die `shuffle`-Funktion aufruft.