EINFÜHRUNG IN DIE PROGRAMMIERUNG

LOOPS

DHBW MANNHEIM
WIRTSCHAFTSINFORMATIK (DATA SCIENCE)

Markus Menth
Martin Gropp

FOR-SCHLEIFE

CONTROL STRUCTURES

Until now:

- Sequential execution of **statements**
- Conditional Execution: `if`, `elif`, `else`

Now:

- Repeated Execution: `for`, `while`

SCHLEIFEN: EINFÜHRUNG

```
print(0)
print(1)
print(2)
...
```

Alles sehr ähnlich...
Schema:

```
print(i)
```

Wenn jetzt `i` der Reihe nach die Werte 0, 1, 2, ... annimmt, ...

FOR-SCHLEIFE

```
for i in range(10):
    print(i)
```

Führe einen Block von Anweisungen für jedes Element von `range(10)` aus.
`range(10)`:
die Zahlen von 0 (einschließlich) bis 10 (ausschließlich), also 0 bis 9.
Wie bei `if` ist die Einrückung wichtig! Der eingerückte Block wird wiederholt.

BEISPIEL: SUMME

Problem: Berechnen Sie die Summe der Zahlen von 1 bis 10!

BEISPIEL: SUMME

Problem: Berechnen Sie mit einer Schleife die Summe der Zahlen von 1 bis 10!

```
summe = 0
for i in range(11):
    summe += i

print(summe)
```

BEISPIEL: SUMME

ALTERNATIVE LÖSUNG?

```
summe = 0
summe += 1
summe += 2
summe += 3
summe += 4
summe += 5
summe += 6
summe += 7
summe += 8
summe += 9
summe += 10

print(summe)
```

BEISPIEL: SUMME

Problem: Wie addieren Sie die Zahlen 1 bis 1000?

```
summe = 0
for i in range(1001):
    summe += i

print(summe)
```

(Ja, $\frac{n \cdot (n-1)}{2}$ geht natürlich auch...)

RANGE

`range(n)`:
die Zahlen von 0 (einschließlich) bis *n* (ausschließlich)
`range(a, n)`:
die Zahlen von *a* (einschließlich) bis *n* (ausschließlich)
`range(a, n, s)`:
...mit Schrittgröße *s*; *s* darf auch negativ sein
Dokumentation: **range**

BEISPIEL: FAKULTÄT

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \cdots \cdot 1$$

BEISPIEL: FAKULTÄT

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \cdots \cdot 1$$

```
n = 10
fak = 1
for i in range(1, n+1):
    fak *= i
```

AUFGABE: EINMALEINS

Verwenden Sie zwei verschachtelte Schleifen, um die "Einmaleinse" von 2 bis 10 auszugeben! Geben Sie vor jedem Einmaleins eine Titel dafür aus!

```
=== 2er-Einmaleins ===
1*2 = 2
2*2 = 4
...
=== 3er-Einmaleins ===
...
```

AUFGABE: EINMALEINS

LÖSUNG

```
for i in range(2, 11):
    print(f'=== {i}er-Einmaleins ===')
    for j in range(1, 11):
        print(f'{j}*{i} = {j*i}')
```

AUFGABE: PRIMZAHLEN

Schreiben Sie ein Programm, das alle Primzahlen zwischen zwei natürlichen Zahlen a und b mit $a < b$ findet!
Verwenden Sie für jede in Frage kommende Zahl den Modulo-Operator %, um zu überprüfen, ob diese Zahl durch andere Zahlen teilbar ist!

AUFGABE: PRIMZAHLEN

LÖSUNG

```
a = 4
b = 25


for i in range(a, b+1):
    prime = True
    for j in range(2, i):
        if i % j == 0:
            prime = False
            break


    if prime:
        print(f'{i} ist prim')
```

ELSE

Die Lösung der letzten Aufgabe lässt sich auch noch kompakter schreiben:

```
for i in range(a, b+1):
    for j in range(2, i):
        if i % j == 0:
            break
    else:
        print(f'{i} ist prim')
```

Der `else`-Block einer Schleife wird ausgeführt, nachdem alle Wiederholungen *normal* durchgelaufen sind. Der Block wird also am Ende ausgeführt, wenn die Schleife nicht durch ein `break` abgebrochen wurde.

AUSBLICK 🔬

`for`-Schleifen verwendet man in Python, um über alle Arten von *Iterables* zu iterieren. Dazu gehört nicht nur `range` sondern auch Strings, Tupel, Listen, ...

```
for c in 'foo':
    print(c)
```

```
f
o
o
```

```
for x in ('a', 2, 'c'):
    print(x)
```

a
2
c

WHILE-SCHLEIFE

WHILE-SCHLEIFE

Die `while`-Schleife ist flexibler als die `for`-Schleife. Der Block wird solange ausgeführt, wie eine Bedingung erfüllt ist.

```
summe = 0
i = 1
while i < 10:
    summe += i
    i += 1

print(summe)
```

(Ist die Bedingung von Anfang an nicht erfüllt, wird der Block nicht ausgeführt.)

AUSGABE

```
45
```

WHILE-SCHLEIFE

```
summe = 0
i = 1
while i <= n:
    summe += i
    i += 1

print(summe)
```

Ausgabe für *n=4*?

10

Ausgabe für *n=0*?

0

AUFGABE: FOR → WHILE

Schreiben Sie die folgende *for*-Schleife als *while*-Schleife!

```
for i in range(10, 4, -1):
    print(i)
```

AUFGABE: FOR → WHILE

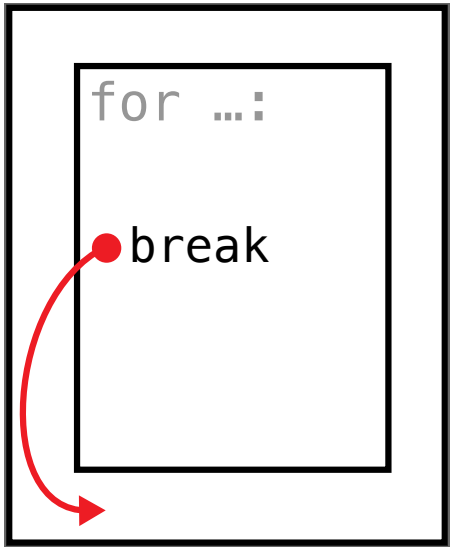
LÖSUNG

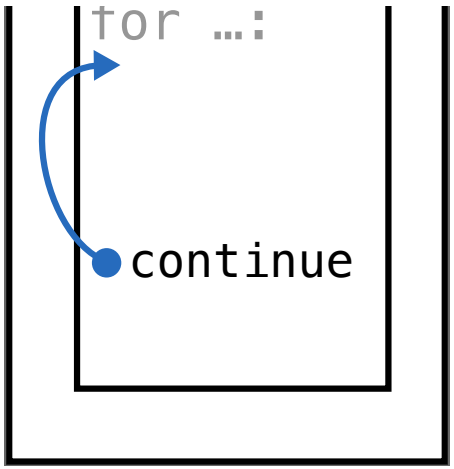
```
i = 10
while i > 4:
    print(i)
    i -= 1
```

BREAK UND CONTINUE

BREAK UND CONTINUE

- **break** bricht die Schleife ab,
  die Ausführung wird hinter der Schleife fortgesetzt.
- **continue** überspringt den Rest des aktuellen Schleifendurchlaufs und setzt die Ausführung mit dem nächsten Durchlauf (z. B. i + 1) fort.




for …:
continue

BREAK UND CONTINUE

```
while True:
    if ‹Bedingung›:
        break
```

```
for i in range(20):
    if i % 2 == 0:
        continue

    print(i)
```

AUFGABEN

AUFGABE: ZUFALLSZAHLEN

In Python können Sie mit der Funktion `random.random()` eine Zufallszahl aus dem Intervall $[0; 1)$ erzeugen:

```
from random import random
x = random()
```

Führen Sie diesen Befehl solange aus, bis Sie ein $x > 0.999$ erzeugt haben. Geben Sie danach aus, wie viele Versuche Ihr Programm dafür gebraucht hat:

```
Beim ⟨n⟩. Versuch war x zum ersten Mal größer als 0.999.
```

AUFGABE: ZUFALLSZAHLEN

LÖSUNG

```
from random import random



n = 0
while True:
    n += 1
    x = random()


    if x > 0.999:
        break


print(f'Beim {n}. Versuch war x zum ersten Mal größer als
0.999.')
```

AUFGABE

Reverse the order of a string!

AUFGABE

GNUSÖL

```
s = 'foobar'
t = ''
for c in s:
    t = c + t
```

AUFGABE

Write a program which prints all numbers which are

- divisible by 7,
- not a multiple of 5, and
- between 2000 and 3200 (both included)

AUFGABE

LÖSUNG

```
for i in range(2000, 3201):
    if (i % 7 == 0) and (i % 5 != 0):
        print(i)
```

AUFGABE

Implementieren Sie die **Caesar-Chiffre**!

- Speichern Sie einen Eingabetext und einen Schlüssel (also um wie viele Stellen ein Zeichen verschoben werden soll).
- Sie können davon ausgehen, dass der Eingabetext ausschließlich aus Kleinbuchstaben (ohne Leerzeichen, Sonderzeichen etc.) besteht.
- Ver- und entschlüsseln Sie den Eingabetext und geben Sie beides aus!

Die vordefinierten Methoden `ord` und `chr` helfen Ihnen, Zeichen aus einem String in Zahlen umzuwandeln und umgekehrt. Den Zeichen a bis z sind die aufsteigenden Zahlencodes 97 bis 122 zugeordnet:

`ord('a') = 97, chr(122) = 'z'`

Der Modulo-Operator `%` könnte Ihnen bei dieser Aufgabe gute Dienste leisten!

AUFGABE

LÖSUNG

```
text = 'helloworld'
key = 4


a = ord('a') # 97


encrypted = ''
for c in text:
    n = ord(c) - a
    e = (n + key) % 26
    s = chr(e + a)
    encrypted += s


print(encrypted)
```

```
lippsasvph
```

AUFGABE

LÖSUNG

```
encrypted = 'lippsasvph'
key = 4


decrypted = ''
for c in encrypted:
    n = ord(c) - a
    d = (n - key) % 26
    s = chr(d + a)
    decrypted += s


print(decrypted)
```

```
helloworld
```

AUFGABE

Write a Python program to compute the **greatest common divisor** (GCD) of two positive integers using **Euclid's algorithm**

AUFGABE

LÖSUNG

```
a = 72
b = 52


while b != 0:
    h = a % b
    a = b
    b = h



print(a)
```

AUFGABE

Compute and display **Fibonacci numbers**!

AUFGABE

LÖSUNG

```
a, b = 0, 1


while a < 10:
  print(a)
  a, b = b, a+b
```