Der Datentyp list

Ein sequentieller, veränderlicher Datentyp

Definition

Eine Liste ist eine geordnete Menge von Werten, wobei jeder Wert mit einem Index angesprochen wird.

Die Werte, die die Liste bilden, werden ihre Elemente genannt. Listen haben Ähnlichkeiten mit Strings, die stets geordnete Mengen von Zeichen sind.

Die Elemente von Listen können dagegen beliebige Objekte sein. Listen und Strings und andere Dinge, die sich wie geordnete Mengen verhalten, werden als Sequenzen bezeichnet.

```
x = 5
a = [1, 3, 'new york', x]
```

Auf Listenelemente zugreifen

Listenelemente können über den Index adressiert werden

```
fruit = ["banana", "apple", "quince"]

print(fruit[0])
banana
```

Verschachtelte Listen

Listen lassen sich auch problemlos verschachteln.

```
a = [[0,1], [1,1], [0,3]]
```

Ein-dimensionale Liste (Vektor)

```
n = [1, 2, 3]
```

Zwei-dimensionale Liste (Matrix)

```
n = [
    [0, 2, 4],
    [6, 8, 10]
]
```

drei-dimensionale Liste (Tensor)

```
n = [[[0, 2, 4], [1, 2, 3]], [[3, 3, 3], [2, 2, 2]]]
```

Unterschied Listen - Strings

Listen sind im Gegensatz zu Strings veränderlich

```
fruit = ["banana", "apple", "quince"]
fruit[0] = "pear"
["pear", "apple", "quince"]
```

mit Strings geht das nicht!

```
string1 = "Welcome to the Blue Mountains!"
string1[0] = "B" # Fehler
```

Listen Methoden

Es gibt in Python zahllose Methoden, die sich auf Listen anwenden können. Listen sind veränderlich, wir können also im Gegensatz zu Strings die Listen direkt manipulieren.

```
a = [1, 2, 3]
a.append(4)
```

es wurde ein Element an die Liste angehängt

wichtige Listen Methoden

a = [2, 4, 5, 7, "Woody"]

a.append(4) => Element 4 zur Liste hinzufügen

a.clear() => löscht alle Elemente der Liste

a.count() => Anzahl der Elemente

a.index("Woody") => gibt den Index zurück, an dem Element mit Wert "Woody" steht

a.pop() => gibt das letzte Element zurück und löscht es aus der Liste

a.pop(2) => gibt das Element an Index 2 zurück und löscht es aus der Liste

a.remove("Woody") => löscht das erste gefundene Element mit Wert "Woody"

a.reverse() => Liste umkehren

a.sort() => sortiert die Liste aufsteigend

a.sort(reverse=True) => sortiert die Liste absteigend

Referenzen

Weist man einer Variablen eine Liste zu, speichert die Variable nicht die Liste an sich, d.h. die Werte, sondern nur die **Referenz auf die Liste.**

a = [1,2,3]

Diese Tatsache, die alle verändlichen Datentypen betrifft, später zum Beispiel auch eigene Objekte, ist von fundamentaler Bedeutung.

So ist zum Beispiel ein einfaches Kopieren von a nicht möglich.

b = a

In b ist jetzt ebenfalls die Referenz auf die Liste gespeichert. Wird a verändert, ändert sich auch b. Eigentlich ändert sich nur die Referenz und die ist in beiden Fällen dasselbe Objekt.

Listen klonen

Wenn wir eine Liste verändern möchten und auch ein Kopie der originalen Liste behalten möchten, müssen wir die Möglichkeit haben, eine Kopie der Liste zu erzeugen und nicht nur einen Verweis auf sie zu haben.

Dieser Vorgang wird **Klonen** genannt.

Der einfachste Weg eine Liste zu klonen ist, den Abschnitt-Operator zu verwenden:

```
a = [1, 2, 3]

b = a[:]
```

b referenziert jetzt auf eine neue Liste und kann verändert werden, ohne a zu verändern.

Verschachtelte Listen klonen

Das Klonen von Listen, die Listen oder andere komplexe Datentypen enthalten, geht nicht mit dem [:] Operator, da immer nur die erste Ebene der Listen geklont wird.

Für dieses Problem gibt es das Modul copy.

```
from copy import deepcopy
a = [1, 2, 2]
b = deepcopy(a)
```

Listen sortieren

Listen lassen sich mit der Listen-Methode sort() sortieren. Listen lassen sich auch absteigend sortieren.

```
a = [3, 5, 1]
a.sort(reverse=True)
```

Verschachtelte Listen zu sortieren ist aufwändiger.