Programmieren – 12. Klassen und Rückblick

Ingenieurinformatik Teil 1, Wintersemester 2026/27

David Straub

Warm-up: Was kommt heraus?

Auf Papier – ohne Computer:

1. `print(0.1 + 0.2 == 0.3)`
2. 1001_2 – dezimal?
3. `print("ab" * 3)`

Heute

- **Eigene Datentypen** bauen: `class`
- Der **Semesterrückblick** als ein Programm, das alles verbindet
- Und: wie es weitergeht – als Nächstes kommt die **Probeklausur**

Eigene Datentypen mit Klassen

Die Frage: Was gehört zusammen?

Eine Messung besteht aus Zeitpunkt **und** Wert – zwei Variablen, die zusammengehören:

```
zeit_1 = 9.5
wert_1 = 21.4

zeit_2 = 10.0
wert_2 = 22.1
# ... und bei 100 Messungen?
```

Kennen Sie aus Woche 7 – aber Listen sammeln **gleichartige** Werte. Hier gehören **verschiedene** Dinge zu einem Ding.

`class`: ein eigener Datentyp

```
class Messung:
    def __init__(self, zeit, wert):
        self.zeit = zeit
        self.wert = wert

m1 = Messung(9.5, 21.4)
m2 = Messung(10.0, 22.1)

print(m1.wert)
print(m2.zeit)
```

- `class` definiert den **Typ** – `Messung(...)` erzeugt ein Exemplar davon
- Typen kennen Sie längst: `int`, `str`, `list` – jetzt bauen Sie eigene
- `__init__` läuft beim Erzeugen; `self` ist „dieses Exemplar“; `self.wert = ...` speichert **im Exemplar**

Methoden: Funktionen, die zum Typ gehören

```
class Messung:
    def __init__(self, zeit, wert):
        self.zeit = zeit
        self.wert = wert

    def fahrenheit(self):
        return self.wert * 9 / 5 + 32

m = Messung(9.5, 21.4)
print(m.fahrenheit())
```

- Die Punkt-Schreibweise kennen Sie: `werte.append(...)` war die ganze Zeit genau das – eine Methode
- Erster Parameter immer `self` – darüber kommt die Methode an die Daten ihres Exemplars

Mini-Aufgabe (3 min)

Dieses Programm stürzt mit einem `AttributeError` ab. Ergänzen Sie den Konstruktor, sodass es läuft:

```
class Batteriezelle:
    def __init__(self, kapazitaet):
        self.kapazitaet = kapazitaet

zelle = Batteriezelle(4800)
print(zelle.kapazitaet)
print(zelle.strom)
```

Hinweis: `strom` soll als zweiter Wert beim Erzeugen übergeben werden: `Batteriezelle(4800, 350)`

Mini-Aufgabe (3 min)

Ergänzen Sie in `Batteriezelle` eine Methode `laufzeit(self)`, die die Laufzeit in Stunden zurückgibt (`return`) – Kapazität geteilt durch Strom.

```
zelle = Batteriezelle(4800, 350)
print(f"{zelle.laufzeit():.1f} h")
```

Die Aufgabe aus Woche 2 – jetzt als Datentyp mit eingebautem Verhalten.

Debug-Aufgabe (3 min)

Der Aufruf schlägt fehl. Finden Sie den Fehler:

```
class Sensor:
    def __init__(self, name):
        self.name = name

    def status():
        return f"Sensor {self.name} bereit"

s = Sensor("Temperatur A")
print(s.status())
```

Wozu das Ganze?

- Zusammengehörige **Daten** und die passenden **Operationen** bleiben dauerhaft beisammen
- Sie verstehen jetzt, was `werte.append(7)` oder `name.upper()` wirklich sind: Methoden von Typen, die andere gebaut haben
- Ab jetzt können Sie beide Seiten: Typen **benutzen** – und Typen **bauen**

Semesterrückblick

Das Semester in einem Programm (Denkphase: 6 min, auf Papier)

Alles, was Sie können, in einer Aufgabe:

1. Anzahl `n` **einlesen**, ungültige Eingaben wiederholen (1 bis 50)
2. `n` Zufalls-Messwerte zwischen 1 und 100 erzeugen (`randint`)
3. `mittelwert(werte)` und `maximum(werte)` als **Funktionen** – ohne `max()`
4. **Plot** der Messreihe, Maximum als roter Punkt markiert
5. **Titel per** `if`: Maximum über 90 ⟶ `Ausreißer prüfen!`, sonst ⟶ `Messreihe ok`

Notieren Sie: Welcher Baustein stammt aus welcher Woche?

Gemeinsam live – in Etappen

Wir bauen das Programm jetzt zusammen, Etappe für Etappe.

Etappe	Baustein	Woche
1	Eingabe + Validierungsschleife	4, 5
2	Liste füllen mit `randint`	7
3	Funktionen: Mittelwert, Maximum	6, 7
4	Plot + markierter Punkt	10
5	Titel per `if`	3, 10

Ausblick: NumPy

Im weiteren Studium rechnen Sie mit größeren Datenmengen – dafür gibt es eine Bibliothek, die Sie kennenlernen werden:

```
import numpy as np

werte = np.array([3.2, 4.1, 5.0])
print(werte * 2)
print(werte.mean())
```

Ganze Datenreihen auf einmal verrechnen, ohne Schleife – mehr davon in späteren Semestern. Heute nur ein Blick über den Zaun.

Wie es weitergeht

Der nächste Termin ist die Probeklausur. 60 Minuten, Papier, eigene Unterlagen erlaubt – exakt wie die echte Klausur. Besprechung sofort danach, Musterlösung am selben Tag.

Wenn Sie vorher üben wollen:

- KI-Quizze auf **OneTutor**: https://hm.onetutor.ai/
- Ihre eigenen Unterlagen ordnen – Sie dürfen sie in Probeklausur und Klausur benutzen!

Mini-Check

1. Was macht `__init__` – und wann läuft es?
2. Was ist `self`?
3. `m = Messung(9.5, 21.4)` – wie greifen Sie auf den Wert zu?
4. Aus welcher Woche stammt die Validierungsschleife?

Schöne Ferien! 🎄

Wir sehen uns zur **Probeklausur** in der ersten Januarwoche.

- Üben: KI-Quizze auf OneTutor: https://hm.onetutor.ai/
- Fragen: auch in den Ferien im Matrix-Chat