1.7 Übung: for- oder while-Schleife?

Leitidee

Du kennst jetzt zwei Arten von Schleifen: die for-Schleife und die while-Schleife. In diesem Abschnitt übst du, beide sicher einzusetzen. Dabei geht es nicht nur um das Programmieren selbst. Bei jeder Aufgabe überlegst du **zuerst**, welche Schleifenart besser passt, und begründest deine Wahl. Erst dann schreibst du das Programm. Diese Entscheidung zu treffen, ist eine wichtige Fähigkeit beim Programmieren – sie macht Programme kürzer und schützt vor Endlosschleifen.

Lernziele

Nach diesem Abschnitt kannst du

- für eine Aufgabe begründet zwischen for- und while-Schleife wählen,
- ein Programm mit Schleife lesen und seine Ausgabe vorhersagen,
- Summen und Produkte mit einer Schleife berechnen,
- Eingaben mit einer while-Schleife wiederholen lassen,
- deine Programme mit sinnvoll gewählten Testfällen überprüfen.

1.7.1 Das brauchst du

Die beiden Schleifenarten im Vergleich:

	for-Schleife	**while-Schleife**
Wann?	Anzahl der Durchläufe steht beim Start fest	Ende hängt von einer Bedingung ab
Kopf	`for i in range(1, 6):`	`while zahl != 0:`
Startwert	steckt in `range`	vor der Schleife selbst setzen
Veränderung	automatisch	im Rumpf selbst programmieren
Gefahr	Stoppwert gehört nicht dazu	Endlosschleife

So sehen die beiden Grundformen aus:

```
1 for i in range(1, 4):
2     print("Durchlauf", i)
3
4 zahl = 1
5 while zahl < 10:
6     print("zahl ist", zahl)
7     zahl = zahl * 2
```

AUSGABE

```
Durchlauf 1
Durchlauf 2
Durchlauf 3
zahl ist 1
zahl ist 2
zahl ist 4
zahl ist 8
```

Merke

- **for-Schleife:** Die Anzahl der Durchläufe steht vorher fest.
- **while-Schleife:** Das Ende hängt von einer Bedingung ab, die Anzahl ist vorher unbekannt.

Bei jeder Aufgabe unten gilt: Überlege zuerst, welche Schleifenart passt, bevor du programmierst!

Zwei Muster brauchst du immer wieder:

- **Zusammenzählen:** Summenvariable **vor** der Schleife auf `0` setzen, **in** der Schleife erhöhen (`summe = summe + zahl` oder `summe += zahl`), **nach** der Schleife ausgeben.
- **Eingaben wiederholen:** erste Eingabe **vor** der while-Schleife, Bedingung prüft die Eingabe, nächste Eingabe **am Ende** des Rumpfs.

Tipp

So testest du ein Programm mit Schleife. Probiere nicht nur eine „normale“ Eingabe aus, sondern auch Grenzfälle: die kleinste erlaubte Zahl (oft 0 oder 1), eine Eingabe, bei der die Schleife gar nicht durchlaufen wird, und eine, bei der sie sehr oft läuft. Rechne das erwartete Ergebnis vorher von Hand aus und vergleiche.

1.7.2 Aufträge

Arbeitsauftrag 79: Programm lesen

Lies das Programm, **ohne** es auszuführen.

1. Gib an, was das Programm ausgibt.
2. Gib an, wie oft der Schleifenrumpf ausgeführt wird.
3. Begründe, warum hier eine while-Schleife verwendet wird.

```
1 zahl = 1
2 while zahl < 50:
3     zahl = zahl * 3
4 print(zahl)
```

Deine Antworten:

Fakultät

In der Mathematik schreibt man $n!$ (gesprochen „n Fakultät“) für das Produkt aller ganzen Zahlen von 1 bis n. Zum Beispiel ist $4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$ und $5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120$. Für $0!$ legt man den Wert 1 fest. Die Fakultät gibt zum Beispiel an, auf wie viele Arten man n Personen in einer Reihe aufstellen kann.

Arbeitsauftrag 80: Fakultät

Berechne die Fakultät einer eingegebenen Zahl n ($n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$). Welche Schleifenart passt hier besser? Begründe deine Wahl, bevor du programmierst.

Tipp

Die Fakultät berechnest du fast wie eine Summe – nur wird multipliziert statt addiert. Überlege genau, mit welchem Wert die Produktvariable **vor** der Schleife starten muss. Was passiert, wenn du sie wie eine Summe mit 0 beginnen lässt?

Begründung:

Gymnasium Ottweiler

Arbeitsauftrag 81: Passwortabfrage

Frage so lange nach einem Passwort, bis das korrekte Passwort `informatik10` eingegeben wurde. Gib danach „Zugang gewährt“ aus. Welche Schleifenart passt hier? Begründe.

Tipp

Das Passwort ist ein Text. Die Eingabe musst du deshalb **nicht** umwandeln. Texte vergleichst du wie Zahlen mit `==` und `!=`. Das richtige Passwort steht im Programm in Anführungszeichen.

Begründung:

Primzahl-Test

Eine **Primzahl** ist eine natürliche Zahl größer als 1, die nur durch 1 und durch sich selbst teilbar ist, zum Beispiel 2, 3, 5, 7, 11 und 13. Die Zahl 9 ist keine Primzahl, weil sie durch 3 teilbar ist. Die 1 ist nach Festlegung keine Primzahl.

Ob eine Zahl durch eine andere teilbar ist, prüfst du mit dem Rest-Operator `%`: Ist `zahl % teiler == 0`, geht die Division ohne Rest auf.

Arbeitsauftrag 82: Primzahl-Test

Prüfe, ob eine eingegebene Zahl eine Primzahl ist. Tipp: Teste, ob sie durch eine Zahl zwischen 2 und sich selbst minus 1 teilbar ist.

Tipp

Arbeite mit einer Variablen, die sich ein Ergebnis merkt: Nimm zuerst an, die Zahl **ist** eine Primzahl (`ist_primzahl = True`). Probiere dann mit einer Schleife alle möglichen Teiler aus. Findest du einen, der ohne Rest aufgeht, war die Annahme falsch: `ist_primzahl = False`. Denke an den Sonderfall 1.

Arbeitsauftrag 83 (Zusatz): Sparziel

Auf einem Sparkonto liegen 100 €. Jedes Jahr kommen 5 % Zinsen dazu, das Guthaben wird also mit 1,05 multipliziert. Schreibe ein Programm, das berechnet, nach wie vielen Jahren das Guthaben zum ersten Mal mindestens 200 € beträgt. Gib die Anzahl der Jahre und das Guthaben (auf Cent gerundet) aus. Begründe deine Wahl der Schleifenart.

Begründung:

1.7.3 Weitere Übungen

Hier kannst du das Gelernte weiter üben. Die Punkte im Titel zeigen, wie schwer eine Aufgabe ist: ●○○ leicht, ●●○ mittel, ●●● knifflig. Fang mit den leichten an.

Arbeitsauftrag 84: Welche Schleife passt? ●○○

Entscheide jeweils, ob eine for- oder eine while-Schleife besser passt, und begründe kurz. Du musst die Programme nicht schreiben.

Situation	for oder while?
a) das Taschengeld von 12 Monaten zusammenzählen	
b) Zahlen einlesen, bis eine negative Zahl eingegeben wird	
c) alle geraden Zahlen von 2 bis 50 ausgeben	
d) so lange nachfragen, bis `ja` oder `nein` eingegeben wird	
e) eine eingegebene Zahl so oft halbieren, bis sie kleiner als 1 ist	
f) für jede der 25 Personen einer Klasse eine Note einlesen	

Begründungen zu a) bis f):

Arbeitsauftrag 85: Aus for wird while ●○○

Schreibe ein Programm mit einer **while**-Schleife, das genau dasselbe ausgibt wie das folgende Programm. Denke an Startwert, Bedingung und Veränderung der Zählvariablen.

```
for i in range(5, 26, 5):
    print(i)
```

Arbeitsauftrag 86: Nur die ungeraden ●○○

Lies das Programm, **ohne** es auszuführen.

1. Gib an, was das Programm ausgibt.
2. Gib an, wie oft die Zeile `summe = summe + zahl` ausgeführt wird.

```
1 summe = 0
2 for zahl in range(1, 6):
3     if zahl % 2 == 1:
4         summe = summe + zahl
5 print(summe)
```

Meine Antworten:

Arbeitsauftrag 87: Potenzieren ohne ** ●○○

Das Programm soll eine Potenz berechnen, ohne den Operator `**` zu benutzen. Dafür wird die Basis so oft mit sich selbst multipliziert, wie der Exponent angibt: $2^5 = 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2$. Ergänze die drei Lücken `___`. Teste mit Basis `2` und Exponent `10` (Ergebnis 1024) und mit Basis `5` und Exponent `0` (Ergebnis 1).

```
basis = int(input("Basis: "))
exponent = int(input("Exponent: "))
ergebnis = ___
for i in range(___):
    ergebnis = ___
print(basis, "hoch", exponent, "ist", ergebnis)
```

Arbeitsauftrag 88: Fehler finden: Vielfache von 7 zählen ●●○

Das Programm soll zählen, wie viele Zahlen von 1 bis 70 durch 7 teilbar sind. Richtig wäre `Anzahl: 10`. Es enthält **drei Fehler**.

```
for zahl in range(1, 70):
    if zahl % 7 = 0:
        anzahl = anzahl + 1
print("Anzahl:", anzahl)
```

1. Finde die drei Fehler. Notiere jeweils die Zeile und was falsch ist.
2. Schreibe das Programm richtig auf.

Die Fehler:

Arbeitsauftrag 89: Alle Teiler einer Zahl ●●○

Schreibe ein Programm, das eine ganze Zahl einliest und alle ihre Teiler ausgibt. Am Ende soll es ausgeben, wie viele Teiler es sind. Begründe, welche Schleifenart du verwendest.

So könnte es aussehen:

AUSGABE

```
Zahl: 12
1
2
3
4
6
12
Anzahl der Teiler: 6
```

Begründung:

Gymnasium Ottweiler

Arbeitsauftrag 90: Gerade und ungerade zählen ●●○

Schreibe ein Programm, das so lange ganze Zahlen einliest, bis `0` eingegeben wird. Danach soll es ausgeben, wie viele der eingegebenen Zahlen gerade und wie viele ungerade waren. Die `0` zählt nicht mit. Teste mit `4`, `7`, `10`, `3`, `8`, `0` und mit sofort `0`.

Arbeitsauftrag 91: Bestweite beim Weitsprung ●●●

Beim Weitsprung werden die Weiten eines Versuchs nach dem anderen eingegeben (in Metern, als Kommazahl). Die Eingabe `0` beendet die Eingabe. Schreibe ein Programm, das danach die **Anzahl der Versuche** und die **beste Weite** ausgibt. Wurde gar keine Weite eingegeben, soll es „Keine Weite eingegeben.“ ausgeben.

So könnte es aussehen:

AUSGABE

```
Weite in m (0 = Ende): 3.85
Weite in m (0 = Ende): 4.10
Weite in m (0 = Ende): 3.92
Weite in m (0 = Ende): 0
Versuche: 3
Bestweite: 4.1 m
```

Tipp

Merke dir die bisher beste Weite in einer Variablen `beste`. Weil alle Weiten größer als 0 sind, kann sie bei 0 starten. Bei jeder neuen Weite fragst du: Ist sie größer als die bisher beste? Dann wird sie die neue beste Weite.

1.7.4 Das Wichtigste in Kürze

- Vor dem Programmieren entscheiden: Steht die Anzahl der Durchläufe beim Start fest → `for`. Hängt das Ende von einer Bedingung oder Eingabe ab → `while`.
- Summen starten bei 0, Produkte bei 1. Der Startwert gehört **vor** die Schleife.
- Bei while-Schleifen muss sich die Bedingung im Rumpf ändern können, sonst terminiert die Schleife nicht.
- Teste Programme mit Grenzfällen, besonders mit Eingaben, bei denen die Schleife gar nicht oder nur einmal läuft.

Selbstcheck

- ☐ Ich kann begründen, ob für eine Aufgabe eine for- oder eine while-Schleife besser passt.
- ☐ Ich kann die Ausgabe eines Programms mit Schleife vorhersagen und mit einem Ablaufprotokoll überprüfen.
- ☐ Ich kann mit einer Schleife Summen und Produkte berechnen und kenne den richtigen Startwert.
- ☐ Ich kann eine Eingabe mit einer while-Schleife so lange wiederholen lassen, bis sie stimmt.
- ☐ Ich kann zu einem Programm sinnvolle Testfälle angeben, auch Grenzfälle.