Programmieren – 8. Strings

Ingenieurinformatik Teil 1, Wintersemester 2026/27

David Straub

Warm-up: Was gibt der Code aus?

```
werte = [3, 8, 5]
werte.append(2)
print(werte[-1])
print(len(werte))
```

Heute

- Strings im Detail: **Index, Slicing**, nützliche Methoden
- **Verschachtelte Schleifen** und der **for↔while-Umbau**
- Zum Abschluss: **Mini-Probeklausur** (15 min, zählt nicht – zeigt Ihnen, wo Sie stehen)

Strings als Zeichenketten

Index und Länge – wie bei Listen

```
wort = "Messwert"

print(wort[0])
print(wort[-1])
print(len(wort))
```

Strings sind Ketten von Zeichen – Index ab 0, `-1` vom Ende, `len` wie gehabt.

Über Zeichen iterieren

```
for zeichen in "NASA":
    print(zeichen)
```

Und damit funktioniert das Akkumulator-Muster aus Woche 7 auch für Text:

```
text = "Programmieren"
anzahl = 0
for zeichen in text:
    if zeichen == "e":
```

```
        anzahl += 1
print(anzahl)
```

Mini-Aufgabe (3 min)

Schreiben Sie eine Funktion `zaehle_e(text)`, die zurückgibt (`return`), wie oft der Buchstabe `e` in `text` vorkommt.

Testen Sie:

```
print(zaehle_e("Programmieren"))
print(zaehle_e("Elektrotechnik"))
```

Beim zweiten Test: Stimmt das Ergebnis mit Ihrer Erwartung überein?

Groß und klein: `upper()` und `lower()`

```
name = "Elektrotechnik"

print(name.upper())
print(name.lower())
print(zaehle_e(name.lower()))
```

- Das große `E` zählte eben nicht mit – `lower()` löst genau das
- `in` prüft auch Teilstrings: `"tech" in name` ⟶ `True`

Slicing: Teilstücke ausschneiden

```
txt = "ABCDEFGHIJ"

print(txt[2:5])    # CDE
print(txt[:3])     # ABC
print(txt[6:])     # GHIJ
print(txt[-3:])    # HIJ
print(txt[3:-1])   # DEFGHI
```

- `[start:stopp]` – Start **dabei**, Stopp **nicht** (wie bei `range`!)
- Weggelassen heißt: vom Anfang bzw. bis zum Ende
- Negative Werte zählen vom Ende
- Funktioniert **genauso für Listen**: `werte[1:3]`

Vorhersage-Aufgabe (3 min)

Aufschreiben, dann ausführen:

```
txt = "ABCDEFGHIJKLMNOP"

print(txt[0:5])
print(txt[11:-1])
print(txt[-3:-1])
print(txt[-3:])
print(txt[-3])
```

Text aufbauen: `+=` und `"#" * n`

```
balken = "#" * 5
print(balken)

zeile = ""
for i in range(4):
    zeile += "ab"
print(zeile)
```

- `"#" * n` – n-fache Wiederholung (aus Woche 2!)
- `zeile += ...` – das Akkumulator-Muster für Text: leer starten, anhängen

Verschachtelte Schleifen

Schleife in der Schleife

```
for i in range(1, 4):
    for j in range(1, 3):
        print(i, j)
```

- Die **innere** Schleife läuft für jeden Durchlauf der äußeren **komplett** durch
- Von Hand verfolgen: Wie viele Zeilen Ausgabe? In welcher Reihenfolge?

Vorhersage-Aufgabe (3 min)

Aufschreiben (Trace-Tabelle!), dann ausführen:

```
for i in range(3):
    for j in range(i):
        print("*", end="")
    print("!")
```

Der for↔while-Umbau

Jede `for`-Schleife lässt sich als `while`-Schleife schreiben – wer beide Richtungen beherrscht, hat den Kontrollfluss wirklich verstanden:

```
for i in range(5):
    print(i)
```

```
i = 0
while i < 5:
    print(i)
    i += 1
```

Rezept: **Startwert** vor die Schleife, **Bedingung** `< stopp`, **Erhöhung** ans Blockende.

Mini-Aufgabe (3 min)

Bauen Sie in eine `while`-Schleife um – gleiche Ausgabe, gleicher Kontrollfluss:

```
for i in range(2, 8):
    print(i, end=" ")
```

Mini-Probeklausur

So läuft es

- Sie bekommen ein **Aufgabenblatt** – drei kurze Aufgaben zum Stoff der Wochen 2 bis 7
- **15 Minuten**, alleine, auf Papier – ohne Ausführen, wie in der echten Klausur
- Danach korrigieren wir **gemeinsam**
- Zählt nicht. Zeigt Ihnen ehrlich, wo Sie nach 8 Wochen stehen – und was Sie gezielt üben sollten

Bis nächste Woche!

Nehmen Sie aus der Mini-Probeklausur mit: **Welche Aufgabenart braucht bei Ihnen noch Übung?** Genau dafür gibt es die OneTutor-Quizze: https://hm.onetutor.ai/

Nächste Woche: **Funktionen 2 und Struktogramme** – Funktionen, die mehr können, und: Programme entwerfen, bevor Sie tippen

- Fragen: jederzeit im Matrix-Chat