Booleans und Funktionen

Richard Müller, Tom Felber

4. November 2021

Python-Kurs

Gliederung

1. Wiederholung
2. Booleans
3. Funktionen

Wiederholung

Wiederholung

Beim letzten Mal

- Zahlen
- Listen

```
liste = [1337, "Hallo Welt", True]
```

- for-Schleife

```
for i in liste:
    print(i)
```

- while-Schleife

```
j = 0
while j < len(liste):
    print(liste[j])
    j += 1 # wie j = j + 1
```

Slices

Um mehrere Elemente einer Liste auszuwählen, kann folgende Notation verwendet werden: `[Start:Ende]`. Dabei können jeweils Start und Ende optional weggelassen werden. Bei jedem solchen Zugriff wird eine neue Liste erzeugt.
Beispiele:

liste = [0,1,2,3,4,5,6]	Wert
`liste[2:4]`	[2,3]
`liste[3:]`	[3,4,5,6]
`liste[:3]`	[0,1,2]
`liste[:]`	[0,1,2,3,4,5,6]

Booleans

Booleans

Booleans können die Werte `True` oder `False` annehmen.
Die Keywords `and`, `or` können benutzt werden, um Booleans logisch zu verbinden.
Mit `not` kann ein Boolean invertiert werden:

mit Operator	Wert
`True and False`	False
`True and True`	True
`True or False`	True
`False or False`	False
`not True`	False
`not False`	True
`not not True`	True

Booleans

Kombinationen der Operatoren sind auch möglich. Dabei können Klammern gesetzt werden, um die Reihenfolge der Auswertung zu bestimmen. Wenn keine gesetzt werden, wird `and` vor `or` ausgewertet. `not` wird zuallererst ausgewertet.

mit Operator	Wert
`False and True or True`	True
`False and (True or True)`	False
`(not not not False and True and not True) or (True and True)`	?

Funktionen

Funktionen - Einleitung

$$f(x) = 3x^5 + 8x^4 + 42x^3 + x$$

Funktionen - Einleitung

Funktionen sind immer dann gut geeignet, wenn man den selben Code an mehreren Stellen ausführen muss.

Zum Beispiel ist es einfacher, `print()` zu benutzen, als jedes Mal den kompletten Code dieser Funktion in sein eigenes Programm zu kopieren.

Funktionen - Aufbau

Die einfachst mögliche Funktion könnte so aussehen:

```
1 def my_funtion():
2     print("Dies ist eine Funktion")
```

Jede Funktion besteht aus folgenden Teilen:

- dem Schlüsselwort `def`
- dem Name der Funktion - grundsätzlich beliebig, aber man sollte die Konvention beachten (alles klein, mehrere Worte mit Unterstrich trennen)
- zwei Runde Klammern, in denen Argumente definiert werden können
- dem Code, der ausgeführt werden soll, eingerückt unter dem Funktionskopf. **Man kann in einer Funktion alles machen, was man auch außerhalb kann.**

Funktionen - Aufruf

Der Code innerhalb einer Funktion wird nicht sofort ausgeführt, wenn die Funktion definiert wird. Stattdessen muss man sie explizit aufrufen:

```
# irgendwo im Programmablauf
my_function()
```

Funktionen - Argumente

In den Klammern können Argumente festgelegt werden, die dann beim Aufruf mit Werten belegt werden müssen:

```
1 # beliebig viele Argumente mit Komma getrennt in die
      Klammern
2 def potenz(x, y):
3     print(x**y)
4
5 # Argumente werden in der Reihenfolge belegt, wie sie
      angegeben wurden
6 potenz(2, 3) # x = 2, y = 3
```

Funktionen - Return

Möchte man einen Wert zurückgeben, ihn also auch außerhalb der Funktion noch verwenden, kann man das mit dem `return`-Statement tun:

```
1 def potenz(x, y):
2     # x^y wird zurückgegeben
3     return x**y
4
5 p = potenz(2, 3) # p = 8
```

Das `return`-Statement bricht die Funktion ab. Alles was danach noch käme, wird nicht mehr ausgeführt.

Funktionen - Default Argumente

Man kann Argumenten Default-Werte mitgeben. Somit wird dieses Argument optional und muss nicht zwingend während des Aufrufs belegt werden. Gibt man ihm trotzdem einen Wert, so wird der Default-Wert überschrieben:

```
1 def potenz(x, y=2):
2     return x**y
3
4 p = potenz(2) # p = 4
5 p = potenz(2, 3) # p = 8
```

Default Argumente müssen in den Klammern der Funktionsdefinition immer als letztes angegeben werden.