DS 11 – Listen: viele Werte unter einem Namen

Informatik 10 – MINT-Zweig | Datenstrukturen

Lernziele

Ich kann

- eine Liste (ein Feld) als Datenstruktur beschreiben, die mehrere Elemente der Reihe nach zu einer Einheit zusammenfasst,
- zwischen dem **Index** und dem **Element** an dieser Stelle unterscheiden,
- über den Index lesend und schreibend auf einzelne Listenelemente zugreifen,
- Elemente mit `append`, `insert`, `pop` und `remove` hinzufügen und entfernen,
- mit einer Schleife alle Elemente einer Liste durchlaufen,
- die Fehlermeldung `IndexError` erklären und den Fehler beheben.

🗣️ Sprachhilfe

- *„Die 5 ist an Stelle 1."* → „Das Listenelement am Index 1 hat den Wert 5."
- *„Ich mach die 3 zur 5."* → „Das Listenelement am Index 1 wird von 3 auf 5 verändert."
- *„Die 6 kommt hinten dran."* → „Mit `append` wird das Element 6 am Ende der Liste hinzugefügt."
- *„Da kommt ein Fehler."* → „Beim Zugriff auf die ungültige Listenposition 5 erhalte ich die Fehlermeldung `IndexError`."

1 Warum Listen?

Speicherst du fünf Noten in `note1` bis `note5`, hat das zwei Schwächen: Kommt eine sechste Note dazu, brauchst du eine neue Variable **und** musst jede Rechnung ändern. Außerdem kann keine Schleife über `note1` bis `note5` laufen, denn für Python sind das fünf unabhängige Namen.

Mit einer **Liste** speicherst du alle Noten in **einer** Variablen:

```
noten = [2, 3, 1, 4, 2]
print(noten)
print(type(noten))

staedte = ["Saarbrücken", "Trier", "Mainz"]
leer = []
print(staedte)
print(leer)
```

```
[2, 3, 1, 4, 2]
<class 'list'>
['Saarbrücken', 'Trier', 'Mainz']
[]
```

- Eine Liste schreibt man in **eckigen Klammern**, die Werte werden durch Kommas getrennt. Der Datentyp heißt `list`.
- Die Werte heißen **Elemente**. Sie stehen in fester Reihenfolge, wie die Waggons eines Zuges: Eine Liste ist eine **lineare** Datenstruktur.
- Zeichenketten in einer Liste zeigt Python mit `'…'` an. Das ist nur die Anzeige.
- Die leere Liste `[]` legt man an und füllt sie später nach und nach.

In anderen Sprachen wie Java heißt diese Datenstruktur **Feld** (*array*) und hat meist eine feste Länge. Eine Python-Liste kann wachsen und schrumpfen.

> 📌 **Merksatz:** Eine **Liste** (in Python: `list`) fasst mehrere **Elemente zu einer Einheit** zusammen, z. B. `noten = [2, 3, 1, 4, 2]`.

2 Index und Element

Denk an die Schließfächer in deiner Schule: Jedes Fach hat eine **Nummer** an der Tür, und darin liegt ein **Inhalt**. Wer „Fach 2“ sagt, meint die Nummer, nicht den Inhalt. Bei einer Liste heißt die Nummer **Index** (ab 0 gezählt wie bei Zeichenketten), der Inhalt **Element**.

Index	0	1	2	3	4
Element	2	3	1	4	2

Lesender Zugriff

Den Index schreibst du in eckigen Klammern hinter den Namen der Liste.

```
noten = [2, 3, 1, 4, 2]
print(noten[0])
print(noten[2])
print(len(noten))
print(noten[len(noten) - 1])
print(noten[-1], noten[-2])
```

```
2
1
5
2
2 4
```

- `noten[2]` liefert die `1`, also das **dritte** Element. Die Zahl in den eckigen Klammern ist der Index, nicht der Wert!
- Das Element `2` kommt zweimal vor: am Index 0 und am Index 4.
- Die Liste hat fünf Elemente, das letzte hat den Index `len(noten) - 1`, also 4. Kürzer: `noten[-1]`.

> 📌 **Merksatz:** Jedes Element besitzt einen **Index**, beginnend bei `0`. Der Index ist **nicht** dasselbe wie der gespeicherte Wert! **Lesender Zugriff:** `liste[index]`. Die Anzahl der Elemente liefert `len(liste)`.

🔮 Übung 1 – Index oder Element?

Gegeben ist `zahlen = [3, 0, 4, 1, 2]`. Bestimme **ohne Computer** die Werte und trage sie in die Tabelle ein. An welchem Index steht das Element `4`? Prüfe danach in der Codezelle.

`zahlen[1]`	`len(zahlen)`	`zahlen[-1]`	`zahlen[zahlen[0]]`	Index von `4`

```
zahlen = [3, 0, 4, 1, 2]
# TODO: Prüfe deine Vorhersage mit print(...)
```

3 Schreibender Zugriff: ein Element ändern

Ein Schließfach kann man ausräumen und neu füllen, die Nummer bleibt gleich. Steht der Zugriff mit Index **links** vom `=`, wird das Element **überschrieben**.

```
1 noten = [2, 3, 1, 4, 2]
2 noten[1] = 5
3 print(noten)
4 noten[2] = noten[2] + 1
5 print(noten)
```

```
[2, 5, 1, 4, 2]
[2, 5, 2, 4, 2]
```

- **Zeile 2:** Das Element am Index 1 bekommt den Wert `5`. Der alte Wert `3` ist weg.
- **Zeile 4:** Wie bei `x = x + 1` wird zuerst rechts gerechnet: `noten[2]` ist `1`, plus 1 ergibt `2`.

Die Länge der Liste ändert sich dabei nicht.

> 📌 **Merksatz: Schreibender Zugriff:** `liste[index] = neuer_wert` überschreibt das Element an dieser Position.

⚠️ Stolpersteine

Führe die drei Zellen aus und lies jeweils die letzte Zeile der Fehlermeldung.

```
1 # 1. Eine Zeichenkette kann man nicht stellenweise ändern.
2 wort = "Hallo"
3 wort[0] = "B"
```

```
Traceback (most recent call last):
  File "<Zelle 5>", line 3, in <module>
TypeError: 'str' object does not support item assignment
```

```
1 # 2. Den Index 5 gibt es bei fünf Elementen nicht.
2 noten = [2, 3, 1, 4, 2]
3 print(noten[5])
```

```
Traceback (most recent call last):
  File "<Zelle 6>", line 3, in <module>
IndexError: list index out of range
```

```
1 # 3. Auch durch eine Zuweisung wird die Liste nicht länger.
2 noten = [2, 3, 1, 4, 2]
3 noten[5] = 1
```

```
Traceback (most recent call last):
  File "<Zelle 7>", line 3, in <module>
IndexError: list assignment index out of range
```

1. `TypeError: 'str' object does not support item assignment`: Zeichenketten kann man nur lesen. Listen sind dagegen **veränderbar**.
2. `IndexError: list index out of range`: Gültig sind nur die Indizes 0 bis 4 (und −1 bis −5).
3. `IndexError: list assignment index out of range`: Zum Hinzufügen braucht man `append` (Teil 4).

Häufigste Ursache: Man zählt aus Gewohnheit ab 1 und schreibt `liste[len(liste)]`. Prüfe bei einem `IndexError` immer: Wie lang ist die Liste gerade, und welchen Index habe ich benutzt?

> 📌 **Merksatz:** Ein Zugriff auf einen ungültigen Index führt zu einem `IndexError`. Bei einer Liste mit `n` Elementen sind nur die Indizes `0` bis `n - 1` gültig.

4 Listen verändern: Elemente hinzufügen und entfernen

Eine Liste kann wachsen und schrumpfen. Dafür gibt es **Methoden**, die man mit einem Punkt anhängt, wie `text.upper()` bei Zeichenketten.

`append` und `insert`

`liste.append(wert)` hängt ein Element **am Ende** an. `liste.insert(index, wert)` fügt es an einer **beliebigen Stelle** ein: Alle Elemente ab dieser Stelle rücken eine Position nach rechts.

```
noten = [2, 5, 1, 4, 2]
noten.append(6)
print(noten)

einkauf = ["Brot", "Milch", "Eier"]
einkauf.insert(1, "Käse")
print(einkauf)
```

```
[2, 5, 1, 4, 2, 6]
['Brot', 'Käse', 'Milch', 'Eier']
```

Die 6 steht jetzt am Index 5. Der Käse steht am Index 1, die Milch ist von Index 1 auf Index 2 gerückt.

`pop` und `remove`

`liste.pop(index)` entfernt das Element **am Index**, ohne Index das **letzte**. `pop` gibt das entfernte Element zurück, so wie eine Funktion mit `return`. `liste.remove(wert)` sucht ein **Element mit diesem Wert** und entfernt das erste Vorkommen.

```
einkauf = ["Brot", "Käse", "Milch", "Eier"]
letztes = einkauf.pop()
print(letztes)
print(einkauf)
erstes = einkauf.pop(0)
print(erstes)
print(einkauf)

einkauf = ["Brot", "Käse", "Milch", "Käse"]
einkauf.remove("Käse")
print(einkauf)
```

```
Eier
['Brot', 'Käse', 'Milch']
Brot
['Käse', 'Milch']
['Brot', 'Milch', 'Käse']
```

Methode	Wirkung	in den Klammern steht
`liste.append(wert)`	hängt `wert` ans Ende an	ein Element
`liste.insert(i, wert)`	fügt `wert` am Index `i` ein	ein Index und ein Element
`liste.pop()`	entfernt das letzte Element und gibt es zurück	nichts
`liste.pop(i)`	entfernt das Element am Index `i` und gibt es zurück	ein Index
`liste.remove(wert)`	entfernt das erste Element mit dem Wert `wert`	ein Element

⚠️ **Stolperstein:** Ist der Wert nicht in der Liste, meldet `remove` einen `ValueError: list.remove(x): x not in list`. Aus einer leeren Liste kann `pop` nichts entfernen: `IndexError: pop from empty list`.

> 📌 **Merksatz:** `append` hängt an, `insert` fügt ein, `pop` entfernt am **Index**, `remove` entfernt ein **Element** mit einem bestimmten Wert.

🔮 Übung 2 – Listen-Vorhersage

Lies das Programm, **ohne** es auszuführen. Zeichne die Liste `tiere` nach jeder Zeile als Reihe von Kästchen mit dem Index darüber. Gib dann die Ausgabe an und prüfe am Computer.

```
tiere = ["Hund", "Katze", "Maus"]
tiere.append("Igel")
tiere[0] = "Pferd"
tiere.insert(2, "Esel")
tiere.remove("Katze")
x = tiere.pop()
print(tiere)
print(x)
print(len(tiere))
```

```
['Pferd', 'Esel', 'Maus']
Igel
3
```

Die Liste nach jeder Zeile und meine Vorhersage:

…

5 Listen mit Schleifen durchlaufen

Über die Elemente einer Liste läufst du wie über die Zeichen einer Zeichenkette. Brauchst du auch den **Index**, läufst du mit `range(len(liste))` über alle gültigen Indizes.

```
staedte = ["Saarbrücken", "Trier", "Mainz", "Koblenz"]
for stadt in staedte:
    print("Ich fahre nach", stadt)

for i in range(len(staedte)):
    print(i, staedte[i])
```

```
Ich fahre nach Saarbrücken
Ich fahre nach Trier
Ich fahre nach Mainz
Ich fahre nach Koblenz
0 Saarbrücken
1 Trier
2 Mainz
3 Koblenz
```

🧪 **Probiere es aus:** Ändere die zweite Schleife zu `for i in range(1, len(staedte) + 1):`. Was gibt das Programm aus, bevor es abbricht? Welche Indizes erzeugt `range` jetzt, und welcher davon ist ungültig?

Eine Liste mit einer Schleife füllen

Ein sehr häufiges Muster: mit einer **leeren Liste** beginnen und in der Schleife ein Element nach dem anderen anhängen.

```python
quadratzahlen = []
for zahl in range(1, 6):
    quadratzahlen.append(zahl * zahl)
print(quadratzahlen)
```

```
[1, 4, 9, 16, 25]
```

Stünde `print` eingerückt in der Schleife, würde die Liste fünfmal ausgegeben, jedes Mal ein Stück länger.

> 📌 **Merksatz:** `for element in liste:` durchläuft die Elemente, `for i in range(len(liste)):` durchläuft die Indizes.

📝 Übungen: Schritt für Schritt selbstständig

Vorgemacht ist das Muster **Eine Liste mit einer Schleife füllen** (`quadratzahlen`). Bei Übung 3 füllst du Lücken `___` aus, bei Übung 4 steht ein Gerüst aus Kommentaren, die Zusatzaufgabe schreibst du allein.

📝 Übung 3 – Einkaufsliste (Lücken füllen)

Lege eine leere Liste an. Frage nacheinander 5 Artikelnamen ab (mit `input()` in einer Schleife) und füge sie mit `append()` hinzu. Gib die vollständige Liste am Ende aus.

Lückencode

```python
einkaufsliste = ___
for i in range(___):
    artikel = input("Artikel eingeben: ")
    einkaufsliste.___(artikel)
print(einkaufsliste)
```

📝 Übung 4 – Index vs. Wert (mit Gerüst)

Gegeben ist `staedte = ["Saarbrücken", "Trier", "Mainz", "Koblenz"]`.

1. Gib den Wert am Index 2 aus.
2. Ändere den Wert am Index 0 zu `"Saarlouis"` und gib die Liste aus.
3. Gib den **Index** der Stadt `"Mainz"` aus.

> 🧰 **Werkzeug:** `liste.index(wert)` liefert den Index des ersten Elements mit diesem Wert. Ist der Wert nicht in der Liste, kommt ein `ValueError`. In DS 12 programmierst du so eine Suche selbst.

```python
staedte = ["Saarbrücken", "Trier", "Mainz", "Koblenz"]

# 1. den Wert am Index 2 ausgeben

# 2. den Wert am Index 0 zu "Saarlouis" ändern und die Liste ausgeben

# 3. den Index von "Mainz" ausgeben (erwartet: 2)
```

🚀 Zusatzaufgabe – Liste umkehren (selbstständig)

Erzeuge aus `zahlen = [1, 2, 3, 4, 5]` mit einer Schleife eine **neue** Liste mit den Elementen in umgekehrter Reihenfolge. Verwende keine eingebaute Funktion zum Umkehren. Kontrolle: `[5, 4, 3, 2, 1]`.

> 💡 **Tipp:** `range(4, -1, -1)` erzeugt 4, 3, 2, 1, 0. Die mittlere Zahl ist die Grenze, die **nicht** mehr erreicht wird.

```python
zahlen = [1, 2, 3, 4, 5]

# Zusatzaufgabe – Liste per Schleife umkehren
```

📌 Das Wichtigste in Kürze

- Eine **Liste** (ein **Feld**) fasst mehrere **Elemente** in einer festen Reihenfolge zu einer Einheit zusammen: `noten = [2, 3, 1, 4, 2]`.
- Jedes Element hat einen **Index**, beginnend bei 0. Index und Element sind verschiedene Dinge: `noten[2]` ist das Element am Index 2.
- **Lesen:** `liste[i]`. **Schreiben:** `liste[i] = wert`. `liste[-1]` ist das letzte Element.
- `len(liste)` liefert die Anzahl der Elemente. Gültige Indizes sind `0` bis `len(liste) - 1`, sonst kommt ein `IndexError`.
- `append` hängt an, `insert` fügt ein, `pop` entfernt am Index, `remove` entfernt ein Element mit einem bestimmten Wert.
- `for element in liste:` durchläuft die Elemente, `for i in range(len(liste)):` durchläuft die Indizes.

Zum Weiterüben: `DS11_Weitere_Uebungen.ipynb`.