Schleifen

Bisher war unser Programmablauf einfach von oben nach unten. Mit Schleifen können wir einen Code-Block mehrmals direkt hintereinander ausführen. Es gibt zwei Sorten Schleifen: `for` und `while`.

while-Schleifen

While-Schleifen arbeiten ähnlich wie if-Statements mit einer Bedingung. Solange diese Bedingung `True` ist, wird die Schleife wiederholt. Beispiel:

In [13]:

```
1 i = 0
2 while i < 10:
3     print(i)
4     i += 1  # identisch mit i = i + 1
5 print("End")
6 i = 0
```

```
0
1
2
3
4
5
6
7
8
9
End
```

Wie wir sehen, ist die Syntax wie bei if/else-Abfragen: die Einleitung (`while ...`) wird mit einem Doppelpunkt beendet und der Code-Block in der Schleife muss eingerückt werden. Diese Schleife prüft vor jedem Durchlauf, ob `i` kleiner als 10 ist. Nur wenn das stimmt, wird der Code-Block in der Schleife ausgeführt. Im ersten Durchgang ist `i` gleich 0, die Bedingung stimmt also. Deshalb wird `0` ausgegeben (`print(i)`) und dann wird `i` hochgezählt. Am Ende der Schleife springt Python wieder zur Bedingung und prüft diese erneut: jetzt ist `i` gleich 1 und damit immer noch kleiner als 10. Die Schleife wird also erneut durchlaufen.

Fast forward: wenn `i` gleich 9 ist, stimmt die Bedingung immer noch, allerdings wird `i` in dem Durchlauf dann auf 10 gesetzt. Das heißt nach diesem Durchlauf wird die Bedingung erneut geprüft, sie stimmt jetzt aber nicht mehr, sie hat also den Wert `False`. Deshalb wird nun die Schleife übersprungen und der Code wird nach der Schleife fortgesetzt. Im obigen Beispiel wird also `End` ausgegeben. Danach wird `i` wieder auf 0 gesetzt, das hat aber keine Auswirkungen mehr auf die Schleife, weil Python jetzt wieder von oben nach unten arbeitet. Zeile 6 hat also keinen Einfluss mehr auf Zeile 2.

for-Schleifen

For-Schleifen arbeiten nicht mit einer Bedingung, sondern mit einer Variable, die mehrere Elemente enthält. Eine Möglichkeit dafür ist ein `range()` -Objekt. Beispiel:

In [8]:

```
for i in range(10):
    print(i)
print("End")
```

```
0
1
2
3
4
5
6
7
8
9
End
```

Auch hier ist die Syntax wieder mit dem Doppelpunkt und der Einrückung. Wie wir sehen ist der Output identisch mit dem der while-Schleife, aber mit weniger Code. Der Aufruf `range(10)` erzeugt eine Zahlensequenz von 0 bis 9. Der Variable `i` wird dabei im ersten Durchlauf der erste Wert der Sequenz zugewiesen, also 0. Im zweiten Durchlauf wird `i` der zweite Wert der Sequenz zugewiesen, also 1. Dies wird so oft wiederholt, bis der letzte Wert der Sequenz erreicht wurde. Danach macht Python mit dem Code nach der Schleife weiter, in diesem Fall also `print("Ende")` .

range()

Das `range()` -Objekt hat drei Parameter: `range(start, stop, step)`

`start` ist dabei der erste Wert in der Zahlensequenz, `stop` ist der erste Wert, der **nicht mehr** in der Zahlensequenz ist, und `step` ist die Schrittweite.

Wir haben oben `range()` nur mit einem Argument aufgerufen. In dem Fall ist die Zahl der `stop` -Wert, `start` ist standardmäßig 0 und `step` ist standardmäßig 1.

Wenn `range()` mit zwei Zahlen verwendet wird, sind das `start` und `stop` . `step` ist in dem Fall standardmäßg 1.

In [31]:

```
for i in range(5):  # von 0 bis 4
    print(i)
```

```
0
1
2
3
4
```

In [32]:

```
for i in range(12, 16):  # von 12 bis 15
    print(i)
```

```
12
13
14
15
```

In [14]:

```
for i in range(12, 24, 3):  # von 12 bis 23 in Dreier-Schritten
    print(i)
```

```
15
18
21
24
```

`range()` kann nur mit Integers arbeiten. Kommazahlen sind nicht möglich.

In [34]:

```
for i in range(2.3, 5.5, 0.5):
    print(i)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-34-33114e24fa0e> in <module>
----> 1 for i in range(2.3, 5.5, 0.5):
      2     print(i)

TypeError: 'float' object cannot be interpreted as an integer
```

Strings

For-Schleifen können auch über Strings laufen. Ein String ist ja nichts anderes als eine Sequenz von Buchstaben und anderen Zeichen. Entsprechend kann auch ein String für eine for-Schleife verwendet werden:

In [51]:

```
1 s = "Hello World!"
2
3 for c in s:
4     print(c)
```

```
H
e
l
l
o

W
o
r
l
d
!
```

Endlosschleifen, `break` und `continue`

Ein Problem, was bei Schleifen auftreten kann, ist dass man (versehentlich) Endlosschleifen programmiert. Das sind in der Regel while-Schleifen, deren Bedingung **nie** `False` wird. Python bleibt also in der Schleife gefangen und der Code kommt nie am Ende an. Beispiel:

In [2]:

```
1 i = 0
2 while i < 10:
3     print(i)
4     i -= 1
```

```
0
-1
-2
-3
-4
-5
-6
-7
-8
-9
-10
-11
-12
-13
-14
-15
-16
-17
-18
```

Dieses Beispiel sieht so ähnlich aus wie das von oben, allerdings wird in Zeile 4 in der Schleife `i` runter gezählt. Das heißt, dass `i` immer kleiner ist als 10, die Bedingung ist also immer erfüllt. Eine solche Schleife kann nur durch einen *Interrupt* beendet werden. In Jupyter Notebook ist das über die Stopp-Taste am oberen

Rand möglich oder indem `i` doppelt gedrückt wird, in einer Standard Python-Console kann mit `Ctrl+C` ein Keyboard-Interrupt erzwungen werden. Falls so etwas in komplexeren Programmen passiert, kann es nötig sein, diese über den Windows-Task-Manager zu beenden.

break

Mit `break` kann eine Schleife vorzeitig beendet werden. Die `break` -Anweisung beendet dabei den aktuellen Durchlauf und springt zum Code nach der Schleife. Beispiel:

In [11]:

```
while True:
    print("Hello")
    break
    print("World")
```

```
Hello
```

`World` wird nicht ausgegeben, weil die `break` -Anweisung die Schleife bereits vorher beendet.

In [53]:

```
pw = "silverfish12"

while True:
    a = input("Please enter the password: ")
    if a == pw:
        break
    print("The password is incorrect!")

print("The password is correct!")
```

```
Please enter the password: fish
The password is incorrect!
Please enter the password: silverfish
The password is incorrect!
Please enter the password: silverfish12
The password is correct!
```

Die Anweisung `while True:` erzeugt prinzipiell eine Endlosschleife (`True` kann nie `False` werden), diese Endlosschleife wird aber unter einer bestimmten Bedingung mit `break` verlassen, nämlich wenn das Passwort korrekt eingegeben wurde. Wenn das Passwort nicht korrekt eingegeben wurde, wird die Passwortabfrage erneut durchgeführt. Diese Abfrage kann auch anders durchgeführt werden: Entweder mit dem Passwort als Bedingung...

In [54]:

```
pw = "silverfish12"
a = ''

while a != pw:
    a = input("Please enter the password: ")

print("The password is correct!")
```

```
Please enter the password: fish
Please enter the password: silverfish
Please enter the password: silverfish12
The password is correct!
```

...oder mit einer Hilfsvariable.

In [55]:

```
pw = "silverfish12"
pw_korrekt = False

while not pw_korrekt:
    a = input("Please enter the password: ")
    if a == pw:
        pw_korrekt = True
        print("The password is correct!")
    else:
        print("The password is incorrect!")
```

```
Please enter the password: fish
The password is incorrect!
Please enter the password: silverfish
The password is incorrect!
Please enter the password: silverfish12
The password is correct!
```

Das Konstrukt mit `while True: ... break` wird von manchen Programmierern als schlechter Stil angesehen, es funktioniert aber wunderbar und ist oft die einfachste bzw. kürzeste Schreibweise für die gewünschte Funktionalität.

continue

Mit `continue` kann auch der aktuelle Durchlauf einer Schleife beendet werden, danach springt Python allerdings wieder zum Beginn der Schleife. Beispiel: Passwortabfrage, bei der zwei Passwörter in Folge korrekt eingegeben werden müssen:

In [57]:

```
pw1 = "silver1"
pw2 = "fish2"

while True:
    a = input("Please enter the FIRST password: ")
    if a != pw1:
        print("Incorrect!")
        continue
    b = input("Please enter the SECOND password: ")
    if b == pw2:
        break
    print("Incorrect!")

print("Acces granted!")
```

```
Please enter the FIRST password: fish
Incorrect!
Please enter the FIRST password: silver1
Please enter the SECOND password: fish
Incorrect!
Please enter the FIRST password: silver1
Please enter the SECOND password: fish2
Acces granted!
```

Wenn das erste Passwort falsch eingegeben wird, springt Python durch die `continue` -Anweisung direkt wieder an den Anfang der Schleife, ohne nach dem zweiten Passwort zu fragen. Nur wenn das erste Passwort korrekt ist, wird nach dem zweiten Passwort gefragt.