

PCEP-30-02 3.1 -
Daten mit Listen
sammeln und
verarbeiten

Lernziele (PCEP 3.1)

- Verstehen, wie Listen in Python funktionieren
- Erstellung und Initialisierung von Listen
- Zugriff auf Listenelemente mit Indexing und Slicing
- Wichtige Listenmethoden (`append()`,
`insert()`, `remove()`, `index()`, etc.)
- Iteration durch Listen mit `for`-Schleifen
- Listenoperationen wie Kopieren, Sortieren

Übersicht der Themen

1. Einführung in Listen in Python
2. Erstellen und Initialisieren von Listen
3. Zugriff auf Listenelemente – Indexing und Slicing
4. Die `len()`-Funktion – Anzahl der Elemente bestimmen
5. Wichtige Listenmethoden (`append()`, `insert()`, `remove()`, `index()`, etc.)

7. Listenoperationen: Kopieren, Sortieren, Löschen
(`del`, `sorted()`)
8. Listen in Listen – Matrizen und verschachtelte Strukturen
9. Listenkomprehension (`list comprehensions`)
10. `in` und `not in` Operatoren zur Überprüfung von Listenelementen
11. Häufige Fehler und Best Practices

1. Einführung in Listen in Python

- Eine Liste ist eine **sammlung** von Elementen, die geordnet und veränderbar ist.
- Listen sind **iterierbar** und können verschiedene Datentypen enthalten.

Beispiel für eine Liste:

```
fruits = ["Apfel", "Banane", "Kirsche"]
print(fruits)
```

Ausgabe:

```
[ 'Apfel', 'Banane', 'Kirsche' ]
```

2. Erstellen und Initialisieren von Listen

Listen können auf verschiedene Weisen erstellt werden:

```
# Eine leere Liste
empty_list = []

# Liste mit Werten
numbers = [1, 2, 3, 4, 5]

# Liste mit gemischten Datentypen
mixed = [42, "Python", True]
```

3. Zugriff auf Listenelemente – Indexing und Slicing

Elemente einer Liste können über **Indexing** und **Slicing** abgerufen werden.

```
fruits = ["Apfel", "Banane", "Kirsche"]
print(fruits[0]) # Erstes Element (Apfel)
print(fruits[-1]) # Letztes Element (Kirsche)
```

Slicing (Ausschnitt einer Liste nehmen):

```
numbers = [0, 1, 2, 3, 4, 5]
print(numbers[1:4]) # Ausgabe: [1, 2, 3]
```

Blitzfrage

Wie greift man auf das letzte Element einer Liste zu?

- A) `list[-1]`
- B) `list[0]`
- C) `list[len(list)]`
- D) `list[last]`

Blitzantwort

Richtige Antwort: A) `list[-1]`

- A) Richtig – Negative Indizes beginnen am Ende der Liste.
- B) Falsch – `List[0]` ist das erste Element.
- C) Falsch – `List[Len(List)]` gibt einen IndexError, da Listen bei 0 beginnen.
- D) Falsch – `Last` ist keine vordefinierte Variable in Python.

Erklärung:

Python erlaubt das **Zugreifen auf das letzte Element mit `list[-1]`**, was besonders nützlich ist, wenn die Länge der Liste nicht bekannt ist.

4. Die `len()`-Funktion – Anzahl der Elemente bestimmen

Die `len()`-Funktion gibt die Anzahl der Elemente in einer Liste zurück.

```
numbers = [10, 20, 30, 40]
print(len(numbers)) # Ausgabe: 4
```

Nützlich für:

- Überprüfen der Listengröße vor einer Iteration
- Dynamische Kontrolle von Indexen

5. Wichtige Listenmethoden (`append()`, `insert()`, `remove()`, `index()`, etc.)

Listen haben viele eingebaute Methoden:

```
fruits = ["Apfel", "Banane"]

# Neues Element hinzufügen
fruits.append("Kirsche")

# An einer bestimmten Position einfügen
fruits.insert(1, "Orange")

# Element entfernen
fruits.remove("Banane")

# Index eines Elements abrufen
```

```
pos = fruits.index("Apfel")
```

```
print(fruits)
```

Erwartete Ausgabe:

```
[ 'Apfel', 'Orange', 'Kirsche' ]
```

6. Iteration durch Listen mit for-Schleifen

Mit einer **for**-Schleife können alle Elemente einer Liste durchlaufen werden:

```
numbers = [1, 2, 3, 4, 5]

for num in numbers:
    print(num) # Gibt alle Zahlen der Reihe nach aus
```

Alternative mit **range()**:

```
for i in range(len(numbers)):
    print(numbers[i]) # Zugriff über Index
```

7. Listenoperationen: Kopieren, Sortieren, Löschen (del, sorted())

Listen bieten verschiedene Methoden zur **Bearbeitung und Manipulation** von Daten.

Kopieren einer Liste

```
original = [1, 2, 3]
copy_list = original.copy() # Oder: copy_list = original[:]

print(copy_list) # [1, 2, 3]
```

Sortieren einer Liste

```
numbers = [5, 3, 8, 1]
sorted_numbers = sorted(numbers)    # Gibt eine neue sortierte Liste zu

numbers.sort()    # Sortiert die Liste direkt

print(sorted_numbers)  # [1, 3, 5, 8]
print(numbers)    # [1, 3, 5, 8]
```

Löschen eines Elements mit del

```
fruits = ["Apfel", "Banane", "Kirsche"]
del fruits[1]    # Entfernt "Banane"

print(fruits)  # ['Apfel', 'Kirsche']
```

Blitzfrage

Welche Methode gibt eine **neue sortierte Liste** zurück, ohne die Original-Liste zu ändern?

- A) `list.sort()`
- B) `sorted(list)`
- C) `list.reverse()`
- D) `list.append()`

Blitzantwort

Richtige Antwort: B) `sorted(list)`

- A) Falsch – `sort()` verändert die Original-Liste.
- B) Richtig – `sorted()` gibt eine neue sortierte Kopie der Liste zurück.
- C) Falsch – `reverse()` kehrt nur die Reihenfolge der Liste um.
- D) Falsch – `append()` fügt ein neues Element zur Liste hinzu.

Erklärung:

`sorted(list)` erzeugt eine **neue Liste** mit sortierten Werten, ohne die Original-Liste zu verändern.

8. Listen in Listen – Matrizen und verschachtelte Strukturen

Python-Listen können **andere Listen enthalten**, um **mehrdimensionale Strukturen** zu erstellen.

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

print(matrix[1][2]) # Zugriff auf die Zahl 6
```

9. Listenkomprehension (list comprehensions)

Mit Listenkomprehension lassen sich Listen effizient erstellen.

```
squares = [x**2 for x in range(1, 6)]
print(squares) # [1, 4, 9, 16, 25]
```

Filter in Listenkomprehension

```
even_numbers = [x for x in range(10) if x % 2 == 0]
print(even_numbers) # [0, 2, 4, 6, 8]
```

10. in und not in Operatoren zur Überprüfung von Listenelementen

Mit in und not in kann geprüft werden, ob ein Wert in einer Liste enthalten ist.

```
fruits = ["Apfel", "Banane", "Kirsche"]

print("Apfel" in fruits)  # True
print("Mango" not in fruits) # True
```

Blitzfrage

Was ist das Ergebnis von `print(5 in [1, 2, 3, 4, 5])`?

- A) True
- B) False
- C) Fehler
- D) None

Blitzantwort

Richtige Antwort: A) True

- A) Richtig – *in* überprüft, ob 5 in der Liste vorhanden ist.
- B) Falsch – Die Zahl 5 ist in der Liste enthalten.
- C) Falsch – Keine Fehlermeldung.
- D) Falsch – None wird nicht zurückgegeben.

Erklärung:

Der `in`-Operator gibt `True` zurück, wenn das

gesuchte Element in der Liste enthalten ist.

11. Häufige Fehler

1. Vergessen, eine Liste zu kopieren (unbeabsichtigte Änderungen)

```
a = [1, 2, 3]
b = a  # b zeigt auf die gleiche Liste wie a!

b.append(4)
print(a)  # Ausgabe: [1, 2, 3, 4] (unerwartet)
```

*Lösung: Nutze `a.copy()` oder `a[:]` für
eine echte Kopie.*

2. Indexfehler durch falschen Zugriff

```
numbers = [10, 20, 30]
```

Best Practices

- ✓ Nutze `sorted()` statt `sort()`, wenn die Original-Liste nicht verändert werden soll.
- ✓ Verwende **Listenkomprehension**, um Listen effizient zu erzeugen.
- ✓ Prüfe mit `in`, ob ein Element in einer Liste existiert, statt mit Schleifen zu iterieren.

12. Übungsaufgaben

Aufgabe 1: Kopieren und Sortieren einer Liste

1. Erstelle eine Liste mit Zahlen.
2. Erstelle eine sortierte Kopie mit `sorted()`.
3. Sortiere die Original-Liste mit `sort()`.
4. Vergleiche die Ergebnisse.

```
numbers = [4, 1, 7, 3]
sorted_copy = sorted(numbers)
numbers.sort()

print(sorted_copy)  # [1, 3, 4, 7]
print(numbers)    # [1, 3, 4, 7]
```

13. Challenge 1: Grundlegende Listenoperationen (`append()`, `remove()`)

Schreibe ein Programm, das eine leere Liste erstellt, **3 Namen** hinzufügt und dann den zweiten Namen entfernt.

```
names = []
names.append("Alice")
names.append("Bob")
names.append("Charlie")

names.remove("Bob")
```

```
print(names) # ['Alice', 'Charlie']
```

14. Challenge 2: Zugriff auf Listenelemente und Slicing

1. Erstelle eine Liste mit den Zahlen von 1 bis 10.
2. Gib die ersten 5 Zahlen mit Slicing aus.
3. Gib die letzten 3 Zahlen mit negativem Indexing aus.

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print(numbers[:5])    # [1, 2, 3, 4, 5]
print(numbers[-3:])  # [8, 9, 10]
```

15. Challenge 3: Iteration über Listen mit for-Schleifen

Erstelle eine Liste mit Wochentagen und iteriere mit einer for-Schleife darüber.

```
weekdays = ["Montag", "Dienstag", "Mittwoch", "Donnerstag", "Freitag"

for day in weekdays:
    print(day)
```

16. Challenge 4: Anwendung von list comprehensions

Nutze eine **Listenkomprehension**, um eine Liste mit den Quadraten der Zahlen von 1 bis 10 zu erstellen.

```
squares = [x**2 for x in range(1, 11)]
print(squares) # [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

17. Multiple-Choice Fragen

Frage 1

Welche Methode wird verwendet, um ein Element am Ende einer Liste hinzuzufügen?

- A) `insert()`
- B) `append()`
- C) `extend()`
- D) `push()`

Frage 1 - Antwort

Richtige Antwort: B) `append()`

- A) Falsch – `insert()` fügt ein Element an einer bestimmten Position ein.
- B) Richtig – `append()` fügt ein Element am Ende der Liste hinzu.
- C) Falsch – `extend()` erweitert die Liste mit einer anderen Liste, nicht mit einem einzelnen Element.
- D) Falsch – `push()` existiert nicht in Python.

Erklärung:

`append()` wird verwendet, um **ein einzelnes Element** an das **Ende der Liste** anzuhängen.

Frage 2

Welche Methode entfernt ein Element anhand seines Wertes?

- A) `delete()`
- B) `remove()`
- C) `discard()`
- D) `pop()`

Frage 2 – Antwort

Richtige Antwort: B) `remove()`

- A) Falsch – `delete()` existiert nicht in Python.
- B) Richtig – `remove()` löscht ein Element anhand seines Wertes.
- C) Falsch – `discard()` existiert nur für Sets, nicht für Listen.
- D) Falsch – `pop()` entfernt ein Element anhand des Index, nicht des Wertes.

Erklärung:

Mit `remove()` kann man ein Element direkt aus einer Liste löschen, indem man den **Wert** angibt, nicht den Index.

Frage 3

Wie viele Elemente hat die Liste `numbers = list(range(1, 11))`?

- A) 9
- B) 10
- C) 11
- D) Fehler

Frage 3 – Antwort

Richtige Antwort: B) 10

- A) Falsch – Die Liste enthält die Zahlen von 1 bis 10, also 10 Elemente.
- B) Richtig – `range(1, 11)` erzeugt die Zahlen von 1 bis einschließlich 10.
- C) Falsch – `range(1, 11)` endet vor 11, nicht bei 11.
- D) Falsch – Der Code funktioniert ohne Fehler.

Erklärung:

`range(start, stop)` erzeugt Zahlen von **start** bis **eine Zahl vor stop**, daher sind es 10 Elemente.

Frage 4

Wie überprüft man, ob ein Wert 42 in einer Liste `numbers` enthalten ist?

- A) `numbers.find(42)`
- B) `42 in numbers`
- C) `numbers.contains(42)`
- D) `check(42, numbers)`

Frage 4 – Antwort

Richtige Antwort: B) 42 in numbers

- A) Falsch – `find()` existiert nicht für Listen.
- B) Richtig – `in` überprüft, ob ein Wert in der Liste vorhanden ist.
- C) Falsch – `contains()` ist kein gültiger Listenbefehl in Python.
- D) Falsch – `check()` existiert nicht.

Erklärung:

`in` ist die einfachste Möglichkeit, um zu prüfen, ob ein Wert in einer Liste enthalten ist.

Frage 5

Was gibt `print(len([10, 20, 30, 40]))` aus?

- A) 3
- B) 4
- C) 40
- D) Fehler

Frage 5 – Antwort

Richtige Antwort: B) 4

- A) Falsch – Die Liste enthält vier Elemente, nicht drei.
- B) Richtig – `Len()` gibt die Anzahl der Elemente in einer Liste zurück.
- C) Falsch – `Len()` gibt nicht den letzten Wert der Liste zurück.
- D) Falsch – Der Code gibt keinen Fehler aus.

Erklärung:

Die Funktion `len()` zählt die Anzahl der Elemente in einer Liste.

Frage 6

Was gibt `list(range(5))` zurück?

- A) [1, 2, 3, 4, 5]
- B) [0, 1, 2, 3, 4]
- C) [0, 1, 2, 3, 4, 5]
- D) Fehler

Frage 6 – Antwort

Richtige Antwort: B) [0, 1, 2, 3, 4]

Erklärung:

`range(5)` beginnt standardmäßig bei 0 und läuft bis 4 (eine Zahl vor 5).

Frage 7

Welche Methode wird verwendet, um eine Kopie einer Liste zu erstellen?

- A) copy()
- B) clone()
- C) duplicate()
- D) new_list()

Frage 7 – Antwort

Richtige Antwort: A) `copy()`

Erklärung:

`copy()` erzeugt eine **flache** Kopie der Liste.

Frage 8

Was macht `list.pop()` ohne Parameter?

- A) Entfernt das erste Element
- B) Entfernt das letzte Element
- C) Entfernt ein zufälliges Element
- D) Erzeugt einen Fehler

Frage 8 – Antwort

Richtige Antwort: B) Entfernt das letzte Element

Erklärung:

`pop()` ohne Parameter entfernt **standardmäßig** das letzte Element aus der Liste.

Frage 9

Welche Funktion gibt eine **sortierte Kopie** einer Liste zurück?

- A) `sort()`
- B) `sorted()`
- C) `arrange()`
- D) `order()`

Frage 9 – Antwort

Richtige Antwort: B) `sorted()`

Erklärung:

`sorted()` erstellt eine neue, sortierte Liste, ohne die Original-Liste zu verändern.

Frage 10

Welche Aussage zu Listen ist korrekt?

- A) Listen sind unveränderlich
- B) Listen können verschiedene Datentypen enthalten
- C) Listen haben eine feste Größe
- D) Listen müssen nur Zahlen enthalten

Frage 10 – Antwort

Richtige Antwort: B) Listen können verschiedene Datentypen enthalten

Erklärung:

Listen in Python können **beliebige Datentypen** enthalten, z. B. Zahlen, Strings und sogar andere Listen.
