1.5 Die for-Schleife: Anweisungen wiederholen

Leitidee

Stell dir vor, dein Programm soll die Zahlen von 1 bis 100 ausgeben. Mit dem, was du bisher kannst, bräuchtest du dafür 100 `print`-Zeilen. Das ist mühsam, und bei einem Tippfehler in Zeile 73 suchst du lange. Viel besser wäre eine Anweisung wie: „Wiederhole das 100-mal." Aus der Blockprogrammierung kennst du dafür den Baustein „wiederhole 10-mal". In Python heißt diese Wiederholung **for-Schleife**. Mit ihr gibst du Zahlenreihen aus, berechnest Summen und zeichnest einfache Muster – mit nur zwei oder drei Zeilen Code.

Lernziele

Nach diesem Abschnitt kannst du

- erklären, was eine Schleife ist und wann man eine for-Schleife verwendet,
- eine for-Schleife mit Schleifenkopf und eingerücktem Schleifenrumpf schreiben,
- mit `range` Zahlenfolgen erzeugen, auch mit eigenem Start und eigener Schrittweite,
- mit einer Schleife eine Summe berechnen und dabei `+=` verwenden,
- den Ablauf einer Schleife in einem Ablaufprotokoll Schritt für Schritt verfolgen.

1.5.1 Die Idee: Wiederholen statt abschreiben

Statt dreimal `print("Hallo!")` untereinanderzuschreiben, kannst du Python sagen: „Mache das dreimal." Das sieht so aus:

```
for i in range(3):
    print("Hallo!")
```

AUSGABE

```
Hallo!
Hallo!
Hallo!
```

- **Zeile 1:** Diese Zeile ist der **Schleifenkopf**. Er sagt: „Mache das Folgende dreimal." Was `i` und `range(3)` genau bedeuten, lernst du gleich. Der Kopf endet mit einem **Doppelpunkt**.
- **Zeile 2:** Diese Zeile ist **eingerückt** (4 Leerzeichen). Alles, was eingerückt unter dem Kopf steht, wird wiederholt. Diesen Teil nennt man den **Schleifenrumpf**.

Der Vorteil wird deutlich, wenn du statt `3` die Zahl `100` schreibst: Das Programm bleibt zwei Zeilen lang, gibt aber 100-mal „Hallo!" aus.

Definition: Schleife

Eine **Schleife** ist eine Kontrollstruktur, die einen Block von Anweisungen (den **Schleifenrumpf**) mehrfach hintereinander ausführt. Jede einzelne Ausführung des Schleifenrumpfs heißt **Durchlauf**.

Definition: for-Schleife

Eine **for-Schleife** (auch **Zählschleife**) ist eine Schleife, bei der die Anzahl der Durchläufe schon vor dem Start der Schleife feststeht.

Merke

- Eine **Schleife** wiederholt einen Anweisungsblock mehrfach.
- Die **for-Schleife** wird verwendet, wenn die **Anzahl der Wiederholungen vorher bekannt** ist (z. B. „10-mal“, „für jede Zahl von 1 bis 100“).

1.5.2 Aufbau einer for-Schleife

Die Zählvariable

Im Schleifenkopf steht eine Variable, hier `i`. Sie heißt **Zählvariable** (oder Schleifenvariable). Bei jedem Durchlauf bekommt sie automatisch den nächsten Wert. Du kannst sie im Schleifenrumpf benutzen, zum Beispiel ausgeben:

```
1 for i in range(1, 4):
2     print("Runde", i)
3 print("Fertig")
```

AUSGABE

```
Runde 1
Runde 2
Runde 3
Fertig
```

- **Zeile 1:** `range(1, 4)` liefert der Reihe nach die Zahlen `1`, `2` und `3`. Die Zählvariable `i` bekommt im ersten Durchlauf den Wert `1`, im zweiten den Wert `2`, im dritten den Wert `3`.
- **Zeile 2:** Der Schleifenrumpf. Er gibt in jedem Durchlauf „Runde“ und den aktuellen Wert von `i` aus.
- **Zeile 3:** Diese Zeile ist **nicht** eingerückt. Sie gehört also nicht mehr zur Schleife. Sie wird erst ausgeführt, wenn die Schleife ganz fertig ist – und zwar nur einmal.

Du kannst dir die for-Schleife wie eine Läuferin auf der Sportbahn vorstellen: Sie läuft ihre Runden und zählt dabei mit. Die Zahl, die sie gerade zählt, ist der Wert der Zählvariablen. Nach der letzten Runde hört sie auf und geht zur Bank (Zeile 3).

Der Ablauf Schritt für Schritt

Was genau passiert, zeigt ein **Ablaufprotokoll**. Es listet jede ausgeführte Zeile auf und den Wert von `i` **nach** dieser Zeile:

Schritt	Zeile	Anweisung	i
1	1	`for i in range(1, 4):`	1
2	2	`print("Runde", i)`	1
3	1	`for i in range(1, 4):`	2
4	2	`print("Runde", i)`	2
5	1	`for i in range(1, 4):`	3
6	2	`print("Runde", i)`	3
7	1	`for i in range(1, 4):`	3
8	3	`print("Fertig")`	3

Beachte drei Dinge:

- Python springt nach dem Rumpf (Zeile 2) immer wieder **zurück zum Kopf** (Zeile 1). Dort bekommt `i` den nächsten Wert.
- Im Schritt 7 ist Python noch einmal im Kopf. Jetzt liefert `range(1, 4)` aber keine Zahl mehr. Die Schleife ist zu Ende, `i` behält den letzten Wert `3`, und es geht mit Zeile 3 weiter.
- Die Zeile 1 wird viermal ausgeführt, der Rumpf dreimal.

Merke

Die **Zählvariable** bekommt bei jedem Durchlauf automatisch den nächsten Wert. Du musst sie nicht selbst erhöhen.

Einrückung: Was gehört zur Schleife?

Wie bei `if` entscheidet in Python die **Einrückung**, welche Anweisungen zusammengehören. Der Schleifenrumpf darf mehrere Zeilen haben. Alle müssen gleich weit eingerückt sein. Vergleiche die beiden Programme. Sie unterscheiden sich nur in der Einrückung der letzten Zeile:

```
1 for i in range(3):
2     print("Ich übe Python.")
3     print("Das macht Spaß!")
```

AUSGABE

```
Ich übe Python.
Das macht Spaß!
Ich übe Python.
Das macht Spaß!
Ich übe Python.
Das macht Spaß!
```

```
1 for i in range(3):
2     print("Ich übe Python.")
3 print("Das macht Spaß!")
```

AUSGABE

```
Ich übe Python.
Ich übe Python.
Ich übe Python.
Das macht Spaß!
```

Im ersten Programm gehören beide `print`-Anweisungen zum Rumpf und werden dreimal ausgeführt. Im zweiten Programm gehört nur die erste dazu. Die letzte Zeile wird erst nach der Schleife ausgeführt, also nur einmal.

Stolperstein

Vergisst du die Einrückung, meldet Python einen Fehler, bevor das Programm überhaupt startet:

```
for i in range(3):
print(i)
```

FEHLERMELDUNG

```
  File "programm.py", line 2
    print(i)
    ^^^^^
IndentationError: expected an indented block after 'for' statement on line 1
```

IndentationError heißt „Einrückungsfehler“. Die Meldung sagt: „Nach der for-Anweisung in Zeile 1 wurde ein eingerückter Block erwartet.“ Die Zeichen ^^^^^ zeigen auf die Zeile 2, die eingerückt sein müsste. Rücke den Rumpf mit 4 Leerzeichen (oder der Tab-Taste) ein.

Fehlt der Doppelpunkt am Ende des Kopfes, kommt diese Meldung:

```
for i in range(3)
    print(i)
```

FEHLERMELDUNG

```
SyntaxError: expected ':'
```

Versuch: Probiere es aus

Tippe das Programm mit „Runde“ ab. Ändere die Zahlen in `range(1, 4)` und beobachte, wie sich die Ausgabe ändert. Rücke dann `print("Fertig")` ein. Was passiert jetzt?

1.5.3 Zahlenfolgen mit `range`

`range` heißt auf Deutsch „Bereich“. Es erzeugt eine Folge von ganzen Zahlen, die die Zählvariable nacheinander annimmt. Es gibt drei Formen:

- **`range(stop)`** beginnt bei `0` und endet **vor** `stop`. `range(5)` liefert `0, 1, 2, 3, 4` – genau 5 Zahlen. Diese Form nimmst du, wenn es nur auf die Anzahl der Durchläufe ankommt.
- **`range(start, stop)`** beginnt bei `start` und endet wieder **vor** `stop`. Die Zahl `stop` selbst ist nie dabei.
- **`range(start, stop, schritt)`** zählt mit der **Schrittweite** `schritt` weiter. Mit einer negativen Schrittweite zählt man rückwärts.

Aufruf	erzeugt die Zahlen	Anzahl
`range(5)`	0, 1, 2, 3, 4	5
`range(1, 6)`	1, 2, 3, 4, 5	5
`range(3, 8)`	3, 4, 5, 6, 7	5
`range(0, 10, 2)`	0, 2, 4, 6, 8	5
`range(1, 10, 3)`	1, 4, 7	3
`range(10, 0, -1)`	10, 9, 8, 7, 6, 5, 4, 3, 2, 1	10
`range(5, 1)`	keine Zahl (die Schleife läuft gar nicht)	0

Merke

- `range(n)` erzeugt die Zahlen `0, 1, ..., n-1`.
- `range(start, stop)` erzeugt die Zahlen von `start` bis `stop-1`.
- `range(start, stop, schritt)` erlaubt eine eigene Schrittweite.

Tipp

Eine Eselsbrücke: Die Zahl `stop` ist wie die Ziellinie beim Laufen. Man zählt bis kurz davor, die Linie selbst gehört nicht mehr dazu. Bei `range(start, stop)` gibt es genau `stop - start` Zahlen.

Beispiel: Zahlen ausgeben und das Einmaleins

Das folgende Programm zeigt zwei Schleifen hintereinander. Überlege zuerst, was es ausgibt. Lies dann weiter.

```
1 for i in range(1, 6):
2     print(i)
3
4 print("---")
5
6 for i in range(1, 11):
7     print("7 *", i, "=", 7 * i)
```

AUSGABE

```
1
2
3
4
5
---
7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
7 * 10 = 70
```

- **Zeilen 1–2:** Die erste Schleife gibt die Zahlen von 1 bis 5 aus. Für „1 bis 5“ braucht man `range(1, 6)`, weil der Stoppwert nicht dazugehört.
- **Zeile 4:** Ein Trennstrich. Er steht nicht eingerückt, gehört also zu keiner Schleife und wird einmal ausgegeben.
- **Zeilen 6–7:** Die zweite Schleife läuft mit `i` von 1 bis 10. In jedem Durchlauf wird eine Zeile des Siebener-Einmaleins ausgegeben. `"7 *"` und `"="` sind Texte, `i` und `7 * i` sind Zahlen. Der Wert `7 * i` wird in jedem Durchlauf neu ausgerechnet.
- Beide Schleifen benutzen den Namen `i`. Das ist erlaubt: Die zweite Schleife gibt `i` einfach wieder neue Werte.

Rückwärts zählen

Mit der Schrittweite `-1` zählt die Schleife rückwärts. `for i in range(5, 0, -1):` lässt `i` die Werte `5, 4, 3, 2, 1` annehmen – ein Countdown. Auch hier gehört der Stoppwert `0` nicht mehr dazu.

Stolperstein

Der häufigste Fehler bei `range`: Man vergisst, dass der Stoppwert **nicht** dazugehört. `range(1, 10)` erzeugt die Zahlen von 1 bis **9**, nicht bis 10. Python meldet dabei keinen Fehler – das Programm tut einfach nicht ganz das, was du wolltest. Prüfe deshalb immer die erste und die letzte Zahl der Ausgabe. Für „von 1 bis 10“ schreibst du `range(1, 11)`.

Arbeitsauftrag 53: range-Detektiv

1. Gib an, welche Zahlen die folgenden `range`-Aufrufe erzeugen.
2. Gib zu den Zahlenfolgen in der unteren Tabelle einen passenden `range`-Aufruf an.

Überprüfe deine Antworten am Computer mit einer Schleife, die `i` ausgibt.

Aufruf	erzeugt die Zahlen
`range(4)`	
`range(2, 7)`	
`range(5, 30, 5)`	
`range(9, 3, -2)`	

Zahlenfolge	Aufruf
1, 2, 3, 4, 5, 6, 7, 8	
10, 20, 30, 40, 50	
3, 2, 1, 0	

Arbeitsauftrag 54: Zahlenreihe

Gib mit einer for-Schleife alle geraden Zahlen von 2 bis 20 aus (Tipp: Schrittweite nutzen!).

1.5.4 Summen bilden

Schleifen sind besonders nützlich, wenn viele Zahlen zusammengezählt werden sollen. Denk an eine Kasse im Supermarkt: Am Anfang steht die Anzeige auf 0. Bei jedem Artikel, der über das Band läuft, kommt sein Preis dazu. Am Ende steht der Gesamtbetrag auf der Anzeige.

Genauso geht man im Programm vor. Man braucht eine Variable, die die Zwischensumme speichert. Sie wird **vor** der Schleife auf `0` gesetzt. **In** der Schleife kommt bei jedem Durchlauf eine Zahl dazu. **Nach** der Schleife steht in ihr das Ergebnis. Eine solche Variable heißt **Summenvariable** (Fachwort: **Akkumulator**, von lateinisch *accumulare* = anhäufen).

Das Programm berechnet die Summe der Zahlen von 1 bis 5:

```
1 summe = 0
2 for i in range(1, 6):
3     summe = summe + i
4 print("Summe:", summe)
```

AUSGABE

```
Summe: 15
```

- **Zeile 1:** Die Summenvariable bekommt den Startwert `0` – die Kasse ist leer.
- **Zeile 2:** `i` läuft von 1 bis 5.

- **Zeile 3:** Der aktuelle Wert von `i` wird zur bisherigen Summe addiert. Das Ergebnis wird wieder in `summe` gespeichert. Diese Zeile kennst du schon aus `alter = alter + 1`: Erst wird rechts gerechnet, dann links gespeichert.
- **Zeile 4:** Nicht eingerückt – die Summe wird einmal ausgegeben, wenn die Schleife fertig ist.

Wie die Summe wächst, zeigt das Ablaufprotokoll. Lies vor allem die Spalte `summe` von oben nach unten: 0, 1, 3, 6, 10, 15.

Schritt	Zeile	Anweisung	i	summe
1	1	`summe = 0`	–	0
2	2	`for i in range(1, 6):`	1	0
3	3	`summe = summe + i`	1	1
4	2	`for i in range(1, 6):`	2	1
5	3	`summe = summe + i`	2	3
6	2	`for i in range(1, 6):`	3	3
7	3	`summe = summe + i`	3	6
8	2	`for i in range(1, 6):`	4	6
9	3	`summe = summe + i`	4	10
10	2	`for i in range(1, 6):`	5	10
11	3	`summe = summe + i`	5	15
12	2	`for i in range(1, 6):`	5	15
13	4	`print("Summe:", summe)`	5	15

Im Schritt 1 hat `i` noch keinen Wert (–), denn die Schleife hat noch nicht begonnen. Im Schritt 12 liefert `range` keine Zahl mehr, die Schleife endet.

Merke

Eine Summe berechnet man mit einer Summenvariablen in drei Schritten:

1. **vor** der Schleife: `summe = 0`
2. **in** der Schleife: `summe = summe + zahl`
3. **nach** der Schleife: Ergebnis ausgeben

Die Kurzschreibweise +=

Zeilen wie `summe = summe + i` kommen so oft vor, dass es dafür eine Abkürzung gibt: `summe += i`. Lies das als „`summe` wird um `i` erhöht“. Beide Schreibweisen bewirken genau dasselbe.

Kurzschreibweise	bedeutet	Beispiel (vorher `x = 10`)
`x += 3`	`x = x + 3`	`x` ist danach `13`
`x -= 3`	`x = x - 3`	`x` ist danach `7`
`x *= 3`	`x = x * 3`	`x` ist danach `30`

Im Summenprogramm kannst du also statt Zeile 3 auch `summe += i` schreiben. Du darfst beide Schreibweisen benutzen. Am Anfang ist die lange Form oft leichter zu lesen.

Stolperstein

Die Summenvariable steht in der Schleife. Dann wird sie in jedem Durchlauf wieder auf 0 gesetzt, und alles bisher Gezählte geht verloren:

```
1 for i in range(1, 6):
2     summe = 0
3     summe = summe + i
4 print("Summe:", summe)
```

AUSGABE

```
Summe: 5
```

Am Ende steht in `summe` nur noch die letzte Zahl `5`. Python meldet keinen Fehler, das Ergebnis ist einfach falsch. Der Startwert gehört **vor** die Schleife.

Die Ausgabe ist eingerückt. Dann wird nach jedem Durchlauf eine Zwischensumme ausgegeben:

```
1 summe = 0
2 for i in range(1, 6):
3     summe = summe + i
4     print("Summe:", summe)
```

AUSGABE

```
Summe: 1
Summe: 3
Summe: 6
Summe: 10
Summe: 15
```

Das kann zum Nachprüfen nützlich sein. Wenn nur das Ergebnis gefragt ist, darf `print` aber nicht eingerückt sein.

Arbeitsauftrag 55: Summe berechnen

Berechne mit einer for-Schleife die Summe aller Zahlen von 1 bis 100 (ohne eine fertige Summenfunktion zu verwenden) und gib das Ergebnis aus.

Die Obergrenze eingeben lassen

Die Schleife wird noch nützlicher, wenn die Nutzerin oder der Nutzer die Obergrenze selbst eingibt. Weil `range` nur mit ganzen Zahlen arbeitet, muss die Eingabe mit `int` umgewandelt werden:

```
1 n = int(input("Bis zu welcher Zahl soll ich addieren? "))
2 summe = 0
3 for i in range(1, n + 1):
4     summe += i
5 print("Die Summe von 1 bis", n, "ist", summe)
```

AUSGABE

```
Bis zu welcher Zahl soll ich addieren? 10
Die Summe von 1 bis 10 ist 55
```

- **Zeile 1:** Die Obergrenze wird eingelesen und in eine ganze Zahl umgewandelt.
- **Zeile 3:** Damit `n` selbst noch mitgezählt wird, muss der Stoppwert `n + 1` sein.
- **Zeilen 4–5:** wie im Beispiel oben, mit der Kurzschreibweise `+=`.

Stolperstein

Vergisst du die Umwandlung mit `int`, ist `n` eine Zeichenkette. Damit kann `range` nichts anfangen:

```
1 n = input("Bis zu welcher Zahl soll ich addieren? ")
2 for i in range(1, n):
3     print(i)
```

FEHLERMELDUNG

```
TypeError: 'str' object cannot be interpreted as an integer
```

Die Meldung heißt: „Ein Wert vom Typ `str` kann nicht als ganze Zahl verstanden werden.“ Dieselbe Meldung mit `'float'` bekommst du, wenn du `float` statt `int` benutzt. `range` braucht immer **ganze Zahlen**.

Arbeitsauftrag 56: Ablaufprotokoll

Fülle das Ablaufprotokoll zu dem Programm aus. Trage nach jedem Schritt die Werte von `i` und `x` ein (– bedeutet: noch kein Wert). Gib dann an, was das Programm ausgibt.

```
1 x = 0
2 for i in range(3):
3     x = x + 2 * i
4 print("x =", x)
```

Schritt	Zeile	Anweisung	Wert von `i`	Wert von `x`
1	1	`x = 0`	-	
2	2	`for i in range(3):`		
3	3	`x = x + 2 * i`		
4	2	`for i in range(3):`		
5	3	`x = x + 2 * i`		
6	2	`for i in range(3):`		
7	3	`x = x + 2 * i`		
8	2	`for i in range(3):`		
9	4	`print("x =", x)`		

Ausgabe:

1.5.5 Muster ausgeben

Mit Schleifen kann man auch Bilder aus Zeichen zeichnen. Dafür brauchst du ein kleines Hilfsmittel.

Werkzeug: Texte vervielfachen mit *

Multiplizierst du eine Zeichenkette mit einer ganzen Zahl, wird der Text so oft hintereinandergehängt:

```
print("ab" * 3)
print("-" * 20)
```

AUSGABE

```
ababab
--------------------
```

Die Zahl muss eine ganze Zahl (`int`) sein. Sie darf auch eine Variable sein, zum Beispiel die Zählvariable einer Schleife.

Arbeitsauftrag 57 (Zusatz): Sternendreieck

Gib mit einer for-Schleife folgendes Muster aus:

```
*
**
***
****
*****
```

So kannst du es sagen

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

- *„Das wird fünfmal gemacht.“ → „Die Anweisungen im Schleifenrumpf werden fünfmal wiederholt.“*
- *„i wird immer eins mehr.“ → „Der Wert der Zählvariablen `i` wird bei jedem Durchlauf um 1 erhöht.“*
- *„Da kommt 15 raus.“ → „Die Schleife addiert die Zahlen von 1 bis 5 zur Summenvariablen. Nach der Schleife gibt das Programm den Wert 15 aus.“*

1.5.6 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 58: Wie oft läuft die Schleife? ●○○

Bestimme für jeden Schleifenkopf, wie oft die Schleife durchlaufen wird und welchen Wert `i` im ersten und im letzten Durchlauf hat. Überprüfe am Computer, indem du in den Rumpf `print(i)` schreibst.

Schleifenkopf	Anzahl Durchläufe	erster Wert	letzter Wert
`for i in range(6):`			
`for i in range(1, 6):`			
`for i in range(3, 10):`			
`for i in range(0, 20, 5):`			
`for i in range(1, 2):`			
`for i in range(4, 4):`			

Arbeitsauftrag 59: Runde für Runde ●○○

Lies das Programm, **ohne** es auszuführen. Schreibe auf, was es ausgibt. Überprüfe deine Vorhersage danach am Computer.

```
1 for i in range(1, 4):
2     print("Runde", i)
3     print("---")
4 print("Spiel vorbei")
```

Meine Vorhersage:

Arbeitsauftrag 60: Einmaleins nach Wahl ●○○

Das Programm soll eine Zahl einlesen und ihr Einmaleins von 1 bis 10 ausgeben, zum Beispiel für 9 die Zeilen `9 * 1 = 9` bis `9 * 10 = 90`. Ergänze die drei Lücken `___`.

```
1 zahl = int(input("Welche Reihe? "))
2 for i in range(___, ___):
3     print(zahl, "*", i, "=", ___)
```

Arbeitsauftrag 61: Quadratzahlen ●○○

Schreibe ein Programm, das mit einer for-Schleife die Quadratzahlen von 1 bis 10 ausgibt. Jede Zeile soll so aussehen wie im Beispiel. Verwende den Operator `**`.

AUSGABE

```
1 hoch 2 ist 1
2 hoch 2 ist 4
3 hoch 2 ist 9
```

Arbeitsauftrag 62: Fehler finden: Ungerade Zahlen addieren ●●○

Das Programm soll die Summe der ungeraden Zahlen von 1 bis 19 berechnen (1 + 3 + 5 + … + 19) und **einmal** ausgeben. Richtig wäre `Summe: 100`. Das Programm läuft ohne Fehlermeldung, enthält aber **drei Fehler**.

```
1 for i in range(1, 19, 2):
2     summe = 0
3     summe = summe + i
4     print("Summe:", summe)
```

1. Führe das Programm aus und beschreibe, was falsch ist.
2. Finde die drei Fehler und schreibe das Programm richtig auf.

Die Fehler:

Gymnasium Ottweiler

Arbeitsauftrag 63: Ablaufprotokoll: Abziehen ●●○

Fülle das Ablaufprotokoll zu dem Programm aus. Trage nach jedem Schritt die Werte von `i` und `wert` ein (– bedeutet: noch kein Wert). Gib dann an, was das Programm ausgibt.

```
1 wert = 10
2 for i in range(2, 8, 2):
3     wert = wert - i
4 print("wert =", wert)
```

Schritt	Zeile	Anweisung	i	wert
1	1	`wert = 10`		
2	2	`for i in range(2, 8, 2):`		
3	3	`wert = wert - i`		
4	2	`for i in range(2, 8, 2):`		
5	3	`wert = wert - i`		
6	2	`for i in range(2, 8, 2):`		
7	3	`wert = wert - i`		
8	2	`for i in range(2, 8, 2):`		
9	4	`print("wert =", wert)`		

Ausgabe:

Arbeitsauftrag 64: Sparschwein ●●○

Im Sparschwein liegt schon ein Startbetrag. Jede Woche kommt derselbe Betrag dazu. Schreibe ein Programm, das den Startbetrag, den Betrag pro Woche und die Anzahl der Wochen (alles ganze Zahlen) einliest. Es soll für jede Woche den Stand ausgeben.

So könnte es aussehen:

AUSGABE

```
Startbetrag in Euro: 10
Betrag pro Woche in Euro: 3
Wie viele Wochen? 4
Nach Woche 1 sind 13 Euro im Sparschwein.
Nach Woche 2 sind 16 Euro im Sparschwein.
Nach Woche 3 sind 19 Euro im Sparschwein.
Nach Woche 4 sind 22 Euro im Sparschwein.
```

Arbeitsauftrag 65: Notendurchschnitt ●●●

Schreibe ein Programm, das zuerst einliest, wie viele Noten es gibt. Danach soll es mit einer for-Schleife jede Note einzeln einlesen. Am Ende soll es die Summe der Noten und den Durchschnitt ausgeben, auf zwei Nachkommastellen gerundet.

Teste mit den Noten 2, 3, 1 und 2 (Durchschnitt 2.0) und mit 2, 2 und 1 (Durchschnitt 1.67). Was passiert, wenn du am Anfang `0` eingibst? Verbessere dein Programm so, dass es dann eine Meldung ausgibt, statt abzubrechen.

Tipp

Die Eingabe der einzelnen Noten gehört **in** den Schleifenrumpf. Dann wird sie in jedem Durchlauf neu ausgeführt. Den Durchschnitt berechnest du **nach** der Schleife: Summe geteilt durch Anzahl.

1.5.7 Das Wichtigste in Kürze

- Eine **Schleife** wiederholt einen Anweisungsblock (den **Schleifenrumpf**) mehrfach.
- Die **for-Schleife** nimmt man, wenn die Anzahl der Wiederholungen vorher feststeht.
- Aufbau: `for i in range(...):` – Kopf mit Doppelpunkt, darunter der eingerückte Rumpf. Was nicht eingerückt ist, kommt nach der Schleife.
- Die **Zählvariable** bekommt bei jedem Durchlauf automatisch den nächsten Wert aus `range`.
- `range(n)` liefert `0` bis `n-1`, `range(start, stop)` liefert `start` bis `stop-1`, `range(start, stop, schritt)` zählt mit eigener Schrittweite. Der Stoppwert gehört **nie** dazu.
- Summen: Summenvariable **vor** der Schleife auf 0 setzen, **in** der Schleife erhöhen, **nach** der Schleife ausgeben. `summe += i` ist die Kurzform von `summe = summe + i`.

Selbstcheck

- ☐ Ich kann erklären, was eine Schleife ist und wann man eine for-Schleife verwendet.
- ☐ Ich kann eine for-Schleife mit Doppelpunkt und richtig eingerücktem Rumpf schreiben.
- ☐ Ich kann zu einem `range`-Aufruf angeben, welche Zahlen er erzeugt – und umgekehrt.
- ☐ Ich kann mit einer Schleife eine Summe berechnen und weiß, wo `summe = 0` stehen muss.
- ☐ Ich kann `+=` benutzen und weiß, was es bedeutet.
- ☐ Ich kann zu einer Schleife ein Ablaufprotokoll aufstellen.