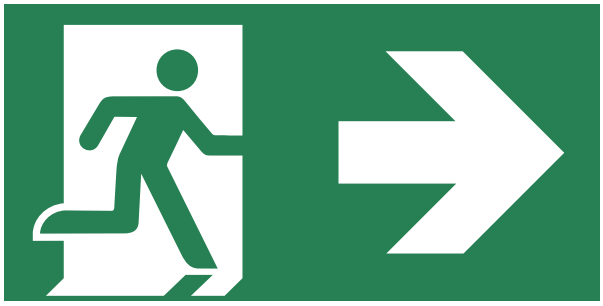
Programmieren – Praktikum

Ingenieurinformatik Teil 1, Wintersemester 2025/26

David Straub

Sicherheitsunterweisung für Benutzer der des Verbundlabors KCA

- **Fluchtwege** von jedem Raum links und rechts auf den Flur in das Treppenhaus
- an der Flurdecke sind **grüne beleuchtete Hinweisschilder** als Fluchtwegmarkierung angebracht
- die **Feuerlöscher** befinden sich im Flur und sind mit **roten Hinweisschildern** an den Seitenwänden gekennzeichnet
- die **Feuermelder** befinden sich in beiden Treppenhäusern
- im Brandfall **keinen Aufzug benützen**; Begründung: möglicher Stromausfall
- im Brandfall die **Fenster geschlossen halten**
- wichtige Informationen sind im Raum **ausgehängt**: Raumnutzungsordnung, …
- **Not-Aus-Schalter** sind in allen Räumen vorhanden



Gliederung

- Termin 3
- Termin 4
- Termin 5
- Termin 6

Termin 1

Datentypen

- `int` (Integer, Ganzzahlen): 1, 42, -7
- `float` (Gleitkommazahlen): 3.14, -0.001, 2.0
- `str` (String, Zeichenkette): `"Hallo"`, `'a'`, `"123"`
- `bool` (Boolean, Wahrheitswert): `True`, `False`

Typumwandlung

```
# Eingabe ist immer ein String
alter_str = input("Wie alt bist du? ")
print(alter_str + 1)  # Fehler! # String + Integer geht nicht
alter = int(alter_str)  # Umwandlung in Integer
print(alter + 1)  # Jetzt geht's
```

Operatoren

- Arithmetische Operatoren: `+`, `-`, `*`, `/`, `//` (Ganzzahldivision), `%` (Modulo), `**` (Exponentiation)
- Vergleichsoperatoren: `==`, `!=`, `<`, `>`, `<=`, `>=`
- Logische Operatoren: `and`, `or`, `not`

Die `input`-Funktion

```
name = input("Wie ist dein Name? ")
print("Hallo " + name)
```

f-Strings

```
name = "Alice"
alter = 30
print(f"Hallo {name}, du bist {alter} Jahre alt.")
```

Verzweigungen

```python
temperatur = float(input("Wie ist die Temperatur draußen? (in °C): "))
if temperatur < 0:
    print("Kalt - Winterjacke anziehen!")
elif temperatur <= 20:
    print("Mild - Pullover reicht")
else:
    print("Warm - T-Shirt-Wetter!")
```

Aufgabe 1: imperiale Einheiten

Schreiben Sie ein Programm, mit dem die in der Luftfahrt verbreiteten imperialen Einheiten Fuß, Seemeilen und Knoten in das metrische System (Meter, m/s) umgerechnet werden können.

Das Programm soll zunächst Fragen, welche der drei Einheiten umgerechnet werden soll. Anschließend soll der Wert der Einheit abgefragt werden, der umgerechnet werden soll. Das Programm soll dann den umgerechneten Wert ausgeben.

Umrechnungswerte

- 1 ft = 0.3048 m
- 1 NM = 1852 m
- 1 kn = 1 NM/h

Aufgabe 2: Schwebedauer

Ein Multicopter benötigt im Schwebeflug eine Leistung von

$$P = \kappa \frac{T^{3/2}}{\sqrt{2\rho A}}$$

κ: dimensionlose Effizienz < 1, $T = mg$: Schubkraft, ρ: Luftdichte, $A = n\pi r^2$: Rotorfläche

$g = 9{,}81 \frac{\mathrm{m}}{\mathrm{s}^2}$, $\rho_{\text{München}} \approx 1{,}2 \frac{\mathrm{kg}}{\mathrm{m}^3}$

Der Multicopter hat einen Akku mit der Kapazität 3 Ah und einer durchschnittlichen Spannung von 11.1 V.

Schreiben Sie ein Programm, das die Schwebedauer des Multicopters in Abhängigkeit von der Masse m, der Anzahl n und dem Durchmesser $2r$ der Rotoren berechnet. Nehmen sie $\kappa = 0{,}5$ an.

Termin 2

Aufgabe: Gleitstreckenberechnung

- Schreiben Sie eine Funktion, die die maximale Gleitstrecke eines Segelflugzeugs berechnet. Die Funktion soll die Starthöhe in Metern, den Höhenverlust pro Kilometer Flugstrecke in Metern und die nötige Höhenreserve in Metern als Eingabeparameter erhalten und die maximale Gleitstrecke in Kilometern zurückgeben.
- Verwenden Sie eine separate Funktion zur Ausgabe der Gleitstrecke in einem lesbaren Format (z.B. “Die maximale Gleitstrecke beträgt X km”) inklusive sinnvoller Rundung.
- Schreiben Sie eine Hauptfunktion `main()`, die den Benutzer nach der Starthöhe, dem Höhenverlust und der Höhenreserve fragt, die Funktionen aufruft und die Gleitstrecke ausgibt.

Zusatzaufgabe: Test-Skript

- Schreiben Sie ein Test-Skript, das die Funktion zur Berechnung der Gleitstrecke mit verschiedenen Eingabewerten aufruft und die Ergebnisse überprüft.
- Verwenden Sie das `assert`-Statement, um sicherzustellen, dass die berechneten Gleitstrecken den erwarteten Werten entsprechen.

Beispiel:

```python
def test_negative_starthoehe():
    assert berechne_gleitstrecke(-1, 1, 1) == 0
```

Zusatzaufgabe: Gleitwinkel

- Erweitern Sie die Funktion zur Berechnung der Gleitstrecke, um zusätzlich den Gleitwinkel in Grad zu berechnen und zurückzugeben.
- Der Gleitwinkel θ kann mit der Formel $\tan(\theta) = \frac{\text{Höhenverlust}}{\text{Flugstrecke}}$ berechnet werden.
- Passen Sie die Ausgabe-Funktion an, um auch den Gleitwinkel auszugeben.

Termin 3

Aufgabe: Primzahlbestimmung Teil 1

Schreiben Sie eine Funktion `ist_prim`, die überprüft, ob eine Zahl eine Primzahl ist. Die Funktion soll `True` zurückgeben, wenn die Zahl eine Primzahl ist, und `False`, wenn nicht.

Hinweise:

- Eine Primzahl ist eine natürliche Zahl größer als 1, die nur durch 1 und sich selbst teilbar ist.
- Verwenden Sie eine Schleife, um die Teilbarkeit der Zahl durch alle Zahlen ab 2 zu überprüfen
- Überlegen Sie sich, warum es ausreichen würde, nur bis zur Quadratwurzel der Zahl zu prüfen
- Für die Teilbarkeit kann der Modulo-Operator `%` verwendet werden
- Schreiben Sie eine Testfunktion, die die Korrektheit Ihrer Primzahl-Funktion überprüft (z.B. dass sie `True` für 2, 3, 5, 7 und `False` für 1, 4, 6, 8, 9 zurückgibt).

Primzahlbestimmung Teil 2

Schreiben Sie eine Funktion, die alle Primzahlen bis zu einer gegebenen Zahl `n` findet und in einer Liste zurückgibt.

Hinweise:

- Verwenden Sie Ihre Primzahl-Funktion aus Teil 1, um zu überprüfen, ob jede Zahl bis `n` eine Primzahl ist.

Primzahlbestimmung: Zusatzaufgaben

- Summe der Primzahlen: Schreiben Sie eine Funktion, die die Summe aller Primzahlen bis n berechnet. Beispiel: Für n = 10 → 2 + 3 + 5 + 7 = 17.
- Primzahldifferenzen: Erstellen Sie eine Liste mit den Abständen zwischen aufeinanderfolgenden Primzahlen bis n. Beispiel: Zwischen 2, 3, 5, 7 → Differenzen: `[1, 2, 2]`.
- Primzahlzwillinge: Finden Sie alle Primzahlzwillinge (Paare von Primzahlen, die genau 2 auseinanderliegen, z. B. (3,5), (5,7), (11,13)) bis `n`.

Termin 4

Würfelspiel-Simulator

In dieser Aufgabe programmieren Sie einen Simulator für ein Würfelspiel und analysieren verschiedene Strategien.

Das Spiel „Pig“ oder „Böse Eins“: - Ein Spieler würfelt mehrmals hintereinander - Nach jedem Wurf werden die Augen zur Rundenpunktzahl addiert - Der Spieler kann jederzeit aufhören und die Punkte “sichern” - **Aber**: Bei einer 1 verliert man alle Punkte der aktuellen Runde! - Wer zuerst 100 Punkte erreicht, gewinnt

Ihre Aufgabe: Testen Sie verschiedene Strategien durch Simulation!

Würfelspiel-Simulator (Teil 1)

Teil 1: Grundfunktionen

Schreiben Sie folgende Funktionen:

a) `wuerfle()`: - Gibt eine Zufallszahl zwischen 1 und 6 zurück - Verwenden Sie die passende Funktion aus dem Modul `random`

b) `spiele_runde(anzahl_wuerfe)`: - Würfelt `anzahl_wuerfe` mal und speichert alle Würfe in einer **Liste** - Wenn eine 1 dabei ist: gibt 0 zurück - Sonst: gibt die Summe aller Würfe zurück - Gibt außerdem die Liste der Würfe zurück (Rückgabe eines Tupels aus Zahl und Liste)

Testen Sie beide Funktionen mit `random.seed` für reproduzierbare Ergebnisse.

Würfelspiel-Simulator (Teil 2)

Teil 2: Strategien implementieren

Eine Strategie legt fest, wie oft man maximal würfelt, bevor man aufhört.

Schreiben Sie eine Funktion `spiele_strategie(max_wuerfe, ziel_punkte)`: - `max_wuerfe`: Anzahl Würfe pro Runde (die "Strategie") - `ziel_punkte`: Punkte, die zum Gewinnen nötig sind (z.B. 100) - Die Funktion spielt das Spiel bis zum Erreichen der Zielpunkte: - Speichert die Gesamtpunktzahl in Variable `gesamt`, zählt Runden in `runden` - Ruft in jeder Runde `spiele_runde(max_wuerfe)` auf - Addiert die Rundenpunkte zu `gesamt` - Gibt zurück: Anzahl der benötigten Runden

Testen Sie mit `max_wuerfe=3` und `ziel_punkte=100`.

Würfelspiel-Simulator (Teil 2, Fortsetzung)

Erstellen Sie ein **Struktogramm** für die Funktion `spiele_strategie`.

Würfelspiel-Simulator (Teil 3)

Teil 3: Mehrfache Simulation

Schreiben Sie eine Funktion `simuliere_strategie(max_wuerfe, ziel_punkte, anzahl_spiele)`: - Spielt das Spiel `anzahl_spiele` mal - Speichert die Anzahl benötigter Runden in einer **Liste** - Verwendet `random.seed(i)` vor jedem Spiel (mit `i` als Schleifenvariable) - Gibt die Liste aller Rundenanzahlen zurück

Führen Sie durch: - Simulieren Sie 1000 Spiele für die Strategien "2 Würfe", "3 Würfe", "4 Würfe" und "5 Würfe" - Speichern Sie die Ergebnisse in verschiedenen Variablen ### Würfelspiel-Simulator (Teil 4)

Teil 4: Statistische Auswertung

Schreiben Sie eine Funktion `analysiere_strategie(runden_liste, strategie_name)`: - Berechnet aus der Liste die folgenden Werte: - Durchschnittliche Anzahl Runden (Mittelwert) - Minimale Anzahl Runden - Maximale Anzahl Runden - Standardabweichung: $\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$ - Verwenden Sie `math.sqrt()` für die Wurzel

Würfelspiel-Simulator (Teil 4, Fortsetzung)

Die Funktion `analysiere_strategie` gibt die Ergebnisse formatiert aus:

```
Strategie: [strategie_name]
```

```
Durchschnitt: X.X Runden
Min: X Runden, Max: X Runden
Standardabweichung: X.X
```

Analysieren Sie alle vier Strategien. Welche ist am effizientesten?

Würfelspiel-Simulator: Zusatzaufgaben

Zusatz 1: Optimale Strategie finden

Schreiben Sie eine Schleife, die alle Strategien von 1 bis 10 Würfen testet (jeweils 1000 Spiele) und die durchschnittliche Rundenanzahl in einer Liste speichert. Finden Sie die optimale Strategie (kleinste durchschnittliche Rundenanzahl).

Zusatz 2: Risiko-Analyse

Berechnen Sie für jede Strategie: Wie oft (in Prozent) wird in einer Runde eine 1 gewürfelt und damit die Runde verloren? Verwenden Sie dafür die Wahrscheinlichkeitsrechnung:
$P(\text{keine } 1) = (5/6)^n$

Zusatz 3: Detaillierte Ausgabe

Erweitern Sie `spiele_runde()` so, dass bei gesetztem optionalen Parameter `debug=True` jeder einzelne Wurf ausgegeben wird, z.B.: "Wurf 1: 4, Wurf 2: 6, Wurf 3: 1 → Runde verloren!"

Termin 5

Aufgabe: Visualisierung von Wechselstromgrößen

Visualisieren Sie den zeitlichen Verlauf von Spannung und Strom an verschiedenen Wechselstromwiderständen.

Formeln: - Spannung: $u(t) = U_0 \sin(\omega t)$ - Strom: $i(t) = I_0 \sin(\omega t + \varphi)$

Konstanten: $U_0 = 325$ V, $I_0 = 23$ A, $f = 50$ Hz, $\omega = 2\pi f$

Importieren Sie `matplotlib.pyplot` und `math`.

Visualisierung Teil 1: Daten vorbereiten

a) Definieren Sie die Konstanten U_0, I_0, f und ω.

b) Schreiben Sie zwei **Funktionen** `spannung(t)` und `strom(t, phi)`, die die Formeln für $u(t)$ und $i(t)$ implementieren und jeweils einen Wert zurückgeben.

c) Erstellen Sie mit einer **List Comprehension** eine Liste `t_werte` mit 200 Intervallen von 0 bis 0.04 s (zwei Perioden).

Hinweis: Formel für den i-ten Zeitpunkt: $t_i = i \cdot \frac{0{,}04}{200}$ für $i = 0, 1, \ldots, 200$

Visualisierung Teil 2: Ohmscher Widerstand

Erstellen Sie einen Plot für den **ohmschen Widerstand** ($\varphi = 0$):

a) Berechnen Sie `u_werte` und `i_werte` mit **List Comprehensions**, die Ihre Funktionen aufrufen.

b) Plotten Sie beide Kurven in einem Diagramm: - Spannung: rote durchgezogene Linie - Strom: blaue gestrichelte Linie

c) Fügen Sie hinzu: Gitter, Achsenbeschriftungen, Titel

d) Zeigen Sie den Plot an oder speichern Sie ihn.

Visualisierung Teil 3: Spule

Erstellen Sie einen Plot für eine **Spule** ($\varphi = -\pi/2$):

a) Berechnen Sie `u_werte` und `i_werte` mit den Funktionen und der neuen Phasenverschiebung.

b) Plotten Sie beide Kurven: - Spannung: rote durchgezogene Linie - Strom: grüne gepunktete Linie

c) Markieren Sie den Punkt bei $t = 0{,}005$ s auf der Spannungskurve mit einem roten Kreis.

d) Fügen Sie Gitter, Beschriftungen und Titel hinzu.

Visualisierung Teil 4: Kondensator

Erstellen Sie einen Plot für einen **Kondensator** ($\varphi = +\pi/2$):

a) Berechnen Sie `u_werte` und `i_werte` mit den Funktionen und der neuen Phasenverschiebung.

b) Plotten Sie beide Kurven: - Spannung: rote durchgezogene Linie - Strom: orange durchgezogene Linie

c) Markieren Sie den Punkt bei $t = 0{,}010$ s auf der Stromkurve mit einem schwarzen Quadrat.

d) Fügen Sie Gitter, Beschriftungen und Titel hinzu.

Visualisierung: Zusatzaufgaben

Zusatz 1: Erstellen Sie eine Figur mit drei Subplots (1 Zeile, 3 Spalten), die alle drei Fälle nebeneinander zeigt. Verwenden Sie `plt.subplot()` (→ Dokumentation).

Zusatz 2: Fügen Sie den einzelnen Plots Legenden hinzu. Verwenden Sie `plt.legend()` (→ Dokumentation).

Zusatz 2: Die Momentanleistung ist $p(t) = u(t) \cdot i(t)$. Berechnen Sie und visualisieren Sie die Leistung für alle drei Fälle in separaten Plots. Was fällt bei der Spule und beim Kondensator auf?

Zusatz 3: Schreiben Sie eine Funktion `plot_phasenverschiebung(phi_grad)`, die Spannung und Strom für eine beliebige Phasenverschiebung in Grad plottet. Testen Sie mit verschiedenen Werten.

Termin 6

Advent of Code

Advent of Code ist ein Programmierwettbewerb mit täglichen Rätseln vom 1. bis 25. Dezember.

Aufgabe: Lösen Sie Day 1 in Python und **zeigen Sie mir Ihren Code**.

Regeln:

- **Keine KI-Tools** (ChatGPT, Copilot, etc.)
- Dokumentation, Google, gegenseitige Hilfe erlaubt

Sie brauchen einen Account auf adventofcode.com (Login mit GitHub, Google, etc.)

Wenn Sie fertig sind: Machen Sie mit Tag 2, 3, … so weit wie Sie kommen!