

Ugh, today's kids are forgetting the old-fashioned art of absentmindedly reading the same half-page of a book over and over and then letting your attention wander and picking up another book.

9. Schleifen

In diesem Kapitel möchten wir ein neues Programmierkonzept kennenlernen, um eine Folge von Befehlen durch Python wiederholt ausführen zu lassen. Mit einer **Schleife** kann Python die Wiederholung von Befehlen direkt übernehmen, ohne dass wir die Befehle mehrmals eintippen müssen. Es gibt in Python zwei Schleifentypen. Wir kümmern uns zunächst nur um einen Schleifentyp – die **for**-Schleife. Der andere Schleifentyp (**while**-Schleife) verschieben wir auf einen späteren Zeitpunkt. Die Lernziele für dieses Kapitel lauten:

- ☐ Sie verwenden eine **for**-Schleife, um Befehle wiederholt ausführen zu lassen und somit ein Python-Programm kompakter zu gestalten.
- ☐ Sie notieren den Programmablauf für ein gegebenes Python-Programm, ohne dabei das Programm am Computer ausführen zu lassen.

9.1 Quadrat ultra kompakt

Bislang waren die „Turtle-Programme“ meist sehr repetitiv. Bereits im Programm für das Quadrat mit der Seitenlänge 100 (Listing 3.1) wiederholt sich der Funktionsaufruf `fd(100)` und `lt(90)` jeweils viermal. Wir können nun in Python ein Programm erstellen (siehe Listing 9.1), um diese beiden Funktionsaufrufe durch Python viermal wiederholen zu lassen.

```
1 import turtle
2
3 for i in [1, 2, 3, 4]:
4     turtle.fd(100)
5     turtle.lt(90)
6 turtle.done()
```

Listing 9.1: Zeile vier und fünf wird jeweils viermal ausgeführt (`quadrat_loop.py`).


Die **for**-Schleife besteht aus zwei Teilen (Schleifenkopf und Schleifenkörper):

- **Schleifenkopf:** **for** und **in** sind Schlüsselwörter von Python und wir müssen diese exakt so notieren. Dazwischen müssen wir eine **Variable** angeben. Danach notieren wir in den **eckigen Klammern** so viele Werte, wie Wiederholungen durchgeführt werden sollen. Den Schleifenkopf beenden wir mit einem **Doppelpunkt**.

■ **Beispiel 9.1** In Listing 9.1 befindet sich der Schleifenkopf in Zeile 3. Die Variable ist `i` und es sind vier Werte in den eckigen Klammern notiert. Es werden deshalb vier Wiederholungen durchgeführt. ■

- **Schleifenkörper:** In diesem Schleifenteil notieren wir die Befehle, welche wiederholt ausgeführt werden sollen. Diese Befehle müssen wir gleichmässig **einrücken**. Alle eingerückten Befehle müssen denselben Abstand zum „Rand“ besitzen. Die Einrückung dient **nicht** zur optischen „Verschönerung“. Nur durch die Einrückung (eng. indentation) weiss Python, welche Befehle wiederholt werden müssen. Die eingerückten Befehle bilden den Schleifenkörper.

■ **Beispiel 9.2** In Listing 9.1 bilden die Zeilen 4 und 5 den Schleifenkörper, da diese Zeilen eingerückt sind. Zeile 6 gehört nicht mehr dazu, da die Zeile nicht mehr eingerückt ist. ■

 **Clean Code 5 — Leerzeichen 2.** In den eckigen Klammern wird **nach** einem Komma ein Leerzeichen eingefügt.

9.2 Typische Fehlerquellen


Achten Sie beim Programmieren auf die folgenden, typischen Fehler:

- Schlüsselwörter falsch geschrieben.
- Doppelpunkt am Ende des Schleifenkopfs vergessen.
- Schleifenkörper ist nicht korrekt eingerückt.

Listing 9.2 zeigt ein fehlerhaftes Programm mit drei Fehlern. In Zeile 3 ist das Schlüsselwort `IN` nicht korrekt geschrieben und es fehlt der Doppelpunkt am Ende der Zeile. In Zeile 5 ist die Einrückung nicht korrekt. Der Befehl muss um ein Leerzeichen nach links gerückt werden.

```
1 import turtle
2
3 for k IN [1, 2, 3, 4]
4     turtle.fd(100)
5     turtle.lt(90)
6 turtle.done()
```

Listing 9.2: Fehlerhaftes Programm (`quadrat_loop_fehler.py`).

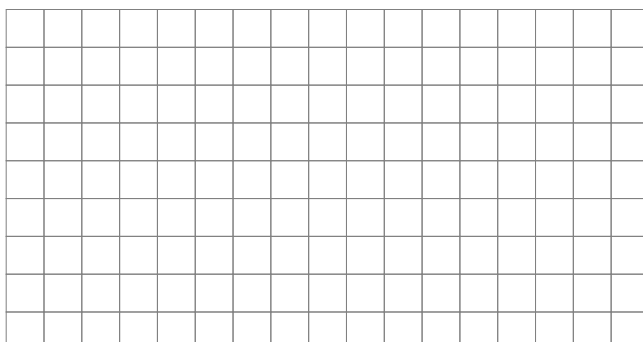
 Die Variable im Schleifenkopf muss nicht `i` heissen. Es handelt sich um eine „gewöhnliche“ Variable. Sie können den Namen selbst wählen.

9.3 Aufgaben

In den folgenden Aufgaben setzen Sie sich mit der `for`-Schleife auseinander.

9.3.1 Aufgabe 1

Kürzen Sie das Programm aus Listing 9.3 mit einer `for`-Schleife ab. Notieren Sie das Programm direkt hier auf dem Papier.



```
1 import turtle
2
3 turtle.fd(50)
4 turtle.lt(120)
5 turtle.fd(50)
6 turtle.lt(120)
7 turtle.fd(50)
8 turtle.lt(120)
9 turtle.done()
```

Listing 9.3: Gleichseitiges Dreieck.

9.3.2 Aufgabe 2

Notieren Sie mithilfe einer **for**-Schleife ein Programm, das ein regelmäßiges 6-Eck (siehe Abbildung 9.1) zeichnet. Alle Befehle zum Zeichnen müssen im Schleifenkörper notiert werden.

H Ein regelmäßiges n -Eck ist ein Vieleck mit n gleich langen Seiten, dessen n Innenwinkel alle gleich gross sind.

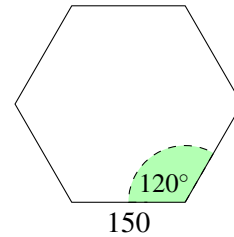
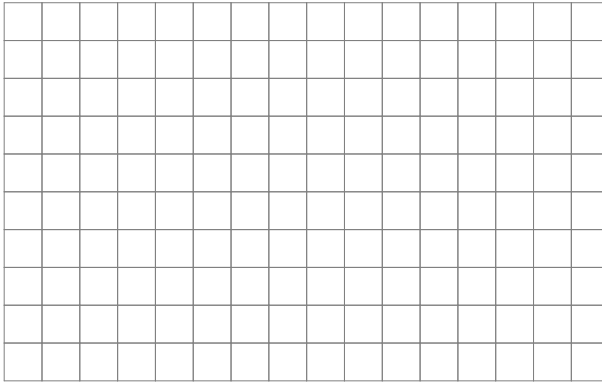


Abbildung 9.1: Ein regelmäßiges 6-Eck mit Seitenlänge 150.

9.4 Programmablauf

Mit dem Programmablauf meinen wir die konkrete Reihenfolge der Befehle, wenn wir ein Programm ausführen. Wir notieren darin die Reihenfolge der Code-Zeilen. Zu jeder Code-Zeile notieren wir auch noch den Speicherinhalt aller selbstdefinierten Variablen.

■ **Beispiel 9.3** Wir schauen uns den Programmablauf aus Listing 9.4 in Tabelle 9.1 an.

| Code-Zeile | Speicherinhalt der Variablen |
|------------|------------------------------|
| | Variablenname |
| | i |
| 1 | ∅ |
| 2 | ∅ |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 |
| 3 | 2 |
| 4 | 2 |
| 5 | 2 |
| 3 | 3 |
| 4 | 3 |
| 5 | 3 |
| 3 | 4 |
| 4 | 4 |
| 5 | 4 |
| 3 | 4 |
| 6 | 4 |

Tabelle 9.1: Wir notieren den Speicherinhalt immer **nachdem** die Code-Zeile ausgeführt wurde.

```

1  import turtle
2
3  for i in [1, 2, 3, 4]:
4      turtle.fd(100)
5      turtle.lt(90)
6  turtle.done()

```

Listing 9.4: Zeile vier und fünf wird jeweils viermal ausgeführt (`quadrat_loop.py`).

Wichtig! Der gespeicherte Wert in der Variablen `i` wird während des Programmablaufs verändert. Der zuvor gespeicherte Wert in der Variablen `i` wird durch den neuen Wert **ersetzt**. Wir sagen auch, dass der Wert überschrieben wird. Der zuvor gespeicherte Wert ist nicht mehr vorhanden. ■

Durch den Programmablauf aus Tabelle 9.1 wird nun auch deutlich, warum ein benannter Speicherplatz als **Variable** bezeichnet wird. Der Wert im Speicherplatz kann **variieren**.

9.5.2 Aufgabe 2

Das Listing 9.6 ist gegeben. Lesen Sie das Programm zunächst durch. Klären Sie unbekannte Elemente. Beginnen Sie dann mit den Teilaufgaben.

```
1 import math
2
3 for k in [49, 64, 100, 16, 36, 4]:
4     quadratwurzel = math.sqrt(k)
5     print(f"Die Quadratwurzel von {k} ist {quadratwurzel}.")
```

9.6 Tipparbeit sparen

9.6.1 range-Funktionsaufruf

Wir können den Schleifenkopf etwas kompakter gestalten, in dem wir die eckigen Klammern inklusive Inhalt durch einen `range`-Funktionsaufruf ersetzen. Listing 9.7 zeigt den Einsatz der **eingebauten Funktion**. Wir können uns den Funktionsaufruf so vorstellen, als würden wir in den eckigen Klammern die Zahlen 0, 1, 2 und 3 notieren (siehe Listing 9.8).

```
1 import turtle
2
3 for i in range(4):
4     turtle.fd(100)
5     turtle.lt(90)
6 turtle.done()
```

Listing 9.7: `quadrat_loop_range.py`

```
1 import turtle
2
3 for i in [0, 1, 2, 3]:
4     turtle.fd(100)
5     turtle.lt(90)
6 turtle.done()
```

Listing 9.8: `quadrat_loop.py`

Es gibt verschiedene Möglichkeiten die `range`-Funktion aufzurufen. Die **Reihenfolge** der Argumente ist dabei stets zu beachten.

- `range(stop)`: erzeugt den Zahlenbereich 0 bis $\text{stop} - 1$
- `range(start, stop)`: erzeugt den Zahlenbereich start bis $\text{stop} - 1$
- `range(start, stop, step)`: wie zuvor, aber mit der Schrittgrösse `step`

Wichtig! Das Argument `start` ist in der ersten Variante optional. Standardmässig wird dann für `start` die Zahl 0 benutzt. In der ersten und zweiten Variante ist das Argument `step` nicht vorhanden. Es ist auch optional. Es wird dann standardmässig 1 für `step` benutzt. ■

■ **Beispiel 9.4** Konkrete `range`-Funktionsaufrufe und deren Ergebnisse:

- `range(5)` erzeugt die Zahlen 0, 1, 2, 3 und 4.
- `range(3, 6)` erzeugt die Zahlen 3, 4 und 5.
- `range(0, 10, 2)` erzeugt die Zahlen 0, 2, 4, 6 und 8.

■

9.6.2 import-Aliasing

Unsere Programme werden noch kompakter, wenn wir bei `import`-Anweisungen einen Alias (dt. Parallelbezeichnung) verwenden. Dadurch müssen wir nicht jedes Mal den vollständigen Modulnamen verwenden, sondern den von uns gewählten Alias. Listing 9.9 zeigt ein Beispiel.

```
1 import turtle as t
2
3 for i in range(4):
4     t.fd(100)
5     t.lt(90)
6 t.done()
```

Listing 9.9: Das Turtle-Modul erhält in Zeile 1 die Bezeichnung `t`. Danach können wir die Befehle aus diesem Modul nur noch mit dieser Bezeichnung verwenden (`quadrat_alias.py`).

Wir können für **jeden Modulimport** einen Alias benutzen. Dazu erweitern wir die `import`-Anweisung und verwenden am Ende das Schlüsselwort `as` mit einem gültigen Namen.

Wichtig! Zwei `import`-Anweisung sollten **nicht** den gleichen Alias besitzen. ■

9.7 Aufgaben

In den Aufgaben setzen Sie sich mit dem `range`-Funktionsaufruf und `import`-Aliasing auseinander.

9.7.1 Aufgabe 1

1. Notieren Sie die Konsolenausgabe für das Programm aus Listing 9.10.

| | | | | | | | | | |
|--|--|--|--|--|--|--|--|--|--|
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

```
1 for zahl in range(8):
2     print(zahl)
```

Listing 9.10: Aufgabe 1

2. Notieren Sie die Konsolenausgabe für das Programm aus Listing 9.11.

| | | | | | | | | | |
|--|--|--|--|--|--|--|--|--|--|
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

```
1 for zahl in range(1, 8):
2     print(zahl)
```

Listing 9.11: Aufgabe 2

3. Notieren Sie die Konsolenausgabe für das Programm aus Listing 9.12.

| | | | | | | | | | |
|--|--|--|--|--|--|--|--|--|--|
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

```
1 for zahl in range(8, 1):
2     print(zahl)
```

Listing 9.12: Aufgabe 3

4. Notieren Sie die Konsolenausgabe für das Programm aus Listing 9.13.

| | | | | | | | | | |
|--|--|--|--|--|--|--|--|--|--|
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

```
1 for zahl in range(1, 50, 5):
2     print(zahl)
```

Listing 9.13: Aufgabe 4

5. Notieren Sie die Konsolenausgabe für das Programm aus Listing 9.14.

| | | | | | | | | | |
|--|--|--|--|--|--|--|--|--|--|
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

```
1 for zahl in range(50, -20, -10):
2     print(zahl)
```

Listing 9.14: Aufgabe 5

6. Notieren Sie die Konsolenausgabe für das Programm aus Listing 9.15.

| | | | | | | | | | |
|--|--|--|--|--|--|--|--|--|--|
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

```
1 for zahl in range(1900, 2051, 25):
2     print(zahl)
```

Listing 9.15: Aufgabe 5

9.7.2 Aufgabe 2

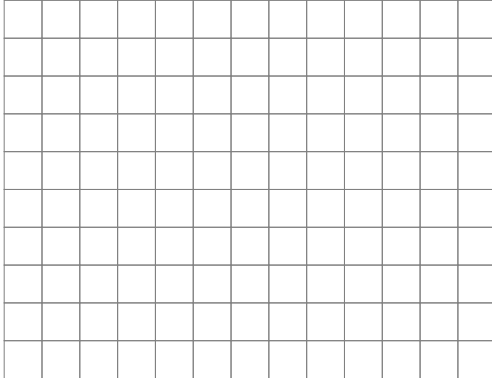
Passen Sie das Programm aus Listing 9.16 so an, dass im Schleifenkopf ein passender `range`-Funktionsaufruf stattfindet. Sie können die Korrektur direkt im Listing vornehmen.

```
1 for i in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:
2     print(f"Der Körper wird zum {i}. Mal ausgeführt.")
```

Listing 9.16: Ersetzen Sie den Schleifenkopf.

9.7.3 Aufgabe 3

Schreiben Sie das Programm aus Listing 9.17 so um, dass ein `import`-Alias, ein passender `range`-Funktionsaufruf und eine passende Variable für die Seitenlänge benutzt wird.



```
1 import turtle
2
3 for i in [1, 2, 3, 4, 5, 6]:
4     turtle.fd(150)
5     turtle.lt(60)
6 turtle.done()
```

Listing 9.17: Es wird ein regelmässiges 6-Eck mit der Seitenlänge 150 gezeichnet.

9.7.4 Aufgabe 4

Es ist das Programm aus Listing 9.18 gegeben. Studieren Sie das Programm. Klären Sie unbekannte Programmier-elemente. Lösen Sie dann die folgende Aufgabe: Schreiben Sie das Programm so um, dass ein `import`-Alias und ein passender `range`-Funktionsaufruf. Schaffen Sie es auch, dass der `range`-Funktionsaufruf eine sinnvolle Variable benutzt?

```
1 import random
2 import math
3
4 for i in [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]:
5     zahl = random.randrange(1, 100)
6     quadratwurzel = math.sqrt(zahl)
7     print(f"Die Quadratwurzel von {zahl} ist {quadratwurzel}.")
```

Listing 9.18: Es werden Quadratwurzeln berechnet.

