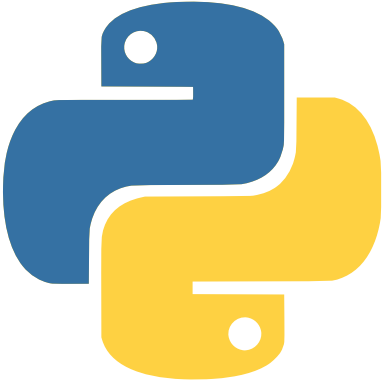
Universität Stuttgart
Projekt digit@L – BOOST. SKILLS. SUPPORT.



Programmierkurs Python

Listen und Sequenzdatentypen

Dominik Göddeke

IANS SimTech

Stiftung Innovation in der Hochschullehre

Listen und Sequenzdatentypen

Listen und Sequenzdatentypen

- **Zusammenfassung** von Variablen in **fester Reihenfolge**

Listen und Sequenzdatentypen

- **Zusammenfassung** von Variablen in **fester Reihenfolge**
- Beispiele

Listen und Sequenzdatentypen

- **Zusammenfassung** von Variablen in **fester Reihenfolge**
- Beispiele
  - Messungen einer physikalischen Größe (Temperaturverläufe etc.)

Listen und Sequenzdatentypen

- **Zusammenfassung** von Variablen in **fester Reihenfolge**
- Beispiele
  - Messungen einer physikalischen Größe (Temperaturverläufe etc.)
  - Datenpunkte für Plots (x und y Koordinaten)

Listen und Sequenzdatentypen

- **Zusammenfassung** von Variablen in **fester Reihenfolge**
- Beispiele
  - Messungen einer physikalischen Größe (Temperaturverläufe etc.)
  - Datenpunkte für Plots (x und y Koordinaten)
  - Matrizen / Vektoren

Listen und Sequenzdatentypen

- **Zusammenfassung** von Variablen in **fester Reihenfolge**
- Beispiele
  - Messungen einer physikalischen Größe (Temperaturverläufe etc.)
  - Datenpunkte für Plots (x und y Koordinaten)
  - Matrizen / Vektoren
  - Buchstabenfolgen (Zusammensetzung einzelner Zeichen zu einer Zeichenkette)

Listen und Sequenzdatentypen

- **Zusammenfassung** von Variablen in **fester Reihenfolge**
- Beispiele
  - Messungen einer physikalischen Größe (Temperaturverläufe etc.)
  - Datenpunkte für Plots (x und y Koordinaten)
  - Matrizen / Vektoren
  - Buchstabenfolgen (Zusammensetzung einzelner Zeichen zu einer Zeichenkette)
- In Python: Datentyp (und Datenstruktur) `list`, auch **Sequenzdatentyp**

Listen und Sequenzdatentypen

- **Zusammenfassung** von Variablen in **fester Reihenfolge**
- Beispiele
  - Messungen einer physikalischen Größe (Temperaturverläufe etc.)
  - Datenpunkte für Plots (x und y Koordinaten)
  - Matrizen / Vektoren
  - Buchstabenfolgen (Zusammensetzung einzelner Zeichen zu einer Zeichenkette)
- In Python: Datentyp (und Datenstruktur) `list`, auch **Sequenzdatentyp**
- Erstellt mit eckigen Klammern

Listen und Sequenzdatentypen

- **Zusammenfassung** von Variablen in **fester Reihenfolge**
- Beispiele
  - Messungen einer physikalischen Größe (Temperaturverläufe etc.)
  - Datenpunkte für Plots (x und y Koordinaten)
  - Matrizen / Vektoren
  - Buchstabenfolgen (Zusammensetzung einzelner Zeichen zu einer Zeichenkette)
- In Python: Datentyp (und Datenstruktur) `list`, auch **Sequenzdatentyp**
- Erstellt mit eckigen Klammern
- Elemente verschiedenen Typs sind erlaubt

Codebeispiele

Operationen mit Listen

Operationen mit Listen

- `x in s`, `x not in s`: Überprüfung, ob `x` in der Liste `s` enthalten ist (`bool`)

Operationen mit Listen

- `x in s`, `x not in s`: Überprüfung, ob `x` in der Liste `s` enthalten ist (`bool`)
- `s+t`: neue Liste als Konkatenation von `s` und `t`

Operationen mit Listen

- `x in s`, `x not in s`: Überprüfung, ob `x` in der Liste `s` enthalten ist (`bool`)
- `s+t`: neue Liste als Konkatenation von `s` und `t`
- `s*n`: `n`-malige Konkatenation von `s` an sich selbst

Operationen mit Listen

- `x in s`, `x not in s`: Überprüfung, ob `x` in der Liste `s` enthalten ist (`bool`)
- `s+t`: neue Liste als Konkatenation von `s` und `t`
- `s*n`: `n`-malige Konkatenation von `s` an sich selbst
- `s[i]`: Listenelement an der Stelle `i`, nullbasierte Indizierung (später)

Operationen mit Listen

- `x in s`, `x not in s`: Überprüfung, ob `x` in der Liste `s` enthalten ist (`bool`)
- `s+t`: neue Liste als Konkatenation von `s` und `t`
- `s*n`: `n`-malige Konkatenation von `s` an sich selbst
- `s[i]`: Listenelement an der Stelle `i`, nullbasierte Indizierung (später)
- `s[i:j]`: Subliste der Elemente zwischen Element `i` und Element `j`

Operationen mit Listen

- `x in s`, `x not in s`: Überprüfung, ob `x` in der Liste `s` enthalten ist (`bool`)
- `s+t`: neue Liste als Konkatenation von `s` und `t`
- `s*n`: `n`-malige Konkatenation von `s` an sich selbst
- `s[i]`: Listenelement an der Stelle `i`, nullbasierte Indizierung (später)
- `s[i:j]`: Subliste der Elemente zwischen Element `i` und Element `j`
- `s[i:j:k]`: wie oben, nur jedes `k`-te Element

Operationen mit Listen

- `x in s`, `x not in s`: Überprüfung, ob `x` in der Liste `s` enthalten ist (`bool`)
- `s+t`: neue Liste als Konkatenation von `s` und `t`
- `s*n`: `n`-malige Konkatenation von `s` an sich selbst
- `s[i]`: Listenelement an der Stelle `i`, nullbasierte Indizierung (später)
- `s[i:j]`: Subliste der Elemente zwischen Element `i` und Element `j`
- `s[i:j:k]`: wie oben, nur jedes `k`-te Element
- `len(s)`: Länge der Liste, Anzahl Elemente

Operationen mit Listen

- `s.count(x)`: Anzahl Vorkommen von `x` in `s`

Operationen mit Listen

- `s.count(x)`: Anzahl Vorkommen von `x` in `s`
- `s.append(x)`: Einfügen von `x` an das Ende der Liste

Operationen mit Listen

- `s.count(x)`: Anzahl Vorkommen von `x` in `s`
- `s.append(x)`: Einfügen von `x` an das Ende der Liste
- `s.insert(i,x)`: Einfügen von `x` an Position `i` der Liste

Operationen mit Listen

- `s.count(x)`: Anzahl Vorkommen von `x` in `s`
- `s.append(x)`: Einfügen von `x` an das Ende der Liste
- `s.insert(i,x)`: Einfügen von `x` an Position `i` der Liste
- `sum(s)`: Summe aller Einträge in der Liste

Operationen mit Listen

- `s.count(x)`: Anzahl Vorkommen von `x` in `s`
- `s.append(x)`: Einfügen von `x` an das Ende der Liste
- `s.insert(i,x)`: Einfügen von `x` an Position `i` der Liste
- `sum(s)`: Summe aller Einträge in der Liste
- `max(s)`: Maximum aller Einträge in der Liste

Operationen mit Listen

- `s.count(x)`: Anzahl Vorkommen von `x` in `s`
- `s.append(x)`: Einfügen von `x` an das Ende der Liste
- `s.insert(i,x)`: Einfügen von `x` an Position `i` der Liste
- `sum(s)`: Summe aller Einträge in der Liste
- `max(s)`: Maximum aller Einträge in der Liste
- `del s[i]`: Löschen des Elements an der Position `i`

Codebeispiele

Indizierung von Listen

Indizierung von Listen

- Indizierung der Elemente erfolgt **nullbasiert**

Indizierung von Listen

- Indizierung der Elemente erfolgt **nullbasiert**
- Erstes Element steht an Position null

Indizierung von Listen

- Indizierung der Elemente erfolgt **nullbasiert**
- Erstes Element steht an Position null
- **Beispiel**: `s = [1, 44, ['s', 'p'], '4', 5]`

Index	0	1	2	3	4
neg. Index	-5	-4	-3	-2	-1
Wert	`1`	`44`	`['s','p']`	`'4'`	`5`

Indizierung von Listen

- Indizierung der Elemente erfolgt **nullbasiert**
- Erstes Element steht an Position null
- **Beispiel**: `s = [1, 44, ['s', 'p'], '4', 5]`

Index	0	1	2	3	4
neg. Index	-5	-4	-3	-2	-1
Wert	`1`	`44`	`['s','p']`	`'4'`	`5`

- Zugriff dann mit `s[i]`, bspw. liefert `s[3]` den Wert `'4'`

Indizierung von Listen

- Indizierung der Elemente erfolgt **nullbasiert**
- Erstes Element steht an Position null
- **Beispiel**: `s = [1, 44, ['s', 'p'], '4', 5]`

Index	0	1	2	3	4
neg. Index	-5	-4	-3	-2	-1
Wert	`1`	`44`	`['s','p']`	`'4'`	`5`

- Zugriff dann mit `s[i]`, bspw. liefert `s[3]` den Wert `'4'`
- Negative Indizes vereinfachen oft **Indexarithmetik**

Indizierung von Listen

- Indizierung der Elemente erfolgt **nullbasiert**
- Erstes Element steht an Position null
- **Beispiel**: `s = [1, 44, ['s', 'p'], '4', 5]`

Index	0	1	2	3	4
neg. Index	-5	-4	-3	-2	-1
Wert	`1`	`44`	`['s','p']`	`'4'`	`5`

- Zugriff dann mit `s[i]`, bspw. liefert `s[3]` den Wert `'4'`
- Negative Indizes vereinfachen oft **Indexarithmetik**
- Kein komplett zyklischer Zugriff

Codebeispiele

Extraktion von Teillisten

Extraktion von Teillisten

- Erfolgt über den **Doppelpunkt-Operator**

Extraktion von Teillisten

- Erfolgt über den **Doppelpunkt-Operator**
- `t = s[i:j:k]`

Extraktion von Teillisten

- Erfolgt über den **Doppelpunkt-Operator**
- `t = s[i:j:k]`
- **Wichtige Regeln** als Konsequenz der nullbasierten Indizierung

Extraktion von Teillisten

- Erfolgt über den **Doppelpunkt-Operator**
- `t = s[i:j:k]`
- **Wichtige Regeln** als Konsequenz der nullbasierten Indizierung
  - **Startindex** `i` stets **inklusive**, Default ist 0

Extraktion von Teillisten

- Erfolgt über den **Doppelpunkt-Operator**
- `t = s[i:j:k]`
- **Wichtige Regeln** als Konsequenz der nullbasierten Indizierung
  - **Startindex** `i` stets **inklusive**, Default ist 0
  - **Endindex** `j` stets **exklusive**, Default ist `len(s)`

Extraktion von Teillisten

- Erfolgt über den **Doppelpunkt-Operator**
- `t = s[i:j:k]`
- **Wichtige Regeln** als Konsequenz der nullbasierten Indizierung
  - **Startindex** `i` stets **inklusive**, Default ist 0
  - **Endindex** `j` stets **exklusive**, Default ist `len(s)`
  - **Schrittweite** `k`, Default ist 1 (kein Überspringen von Elementen)

Codebeispiele

Impressum, Danksagung und Quellen

Stiftung Innovation in der Hochschullehre

Gefördert durch die Stiftung Innovation in der Hochschullehre im Rahmen des Projekts digit@L, https://stiftung-hochschullehre.de

SimTech

Gefördert mit Mitteln der Deutschen Forschungsgemeinschaft (EXC 2075 - 390740016) im Rahmen der Exzellenzstrategie

---

Autor: Dominik Göddeke, IANS, Universität Stuttgart

ZOERR

Veröffentlicht auf dem Zentralen OER Repositorium Baden-Württemberg, https://www.zoerr.de