Schleifen und logische Verknüpfungen

Schleifenköpfe

for - Schleifen

for-Schleifen haben eine Funktion, nämlich durch iterierbare Objekte/ Bereiche zu laufen

1) Ein Objekt wie eine Liste oder ein String kann mit einer for-Schleife durchlaufen werden
2) Ein definierter „Range-Bereich“ kann mit einer for-Schleife durchlaufen werden

In jedem Fall wird hierfür eine sogenannte Laufvariable benötigt, die jeweils nur innerhalb der Schleife gültig ist.

Beispiel

```
for i in range(0,5):

# Laufvariable i muss nicht vorher deklariert werden, wird als Variable der Schleife erkannt

    print(i)

#Ausgabe:

0

1

2

3

4
```

```
mylist = [0,1,2,3,4]

for i in mylist:

    print(i)

#Ausgabe:

0

1

2

3

4
```

Beispiel

```
for i in range(0,5):

# i nimmt in jedem Schleifendurchlauf die nächsthöhere Zahl im Bereich von 0-4 an (Schrittweite 1)

    print(i)

# 1. Durchlauf: i=0

  2. Durchlauf: i=1

  3. Durchlauf: i=3

  etc.
```

```
mylist = [0,1,2,3,4]

for i in mylist:

# i nimmt in jedem Schleifendurchlauf den Wert des nächsten Elements der Liste an

    print(i)

# mylist ist eine Liste und damit ein iterierbares Objekt (man kann „von vorn nach hinten drüberlaufen“)
```

Wichtig!

Eine Range kann auch rückwärts laufen:
for i in range(5,0,-1)

Eine Laufvariable kann einen beliebigen Namen haben:
for zahl in range(…)
for item in mylist
for buchstabe in satz...

Ein String ist ebenfalls ein iterierbares Objekt; die einzelnen Symbole sind die Elemente, über die „gelaufen“ wird:
satz = ‚Hello World!‘

```
for buchstabe in satz:
    print(buchstabe)
```

Wichtig!

Eine Laufvariable ist immer eine neu definierte Variable und greift nicht auf bereits existierende Variablen zu, auch wenn sie denselben Namen haben!

```
a = ‚a‘
satz = ‚Das ist ein Beispielsatz‘
counter = 0

for a in satz:
    counter = counter + 1
```

a ist eine Variable, die den String ‚a‘ speichert
a Ist eine Laufvariable, die jeden Wert im String annimmt

D.h. a nimmt pro Durchlauf immer den nächsten Buchstaben an und jedes Mal zählt counter hoch
→ es kann auf diese Weise nicht überprüft werden, ob ein Buchstabe des Satzes == a ist; die Laufvariable greift nicht auf die vorher definierte Variable zu!

While-Schleifen

While-Schleifen haben die Funktion nur unter einer bestimmten Bedingung zu laufen;

d.h. While-Schleifen brauchen ein Abbruchkriterium, damit sie nicht unbegrenzt weiterlaufen. Also muss die Bedingung ab einem bestimmten Punkt nicht mehr erfüllt sein.

Im Gegensatz zu for-Schleifen müssen die Items, auf die sich die Bedingung bezieht, schon außerhalb der Schleife existieren.

Beispiel

```
counter = 0

while counter <= 5 :

# solange die vorher definierte Variable
<= 5 ist, läuft die Schleife

    print(‚Der Countdown läuft.‘)

    counter = counter + 1

#Abbruchkriterium: der counter wird bei
jedem Schleifendurchlauf um den Wert 1
erhöht, sodass er nach 6 Durchläufen die
Bedingung nicht mehr erfüllt
```

```
found = False

zahl = 6

while found is False :

# solange die vorher definierte Variable
<= 5 und die Boolvariable False ist, läuft
die Schleife

    eing = int(input(‚Zahl eingeben‘))

    if eing == zahl:

        found = True

#Abbruchkriterium: Sobald die Zahl richtig
erraten wird, wird die Boolvariable auf
True gesetzt, sodass nach diesem
Durchlauf die Bedingung nicht mehr
erfüllt ist.
```

Logische Verknüpfungen

Wichtig!

Operatoren: and, or, (not)

Logische Verknüpfungen verbinden Aussagen/ Bedingungen miteinander, um komplexere Aussagen zu schaffen

if i == ‚a‘ and counter < 5:
→ nur wenn die Variable i gleich dem String ‚a‘ ist und eine Countervariable kleiner 5 wird das Statement ausgeführt

if i == ‚a‘ or i == ‚e‘:
→ wenn die Variable i gleich dem String ‚a‘ oder dem String ‚e‘ ist, wird das Statement ausgeführt

while is_element is false:
→ solange die Boolvariable is_element false ist, läuft die Schleife; dies ist äquivalet zu:

while is_element is not true:

Wichtig!

Logische Verknüpfungen verbinden Aussagen, keine Items/ Objekte, d.h.:

```
zahl1 = 5

if zahl1 == 4 or 5:
# funktioniert nicht; 5 ist ein Integer und keine Aussage

if zahl1 == 4 or zahl1 == 5:
#funktioniert, auch wenn es gedoppelt wirkt; hier werden zwei Aussagen miteinander verknüpft, die sich auf dieselbe Variable beziehen
```

Zu verstehen als:

```
If (zahl1 == 4) or (zahl1 == 5):
```