

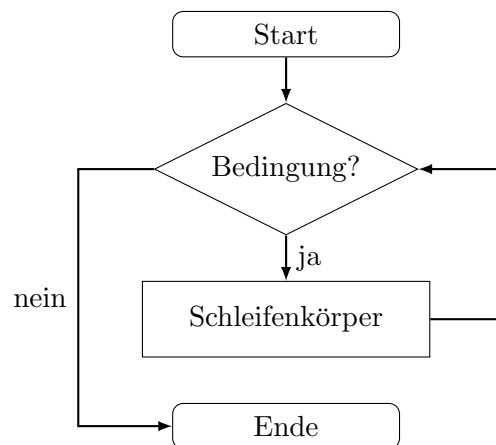
1 Schleifen – ein Blick auf den Prozessor

Ein Programm besteht im Wesentlichen aus einer Folge von Anweisungen, die der Prozessor nacheinander ausführt. Eine Schleife durchbricht diese lineare Abfolge: Ein Teil des Programms wird wiederholt ausgeführt.

Aus Sicht des Prozessors besteht eine Schleife im Wesentlichen aus drei Schritten:

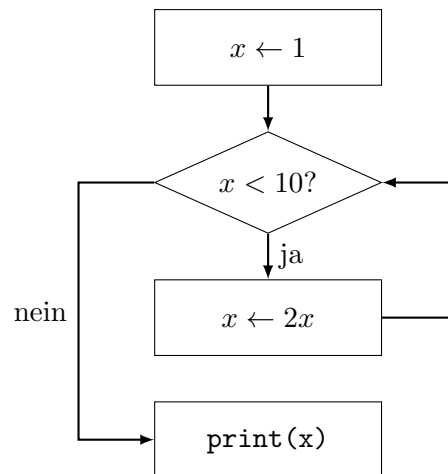
1. Eine Anweisung bzw. Bedingung wird ausgewertet.
2. Abhängig vom Ergebnis wird entschieden, ob der Schleifenkörper ausgeführt wird.
3. Anschliessend springt der Prozessor wieder zu einer früheren Stelle im Programm zurück.

Ein einfacher Programmablaufplan (PAP) für eine Schleife sieht beispielsweise so aus:



Die entscheidende Eigenschaft ist der *Rücksprung*: Nach der Ausführung des Schleifenkörpers wird die Bedingung erneut geprüft. Dadurch kann derselbe Programmabschnitt beliebig oft ausgeführt werden.

1.1 Übung: Einen PAP auswerten



Welche Ausgabe erzeugt der PAP?

16

Wie oft wird die Anweisung $x \leftarrow 2x$ ausgeführt? 4-mal (1 → 2, 2 → 4, 4 → 8, 8 → 16)

2 Schleifen in Python

Python stellt im Wesentlichen zwei Schleifenkonstrukte zur Verfügung:

- **for**: Wiederholung für eine vorgegebene Folge von Werten bzw. für die Elemente eines iterierbaren Objekts,
- **while**: Wiederholung, solange eine Bedingung erfüllt ist.

Welcher Schleifentyp geeignet ist, hängt vor allem davon ab, *wodurch die Wiederholung bestimmt wird*.

2.1 Die for-Schleife als Zählschleife

Wenn bekannt ist, wie oft eine Anweisung oder ein Programmblock ausgeführt werden soll, verwendet man häufig eine **for**-Schleife zusammen mit **range()**.

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

Die Ausgabe lautet:

```
0  
1  
2  
3  
4
```

Wichtig ist, dass der Endwert bei **range(5)** nicht mehr zur Folge gehört.

Man kann auch einen Anfangswert und eine Schrittweite angeben:

```
for i in range(2, 11, 2):  
    print(i)
```

Die Ausgabe ist:

```
2  
4  
6  
8  
10
```

2.2 Die for-Schleife für iterierbare Objekte

Eine `for`-Schleife muss nicht zählen. Sie kann unmittelbar über die Elemente einer Liste oder eines anderen iterierbaren Objekts laufen.

```
namen = ["Anna", "Ben", "Clara"]  
  
for name in namen:  
    print(name)
```

Hier wird die Variable `name` nacheinander mit den drei Elementen der Liste belegt.

Das ist häufig besser lesbar als eine Zählschleife mit Indizes:

```
namen = ["Anna", "Ben", "Clara"]  
  
for i in range(len(namen)):  
    print(namen[i])
```

Wenn der Index selbst nicht benötigt wird, ist die erste Variante vorzuziehen.

2.3 Wann verwendet man `for`?

Eine `for`-Schleife ist besonders geeignet, wenn

- eine bestimmte Anzahl von Wiederholungen vorgesehen ist,
- alle Elemente einer Liste, eines Tupels oder einer anderen Sequenz bearbeitet werden sollen,
- über einen Zahlenbereich iteriert werden soll,
- die Wiederholung durch eine vorhandene Folge von Werten bestimmt wird.

Typisch ist beispielsweise:

```
summe = 0

for x in [3, 5, 2, 7]:
    summe += x

print(summe)
```

2.4 Übung: Zählschleife

Was gibt das folgende Programm aus?

```
s = 0

for i in range(1, 6):
    s += i

print(s)
```

15

Wie oft wird der Schleifenkörper ausgeführt?

6 – 1 = 5-mal

2.5 Übung: Listen

Was gibt das folgende Programm aus?

```
zahlen = [2, 4, 7, 3]

for x in zahlen:
    if x > 3:
        print(x)
```

4
7

Erklären Sie in einem Satz, warum hier keine Zählvariable benötigt wird.

Weil `for x in zahlen:` nacheinander alle Elemente der Liste durchläuft.

2.6 Die while-Schleife

Eine `while`-Schleife wird verwendet, wenn sich die Anzahl der Wiederholungen nicht einfach durch eine Folge von Werten beschreiben lässt. Der Schleifenkörper wird ausgeführt, solange eine Bedingung wahr ist.

```
x = 1

while x < 20:
    print(x)
    x *= 2
```

Die Ausgabe lautet:

```
1
2
4
8
16
```

Die Bedingung wird *vor* jedem Durchlauf geprüft. Deshalb spricht man auch von einer *fussgesteuerten* Schleife?

2.7 Wann verwendet man while?

Eine `while`-Schleife ist besonders geeignet, wenn

- die Anzahl der Wiederholungen vorher nicht bekannt ist,
- die Wiederholung von einer Bedingung abhängt,
- ein Prozess so lange laufen soll, bis ein bestimmter Zustand erreicht ist.

Beispielsweise kann man eine Zahl so lange verdoppeln, bis eine Grenze erreicht ist:

```
x = 3

while x <= 100:
    x *= 2

print(x)
```

2.8 Übung: while

Was gibt das folgende Programm aus?

```
x = 20

while x > 1:
    print(x)
    x //= 2
```

Warum endet die Schleife?

Weil früher oder später der Wert von **x** kleiner als 1 wird.

2.9 Übung: for oder while?

Entscheiden Sie jeweils, welcher Schleifentyp geeigneter ist.

- (a) Die Zahlen von 1 bis 100 sollen ausgegeben werden.

for

- (b) Eine Liste mit Namen soll vollständig durchlaufen werden.

for

- (c) Ein Programm soll so lange rechnen, bis ein bestimmter Fehler kleiner als 10^{-8} ist.

while

- (d) Eine Eingabe soll so lange wiederholt werden, bis der Benutzer eine gültige Zahl eingibt.

while

3 Steuerung einer Schleife

Mit **continue** und **break** kann man den normalen Ablauf einer Schleife gezielt verändern.

3.1 continue

Mit **continue** wird der aktuelle Schleifendurchlauf vorzeitig beendet. Die Schleife wird anschliessend mit dem nächsten Durchlauf fortgesetzt.

```
for x in range(1, 7):
    if x == 4:
        continue
    print(x)
```

Die Ausgabe ist:

```
1
2
3
5
6
```

Der Wert 4 wird also nicht ausgegeben.

3.2 break

Mit **break** wird die gesamte Schleife sofort beendet.

```
for x in range(1, 10):
    if x == 5:
        break
    print(x)
```

Die Ausgabe lautet:

```
1
2
3
4
```

Der Unterschied lässt sich so zusammenfassen:

Schlüsselwort	Bedeutung
<code>continue</code>	aktuellen Durchlauf abbrechen
<code>break</code>	gesamte Schleife abbrechen

3.3 while True und der gezielte Schleifenabbruch

Manchmal soll eine Schleife zunächst ohne eine natürliche Abbruchbedingung gestartet werden. Dafür kann man schreiben:

```
while True:
    ...
```

Die Bedingung **True** ist immer wahr. Die Schleife würde daher von selbst nie enden. Ein **break** sorgt für den Abbruch.

```
x = 1

while True:
    print(x)
    x *= 2
    if x > 10:
        break
```

Die Ausgabe ist:

1
2
4
8

Diese Form ist besonders nützlich, wenn die Abbruchbedingung erst *innerhalb* des Schleifenkörpers sinnvoll geprüft werden kann.

Ein typisches Beispiel ist die Verarbeitung von Eingaben:

```
while True:
    text = input("Eingabe: ")
    if text == "ende":
        break

    print("Sie haben", text, "eingegeben.")
```

Die Schleife läuft hier so lange, bis das spezielle Abbruchsignal "ende" eingegeben wird.

3.4 Übung: continue

Was gibt das folgende Programm aus?

```
for i in range(1, 8):
    if i % 2 == 0:
        continue
    print(i)
```

1
3
5
7

Was bewirkt hier die Anweisung `continue`?

`continue` beendet den aktuellen Schleifendurchlauf vorzeitig und fährt mit dem nächsten Schleifendurchlauf fort. Dadurch werden die geraden Zahlen nicht ausgegeben.

3.5 Übung: break

Was gibt das folgende Programm aus?

```
x = 2

while x < 100:
    print(x)
    if x >= 20:
        break
    x *= 2
```

2
4
8
16
32

An welcher Stelle wird die Schleife beendet?

Die Schleife wird durch die Anweisung **break** beendet, sobald **x** den Wert 32 erreicht hat.

3.6 Übung: while True

Vervollständigen Sie das Programm so, dass die Zahlen 1, 2, 3, 4, 5 ausgegeben werden und die Schleife danach beendet wird. *Einrückung: 2 Leerzeichen*

x	=	1																
w	h	i	l	e		T	r	u	e	:								
		p	r	i	n	t	(	x	)									
		x	=	x	+	1												
		i	f		x	>	5	:										
					b	r	e	a	k									

3.7 Übung: Verständnisfrage

Erklären Sie den Unterschied zwischen den folgenden beiden Konstruktionen:

```
while x < 10:  
    ...
```

und

```
while True:  
    ...  
    if x >= 10:  
        break
```

- Bei `while x < 10:` wird die Abbruchbedingung vor jedem Schleifendurchlauf geprüft. Sobald sie nicht mehr erfüllt ist, wird die Schleife beendet.
- Bei `while True:` ist die Schleifenbedingung immer erfüllt. Die Schleife wird deshalb erst durch die Anweisung `break` beendet, die hier ausgeführt wird, sobald `x >= 10` gilt. Die Abbruchbedingung wird also innerhalb des Schleifenkörpers geprüft.

Gibt es Situationen, in denen die zweite Form übersichtlicher sein kann?

Ja. Die zweite Form kann übersichtlicher sein, wenn die Abbruchbedingung erst innerhalb des Schleifenkörpers sinnvoll geprüft werden kann oder wenn es mehrere mögliche Abbruchbedingungen gibt.

4 Übungen zur Wiederholung

4.1 Übung: Ausgabe bestimmen

Welche Ausgabe erzeugt das folgende Programm?

```
s = 0  
  
for i in range(2, 8):  
    if i == 5:  
        continue  
  
    s += i  
  
print(s)
```

22

4.2 Übung: Verschachtelte Schleifen

Welche Ausgabe erzeugt das folgende Programm?

```
x = 1
while x < 20:
    if x == 8:
        break
    print(x)
    x *= 2
```

1
2
4

Warum wird die Zahl 8 nicht ausgegeben?

Bei `x = 8` beendet `break` die Schleife, bevor die Zahl durch `print(x)` ausgegeben werden kann.

4.3 Übung: Eine Schleife selbst entwerfen

Schreiben Sie eine `for`-Schleife, die die Quadrate der Zahlen von 1 bis 5 ausgibt:

f	o	r		i		i	n		r	a	n	g	e	(	1	,	6	)	:						
				p	r	i	n	t	(	i	*	*	2	)											

4.4 Übung: Eine while-Schleife selbst entwerfen

Schreiben Sie eine `while`-Schleife, die bei 100 beginnt und die Zahl jeweils halbiert, solange sie grösser als 1 ist. Geben Sie jeden Wert aus.

a	=	1	0	0																				
w	h	i	l	e		a	>	1	:															
		p	r	i	n	t	(	a	)															
		a	=	a	/	2																		

4.5 Übung: continue oder break?

Welche Anweisung passt jeweils?

1. Eine bestimmte Zahl soll übersprungen werden.

`continue`

2. Die Schleife soll vollständig beendet werden, sobald ein Fehler gefunden wurde.

`break`

3. Bei einer Liste sollen nur die positiven Zahlen verarbeitet werden.

`continue`

4.6 Übung: Gesamtverständnis

Beantworten Sie kurz:

1. Was ist der wesentliche Unterschied zwischen `for` und `while`?

`for` durchläuft ein iterierbares Objekt und `while` wiederholt einen Block, so lange eine Bedingung wahr ist.

2. Was bewirkt `continue`?

`continue` beendet den aktuellen Schleifendurchlauf und setzt die Schleife mit dem nächsten Durchlauf fort.

3. Was bewirkt `break`?

`break` bricht die gesamte Schleife ab.

4. Warum kann `while True` trotzdem eine endliche Schleife darstellen?

Weil eine Endlosschleife mit `break` vorzeitig verlassen werden kann.

5. Warum muss bei einer `while`-Schleife darauf geachtet werden, dass sich die Schleifenbedingung irgendwann ändert?

Weil die `while`-Schleife sonst unendlich weiterläuft.