Merkblatt - Funktionen und Methoden

May 7, 2020

1 Merkblatt: Funktionen & Methoden

1.1 Funktionen

Funktionen sind zusammengefasste Codeblöcke. Mittels Funktionen können wir es vermeiden, mehrmals verwendete Codeblöcke zu wiederholen. Wir definieren stattdessen einmal eine Funktion, die diese Codeblöcke enthält und brauchen an weiteren Stellen nur noch (kurz) die Funktion aufzurufen, ohne die in ihr enthaltenen Codezeilen zu kopieren.

1.1.1 Eine Funktion definieren und aufrufen

Wir haben schon einige Funktionen kennengelernt, die uns Python zur Verfügung stellt. Die Funktion, die wir bislang wohl am häufigsten verwendet haben, ist die print-Funktion:

[1]:
```
print("HALLO WELT")
```

```
HALLO WELT
```

Wenn wir eine eigene Funktion verwenden wollen, müssen wir sie zuerst definieren. Eine solche Funktionsdefinition hat die allgemeine Syntax:

def Funktionname(): Code

[4]:
```
def multi_print():
    print("Hallo Welt!")
    print("Hallo Welt!")
```

Um eine Funktion auszuführen, die definiert wurde, schreiben wir: **Funktionname()**

[5]:
```
multi_print()
```

```
Hallo Welt!
Hallo Welt!
```

1.1.2 Funktionen mit einem Argument

Man kann Funktionen ein **Argument** übergeben, d.h. einen Wert, von dem der Code innerhalb der Funktion abhängt.

def Funktionsname(Argument): Code in dem mit dem spezifischen Argument gearbeitet wird

[6]:
```
def multi_print2(name):
    print(name)
    print(name)

multi_print2("HALLO")
multi_print2("WELT")
```

```
HALLO
HALLO
WELT
WELT
```

Du kannst dir einen solchen Parameter als eine zu einer Funktion gehörige Variable vorstellen. Vermeide es, einen Funktionsparameter wie eine bereits bestehende Variable zu benennen - Verwirrungsgefahr!

[7]:
```
name = "MARS"

def multi_print2(name):
    print(name)
    print(name)

multi_print2("HALLO")
multi_print2("WELT")

print(name)
```

```
HALLO
HALLO
WELT
WELT
MARS
```

Du siehst, dass der Wert der Variable *name* keinen Einfluss auf das Argument *name* der Funktion hat!

1.1.3 Weitere Funktionen in Python

Auch die len-Funktion für Listen kennst du schon. :-)

[1]:
```
print(len(["Hallo", "Welt"]))
```

```
2
```

Du kannst die len-Funktion auch auf Strings anwenden.

[2]:
```
print(len("Hallo"))
```

```
5
```

Eine Übersicht über Funktionen in Python findest du hier: https://docs.python.org/3/library/functions.html

1.1.4 Funktionen mit mehreren Argumenten

Eine Funktion darf auch mehrere Argumente enthalten.

def Funktionenname(Argument1, Argument2, ...): Code in dem mit Argument1, Argument2,... gearbeitet wird

[1]:
```
def multi_print(name, count):
    for i in range(0, count):
        print(name)

multi_print("Hallo!", 5)
```

```
Hallo!
Hallo!
Hallo!
Hallo!
Hallo!
```

1.1.5 Funktionen in Funktionen

Funktionen können auch ineinander geschachtelt werden:

[5]:
```
def weitere_funktion():
    multi_print("Hallo!", 3)
    multi_print("Welt!", 3)
```

[6]:
```
weitere_funktion()
```

```
Hallo!
Hallo!
Hallo!
Welt!
Welt!
Welt!
```

1.1.6 Einen Wert zurückgeben

Bislang führen wir mit Funktionen einen Codeblock aus, der von Argumenten abhängen kann. Funktionen können aber auch mittels des Befehls **return** Werte zurückgeben:

[2]:
```
def return_element(name):
    return name

print(return_element("Hi"))
```

```
Hi
```

Solche Funktionen mit return können wir dann wie Variablen behandeln:

[10]:
```
def return_with_exclamation(name):
    return name + "!"

if return_with_exclamation("Hi") == "Hi!":
    print("Right!")
else:
    print("Wrong.")
```

```
Right!
```

[1]:
```
def maximum(a, b):
    if a < b:
        return b
    else:
        return a

result = maximum(4, 5)
print(result)
```

```
5
```

2 Funktionen vs. Methoden

2.0.1 Funktionen

Bei ihrem Aufruf stehen Funktionen "für sich" und das, worauf sie sich beziehen steht ggf. als Argument in den Klammern hinter ihnen:

[36]:
```
liste = [1, 2, 3]
```

[28]:
```
print(liste)
```

```
[1, 2, 3]
```

[29]:
```
print(len(liste))
```

```
3
```

2.0.2 Methoden

Daneben kennen wir aber auch schon Befehle, die mit einem Punkt an Objekte angehängt werden. Eine Liste ist ein solches **Objekt**. Jedes Objekt hat Methoden, auf die wir zurückgreifen können. Diese Methoden können wir aber nicht auf ein Objekt eines anderen Typs anwenden (meistens zumindest).

Schauen wir uns einige nützliche Methoden des Listen-Objektes an :-) (du brauchst sie dir nicht alle merken)

[37]:
```
# ein Element anhängen
liste.append(4)

print(liste)
```

```
[1, 2, 3, 4]
```

[39]:
```
# ein Element an einem bestimmten Index entfernen
liste.pop(2)
```

[39]: 2

[ ]:
```
# wir sehen, dass die Methode nicht die aktualisierte Liste, sondern das␣
↪entfernte Element liefert
```

[40]:
```
print(liste)
```

```
[1, 4, 3, 4]
```

[41]:
```
 # Ein Element an einer bestimmten Stelle einfügen
# das erste Argument bei insert gibt an, welches Element in die Liste eingefügt␣
↪wird,
# das zweite Argument bei insert gibt an, an welcher Stelle das Element␣
↪eingefügt wird;
# beachte, dass der Index des ersten Elements in einer Liste 0 ist!
liste.insert(1, 4)

print(liste)
```

[ ]:
```
# ein Element entfernen
liste.remove(4)

print(liste)
```

[21]:
```
# den Index eines Elementes angeben (die erste Stelle, an der es vorkommt)
print(liste.index(3))
```

```
4
```

[23]:
```
print(liste.index(4))
```

```
1
```

[24]:
```
print(liste.count(4))
```

```
3
```

[19]:
```
# mit reverse können wir die Reihenfolge einer Liste umkehren
liste.reverse()
print(liste)
```

```
[1, 4, 4, 4, 3]
```

[ ]:
```

```