Schleifen in Python Repetition

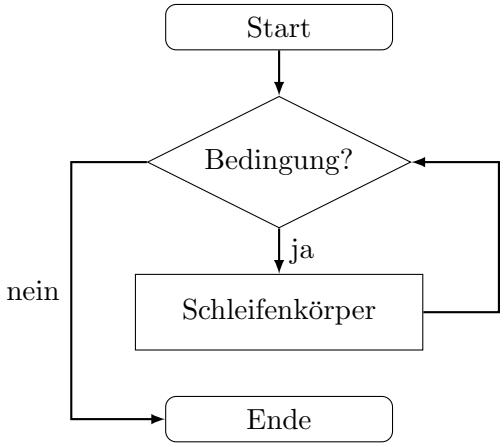
1 Schleifen – ein Blick auf den Prozessor

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:

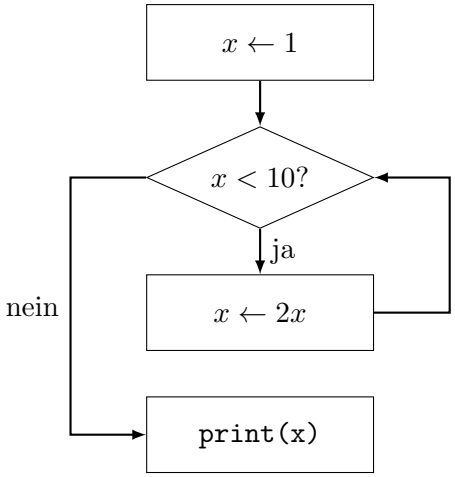
1. Eine Anweisung bzw. Bedingung wird ausgewertet.
2. Abhängig vom Ergebnis wird entschieden, ob der Schleifen- körper ausgeführt wird.
3. Anschliessend springt der Prozessor wieder zu einer früheren Stelle im Programm zurück.

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? _____

Wie oft wird die Anweisung $x \leftarrow 2x$ ausgeführt? _____

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)
```

Wie oft wird der Schleifenkörper ausgeführt?

2.5 Übung: Listen

Was gibt das folgende Programm aus?

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

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

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

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?

2.9 Übung: `for` oder `while`?

Entscheiden Sie jeweils, welcher Schleifentyp geeigneter ist.

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

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

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

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

---

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**
`continue`	aktuellen Durchlauf abbrechen
`break`	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)
```

Was bewirkt hier die Anweisung `continue`?

3.5 Übung: `break`

Was gibt das folgende Programm aus?

```
x = 2

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

An welcher Stelle wird die Schleife beendet?

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.

```
x = 1

while True:
    print(x)

    ----------------------

    ----------------------

    ----------------------
```

3.7 Übung: Verständnisfrage

Erklären Sie den Unterschied zwischen den folgenden beiden Konstruktionen:

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

und

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

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

---

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)
```

4.2 Übung: Verschachtelte Schleifen

Welche Ausgabe erzeugt das folgende Programm?

```
x = 1

while x < 20:
    if x == 8:
        break

    print(x)
    x *= 2
```

Warum wird die Zahl 8 nicht ausgegeben?

4.3 Übung: Eine Schleife selbst entwerfen

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

```
1
4
9
16
25
```

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.

4.5 Übung: `continue` oder `break`?

Welche Anweisung passt jeweils?

1. Eine bestimmte Zahl soll übersprungen werden.

   _____

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

   _____

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

   _____

4.6 Übung: Gesamtverständnis

Beantworten Sie kurz:

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

   _____

2. Was bewirkt `continue`?

   _____

3. Was bewirkt `break`?

   _____

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

   _____

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

   _____