LU05c - Iteration

Eine Iteration führt einen Programmteil nie, einmal oder mehrmals aus. Solange die Bedingung erfüllt ist, wird der Programmteil innerhalb der Iteration ausgeführt.

Einführung

Die Verwendung von Schleifen erleichtert dir die Umsetzung von sich wiederholenden Programmteilen. Anstatt den (fast) gleichen Code immer wieder zu schreiben, solltest du eine Iteration einsetzen. Schauen wir uns ein Beispiel an, welches die Summe von 5 Zahlen bildet.

```
summ = 0

print('Input a number: ')
summ = summ + int(input())

print('Input a number: ')
summ = summ + int(input())

print('Input a number: ')
summ = summ + int(input())

print('Input a number: ')
summ = summ + int(input())

print('Input a number: ')
summ = summ + int(input())

print('The sum of the numbers is ' +
str(summ))
```

Das Programm erledigt die Aufgabe, aber nicht auf elegante Weise. Was wäre, wenn das Programm hundert oder vielleicht tausend Zahlen lesen und deren Summe ausgeben müsste? Was wäre, wenn das Programm nur drei Zahlen lesen müsste?

Das Problem lässt sich mit einer Schleife lösen, die sowohl die Summe als auch die Anzahl der gelesenen Eingaben festhält. Das Programm, das die Summe von fünf Zahlen ausgibt, sieht nun wie folgt aus:

```
numbers_read = 0
summ = 0

while (numbers_read < 5):
    summ = summ + int(input('Input number'))
```

```
    numbers_read = numbers_read + 1

print('The sum of the numbers is ' +
str(summ))
```

Schleifen und Endlosschleifen

Eine Schleife besteht aus einer Bedingung, die festlegt, ob der Code innerhalb der Schleife wiederholt werden soll oder nicht, sowie aus einem Block, der den zu wiederholenden Quellcode enthält. Eine Schleife hat die folgende Form.

```
while (_statement_):
    # The content of the block
    # The block can have an unlimited amount
of content
```

Wir verwenden zunächst den Wert `True` als Anweisung der Schleife. Auf diese Weise wird die Ausführung der Schleife endlos wiederholt. Dies ist der Fall, wenn die Programmausführung zum ersten Mal bei der Schleifenanweisung ankommt, und auch, wenn sie das Ende des Schleifenblocks erreicht.

Die Ausführung der Schleife erfolgt zeilenweise. Die folgenden Programmausgaben kann ich unendlich oft programmieren.

```
while True:
    print('I can program!')
```

```
I can program!
I can program!
I can program!
I can program!
I can program!
...
```

Ein Programm, das unendlich lange läuft, beendet sich nicht von selbst. In Python kann es normalerweise mit dem Befehl `Strg-C` beendet werden.

While-Schleife mit einer Bedingung

Oben haben wir eine Schleife mit dem booleschen Wert `True` in der Bedingung verwendet, was bedeutet, dass die Schleife für immer weiterläuft (oder bis die Schleife mit dem Befehl `break` beendet wird).

Die Bedingung einer Schleife muss einen Ausdruck enthalten, genau wie die Bedingung einer `if`-

Anweisung. Der Wert `True` kann durch einen Ausdruck ersetzt werden, der bei der Ausführung des Programms ausgewertet wird. Der Ausdruck wird auf die gleiche Weise definiert wie die Bedingung einer Selektion.

Der folgende Code gibt die Zahlen 1,2,...,5 aus. Wenn der Wert der Variablen number gröser als 5 ist, wird die `while`-Bedingung als `false` ausgewertet und die Ausführung der Schleife endgültig beendet.

```
number = 1

while (number < 6):
    print(number)
    number+= 1
print('Ready!')
```

```
1
2
3
4
5
Ready!
```

Der obige Code kann wie folgt gelesen werden: „Solange der Wert der variablen Zahl kleiner als 6 ist, drucke den Wert der variablen Zahl und erhöhe den Wert der variablen Zahl um eins“.

Oben wird der Wert der Variablen `number` bei jeder Ausführung des Schleifenkörpers um eins erhöht.

Beenden einer Schleife

Die Schleifenanweisung kann mit dem Befehl `break` abgebrochen werden. Wenn ein Computer den Befehl `break` ausführt, geht die Programmausführung zum nächsten Befehl über, der auf den Schleifenblock folgt.

Das folgende Beispiel ist ein Programm, das Zahlen von eins bis fünf ausgibt. Beachte die Variable `number`, welche vor der Schleife definiert wird. Auf diese Weise kann die Variable innerhalb der Schleife inkrementiert werden und die Änderung bleibt zwischen mehreren Iterationen der Schleife erhalten.

```
number = 1

while True:
    print(number)
    if (number >= 5):
        break

    number = number + 1
```

```
print('Ready!')
```

```
1
2
3
4
5
Ready!
```

Der Ausstieg aus der Schleife erfolgt, wenn der Benutzer eine bestimmte Eingabe vornimmt oder wenn eine in der Schleife durchgeführte Berechnung zum gewünschten Ergebnis führt. Diese Art von Programmen enthält sowohl eine Schleife, die zur Definition eines zu wiederholenden Abschnitts dient, als auch eine bedingte Anweisung, mit der geprüft wird, ob die Bedingung zum Verlassen der Schleife erfüllt ist oder nicht.

Innerhalb einer Schleife können auch Benutzereingaben abgefragt werden. Die Variablen, die üblicherweise in Schleifen verwendet werden, werden vor der Schleife definiert, während Variablen (wie der vom Benutzer gelesene Wert), die für die Schleife spezifisch sind, innerhalb der Schleife definiert werden.

Im folgenden Beispiel fragt das Programm den Benutzer, ob er die Schleife verlassen will oder nicht. Wenn der Benutzer die Zeichenkette „y“ eingibt, geht die Ausführung des Programms zu dem auf den Schleifenblock folgenden Befehl über, woraufhin die Ausführung des Programms endet.

```
while True:
    print('Exit? (y exits)')
    message = input()
    if (message == 'y'):
        break

    print('Ok! Let's carry on!')

print('Ready!')
```

```
Exit? (y exits)
User: <no>
Ok! Let's carry on!
Exit? (y exits)
User: <non>
Ok! Let's carry on!
Exit? (y exits)
User: <y>
Ready!
```

Im vorherigen Beispiel hat das Programm Eingaben vom Typ String vom Benutzer gelesen. Das Programm kann auch mit anderen Typen von Variablen implementiert werden. Das folgende Programm fragt Zahlen vom Benutzer ab, bis dieser eine `0` eingibt.

```
while True:
    number = int(input('Input a number, 0 to
quit'))
    if (number == 0):
        break

    print('You input ' + str(number))

print('Done, thank you!')
```

```
Input a number, 0 to quit
User: <5>
You input 5
Input a number, 0 to quit
User: <-2>
You input -2
Input a number, 0 to quit
User: <y>
Done, thank you!
```

Rückkehr zum Anfang der Schleife

Wenn die Ausführung das Ende der Schleife erreicht, beginnt die Ausführung wieder am Anfang der Schleife. Dies bedeutet, dass alle Befehle in der Schleife ausgeführt wurden. Sie können auch von anderen Stellen als dem Ende mit dem Befehl `continue` zum Anfang zurückkehren. Wenn der Computer den Befehl `continue` ausführt, springt die Ausführung des Programms an den Anfang der Schleife.

Das folgende Beispiel demonstriert die Verwendung des `continue`-Befehls. Das Programm fordert den Benutzer auf, positive Zahlen einzugeben. Wenn der Benutzer eine negative Zahl oder eine Null eingibt, gibt das Programm die Meldung „Ungültige Zahl! Versuchen Sie es erneut“, woraufhin die Ausführung zum Anfang der Schleife zurückkehrt. Im vorherigen Beispiel hat das Programm Eingaben vom Typ String vom Benutzer gelesen. Ähnliche Programme mit anderen Eingabetypen sind ebenfalls möglich. Im folgenden Beispiel wird der Benutzer nach Zahlen gefragt, bis er eine `0` eingibt.

```
while True:
    number = int(input('Insert positive
integers'))

    if (number <= 0):
        print('Invalid number! Try again.')
        continue

    print('Your input was ' + str(number))
```

Das Programm im obigen Beispiel wird unendlich oft wiederholt, da der `break`-Befehl zum Verlassen der Schleife nicht verwendet wird. Um die Schleife zu verlassen, muss der break-Befehl hinzugefügt werden.

Im folgenden Beispiel wird das Programm so geändert, dass der Benutzer aufgefordert wird, positive Zahlen einzugeben. Wenn der Benutzer eine negative Zahl eingibt, teilt ihm das Programm mit, dass die Zahl ungültig war und kehrt zum Anfang der Schleife zurück. Wenn die Zahl `0` war, verlässt das Programm die Schleife.

```
while True:
    number = int(input('Insert positive
integers'))

    if (number == 0):
        break

    if (number <= 0):
        print('Invalid number! Try again.')
        continue

    print('Your input was ' + str(number))
```

Rechnen mit Schleifen

Schleifen werden bei der Berechnung vieler verschiedener Dinge verwendet. Zum Beispiel verwenden Programme, die eine unbestimmte Anzahl von vom Benutzer eingegebenen Werten verarbeiten, Schleifen. Diese Art von Programmen gibt in der Regel nach dem Ende der Schleife eine Art Statistik über die gelesenen Zahlen oder andere Eingaben aus.

Damit das Programm Informationen aus der Schleifenausführung nach der Schleife ausdrucken kann, müssen die Informationen während der Schleife gespeichert und geändert werden.

Wenn die Variable, die zum Speichern der Daten verwendet wird, innerhalb der Schleife eingeführt wird, ist die Variable nur innerhalb dieser Schleife verfügbar und nirgendwo sonst.

Wir wollen ein Programm erstellen, das die Anzahl der vom Benutzer eingegebenen Einsen zählt und ausgibt. Erstellen wir zunächst eine nicht funktionierende Version und untersuchen wir die Wirkung der Blöcke.

```
# The task is to read an input from the user
while True:

    # The task is to keep count of number
ones
    ones = 0

    # The task is to ask the user for an
input and read a number from the user
    number = int(input('Input a number (0
exits)'))

    # The task is to exit the loop if the
```

```
user
    # has inputted zero
    if (number == 0):
        break

    # The task is to increase the amount of
ones
    # if the user inputs a number one
    if (number == 1):
        ones = ones + 1 #can also achieve the
same thing by ones += 1

# The task is to print out the total of ones
# This doesn't work because the variable ones
has been
# introduced within the loop
print('The total of ones: ' + str(ones))
```

Das vorherige Programm funktioniert nicht, weil die Variable `ones` innerhalb der Schleife eingeführt wird und versucht wird, sie nach der Schleife am Ende des Programms zu verwenden. Die Variable existiert nur innerhalb der Schleife. Wenn die Druckanweisung `print('The total of ones: ' + ones)` innerhalb der Schleife wäre, würde das Programm funktionieren, aber nicht in der gewünschten Weise. Schauen wir uns das als nächstes an.

```
# The task is to read an input from the user
while True:

    # The task is to keep count of number
ones
    ones = 0

    # The task is to ask the user for an
input and read a number from the user
    number = int(input('Input a number (0
exits)'))

    # The task is to exit the loop if the
user
    # has inputted zero
    if (number == 0):
        break

    # The task is to increase the amount of
ones
    # if the user inputs a number one
    if (number == 1):
        ones = ones + 1

    # The task is to print out the total of
ones
```

```
    print('The total of ones: ' + str(ones))
```

Das Beispiel funktioniert zwar, aber nicht wie gehofft. Wenn wir einige Eingaben ausprobieren, erhalten wir dies:

```
Insert a number (0 exits)
User: <5>
The total of ones: 0
Insert a number (0 exits)
User: <1>
The total of ones: 1
Insert a number (0 exits)
User: <1>
The total of ones: 1
Insert a number (0 exits)
User: <2>
The total of ones: 0
Insert a number (0 exits)
User: <0>
```

Der Grund dafür liegt in der Variable `ones`, welche wir innerhalb der Schleife definiert haben. Dadurch wird die Variable bei jedem Durchlauf der Schleife neu erstellt. Wenn eine Variable ihren Wert über mehrere Durchläufe behalten soll, musst du sie **vor** der Iteration definieren.

In dem folgenden Beispiel berechnet das Programm die Gesamtzahl der eingegebenen Einsen. Die Eingaben werden gelesen, bis der Benutzer eine `0` eingibt, woraufhin das Programm die Gesamtzahl der eingegebenen Einsen ausgibt. Das Programm verwendet die Variable `ones`, um die Anzahl der Einsen zu erfassen.

```
# The task is to keep track of number ones
ones = 0

# The task is to read an input from the user
while True:
    # The task is to ask the user for an
input and read a number from the user
    number = int(input('Input a number (0
exits)'))

    # The task is to exit the loop if the
user
    # has inputted zero
    if (number == 0):
        break

    # The task is to increase the amount of
ones
    # if the user inputs a number one
    if (number == 1):
        ones = ones + 1
```

```
# The task is to print out the total of ones
print('The total of ones: ' + str(ones))
```

```
Insert a number (0 exits)
User: <1>
Insert a number (0 exits)
User: <2>
Insert a number (0 exits)
User: <1>
Insert a number (0 exits)
User: <-1>
Insert a number (0 exits)
User: <0>
Total of ones: 2
```

Manchmal müssen Sie mehrere Variablen verwenden. Das folgende Beispiel zeigt ein Programm, das Zahlen vom Benutzer liest, bis der Benutzer `0` schreibt. Dann gibt das Programm die Anzahl der positiven und negativen Zahlen sowie den Prozentsatz der positiven Zahlen von allen gegebenen Zahlen aus.

```
# For saving number of numbers
number_of_positives = 0
number_of_negatives = 0

# For repeatedly asking for numbers
while True:
    # The task is to ask the user for an
input and read a number from the user
    number_from_user = int(input('Input a
number (0 exits)'))

    # For breaking the loop when user writes
0
    if (number_from_user == 0):
        break

    # For increasing number_of_positives by
one
    # when user gives a positive number
    if (number_from_user > 0):
        number_of_positives =
number_of_positives + 1

    # For increasing number_of_negatives by
one
    # when user gives a negative number
    if (number_from_user < 0):
        number_of_negatives =
```

```
number_of_negatives + 1

    # Also could have used:
    # if (number_from_user > 0):
    #     number_of_positives =
number_of_positives + 1
    # else:
    #     number_of_negatives =
number_of_negatives + 1
    #

# For printing the number of positive numbers
print('Positive numbers: ' +
str(number_of_positives))
# For printing the number of negative numbers
print('Negative numbers: ' +
str(numberOfNegative))

# For printing the percentage of positive
numbers from all numbers
if (number_of_positives + number_of_negatives
> 0):
    # Print only if user has given numbers
    # to avoid dividing by zero
    percentageOfPositives = 100.0 *
number_of_positives / (number_of_positives +
number_of_negatives)
    print('Percentage of positive numbers: '
+ str(percentageOfPositives) + '%')
```

M319-LU05, M319-E1G, M319-E1F

Kevin Maurizi, Marcel Suter

Diese Theorieseite ist eine übersetzte und Theorieseite Aufgabe von Scott Morgan, verwendet unter CC BY NC SA.

From:
https://wiki.bzz.ch/ - **BZZ - Modulwiki**

Permanent link:
https://wiki.bzz.ch/de/modul/m319/learningunits/lu05/iteration

Last update: **2025/06/23 07:45**