

Einführung in die Programmierung mit Python

Lucerne University of
Applied Sciences and Arts

**HOCHSCHULE
LUZERN**

Syllabus

1. Python installieren; Jupyter Notebooks
2. Grundlagen: Arithmethik, Variablen und Datentypen
3. Methoden; Listen; Bedingte Anweisungen
- 4. Schleifen**
5. Benutzerdefinierte Funktionen
6. Datenaufbereitung und Grafische Darstellungen

Kontrollfluss

Schleifen

- Eine Schleife wird verwendet, wenn eine Gruppe von Anweisungen mehr als einmal ausgeführt werden soll.
- Python hat zwei Arten von Schleifen: `while`-Schleifen und `for`-Schleifen.
- Eine `while`-Schleife wird verwendet, um Code auszuführen, solange eine Bedingung wahr ist.
- Eine `for`-Schleife wird verwendet, um Code eine vorher festgelegte Anzahl von Malen auszuführen.

While-Schleifen

- Ähnlich wie `if`, aber springt nach dem Ende des `while`-Blocks zurück zur `while`-Anweisung.

```
In [1]: x = -1 # warum ist dies nötig?  
while x < 0 or x > 9:  
    x = int(input("Geben Sie eine Zahl zwischen 0 und 9 ein: "))  
    if x < 0:  
        print("Sie haben eine negative Zahl eingegeben.")  
    elif x > 9:  
        print("Sie haben eine Zahl größer als 9 eingegeben.")  
print("Danke. Sie haben " + str(x) + " eingegeben.")
```

Sie haben eine negative Zahl eingegeben.

Danke. Sie haben 9 eingegeben.

- Alternative Implementierung:

```
In [2]: while True:  
    x = int(input("Geben Sie eine Zahl zwischen 0 und 9 ein: "))  
    if x < 0:  
        print("Sie haben eine negative Zahl eingegeben.")  
    elif x > 9:  
        print("Sie haben eine Zahl größer als 9 eingegeben.")  
    else:  
        print("Danke. Sie haben " + str(x) + " eingegeben.")  
        break # Beendet die innerste umschließende Schleife.
```

Sie haben eine negative Zahl eingegeben.

Sie haben eine Zahl größer als 9 eingegeben.

Danke. Sie haben 9 eingegeben.

Bemerkung: Wie `if`-Blöcke können auch `while`-Schleifen einen `else`-Block haben. Dieser wird ausgeführt, wenn die Bedingung falsch ist (oder wird). Dasselbe gilt natürlich, wenn man den Code im `else`-Block *nach* der Schleife platziert, wie in unserer ersten Implementierung, aber die beiden Ansätze unterscheiden sich, wenn eine `break`-Anweisung vorhanden ist. Hier ist unsere erste Implementierung noch einmal, modifiziert, um dies zu verwenden:

```
In [3]: x = -1 # warum ist dies nötig?  
while x < 0 or x > 9:  
    x = int(input("Geben Sie eine Zahl zwischen 0 und 9 ein: "))  
    if x < 0:  
        print("Sie haben eine negative Zahl eingegeben.")  
    elif x > 9:  
        print("Sie haben eine Zahl größer als 9 eingegeben.")  
    else:  
        print("Danke. Sie haben " + str(x) + " eingegeben.")
```

Danke. Sie haben 9 eingegeben.

Übung: Zahlenratespiel

Implementieren Sie das folgende Spiel: Der Computer wählt eine Zufallszahl, und der Spieler muss sie erraten. Nach dem Raten erhält der Spieler Rückmeldung, ob er korrekt, zu niedrig oder zu hoch geraten hat. Das Spiel endet, wenn die richtige Zahl erraten wurde. Beginnen Sie mit dem folgenden Gerüst.

```
In [4]: import random # ein Standardbibliotheksmodul (d.h. eingebaut). Mehr dazu später.
lower = 0
upper = 100
to_guess = random.randint(lower, upper) # eine Funktion aus dieser Bibliothek.
current_guess = -1
while current_guess != to_guess:
    break
```

for-Schleifen

Dies ist die andere Art von Schleife, die von Python unterstützt wird. Sie iteriert über die Elemente einer Sequenz (z.B. einer Liste):

```
In [5]: for letter in "Python":  
    print(letter)
```

```
P  
y  
t  
h  
o  
n
```

letter wird als Schleifenvariable bezeichnet. Jedes Mal, wenn der Schleifenkörper ausgeführt wird, nimmt sie der Reihe nach den Wert jedes Elements der Sequenz an.

Range-Objekte

range -Objekte sind oft nützlich in Verbindung mit for -Schleifen, insbesondere für numerische Berechnungen. Sie repräsentieren Sequenzen von Ganzzahlen.

Syntax:

```
In [6]: myrange = range(1, 10, 2) # Start, Stop, Schritt; vgl. Slicing  
print(myrange)
```

```
range(1, 10, 2)
```

```
In [7]: type(myrange)
```

```
Out[7]: range
```

```
In [8]: list(myrange)
```

```
Out[8]: [1, 3, 5, 7, 9]
```

- Wie Sie sehen können, müssen wir das `range`-Objekt in eine Liste umwandeln, um seinen Inhalt zu sehen (dies sollte in der Praxis nicht getan werden).
- Grund: Ein `range` materialisiert seinen Inhalt nicht bei der Erstellung, sondern erst bei der Verwendung.
- Dies wird als `lazy` Berechnung bezeichnet.
- Vorteil: `range`-Objekte können riesig sein, ohne Speicher zu verbrauchen.
- Verwendung in `for`-Schleifen:

```
In [9]: squares = []
for element in range(1, 11): # Schritt ist optional
    squares.append(element ** 2)
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Quiz: Was berechnet der folgende Code?

In [10]:

```
n = 5
result = 1
for i in range(1, n+1):
    result = result * i # oder kürzer: result *= i
print(result)
```

120

for -Schleifen unterstützen break für ein frühzeitiges Abbrechen und eine else - Klausel, die ausgeführt wird, wenn die Schleife normal (d.h. nicht über break) endet:

In [11]:

```
import random
haystack = random.sample(range(1, 11), 10)
print(haystack)
```

```
[4, 7, 10, 1, 6, 9, 2, 5, 3, 8]
```

In [12]:

```
needle = 6
counter = 0
for elem in haystack:
    if elem == needle:
        print(needle, "gefunden an Position " + str(counter) + ".")
        break
    counter += 1
else:
    print(needle, " nicht gefunden.")
```

```
6 gefunden an Position 4.
```

(wir hätten natürlich einfach haystack.index(needle) verwenden können).

Schleifen können verschachtelt werden:

```
In [13]: for row in range(1, 6):
    for col in range(row):
        print("*", end="")
    print()
```

```
*
```

```
**
```

```
***
```

```
****
```

```
*****
```

Übung

Vervollständigen Sie das folgende Codesnippet, sodass es die Primzahlen bis N ausgibt (derzeit gibt es alle Zahlen zwischen 2 und N aus). Hinweise:

- `pass` ist ein syntaktischer Platzhalter. Dies müssen Sie ersetzen.
- Der "Modulo"-Operator `%` gibt den Rest nach der Division zurück, d.h., `a % b == 0` ist True, wenn und nur wenn a durch b teilbar ist.
- Verwenden Sie `break`.

```
In [14]: N = 100
for i in range(2, N+1):
    for j in range(2, i):
        pass
    else:
        pass
```

Hinweis: Dieser Algorithmus ist nicht effizient. Ein effizienterer Algorithmus ist das [Sieb des Eratosthenes](#).

Hausaufgabe

Erweitern Sie das Zahlenratespiel

1. Der Spieler hat nur eine begrenzte Anzahl von Versuchen, sagen wir $n=10$. Wenn diese erreicht ist, stoppt das Spiel mit einer Nachricht an den Spieler.
2. Der Spieler kann das Spiel durch Eingabe einer negativen Zahl beenden. Der Computer sagt dann "Tschüss.".
3. Verbesserte Rückmeldung: Wenn der Spieler zu hoch rät (z.B. `to_guess` ist 50 und der Benutzer rät 75), fragt der Computer in der nächsten Runde nach einer Zahl zwischen 0 und 75. Analog, wenn die Schätzung zu niedrig ist.

Weitere Hausaufgaben

while

- Intermediate Übungen 9a, 9b von <https://holypython.com/intermediate-python-exercises/>
- Optional: Übung 9 (schwierig) von <https://www.w3resource.com/python-exercises/python-conditional-statements-and-loop-exercises.php>

Hinweis für die Musterlösung von Übung 9: Zwei Variablen können gleichzeitig wie folgt zugewiesen werden:

```
In [15]: a, b = 1, 2  
        print(a, b)
```

```
1 2
```

for

- Übungen 1, 4, 6 von <https://www.w3resource.com/python-exercises/python-conditional-statements-and-loop-exercises.php>