Datenstrukturen in Python: Listen

Jacques Mock Schindler

13.11.2024

Eine Datenstruktur kann allgemein als Bündelung von Daten und Funktionalitäten verstanden werden. Wie das funktioniert, soll am Beispiel von Python-Listen gezeigt werden.

Python-Listen können dazu verwendet werden, einer Variabel mehrere Werte zuzuweisen. Die Werte behalten dabei grundsätzlich ihre ursprüngliche Reihenfolge bei. Listen werden in der Regel nach der Art Ihrer Elemente benannt. Wenn eine Python-Liste von Namen erstellt wird, wird sie beispielsweise der Variablen `names` zugewiesen. Die Variable wird als Kennzeichen dafür, dass sie auf eine Python-Liste verweist, in den Plural gesetzt.

Hintergrund der Datenstruktur Liste in Python (Exkurs)

Im Verlauf der Entwicklung haben sich für die Ablage einer Datenreihe die Datenstrukturen Liste und Array herausgebildet. Die beiden Datenstrukturen weisen Ähnlichkeiten auf, unterscheiden sich aber auch in wesentlichen Punkten. Ähnlich sind sie sich darin, dass sie die Daten in einer vorgegebenen Reihenfolge speichern.
Unterschiede bestehen bezüglich der Anzahl Elemente, die sie aufnehmen können, sowie bezüglich des Zugriffs auf die einzelnen Elemente. Eine Liste kann grundsätzlich eine unbeschränkte Anzahl von Elementen aufnehmen. Begrenzt ist ihre Länge lediglich durch den physischen Speicherplatz des Computers. Im Gegensatz dazu muss bei einem Array grundsätzlich von Anfang an festgelegt werden, wie viele Elemente es aufnehmen kann.
Wenn auf ein Element einer Liste zugegriffen werden soll, muss in einer Liste am Beginn der Liste gestartet werden und dann Element für Element durch die Liste durchgegangen werden, bis man beim gewünschten Element angekommen ist. In einem Array ist der direkte Zugriff auf jedes Element möglich.

Die Datenstruktur der Liste in Python verbindet aus der Sicht des Programms die Eigenschaften von Listen und Arrays. Eine Python-Liste kann jederzeit zusätzliche Elemente aufnehmen oder nicht mehr benötigte Elemente können gelöscht werden. Trotzdem kann mit Hilfe eines Index' direkt auf jeden Wert

zugegriffen werden. Aus technischer Sicht handelt es sich daher bei Python-Listen in Python nicht um Listen im allgemeinen Sinn der Informatik, sondern um dynamische Arrays.

Python-Listen erstellen

Am einfachsten werden Python-Listen durch die Eingabe von konkreten Werten des betrachteten Datentypen erstellt. Dazu werden die Werte der Python-Liste durch Kommas getrennt in eckigen Klammern geschrieben:

```
\NormalTok{numbers }\OperatorTok{=}\NormalTok{ [}\DecValTok{1}\NormalTok{,
}\DecValTok{8}\NormalTok{, }\DecValTok{3}\NormalTok{]}
```

Grafisch kann man sich diese Python-Liste folgendermassen vorstellen:

Alternativ kann eine Liste mit Gemüsen erstellt werden:

```
\NormalTok{vegetables }\OperatorTok{=}\NormalTok{
[}\StringTok{\textquotesingle{}Spinat\textquotesingle{}}\NormalTok{,
}\StringTok{\textquotesingle{}Sellerie\textquotesingle{}}\NormalTok{,
}\StringTok{\textquotesingle{}Blumenkohl\textquotesingle{}}\NormalTok{]}
```

Auf Elemente einer Python-Liste zugreifen

Die Elemente einer Python-Liste sind geordnet und indexiert. Geordnet heisst, dass die Python-Liste die Reihenfolge der Elemente so behält, wie sie erstellt wird - zumindest solange die Python-Liste nicht verändert wird. Indexiert heisst, dass jedes Element einer Python-Liste eine Nummer erhält. Allerdings beginnt man mit Null zu zählen. Das bedeutet, dass der `Sellerie` in der Python-Liste `vegetables` den Index 1 hat.

Um das zweite Element der Python-Liste auszugeben, muss Python nach dem Element mit dem Index 1 der entsprechenden Python-Liste gefragt werden:

```
\BuiltInTok{print}\NormalTok{(vegetables[}\DecValTok{1}\NormalTok{])}
```

gibt

```
\NormalTok{Sellerie}
```

Python-Listen haben noch einen zweiten Index. Dieser beginnt beim letzten Element mit -1 und zählt mit einem negativen Vorzeichen bis zum ersten Element hoch. Auf das Element Blumenkohl kann also auch mit `vegetables[-1]` zugegriffen werden.

Über Python-Listen iterieren

Um der Reihe nach auf die einzelnen Elemente einer Liste zuzugreifen, kann mit einer `for` Schlaufe über die Liste iteriert (darüber gelaufen) werden. Das folgende Listing zeigt, wie über die Gemüseliste iteriert werden kann.

```
\ControlFlowTok{for}\NormalTok{ vegetable }\KeywordTok{in}\NormalTok{ vegetables:}
    \BuiltInTok{print}\NormalTok{(vegetable)}
```

Für dieses Vorgehen hat sich eingebürgert, dass man die laufende Variabel (`vgetable`) im Singular der Variabel der Liste (`vegetables`) benennt.
Innerhalb der Schlaufe nimmt die laufende Variabel der Reihe nach jeden Wert der Python Liste an und verarbeitet ihn gemäss den Programmanweisungen im Körper der Schlaufe (hier `print(vegetable)`).

Python-Listen mit list comprehensions erstellen

Python-Listen können auch mit einer sogenannten *list comprehension* erstellt werden. Das folgende Listing zeigt ein einfaches Beispiel einer *list comrprehension*.

```
\NormalTok{example\_numbers }\OperatorTok{=}\NormalTok{ [i }\ControlFlowTok{for}\NormalTok{ i }\KeywordTok{in} \BuiltInTok{range}\NormalTok{(}\DecValTok{1}\NormalTok{,}\DecValTok{21}\NormalTok{)]}
```

Die Datenstruktur enthält also 20 Variablen mit den Werten `example_numbers[0]` = 1, `example_numbers[1]` = 2, ..., `example_numbers[19]` = 20.

Grafisch dargestellt sieht das wie unten dargestellt aus.

Dies ermöglicht es uns, alle Variablen einzeln anzusprechen.

Eine *list comprehension* funktioniert ähnlich, wie die Beschreibung einer Menge in der Mathematik (

$$\{x \in \mathbb{N} \mid 1 \geq x > 21\}$$

).

Der Teil

```
\ControlFlowTok{for}\NormalTok{ i }\KeywordTok{in} \BuiltInTok{range}\NormalTok{(}\DecValTok{1}\NormalTok{,}\DecValTok{21}\NormalTok{)}
```

entspricht dabei der Bedingung (oben rot dargestellt) in der Beschreibung einer Menge. 1 ist der Startwert bei dem die Variabel `i` zu zählen beginnt. Man nennt eine Variable wie `i` eine Laufvariable. 21 ist die obere Grenze. Die Grenze selber wird nicht mitgezählt.

Ein etwas komplexeres Beispiel resultiert, wenn die *list comrehension* mit einer Bedingung verknüpft wird.

```
\NormalTok{even\_example\_numbers = [i for i in range(1,21) if i \% 2 == 0]}
```

In die Python-Liste `even_example_numbers` werden nur jene Werte für `i` aufgenommen, welche die Bedingung `i % 2 == 0` erfüllen. Das sind in diesem Fall die geraden Zahlen von 1 bis (und mit) 20. Dies entspricht der Mengenbeschreibung

$$\{x \in \mathbb{N} \mid 1 \geq x > 21, x \bmod 2 = 0\}$$

.

Anstelle der Funktion `range()` kann in einer *list comprehension* auch eine bereits bestehende Python-Liste verwendet werden. So kann mit einer *list comprehension* eine neue Liste erstellt werden, die aus Elementen einer Liste besteht, welche bestimmte Bedingungen erfüllen. Das folgende Listing soll dies verdeutlichen.

```
\NormalTok{hundert = [i for i in range(1,101)]}
\NormalTok{hundert\_mit\_bedingungen = [x for x in hundert if x \% 7 == 0
and x \textgreater{} 50]}
```

Die zweite *list comprehension* erstellt hier eine Liste mit Zahlen, die durch 7 Teilbar **und** grösser als 50 sind.

Übungen

Die obigen Erklärungen können auf dem hier verlinkten Jupyter Notebook eingeübt werden. Hier finden Sie eine Musterlösung zum Arbeitsblatt.