Schleifen in Python: for und while (Hilfe-Version)

Wiederholungen im Code Schritt für Schritt verstehen

Mit einer Schleife wiederholt der Computer Code automatisch. Du lernst zwei Arten:
• for-Schleife: Du weißt, wie oft wiederholt wird.
• while-Schleife: Es wird wiederholt, solange eine Bedingung stimmt.

Tipp: Bearbeite die Aufgaben der Reihe nach. Bei Code-Aufgaben nutze die Hilfen und die Beispiele.

Wörter einfach erklärt:
• Schleife = Code, der sich wiederholt.
• Bedingung = eine Frage mit Ja oder Nein, z. B. zahl <= 5.
• range(1, 6) = die Zahlen 1, 2, 3, 4, 5 (die 6 ist NICHT dabei).
• print() = zeigt etwas auf dem Bildschirm an.
• Einrücken = Der Code in der Schleife steht etwas weiter rechts (4 Leerzeichen).

1 Welche Aussage über die for-Schleife ist richtig? Tipp: Die for-Schleife nutzt man, wenn man weiß, wie oft etwas wiederholt wird.

- [ ] Eine for-Schleife wird verwendet, wenn die Anzahl der Wiederholungen bekannt ist.
- [ ] Eine for-Schleife wiederholt Code, solange eine Bedingung wahr ist.
- [ ] Eine for-Schleife kann nicht mit range() verwendet werden.
- [ ] Eine for-Schleife ist immer schneller als eine while-Schleife.

Hilfe zur for-Schleife mit range():

range(start, stop) zählt von start bis kurz VOR stop.

Beispiel (gelöst): Gib die Zahlen 1 bis 3 aus.

```
for zahl in range(1, 4):
    print(zahl)
```

Ausgabe: 1, 2, 3

Lies so: „Für jede Zahl von 1 bis 3: gib die Zahl aus.“

2 Aufgabe A: Schreibe ein Programm mit einer for-Schleife, das die Zahlen von 1 bis 5 ausgibt.

Tipps:
1. Beginne so: for zahl in range(___, ___):
2. Die erste Zahl ist 1. Die zweite Zahl ist eins größer als die letzte Zahl, die du sehen willst.
3. In die nächste Zeile (eingerückt): print(zahl)

3 Merksatz: Bei range(start, stop) wird die Zahl _____ nicht mehr durchlaufen. Um die Zahlen 1 bis 10 auszugeben, schreibt man range(1, _____).

Hilfe zur while-Schleife:

Eine while-Schleife läuft, solange die Bedingung stimmt. Du musst die Zahl in der Schleife selbst erhöhen. Sonst endet die Schleife nie.

Beispiel (gelöst): Gib die Zahlen 1 bis 3 aus.

```
zahl = 1
while zahl <= 3:
    print(zahl)
    zahl = zahl + 1
```

Schritte: 1. Startwert festlegen. 2. Bedingung prüfen. 3. Etwas tun. 4. Zahl um 1 erhöhen.

4 Aufgabe B: Schreibe ein Programm mit einer while-Schleife, das die Summe der Zahlen von 1 bis 5 berechnet (1+2+3+4+5) und ausgibt.

Tipps:
1. Du brauchst zwei Variablen: summe = 0 und zahl = 1.
2. Bedingung: while zahl <= 5:
3. In der Schleife: erst summe = summe + zahl, dann zahl = zahl + 1.
4. Gib die Summe NACH der Schleife aus (nicht eingerückt).

Hilfe zur Endlosschleife:

Eine Endlosschleife hört nie auf. Das passiert, wenn die Bedingung immer wahr bleibt. Meist fehlt die Zeile, die die Zahl erhöht.

Frage dich: Ändert sich die Variable in der Schleife?

5 Aufgabe C: Dieser Code hat einen Fehler.

```
zaehler = 1
while zaehler < 5:
    print(zaehler)
```

Bearbeite in 2 Schritten:
1. Was passiert, wenn man den Code startet? Beginne so: „Der Code gibt immer wieder ... aus, weil ...“
2. Schreibe den Code richtig auf. Tipp: Es fehlt eine Zeile in der Schleife, die zaehler größer macht.

6 Entscheide: wahr oder falsch? Tipp zu Aussage 1: range(5) beginnt bei 0. Tipp zu Aussage 3: Probiere im Kopf, ob du die Variable nach der Schleife noch benutzen kannst.

	richtig	falsch
range(5) erzeugt die Zahlen 1, 2, 3, 4, 5.	☐	☐
Eine while-Schleife kann endlos laufen, wenn die Bedingung nie falsch wird.	☐	☐
Die Schleifenvariable einer for-Schleife ist nach der Schleife nicht mehr verfügbar.	☐	☐
Um die Zahlen 1 bis 100 auszugeben, passt eine for-Schleife besser als eine while-Schleife.	☐	☐

Hilfe zu Listen und if:

Eine Liste enthält mehrere Werte, z. B. zahlen = [2, 7, 3].
Eine for-Schleife kann die Liste Element für Element durchgehen.

`if` prüft eine Bedingung. Nur wenn sie stimmt, wird der eingerückte Code ausgeführt.

Beispiel (gelöst): Nur Zahlen größer als 4 ausgeben.

```
zahlen = [2, 7, 3]
for zahl in zahlen:
    if zahl > 4:
        print(zahl)
```

Ausgabe: 7

7 Aufgabe D: Schreibe ein Programm, das eine Liste durchgeht und nur Zahlen ausgibt, die größer als 5 sind. Nutze diese Liste: zahlen = [2, 7, 3, 9, 1, 6]

Tipps:
1. Beginne mit: for zahl in zahlen:
2. Eingerückt darunter: if zahl > 5:
3. Noch weiter eingerückt: print(zahl)

8 Ordne jede Situation der passenden Schleife zu. Tipp: Weißt du vorher, wie oft wiederholt wird? Dann for. Hängt es davon ab, was passiert? Dann while.

1. Alle Elemente einer bekannten Liste verarbeiten	____	A) for-Schleife
2. Eingaben einlesen, bis „stop“ eingegeben wird	____	B) for-Schleife
3. Genau 10 Mal eine Aktion wiederholen	____	C) for-Schleife
4. Würfeln, bis eine Sechs fällt	____	D) while-Schleife
5. Alle Zahlen von 1 bis 1000 durchlaufen	____	E) while-Schleife

Hilfe zu verschachtelten Schleifen:

Verschachtelt heißt: eine Schleife steckt in einer anderen Schleife. Die innere Schleife läuft bei jedem Durchlauf der äußeren komplett durch.

Beispiel (gelöst): Zahlenpaare mit 1 und 2.

```
for i in range(1, 3):
    for j in range(1, 3):
        print(i, j)
```

Ausgabe:

```
1 1
1 2
2 1
2 2
```

So geht es: i = 1 → j = 1, dann j = 2. Danach i = 2 → j = 1, dann j = 2.

9 Aufgabe E: Schreibe ein Programm, das eine 3×3-Multiplikationstabelle ausgibt (von 1×1 bis 3×3). Beispiel für eine Zeile: 2 × 3 = 6

Tipps:
1. Äußere Schleife: for i in range(1, 4):
2. Innere Schleife (eingerückt): for j in range(1, 4):
3. Rechne: ergebnis = i * j
4. Ausgabe: print(i, "×", j, "=", ergebnis)
5. Denke ans Einrücken!

Lösungen

Schleifen in Python: for und while (Hilfe-Version) · Informatik · Klasse 9

Für dich als Lehrkraft: leichtere Fassung in einer Minute

QR scannen oder lehreule.de/anpassen/schleifen-in-python-klasse-9 öffnen: kostenloses Konto, dann schreibt Lehreule die leichtere Fassung mit Hilfen — oder Zusatzaufgaben für Schnelle, eine Word-Datei, einen QR-Code für die Klasse.

Aufgabe 1: Eine for-Schleife wird verwendet, wenn die Anzahl der Wiederholungen bekannt ist.

Aufgabe 2:
```
for zahl in range(1, 6):
    print(zahl)
```

Ausgabe:
```
1
2
3
4
5
```

Erklärung: range(1, 6) erzeugt die Zahlen 1, 2, 3, 4, 5. Die 6 gehört nicht mehr dazu. Die Schleife nimmt nacheinander jede Zahl und gibt sie mit print() aus. (Auch range(1, 6, 1) mit Schrittweite 1 ist richtig.)

Aufgabe 3: stop, 11

Aufgabe 4:
```
summe = 0
zahl = 1
while zahl <= 5:
    summe = summe + zahl
    zahl = zahl + 1
print(summe)
```

Ausgabe:
```
15
```

Erklärung: Die Schleife läuft, solange zahl kleiner oder gleich 5 ist. In jedem Durchlauf wird zahl zur summe addiert und zahl um 1 größer gemacht. Wenn zahl 6 ist, stimmt die Bedingung nicht mehr und die Schleife stoppt. Das Ergebnis ist 1+2+3+4+5 = 15.

Aufgabe 5: 1. Der Code gibt immer wieder 1 aus, weil sich zaehler nie ändert. Die Bedingung zaehler < 5 bleibt immer wahr. Das Programm läuft endlos (Endlosschleife).

2. Richtiger Code:

```
zaehler = 1
while zaehler < 5:
    print(zaehler)
    zaehler = zaehler + 1
```

Ausgabe:

```
1
2
3
4
```

Die Zeile zaehler = zaehler + 1 (oder zaehler += 1) fehlte. Sie macht zaehler in jedem Durchlauf größer, bis die Bedingung falsch wird.

Aufgabe 6: range(5) erzeugt die Zahlen 1, 2, 3, 4, 5. → **falsch**
Eine while-Schleife kann endlos laufen, wenn die Bedingung nie falsch wird. → **richtig**
Die Schleifenvariable einer for-Schleife ist nach der Schleife nicht mehr verfügbar. → **falsch**
Um die Zahlen 1 bis 100 auszugeben, passt eine for-Schleife besser als eine while-Schleife. → **richtig**

Aufgabe 7:

```
zahlen = [2, 7, 3, 9, 1, 6]
for zahl in zahlen:
    if zahl > 5:
        print(zahl)
```

Ausgabe:

```
7
9
6
```

Erklärung: Die Schleife nimmt nacheinander jede Zahl aus der Liste. Das if prüft, ob die Zahl größer als 5 ist. Nur dann wird sie ausgegeben.

Aufgabe 8: 1 → B, 2 → D, 3 → C, 4 → E, 5 → A

Aufgabe 9:
```
for i in range(1, 4):
    for j in range(1, 4):
        ergebnis = i * j
        print(i, "×", j, "=", ergebnis)
```

Ausgabe:

```
1 × 1 = 1
1 × 2 = 2
1 × 3 = 3
2 × 1 = 2
2 × 2 = 4
2 × 3 = 6
3 × 1 = 3
3 × 2 = 6
3 × 3 = 9
```

Erklärung: Die äußere Schleife zählt i von 1 bis 3. Für jedes i zählt die innere Schleife j von 1 bis 3. So entstehen 3 · 3 = 9 Rechnungen. Jede wird berechnet und ausgegeben. (Auch print(f"{i} × {j} = {ergebnis}") ist richtig.)