DS 7 – Weitere Übungen

Übung: for- oder while-Schleife?

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.

Die Aufgaben stehen auch im Skript im Abschnitt „Weitere Übungen“; die Lösungen im Lösungsheft.

Aufgabe 1: 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	

Deine Antwort:

…

Lösung

✅ **Lösung**

Lösung

Situation	Schleife	Begründung
a) 12 Monate	`for`	Die Anzahl 12 steht fest.
b) bis zur negativen Zahl	`while`	Wie viele Zahlen kommen, entscheidet die Eingabe.
c) gerade Zahlen bis 50	`for`	Die Zahlen stehen fest: `range(2, 51, 2)`.
d) bis `ja` oder `nein`	`while`	Die Anzahl der Fehlversuche ist unbekannt.
e) halbieren bis unter 1	`while`	Wie oft halbiert werden muss, soll erst herauskommen.
f) 25 Noten	`for`	Es sind genau 25 Durchläufe.

Entscheidend ist immer die Frage: Steht beim Start der Schleife fest, wie oft wiederholt wird? Bei e) hängt das Ende von einer Bedingung ab („solange die Zahl mindestens 1 ist“), auch wenn keine Eingabe mehr kommt.

Aufgabe 2: 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.

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

```
5
10
15
20
25
```

Lösung

✅ **Lösung**

Lösung

```python
i = 5
while i < 26:
    print(i)
    i = i + 5
```

```
5
10
15
20
25
```

Lösung

Ausgabe:

```
5
10
15
20
25
```

Lösung

Die drei Angaben aus `range(5, 26, 5)` verteilen sich auf drei Stellen: Der Startwert `5` steht vor der Schleife, der Stoppwert `26` in der Bedingung und die Schrittweite `5` in der letzten Zeile des Rumpfs. Auch `while i <= 25:` ist richtig. Fehlt `i = i + 5`, entsteht eine Endlosschleife.

Aufgabe 3: 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.

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

```
9
```

Deine Antwort:

…

Lösung

✅ **Lösung**

Lösung

Ausgabe:

```
9
```

Lösung

`zahl` läuft von 1 bis 5. Nur bei den ungeraden Zahlen 1, 3 und 5 ist die Bedingung `zahl % 2 == 1` wahr. Die Zeile `summe = summe + zahl` wird also **dreimal** ausgeführt: 1 + 3 + 5 = 9.

zahl	1	2	3	4	5
`zahl % 2 == 1`	`True`	`False`	`True`	`False`	`True`
`summe` danach	1	1	4	4	9

Aufgabe 4: 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).

Lückencode

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

Lösung

✅ **Lösung**

Lösung

```
basis = int(input("Basis: "))
exponent = int(input("Exponent: "))
ergebnis = 1
for i in range(exponent):
    ergebnis = ergebnis * basis
print(basis, "hoch", exponent, "ist", ergebnis)
```

```
Basis: 2
Exponent: 10
2 hoch 10 ist 1024
```

Lösung

Getestet mit den Eingaben: `2`, `10`

Lösung

Ausgabe:

```
Basis: «2»
Exponent: «10»
2 hoch 10 ist 1024
```

Lösung

Wie bei der Fakultät startet die Produktvariable bei **1**. `range(exponent)` sorgt für genau `exponent` Durchläufe. Bei Exponent `0` läuft die Schleife gar nicht, und das Ergebnis bleibt 1. Das stimmt, denn $5^0 = 1$.

Aufgabe 5: 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**.

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

```
  File "<Zelle 6>", line 2
    if zahl % 7 = 0:
       ^^^^^^^^
SyntaxError: cannot assign to expression here. Maybe you meant '==' instead of '='?
```

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

Deine Antwort:

…

Lösung

✅ Lösung

Lösung

1. **Zeile 2:** `=` statt `==` im Vergleich (`SyntaxError`).
2. **Vor Zeile 1:** Die Zählvariable `anzahl` bekommt nie einen Startwert. In Zeile 3 meldet Python deshalb einen `NameError`. Vor die Schleife gehört `anzahl = 0`.
3. **Zeile 1:** Der Stoppwert 70 gehört nicht dazu. Dann fehlt die 70, und es kommt 9 statt 10 heraus. Richtig ist `range(1, 71)`. Dieser Fehler erzeugt keine Fehlermeldung.

Lösung

```
1 anzahl = 0
2 for zahl in range(1, 71):
3     if zahl % 7 == 0:
4         anzahl = anzahl + 1
5 print("Anzahl:", anzahl)
```

```
Anzahl: 10
```

Lösung

Ausgabe:

```
Anzahl: 10
```

Lösung

Die Zählvariable funktioniert wie eine Summenvariable: Startwert 0 vor der Schleife, in der Schleife um 1 erhöhen. Hier wird aber nur erhöht, wenn die Bedingung wahr ist.

Aufgabe 6: 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
```

Deine Antwort:

…

Lösung

✅ **Lösung**

Lösung

```python
zahl = int(input("Zahl: "))
anzahl = 0
for teiler in range(1, zahl + 1):
    if zahl % teiler == 0:
        print(teiler)
        anzahl = anzahl + 1
print("Anzahl der Teiler:", anzahl)
```

```
Zahl: 15
1
3
5
15
Anzahl der Teiler: 4
```

Lösung

Getestet mit den Eingaben: `12`

Lösung

Ausgabe:

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

Lösung

Eine for-Schleife passt: Die möglichen Teiler 1 bis `zahl` stehen fest, sobald die Zahl eingegeben ist. Damit die Zahl selbst mitgeprüft wird, ist der Stoppwert `zahl + 1`. Ausgabe und Zählen stehen im `if`-Block, denn sie sollen nur bei einem echten Teiler passieren. Weitere Testfälle: Die 7 hat genau 2 Teiler (sie ist eine Primzahl), die 1 hat nur einen Teiler.

Aufgabe 7: 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`.

Lösung

✅ **Lösung**

Lösung

```python
gerade = 0
ungerade = 0
zahl = int(input("Zahl (0 zum Beenden): "))
while zahl != 0:
    if zahl % 2 == 0:
        gerade = gerade + 1
    else:
        ungerade = ungerade + 1
    zahl = int(input("Zahl (0 zum Beenden): "))
print("Gerade Zahlen:", gerade)
print("Ungerade Zahlen:", ungerade)
```

```
Zahl (0 zum Beenden): 4
Zahl (0 zum Beenden): 7
Zahl (0 zum Beenden): 0
Gerade Zahlen: 1
Ungerade Zahlen: 1
```

Lösung

Getestet mit den Eingaben: `4`, `7`, `10`, `3`, `8`, `0`

Lösung

Ausgabe:

```
Zahl (0 zum Beenden): «4»
Zahl (0 zum Beenden): «7»
Zahl (0 zum Beenden): «10»
Zahl (0 zum Beenden): «3»
Zahl (0 zum Beenden): «8»
Zahl (0 zum Beenden): «0»
Gerade Zahlen: 3
Ungerade Zahlen: 2
```

Lösung

Eine while-Schleife passt, weil vorher nicht feststeht, wie viele Zahlen kommen. Es gibt zwei Zählvariablen, beide starten vor der Schleife bei 0. Die Verzweigung im Rumpf entscheidet, welche erhöht wird. Die nächste Eingabe steht am Ende des Rumpfs, **nach** der Verzweigung und nicht in einem ihrer Blöcke. Die `0` wird nicht mitgezählt, weil die Schleife bei ihr sofort endet. Bei sofort `0` gibt das Programm zweimal 0 aus.

Aufgabe 8: 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.

Lösung

✅ Lösung

Lösung

```
 1 versuche = 0
 2 beste = 0
 3 weite = float(input("Weite in m (0 = Ende): "))
 4 while weite != 0:
 5     versuche = versuche + 1
 6     if weite > beste:
 7         beste = weite
 8     weite = float(input("Weite in m (0 = Ende): "))
 9 if versuche > 0:
10     print("Versuche:", versuche)
11     print("Bestweite:", beste, "m")
12 else:
13     print("Keine Weite eingegeben.")
```

```
Weite in m (0 = Ende): 4.5
Weite in m (0 = Ende): 5.2
Weite in m (0 = Ende): 0
Versuche: 2
Bestweite: 5.2 m
```

Lösung

Getestet mit den Eingaben: `3.85`, `4.10`, `3.92`, `0`

Lösung

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
```

Lösung

- **Zeilen 3–8:** das bekannte Muster für Eingaben: erste Eingabe vor der Schleife, nächste Eingabe am Ende des Rumpfs.
- **Zeilen 6–7:** Ist die neue Weite größer als die bisher beste, wird sie gespeichert. Sonst bleibt `beste` unverändert. Nach der Schleife steht in `beste` die größte Weite.
- **Zeilen 9–13:** Nach der Schleife entscheidet eine Verzweigung, ob überhaupt Versuche eingegeben wurden.

Python zeigt `4.10` als `4.1` an, denn bei einer Kommazahl lässt es Nullen am Ende weg. Teste auch, ob die Bestweite stimmt, wenn der beste Sprung **der erste** oder **der letzte** ist.