1.6 Die while-Schleife: Wiederholen, solange eine Bedingung gilt

Leitidee

Mit der for-Schleife wiederholst du Anweisungen eine feste Anzahl von Malen. Oft weiß man aber vorher nicht, wie viele Wiederholungen nötig sind. Beim „Mensch ärgere dich nicht“ würfelst du, **bis** eine Sechs kommt – das kann einmal dauern oder zehnmal. Ein Geldautomat fragt die PIN ab, **solange** sie falsch ist. Für solche Fälle gibt es die **while-Schleife**. Sie prüft vor jedem Durchlauf eine Bedingung und hört auf, sobald die Bedingung nicht mehr erfüllt ist. In diesem Abschnitt lernst du, wie man sie schreibt, wie man gefährliche Endlosschleifen vermeidet und wann man besser for und wann besser while verwendet.

Lernziele

Nach diesem Abschnitt kannst du

- eine while-Schleife mit Bedingung und eingerücktem Schleifenrumpf schreiben,
- den Ablauf einer while-Schleife in einem Ablaufprotokoll und einem Programmablaufplan darstellen,
- eine Endlosschleife erkennen, abbrechen und ihre Ursache beheben,
- Eingaben so lange wiederholen lassen, bis sie gültig sind,
- Zufallszahlen mit `random.randint` erzeugen,
- begründet entscheiden, ob eine for- oder eine while-Schleife besser passt.

1.6.1 Wiederholen, solange eine Bedingung wahr ist

Das Wort *while* ist englisch und heißt „solange“. Eine while-Schleife sagt: „**Solange** die Bedingung wahr ist, führe den Rumpf aus.“ Die Bedingung kennst du schon von `if`: Es ist ein Ausdruck, der `True` oder `False` ergibt, zum Beispiel `i <= 3` oder `eingabe != "stopp"`.

Das erste Beispiel zählt von 1 bis 3 – das könnte die for-Schleife auch, aber so siehst du gut, wie die while-Schleife arbeitet:

```
1 i = 1
2 while i <= 3:
3     print(i)
4     i = i + 1
5 print("Ende")
```

AUSGABE

```
1
2
3
Ende
```

- **Zeile 1:** Die Variable `i` bekommt den Startwert `1`. Das musst du bei while selbst erledigen.
- **Zeile 2:** Der Schleifenkopf: `while`, dann die Bedingung, dann ein Doppelpunkt. Vor **jedem** Durchlauf prüft Python, ob `i <= 3` wahr ist. Nur dann wird der Rumpf ausgeführt.
- **Zeilen 3–4:** Der eingerückte Schleifenrumpf. Er gibt `i` aus und erhöht `i` danach um 1. Diese Erhöhung ist wichtig: Ohne sie bliebe `i` immer `1`, und die Bedingung wäre immer wahr.
- **Zeile 5:** Nicht eingerückt – wird ausgeführt, sobald die Bedingung falsch ist.

Das Ablaufprotokoll zeigt, wie Python nach dem Rumpf immer wieder zur Bedingung in Zeile 2 zurückspringt:

Schritt	Zeile	Anweisung	i
1	1	`i = 1`	1
2	2	`while i <= 3:`	1
3	3	`print(i)`	1
4	4	`i = i + 1`	2
5	2	`while i <= 3:`	2
6	3	`print(i)`	2
7	4	`i = i + 1`	3
8	2	`while i <= 3:`	3
9	3	`print(i)`	3
10	4	`i = i + 1`	4
11	2	`while i <= 3:`	4
12	5	`print("Ende")`	4

Im Schritt 11 hat `i` den Wert `4`. Die Bedingung `4 <= 3` ist falsch. Die Schleife **endet**, und Python macht mit Zeile 5 weiter. Am Ende hat `i` also den Wert `4`, obwohl zuletzt `3` ausgegeben wurde.

Definition: while-Schleife

Eine **while-Schleife** (bedingte Schleife) wiederholt ihren Schleifenrumpf, **solange** eine Bedingung wahr ist. Die Bedingung wird **vor** jedem Durchlauf geprüft. Ist sie schon beim ersten Mal falsch, wird der Rumpf kein einziges Mal ausgeführt.

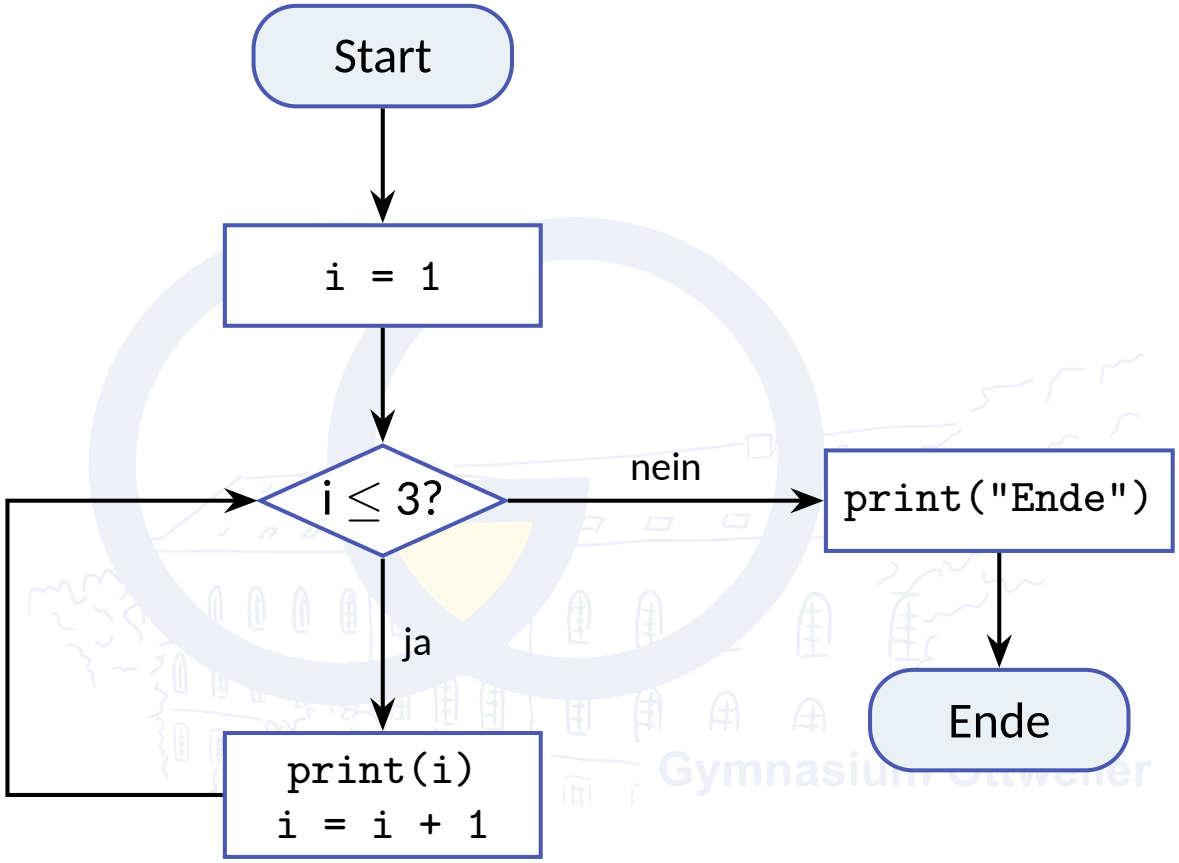
Gymnasium Ottweiler

Merke

- Die **while-Schleife** wird verwendet, wenn die **Anzahl der Wiederholungen vorher nicht bekannt** ist – sie läuft, **solange** eine Bedingung wahr ist.
- Syntax: `while Bedingung:` gefolgt vom eingerückten Anweisungsblock.

Der Ablauf als Bild

Den Ablauf einer Schleife kann man auch zeichnen. In einem **Programmablaufplan** stehen Bedingungen in Rauten und Anweisungen in Rechtecken. Pfeile zeigen, in welcher Reihenfolge es weitergeht. Bei einer Schleife führt ein Pfeil vom Rumpf **zurück zur Bedingung** – daher der Name „Schleife“.



Programmablaufplan der while-Schleife aus dem Beispiel: Solange die Bedingung in der Raute wahr ist (ja), wird der Rumpf ausgeführt und danach die Bedingung erneut geprüft

1.6.2 Beispiel: Zahlen addieren, bis 0 eingegeben wird

Jetzt ein Beispiel, bei dem die for-Schleife nicht weiterhilft. Das Programm soll beliebig viele Zahlen addieren. Wie viele es sind, entscheidet die Nutzerin oder der Nutzer beim Eintippen: Die Eingabe `0` beendet die Schleife.

```
1 summe = 0
2 zahl = int(input("Zahl eingeben (0 zum Beenden): "))
3
4 while zahl != 0:
5     summe = summe + zahl
6     zahl = int(input("Zahl eingeben (0 zum Beenden): "))
7
8 print("Summe aller Eingaben:", summe)
```

AUSGABE

```
Zahl eingeben (0 zum Beenden): 4
Zahl eingeben (0 zum Beenden): 7
Zahl eingeben (0 zum Beenden): 2
Zahl eingeben (0 zum Beenden): 0
Summe aller Eingaben: 13
```

- **Zeile 1:** Die Summenvariable startet bei 0 – genau wie bei der for-Schleife.
- **Zeile 2:** Die **erste** Zahl wird schon **vor** der Schleife eingelesen. Sonst könnte Python die Bedingung in Zeile 4 gar nicht prüfen, denn `zahl` hätte noch keinen Wert.
- **Zeile 4:** Solange die eingegebene Zahl nicht 0 ist, wird der Rumpf ausgeführt.
- **Zeile 5:** Die Zahl wird zur Summe addiert.
- **Zeile 6:** Die **nächste** Zahl wird eingelesen. Danach springt Python zurück zu Zeile 4 und prüft die neue Zahl. Diese Zeile sorgt dafür, dass sich die Bedingung ändern kann.
- **Zeile 8:** Nach der Schleife wird die Summe ausgegeben.

Das Ablaufprotokoll für die Eingaben 4, 7, 2, 0:

Schritt	Zeile	zahl	summe
1	1	–	0
2	2	4	0
3	4	4	0
4	5	4	4
5	6	7	4
6	4	7	4
7	5	7	11
8	6	2	11
9	4	2	11
10	5	2	13
11	6	0	13
12	4	0	13
13	8	0	13

Im Schritt 12 ist `zahl` gleich `0`. Die Bedingung `0 != 0` ist falsch, die Schleife endet. Die `0` selbst wird nicht mehr addiert (das würde aber auch nichts ändern).

Tipp

Das Muster aus diesem Beispiel brauchst du immer wieder, wenn eine Schleife von Eingaben abhängt:

1. **vor** der Schleife die erste Eingabe einlesen,
2. im Kopf die Eingabe prüfen,
3. **am Ende** des Rumpfs die nächste Eingabe einlesen.

Versuch: Probiere es aus

Starte das Programm und gib gleich als Erstes `0` ein. Was passiert? Erkläre mit der Definition der while-Schleife, warum der Rumpf kein einziges Mal ausgeführt wird.

1.6.3 Endlosschleifen

In der while-Schleife bist **du** dafür verantwortlich, dass die Bedingung irgendwann falsch wird. Vergisst du das, hört die Schleife nie auf. Hier soll ein Countdown von 3 herunterzählen. Findest du den Fehler?

```
1 zahl = 3
2 while zahl > 0:
3     print(zahl)
4     zahl = zahl + 1
5 print("Start!")
```

AUSGABE

```
3
4
5
6
7
...
```

In Zeile 4 wird `zahl` erhöht statt verringert. Die Zahl wird also immer größer, und `zahl > 0` bleibt für immer wahr. Das Programm gibt endlos Zahlen aus und erreicht `print("Start!")` nie. Richtig wäre `zahl = zahl - 1`.

Definition: Endlosschleife

Eine **Endlosschleife** ist eine Schleife, deren Bedingung nie falsch wird. Sie **terminiert nicht**, das heißt: Sie endet nie von selbst.

Woran erkennst du eine Endlosschleife? Das Programm wird nicht fertig. Entweder läuft die Ausgabe immer weiter, oder es passiert scheinbar gar nichts mehr (die Schleife läuft dann ohne Ausgabe). In Jupyter steht links neben deinem Programm lange ein Sternchen `[*]`.

Wie brichst du sie ab? In Jupyter klickst du oben auf die Stopp-Schaltfläche (ein kleines Quadrat). In Thonny klickst du auf den roten Stopp-Knopf. In einem Terminal drückst du `Strg` + `C`. Python bricht dann mit dieser Meldung ab:

FEHLERMELDUNG

```
KeyboardInterrupt
```

KeyboardInterrupt heißt „Unterbrechung über die Tastatur“. Das ist kein Fehler in deinem Code, sondern nur die Bestätigung, dass du das Programm angehalten hast. Den eigentlichen Fehler musst du danach selbst suchen.

Merke

Die Bedingung muss sich **innerhalb der Schleife verändern** können, sonst entsteht eine **Endlosschleife**.

Stolperstein

So findest du die Ursache einer Endlosschleife:

1. Welche Variablen stehen in der Bedingung?
2. Wird mindestens eine davon **im Rumpf** verändert? Typische vergessene Zeilen sind `i = i + 1` oder ein neues `input(...)` am Ende des Rumpfs.
3. Wird sie in die **richtige Richtung** verändert, sodass die Bedingung irgendwann falsch wird?
4. Steht die Änderung wirklich **eingerückt** im Rumpf und nicht erst nach der Schleife?

1.6.4 Eingaben prüfen: Wiederholen, bis die Eingabe stimmt

Eine typische Aufgabe für die while-Schleife: Ein Programm soll so lange nachfragen, bis eine **gültige** Eingabe kommt. Wie oft sich jemand vertippt, weiß man vorher nicht. Das folgende Programm akzeptiert nur Schulnoten von 1 bis 6:

```
1 note = int(input("Gib deine Note ein (1 bis 6): "))
2 while note < 1 or note > 6:
3     print("Ungültig! Erlaubt sind nur 1 bis 6.")
4     note = int(input("Gib deine Note ein (1 bis 6): "))
5 print("Gespeichert: Note", note)
```

AUSGABE

```
Gib deine Note ein (1 bis 6): 8
Ungültig! Erlaubt sind nur 1 bis 6.
Gib deine Note ein (1 bis 6): 0
Ungültig! Erlaubt sind nur 1 bis 6.
Gib deine Note ein (1 bis 6): 2
Gespeichert: Note 2
```

- **Zeile 1:** Die erste Eingabe wird vor der Schleife eingelesen.
- **Zeile 2:** Die Bedingung beschreibt eine **ungültige** Eingabe: kleiner als 1 **oder** größer als 6. Solange das zutrifft, wird wiederholt.
- **Zeilen 3–4:** Hinweis ausgeben und neu fragen. Die neue Eingabe überschreibt die alte.
- **Zeile 5:** Hier kommt das Programm nur an, wenn die Note gültig ist.

Tipp

Beim Formulieren der Bedingung hilft die Frage: „**Wann muss ich noch einmal fragen?**“ Die Antwort ist die Bedingung der while-Schleife. Hier: „wenn die Note kleiner als 1 oder größer als 6 ist“.

1.6.5 Zufallszahlen

Bei Spielen braucht man oft den Zufall: einen Würfelwurf oder eine geheime Zahl, die sich das Programm selbst ausdenkt. Dafür gibt es in Python ein Zusatzmodul.

Werkzeug: Zufallszahlen mit random.randint

Das **Modul** `random` ist eine Sammlung von Befehlen rund um den Zufall. Damit du es benutzen kannst, schreibst du **einmal am Anfang** des Programms `import random`. Danach liefert `random.randint(a, b)` eine zufällige ganze Zahl von `a` bis `b`. Anders als bei `range` gehören hier **beide** Grenzen dazu: `random.randint(1, 6)` liefert 1, 2, 3, 4, 5 oder 6 – wie ein Würfel.

Das folgende Programm würfelt so lange, bis eine Sechs fällt, und zählt die Würfe. Wie oft gewürfelt wird, entscheidet der Zufall – ein klarer Fall für while:

```
1  import random
2
3  wurf = random.randint(1, 6)
4  anzahl = 1
5  print("Gewürfelt:", wurf)
6  while wurf != 6:
7      wurf = random.randint(1, 6)
8      anzahl += 1
9      print("Gewürfelt:", wurf)
10 print("Endlich eine Sechs! Anzahl der Würfe:", anzahl)
```

So könnte es aussehen (bei jedem Start kommt etwas anderes heraus):

AUSGABE

```
Gewürfelt: 4
Gewürfelt: 1
Gewürfelt: 6
Endlich eine Sechs! Anzahl der Würfe: 3
```

- **Zeile 1:** Das Modul `random` wird geladen.
- **Zeilen 3–5:** Der erste Wurf wird vor der Schleife ausgeführt, gezählt und ausgegeben.
- **Zeile 6:** Solange keine Sechs gewürfelt wurde, geht es weiter.
- **Zeilen 7–9:** Neu würfeln, den Zähler um 1 erhöhen, den Wurf ausgeben. Durch das neue Würfeln kann sich die Bedingung ändern.
- **Zeile 10:** Nach der Schleife steht fest: Der letzte Wurf war eine Sechs.

So kannst du es sagen

So beschreibst du eine while-Schleife in Fachsprache – vom einfachen zum genauen Satz:

- *„Das läuft, bis man 0 eingibt.“ → „Die Anweisungen im Schleifenrumpf werden wiederholt, solange die eingegebene Zahl ungleich 0 ist.“*
- *„Dann hört es auf.“ → „Sobald die Bedingung `zahl != 0` falsch ist, terminiert die Schleife, und das Programm gibt die Summe aus.“*
- *„Das hängt sich auf.“ → „Die Schleife terminiert nicht, weil der Wert der Variablen `i` im Schleifenrumpf nicht erhöht wird.“*

1.6.6 for oder while?

Viele Aufgaben kann man mit beiden Schleifenarten lösen. Das Zählen von 1 bis 3 sieht mit for und while so aus:

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

AUSGABE

```
1
2
3
```

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

AUSGABE

```
1
2
3
```

Beide Programme geben dasselbe aus. Die for-Schleife ist aber kürzer und sicherer: Startwert, Bedingung und Erhöhung stecken alle in `range(1, 4)`. Bei der while-Schleife musst du an alle drei selbst denken – vergisst du die Erhöhung, entsteht eine Endlosschleife. Umgekehrt eignet sich die for-Schleife nicht für Aufgaben wie „würfle bis zur Sechs“, weil `range` eine feste Anzahl braucht.

Bei der Wahl hilft eine einzige Frage: **Steht beim Start der Schleife fest, wie oft wiederholt wird?** Das ist auch dann der Fall, wenn die Anzahl vorher eingegeben wurde, zum Beispiel bei „Summe von 1 bis n“.

Situation	Schleife	Begründung
Zahlen von 1 bis 100 ausgeben	`for`	genau 100 Durchläufe
Summe von 1 bis `n`, `n` wird eingegeben	`for`	beim Start der Schleife steht `n` fest
Zahlen addieren, bis 0 eingegeben wird	`while`	Ende hängt von der Eingabe ab
Eingabe wiederholen, bis sie gültig ist	`while`	Anzahl der Fehlversuche unbekannt
würfeln, bis eine Sechs fällt	`while`	Ende hängt vom Zufall ab
Ziffern einer Zahl einzeln abarbeiten	`while`	so lange, bis keine Ziffer mehr übrig ist

Merke

Faustregel: Ist die Anzahl der Durchläufe vorher bekannt → `for`. Hängt das Ende von einer Bedingung oder Eingabe ab → `while`.

Arbeitsauftrag 66: for oder while?

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) 20-mal „Hallo“ ausgeben	
b) Wörter einlesen, bis „stopp“ eingegeben wird	
c) das Einmaleins einer eingegebenen Zahl ausgeben	
d) ein Guthaben von 100 € wächst jedes Jahr um 5 %: Nach wie vielen Jahren sind es mindestens 200 €?	

Begründungen zu a) bis d):

Gymnasium Ottweiler

1.6.7 Übungen

Arbeitsauftrag 67: Zahl erraten

Lege eine geheime Zahl im Code fest (z. B. `geheim = 42`). Die Nutzerin oder der Nutzer soll so lange raten dürfen, bis die richtige Zahl eingegeben wurde. Gib nach jedem Versuch „zu hoch“ oder „zu niedrig“ aus.

Tipp

Nutze das Muster aus dem Abschnitt über das Addieren von Eingaben: erste Eingabe vor der Schleife, Bedingung „Zahl ist noch nicht richtig“, neue Eingabe am Ende des Rumpfs. Für „zu hoch“ oder „zu niedrig“ brauchst du im Rumpf eine Verzweigung mit `if` und `else`.

Arbeitsauftrag 68: Ratespiel mit Zufallszahl

Erweitere dein Programm aus dem Auftrag *Zahl erraten*:

1. Die geheime Zahl soll mit `random.randint` zufällig zwischen 1 und 100 gewählt werden.
2. Das Programm soll die Versuche zählen und am Ende ausgeben, wie viele Versuche gebraucht wurden.

Schreibe nur die Zeilen auf, die du ergänzt oder änderst, und gib jeweils an, wo sie stehen.

Arbeitsauftrag 69: Quersumme

Berechne mit einer while-Schleife die Quersumme einer eingegebenen Zahl (z. B. ist die Quersumme von 384 gleich 3 + 8 + 4 = 15).

Tipp: `zahl % 10` liefert die letzte Ziffer, `zahl // 10` entfernt sie.

Tipp

Probiere die beiden Rechenzeichen zuerst einzeln aus: `384 % 10` ergibt `4` (der Rest beim Teilen durch 10), `384 // 10` ergibt `38`. Wiederholst du das mit `38`, bekommst du `8` und `3`. Und mit `3` bekommst du `3` und `0`. Bei `0` ist keine Ziffer mehr übrig – das ist deine Abbruchbedingung.

Arbeitsauftrag 70 (Zusatz): Endlosschleife erkennen

Der folgende Code enthält eine Endlosschleife. Finde den Fehler und korrigiere ihn. Ändere **nicht** die Bedingung `i < 10`, sondern behebe den eigentlichen Fehler im Schleifenrumpf.

```
1 i = 0
2 while i < 10:
3     print(i)
```

Die Fehler:

1.6.8 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 71: In Viererschritten ●○○

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

1. Schreibe auf, was es ausgibt.
2. Gib an, wie oft der Schleifenrumpf ausgeführt wird und welchen Wert `zahl` am Ende hat.

```
1 zahl = 20
2 while zahl > 5:
3     print(zahl)
4     zahl = zahl - 4
5 print("fertig")
```

Gymnasium Ottweiler

Meine Antworten:

Arbeitsauftrag 72: Countdown ●○○

Das Programm soll eine Startzahl einlesen und von dort bis 1 herunterzählen. Danach soll es „Start!“ ausgeben. Ergänze die beiden Lücken ___ und teste mit `5` und mit `0`.

```
zahl = int(input("Starte bei: "))
while ___:
    print(zahl)
    ___
print("Start!")
```

Arbeitsauftrag 73: Anzahl der Spieler ●○○

Ein Brettspiel ist für 2 bis 4 Personen. Schreibe ein Programm, das die Anzahl der Spielerinnen und Spieler so lange abfragt, bis eine gültige Zahl eingegeben wird. Bei einer ungültigen Eingabe soll es einen Hinweis ausgeben. Am Ende soll es `Das Spiel startet mit 3 Personen.` ausgeben (mit der eingegebenen Zahl).

Arbeitsauftrag 74: Endlosschleife oder nicht? ●●○

Entscheide für jedes Programm, ob die Schleife irgendwann endet oder ob eine Endlosschleife entsteht. Begründe kurz. Führe die Programme **nicht** aus.

a)

```
i = 1
while i < 5:
    i = i + 1
```

b)

```
i = 10
while i > 0:
    i = i + 1
```

c)

```
1 i = 1
2 while i != 10:
3     i = i + 2
```

d)

```
1 i = 0
2 while i < 3:
3     print(i)
4 i = i + 1
```

Meine Entscheidungen und Begründungen:

Arbeitsauftrag 75: Fehler finden: Wörter zählen ●●○

Das Programm soll so lange Wörter einlesen, bis `stopp` eingegeben wird. Danach soll es ausgeben, wie viele Wörter eingegeben wurden (ohne `stopp`). Es enthält **drei Fehler**.

```
1 anzahl = 0
2 wort = input("Wort (stopp zum Beenden): ")
3 while wort == "stopp":
4     anzahl = anzahl + 1
5 wort = input("Wort (stopp zum Beenden): ")
6 print("Du hast", anzahl "Wörter eingegeben.")
```

1. Finde die drei Fehler. Notiere jeweils die Zeile und was falsch ist.
2. Schreibe das Programm richtig auf. Teste es mit den Eingaben `Hund`, `Katze`, `Maus`, `stopp` und auch mit sofort `stopp`.

Die Fehler:

Arbeitsauftrag 76: Ablaufprotokoll: Stellen zählen ●●○

Das Programm zählt, wie viele Stellen eine Zahl hat. Fülle das Ablaufprotokoll aus (– bedeutet: noch kein Wert) und gib an, was das Programm ausgibt.

```
1 zahl = 4096
2 stellen = 0
3 while zahl > 0:
4     zahl = zahl // 10
5     stellen = stellen + 1
6 print("Stellen:", stellen)
```

Schritt	Zeile	`zahl`	`stellen`	Schritt	Zeile	`zahl`	`stellen`
1	1			9	3		
2	2			10	4		
3	3			11	5		
4	4			12	3		
5	5			13	4		
6	3			14	5		
7	4			15	3		
8	5			16	6		

Ausgabe:

Arbeitsauftrag 77: Papier falten ●●○

Ein Blatt Papier ist etwa 0,1 mm dick. Faltet man es einmal, ist der Stapel doppelt so dick. Stell dir vor, man könnte das Blatt beliebig oft falten.

1. Schreibe ein Programm, das berechnet, wie oft man falten müsste, bis der Stapel mindestens 1 Meter (1000 mm) dick ist. Gib die Anzahl der Faltungen und die Dicke aus.
2. Begründe, warum hier eine while-Schleife passt.

Begründung:

Arbeitsauftrag 78: Die 3n+1-Folge ●●●

Mit einer Startzahl wird immer wieder so gerechnet:

- Ist die Zahl gerade, wird sie halbiert.
- Ist sie ungerade, wird sie mit 3 multipliziert und dann 1 addiert.

Das wird wiederholt, bis die Zahl 1 erreicht ist. Aus 6 wird zum Beispiel 3, dann 10, 5, 16, 8, 4, 2 und 1. Das sind 8 Schritte.

1. Schreibe ein Programm, das eine Startzahl einliest, jede neue Zahl ausgibt und am Ende die Anzahl der Schritte. Teste mit 6 und mit 7.
2. Was passiert bei der Startzahl `0`? Erkläre.

Zu 2.:

1.6.9 Das Wichtigste in Kürze

- Die **while-Schleife** wiederholt ihren Rumpf, **solange** die Bedingung wahr ist. Die Bedingung wird vor jedem Durchlauf geprüft.
- Aufbau: `while Bedingung:` – darunter der eingerückte Rumpf.
- Startwert und Veränderung der Variablen in der Bedingung musst du selbst programmieren. Ändert sich die Bedingung nie, entsteht eine **Endlosschleife**. Abbrechen mit der Stopp-Schaltfläche oder `Strg` + `C`.
- Bei Eingaben gilt das Muster: erste Eingabe vor der Schleife, nächste Eingabe am Ende des Rumpfs.
- `import random` und `random.randint(a, b)` liefern eine zufällige ganze Zahl von `a` bis einschließlich `b`.
- Steht die Anzahl der Durchläufe vorher fest → `for`. Hängt das Ende von einer Bedingung oder Eingabe ab → `while`.

Selbstcheck

- ☐ Ich kann eine while-Schleife schreiben und erklären, wann sie terminiert.
- ☐ Ich kann zu einer while-Schleife ein Ablaufprotokoll und einen Programmablaufplan erstellen.
- ☐ Ich kann eine Endlosschleife erkennen, abbrechen und den Fehler finden.
- ☐ Ich kann eine Eingabe so lange wiederholen lassen, bis sie gültig ist.
- ☐ Ich kann mit `random.randint` Zufallszahlen erzeugen.
- ☐ Ich kann begründen, ob für eine Aufgabe eine for- oder eine while-Schleife besser passt.

Gymnasium Ottweiler