CloudCommand

Cyber Security

CloudCommand GmbH chr.schumacher@gmx.tm

CloudCommand

Grundlagen der Programmierung

CloudCommand GmbH chr.schumacher@gmx.tm

Schleifen

Einleitung zu Schleifen

Was sind Schleifen?

- grundlegende Programmierkonstruktoren
- werden verwendet für wiederholte Ausführungen von Codeblöcken

Anwendungsfälle

Allgemeine Anwendungen von Schleifen sind:

- Datensammlung und -Analyse
- Automatisierung von (sich wiederholenden) Aufgaben

While-Schleife: Eigenschaften

Die while-Schleife in Python führt einen Codeblock so lange aus, wie eine bestimmte Bedingung erfüllt ist.

```
while Bedingung:
    # Codeblock
```

While-Schleife: Aufbau

- Einleitung mit dem Schlüsselwort „while“
- Bedingung, worauf ein Doppelpunkt folgt
- eingerückter Schleifenkörper

While-Schleife: Funktionsweise

- Programm überprüft, ob Bedingung der Schleife wahr ist
- falls true, wird Schleifenkörper komplett ausgeführt
- am Ende des Schleifenkörpers wird Bedingung erneut geprüft
- Schleifenkörper wird so oft ausgeführt, bis Bedingung false ist
- Schleife wird beendet

While-Schleife: Zähler

Mit einer while-Schleife lassen sich zum Beispiel Zähler programmieren:

- Codeblock zählt von 0 bis zur 4 hoch und bricht mit der 5 ab
- in jedem Durchgang wird der Zähler um 1 erhöht

```
counter = 0
while counter < 5:
    print(counter)
    doSomeStuff(counter)
    counter += 1
```

While-Schleife: Zähler

- Zähler ist innerhalb der Schleife in jedem Durchgang zugreifbar
- kann für weiterführende Programmlogik genutzt werden
  - auch als Akkumulator bezeichnet

```
summe = 0
counter = 0
while counter <= 5:
    summe += counter
    counter += 1
print(summe)
```

While-Schleife: Endlosschleife

- entsteht sobald Bedingung einer while-Schleife immer erfüllt bleibt
- im Regelfall nicht erwünscht
- können zu hoher CPU-Auslastung führen
- es kommt zu keinen Fortschritten im Programm
- zur Vermeidung muss die Bedingung der Schleife genau geprüft werden

Hier ein Beispiel einer Endlosschleife (Zähler wird nicht inkrementiert)

```
counter = 0
while counter < 5:
    print("Endlos")
```

For-Schleife: Eigenschaften

Die for-Schleife ist eine andere Art von Schleife, die sich besonders für die Iteration über Sequenzen eignet. Die grundlegende Syntax ist dabei sehr einfach.

```
for element in sequenz:
    # Codeblock
```

For-Schleife: Eigenschaften

- Iteration über einzelne Elemente von Listen, Dictionaries, Zeichenketten, ...
- können auch komplexere Datentypen (eigene Klassen, die Rückgaben von Datenbanken usw.) enthalten.

```python
#Liste
for num in [1, 2, 3, 4, 5]:
    print(num)

#Dictionary
myDict = { "zutat1": "Mehl",
           "zutat2": "Zucker",
           "zutat3": "Ei",
           "zutat4": "Milch"}

for key in myDict:
    print(f"Schlüssel: {key}, Wert: {myDict[key]}")

#Zeichenkette
for char in "Python":
    print(char)
```

For-Schleife: Aufbau

- Einleitung mit dem Schlüsselwort „for“
- frei wählbarer Bezeichner
- Schlüsselwort „in“
- iterierbares Objekt, worauf ein Doppelpunkt folgt
- eingerückter Schleifenkörper

For-Schleife: Funktionsweise

- jedes Element des iterierbaren Objekts wird mit eigenem Schleifendurchlauf durchlaufen
- Bezeichner greift auf erstes Element im iterierbaren Objekt zu
- Beendung des Schleifendurchlaufs
- nächster Durchlauf greift auf das nächste Element
- bis das Ende des iterierbaren Objekts erreicht ist

For-Schleife: Verschachtelung

Schleifen können auch ineinander verschachtelt werden, der gängigste Anwendungsfall sind zweidimensionale Arrays, Tabellenblätter und ähnliche Datenansammlungen.

```
#2D-Array
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for row in matrix:
    for col in row:
        print(col)

#Schleife ähnlich einer Tabelle
for i in [0, 1, 2, 3]:
    for j in [0, 1, 2, 3, 4]:
        print(i, j)
```

For-Schleife: range

- Helferfunktion range
- gibt eine Liste zurück
- erstes Element n der Liste ist die Null
- das übergebene Argument ist eine ganze Zahl, bei welcher die Funktion stoppt

```
#Schleifen mit handgeschriebenen Arrays
for i in [0, 1, 2, 3]:
    for j in [0, 1, 2, 3, 4]:
        print(i, j)

#die gleiche Schleife mit range()
for i in range(4):
    for j in range(5):
        print(i, j)
```

For-Schleife: range

Funktion range kann alternativ auch mit 2 bzw. 3 Parametern verwendet werden

```
#iteriert [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
for i in range(0, 110, 10):
        print(i)
```

For-Schleife: range

Parameter: range(start, stop, step)

- **Start:** stellt den Anfangswert ein, ansonsten startet range bei 0
- **Stop:** stellt den Endwert fest, muss immer angegeben werden, range hört bei stop-1 auf zu zählen
- **Step:** gibt die Schrittweite an, ansonsten zählt range in Einserschritten

For-Schleife: range

Beispiel

- Start = 2: range fängt bei 2 an zu zählen
- Stop = 10: range hört bei 9 auf zu zählen
- Step = 3: jede dritte Zahl wird gedruckt

Steueranweisungen in Schleifen

In Schleifen können außerdem **Steueranweisungen** verwendet werden. Diese sind:

- break
- continue
- pass
- else

Steueranweisung break

- break beendet die Schleife sofort vollständig
- wird unter Schleifenanweisung, nach bedingter if-Anweisung gesetzt
- wenn externe Bedingung ausgelöst wird, wird Schleife verlassen

Steueranweisung break

Das folgende Beispiel würde die for-Schleife ausführen, bis der Zähler i beginnend von 0 bis 4 hochgezählt wurde und in jedem Iterationsschritt i ausgeben.

- break beendet die Schleife bei i = 3
- nachdem die Werte von 0 bis inklusive 2 durchiteriert wurden

```
for i in range(5):
    if i == 3:
        break
    print(i)
```

Steueranweisung continue

- continue überspringt den aktuellen Iterationsschritt, ohne Schleife zu beenden
- wird unter Schleifenanweisung, nach bedingter if-Anweisung gesetzt
- wenn externe Bedingung ausgelöst wird, wird der Teil der Schleife übersprungen & der Rest zu Ende geführt

Steueranweisung continue

- überspringt den Iterationsschritt, wenn i zur 3 hochgezählt wurde
- fährt danach aber mit dem Wert i = 4 fort
- Schleife als regulär durchlaufen, nicht als abgebrochene Schleife

```
for i in range(5):
    if i == 3:
        continue
    print(i)
```

Steueranweisung pass

- dient als Platzhalter, um Syntaxfehler zu verhindern
- wird unter Schleifenanweisung, nach bedingter if-Anweisung gesetzt
- wenn externe Bedingung ausgelöst wird, dient pass dazu, die Schleife ohne jeglichen Einfluss fortzuführen

Steueranweisung pass

Das folgende Beispiel iteriert komplett von i = 0 bis i = 4 durch, als ob die if-Abfrage nicht existieren würde.

```
for i in range(5):
    if i == 3:
        pass
    print(i)
```

Steueranweisung else

- nicht mit der if-Abfrage zu verwechseln!
- Sobald Anweisung nicht mehr erfüllt ist, wird else-Teil ausgeführt
- nach vollständigem Durchlauf einer Schleife, wird else-Block immer ausgeführt
- ohne Abbruch durch break

Steueranweisung else

So wie in diesem Beispiel:

```
for i in range(3):
    print(i)
else:
    print("Ende der Schleife")
```

Komplexere Schleifen

list comprehensions:

- programmieren kompakter und eleganter Schleifen
- erzeugt Listen nach einem bestimmten Schema

```
[ausdruck for element in sequenz if bedingung]
```

Komplexere Schleifen

Hier ein anschauliches Beispiel mit Vergleich zu einer klassischen for-Schleife und der if-Abfrage darin:

- for-Schleife mit if-Abfrage darin

```
autos = ["VW Golf", "VW Passat", "VW Lupo", "VW up"]
kleinwagen = []

for auto in autos:
    if "Lupo" in auto or "up" in auto:
        kleinwagen.append(auto)

print(kleinwagen)
```

Komplexere Schleifen

Hier ein anschauliches Beispiel mit Vergleich zu einer klassischen for-Schleife und der if-Abfrage darin:

- list comprehension

```
autos = ["VW Golf", "VW Passat", "VW Lupo", "VW up"]

kleinwagen = [auto for auto in autos if "Lupo" in auto or "up" in auto]

print(kleinwagen)
```

Komplexere Schleifen

In diesem Beispiel wird keine bereits vorhandene Liste als Grundlage genommen...

```
quadrate = [x*x for x in range(1, 6)]
print(quadrate)
```

- mit der range-Funktion wird eine Liste der Zahlen von 1 bis 5 generiert
- Liste wird mittels list comprehension in neue Liste überführt
- jedes Element wird mit sich selbst multipliziert

Komplexere Schleifen

Fehlerbehandlung mit hilfe von Schleifen:

```
numbers = [1, 2, 3, 0, 4]

for num in numbers:
    try:
        print(10 / num)
    except ZeroDivisionError:
        print("Division durch Null!")
```

- Schleife verhindert das Teilen durch Null

Anwendungsbeispiel 1: Fibonacci-Reihe

Die Fibonacci-Reihe ist eine unendliche Reihe natürlicher Zahlen.

```
ende = 100

a, b = 0, 1
while b < ende:
    print(b)
    a, b = b, a+b
```

- die ersten beiden Zahlen sind 0 und 1
- jede weitere Zahl ist die Summe ihrer beiden Vorgänger in der Reihe
- damit keine Endlosschleife entsteht, rechnet das Beispiel nur bis zur Zahl 100

Anwendungsbeispiel 2: Zinseszinsrechnung

Zinseszinsrechnung mit einem Verlauf von 1000 € über 10 Jahre bei 5% Zinsen:

```
kapital = 1000  # Startkapital in Euro
zinssatz = 5  # Zinssatz pro Jahr
jahre = 10

for jahr in range(1, jahre + 1):
    kapital *= (1 + (zinssatz / 100))
    print(f"Jahr {jahr}: {kapital:.2f} Euro")
```

Anwendungsbeispiel 3: textbasierte Tools

Textbasierte Tools mit kleinen Menüs

- wichtig für echte Anwendung
- Logik der Menüpunkte passend in eigene Methoden zu kapseln statt kompletten Code in Schleife zu integrieren

```
while True:
    eingabe = input("Schreibe 'exit' zum Beenden oder 'weiter' zum Fortfahren: ")

    match eingabe:
        case 'exit':
            print("Die Schleife wird abgebrochen.")
            break
        case 'weiter':
            print("Die Schleife wird fortgesetzt.")
            continue
        case _:
            print(f"Du hast '{eingabe}' eingegeben. Das ist weder 'exit' noch 'weiter'.")
```

Anwendungsbeispiel 3.1: Datenbank

Es folgen vereinfachte Beispiele für den Einsatz von Schleifen mit einer Datenbank, hier sqlite3:

- Datenbank mit Name und Alter
- Einträge werden ausgelesen
- Zeilenweise Formatierung in der Schleife

```
import sqlite3

connection = sqlite3.connect('meine_datenbank.db')
cursor = connection.cursor()

cursor.execute("SELECT name, alter FROM benutzer")
rows = cursor.fetchall()

for row in rows:
    print(f"Name: {row[0]}, Alter: {row[1]}")

connection.close()
```

Anwendungsbeispiel 3.2: Datenbank

Einfügen neuer Werte in die Datenbank:

- 3 neue Datensätze aus Namen und Alter werden durch die Schleife einzeln in Datenbank übergeben

```
import sqlite3

daten_zum_einfügen = [('Alice', 28), ('Bob', 22), ('Charlie', 30)]
connection = sqlite3.connect('meine_datenbank.db')
cursor = connection.cursor()

for datensatz in daten_zum_einfügen:
    cursor.execute("INSERT INTO benutzer (name, alter) VALUES (?, ?)", datensatz)

connection.commit()
connection.close()
```

Anwendungsbeispiel 3.3: Datenbank

Aktualisieren von Datensätzen ist Anwendungsmöglichkeit für Schleifen:

- Alter der Einträge für Alice und Bob wird aktualisiert
- die anderen Datensätze bleiben unberührt

```
import sqlite3

zu_aktualisierende_daten = [(30, 'Alice'), (22, 'Bob')]
connection = sqlite3.connect('meine_datenbank.db')
cursor = connection.cursor()

for alter, name in zu_aktualisierende_daten:
    cursor.execute("UPDATE benutzer SET alter = ? WHERE name = ?", (alter, name))

connection.commit()
connection.close()
```

CloudCommand

CloudCommand GmbH chr.schumacher@gmx.tm