



# EINFÜHRUNG IN PROGRAMMIERUNG UND DATENBANKEN

joern ploennigs

## grundlagen

Motivation

Computer und  
Architekturen

Programmierung  
und Datentypen

Verzweigungen und  
Schleifen

## modellierung

Fehler und  
Debugging

Objektorientierung u.  
Softwareentwurf

Funktionen und  
Rekursion

# SCHLEIFEN



DALL·E 2: Infinite loop of octopuse, mathematical art by M.C. Escher ceramic dish

# SCHLEIFEN - ALLGEMEIN

- Wird genau wie die if/else-Anweisungen durch eine Bedingung kontrolliert
- Rufen eine Folge von Statements wiederholt auf
- Bekannte Anzahl an Wiederholungen nutzt man For-Loops
  - For-Loop: Wiederhole n-mal
  - For-Each-Loop (Iterator): Wiederhole für jedes Element in einer Sequenz
- Unbekannte Anzahl an Wiederholungen nutzt man While-Loops
  - While-Loop: Wiederhole bis eine Bedingung wahr ist (ggf. nie)
  - Do-While-Loop Wiederhole bis eine Bedingung falsch ist (d.h. mindestens einmal)
  - Repeat-Until-Loop Wiederhole bis eine Bedingung wahr wird (d.h. mindestens einmal)

# SCHLEIFEN – IN PYTHON

- In Python sind verfügbar:
  - `for` (Entspricht For-Each-Loop)
  - `while` (Entspricht While-Loop)
- Funktionieren genau wie alle anderen Steueranweisungen durch Einrückungen nach einer einzeiligen Definition
- Anders als in Funktionen sind Variablen aus Schleifen auch außerhalb der Schleife verfügbar!
- Die anderen Loop-Varianten sind zwar nicht explizit vorhanden, aber funktionell nachbildbar

## SCHLEIFEN - FOR-LOOP

Der **for**-Loop wiederholt ein Block an Statements für alle Elemente in einer Sequenz. **Element** ist dabei eine Variable die immer mit dem aktuellen Element aus der **Sequenz** belegt wird.

```
for Element in Sequenz:  
    # Statement
```

## SCHLEIFEN - FOR-EACH-LOOP BEISPIEL

Der **for**-Loop in Python ist ein For-Each-Loop. Er iteriert immer durch eine Sequenz an Werten, wie z.B. eine Liste. Die Variable des Iterators **e** enthält dabei den aktuellen Wert der Liste **seq**.

```
seq = ['a', 'b', 'c']  
for e in seq:  
    print(e)
```

Ausgabe

a  
b  
c

## SCHLEIFEN - FOR-LOOP BEISPIEL

Der **for**-Loop in Python kann auch als klassischer For-Loop verwendet werden, bei der man **n**-mal etwas wiederholt. Dazu erzeugt man mit der **range(n)** -Funktion eine Folge an Zahlen, durch die dann das For-Each-Loop iteriert wird.

```
n = 3
seq = range(n)
for e in seq:
    print(e)
```

Ausgabe

```
1
2
3
```



## SCHLEIFEN - FOR-LOOP BEISPIEL

Damit kann man dann zum Beispiel eine Liste an Messwerten aus imperialen Fuß in metrische Meter.

```
measurements_foot = [4.2, 2.3, 6.2, 10.5]    # Eingabe
measurements_meter = []                      # Ergebnisse
for feets in measurements_foot:
    measurements_meter.append(0.3048 * feets)

print(measurements_meter)
```

Ausgabe

```
[1.2801600000000002, 0.70104, 1.88976, 3.2004]
```

# SCHLEIFEN - WHILE-LOOP

Die **while**-Loop-Schleife wird so lange ausgeführt so lange bis eine **Bedingung** wahr ist. Da diese am Anfang geprüft wird und von Anfang an falsch sein kann, muss der Inhalt nicht ausgeführt werden.

```
while Bedingung:  
    # Statement
```

Wie beim **if** wird die **Bedingung** auf seinen Wahrheitswert geprüft.

- Ist dieser True läuft die Schleife einmal durch, woraufhin wieder geprüft wird.
- Ist dieser False endet die Schleife und der Code wird nicht (noch einmal) ausgeführt.

# SCHLEIFEN - WHILE-LOOP VS. DO-WHILE-LOOP VS. REPEAT-UNTIL

Im While-Loop kann die Bedingung schon am Anfang falsch sein. Die Schleife *muss also nicht* ausgeführt werden.

Im Do-While-Loop wird die Bedingung erst am Ende geprüft. Die Schleife wird also *immer mindestens einmal* durchlaufen. Das ist jedoch identisch im Verhalten mit einem While-Loop bei der die Bedingung am Anfang als True gesetzt wurde und in der Schleife geändert wird.

Beim Repeat-Until-Loop wird die Bedingung auch erst am Ende geprüft. Die Schleife wird also auch *immer mindestens einmal* durchlaufen. Die Schleife wird allerdings wiederholt bis die Bedingung wahr wird. Es ist also eine Negation des Do-While-Loops.

```
Bedingung = True/False
while Bedingung:
    # Statement
Bedingung = False
```

```
Bedingung = True
while Bedingung:
    # Statement
Bedingung = False
```

```
Bedingung = False
while not Bedingung:
    # Statement
Bedingung = True
```

## SCHLEIFEN - ENDLOSSCHLEIFEN!

While-Schleifen haben keine maximale Wiederholungsanzahl. Sie können potentiell für immer laufen, wenn sich die **Bedingung** nie ändert.

Es ist daher immer wichtig zu durchdenken, ob es den Fall gibt in dem sich die **Bedingung** nicht ändern kann und dann eine zweite Abbruchsbedingung (z.B. maximale Iterationen, maximale Zeit, etc.) hinzuzufügen.



```
Bedingung = True
while Bedingung: # DAS IST EINE ENDLOSSCHLEIFE
    # Statement
    Bedingung = True
```

## SCHLEIFEN - SCHLEIFE VERFRÜHT ABBRECHEN

Manchmal möchte man eine for/while-Schleife verfrüht abbrechen. Hierfür bietet Python den **break**-Befehl. Er bricht die Wiederholschleife an der aktuellen Position ab (ähnlich wie **repeat** in einer Funktion). Damit lassen sich zusätzliche Abbruchbedingungen in der Schleife implementieren.

Der **continue**-Befehl bricht die Schleife nicht ab, sondern beginnt direkt die nächste Iteration. Das ist sinnvoll wenn z.B. eine Filterbedingung nicht erfüllt ist.

```
dinge_in_meinem_rucksack = ["Papier", "Stift", "Apfel", "Brot", "Messer"]
essbares = {"Apfel", "Brot"}
essbar = None

for ding in dinge_in_meinem_rucksack:    # Finde das erste essbare Ding in meinem Rucksack
    if ding not in essbares:
        print(f"Überspringe {ding}")
        continue
    print(f"Essbares gefunden {ding}")
    essbar = ding
    break

print(essbar)
```

## SCHLEIFEN - SCHLEIFE VERFRÜHT ABBRECHEN

Mit **break** lassen sich zusätzliche Abbruchbedingungen realisieren, wie z.B. eine maximale Ausführzeit. Z.B. definieren wir einen zusätzlichen Timeout in einer Endlosschleife von 10 Sekunden mit Hilfe der **time()**-Funktion aus dem Paket **time**, welche die aktuelle Zeit (seit 1970) in Sekunden ausgibt.

```
import time

iterations = 0
start = time.time()
while True:    # DAS IST EINE ENDLOSSCHLEIFE
    iterations += 1
    elapsed = time.time() - start
    if elapsed > 3:
        print(f"Timeout after {iterations} iterations")
        break
```

# PROGRAMMFLUSSKONTROLLE

- Wir kennen nun zwei Werkzeuge:
  - Bedingte Ausführung
    - `if, else, elif`
  - Mehrfachausführung
    - `while, for`

Jedes Programm, was auf jemals auf einem Computer ausgeführt wurde, könnte mit diesen Mitteln geschrieben werden.

Hörsaalfrage

FRAGEN?



DALL·E 2: A psychedelic DJ with a question mark for a head