Scripting 2

TSBE Frühlingssemester 2018

http://smlz.github.io/tsbe-2018fs/scri/

Marco Schmalz

marco.schmalz@gibb.ch

Kursübersicht

1. Tools, Zahlen, Strings und Entscheidungen
2. **Funktionen, Test-driven development, Listen und Schleifen**
3. dicts
4. Files und externe Kommandos
5. Praxistipps, externe Libraries und Repetition

Heute

1. Funktionen
2. Test-driven development
3. Person of the Day: Dave Beazley
4. Listen und Schleifen
5. Crazy Code: Rekursion

Funktionen

Eine Funktion mit dem Namen `sag_hallo` definieren:

```
def sag_hallo():
    print(f"Guten Tag!")
```

Achtung: Doppelpunkt und Einrücken zu Beginn der Funktion.

Die Funktion aufrufen:

```
sag_hallo()   # Gibt die Mitteilung 'Guten Tag!' auf der Konsole aus.
```

Eine Funktion wird durch das Anhängen von runden Klammern aufgerufen bzw. ausgeführt.

Funktionsparameter

Funktionsdefinition:

```
def sag_hallo(name):
    print(f"Guten Tag {name}!")
```

Funktionsaufruf:

```
sag_hallo("Kurt")  #  Gibt die Mitteilung 'Guten Tag Kurt!' auf der Konsole
aus.
```

Es können mehrere durch Komma abgetrennte Funktionsparameter definiert werden.

```
def sag_hallo(vorname, nachname):
    print(f"Guten Tag {vorname} {nachname}!")
```

Funktionen als Abfolge von Befehlen (Prozeduren)

Mit Funktionen können mehrere Befehle zusammen gefasst, und dann später aufgerufen werden.

Beispiel:

```
def begruessung():
    print("Guten Tag!")
    print("Was möchten sie gerne tun?")
    print("Daten eingeben: E")
    print("Daten abfragen: A")

def ende():
    print("Vielen Dank für die Verwendung unseres Programms.")
    print("(c) 2018, Wasserfall GmbH")
```

Diese Art des Programmierens wird oft als *prozedural* bezeichnet.

Funktionen mit Rückgabewerten

Mit der `return` Anweisung, kann ein Wert aus einer Funktion zurück gegeben werden.

```
def summe(a, b):
    return a + b

def kreisumfang(radius):
    return 3.14 * 2 * radius

def anrede(geschlecht, vorname, nachname):
    if geschlecht == "f":
        return f"Liebe {vorname} {nachname}"
    else:
        return f"Lieber {vorname} {nachname}"
```

Aufruf der Funktionen:

```
>>> summe(1, 2)
3
>>> anrede("f", "Marie", "Curie")
'Liebe Marie Curie'
```

Mathematische Funktion

Eine Funktion kann nicht nur als Abfolge von Befehlen betrachtet werden, sondern mathematische Funktion, welche aus einem oder mehreren Eingabewerten (Funktionsparameter) einen Ausgabewert berechnet.

Beispiele:

```
def durchschnitt(a, b):
    return (a + b) / 2

def betrag(a):
    if a < 0:
        return -a
    else:
        return a

def gerade_ungerade(a):
    if a % 2 == 0:
        return "gerade"
    else:
        return "ungerade"
```

Dieser Programmierstil wird als *funktional* Bezeichnet.

Funktionen testen

Mit der `assert`-Anweisung können Bedingungen überprüft werden, welche immer erfüllt sein müssen.

```
assert betrag(5) == 5
assert betrag(-4) == 4

assert durchschnitt(4.5, 5) == 4.75
```

Dies ist die einfachste Form um zu überprüfen, dass die geschriebenen Funktionen wirklich das tun, was sie sollen.

Funktionen bei denen wie bei mathematische Funktionen der Rückgabewert nur von den Funktionsparametern abhängt, lassen sich besonders effizient testen.

Wenn man die Tests (also die `assert`-Anweisungen) schreibt, bevor man die eingentliche Funktion programmiert, spricht man von *Test Driven Development*.

Exkurs 1: Funktionen als Werte

In Python können Funktionen genau so wie zum Beispiel Strings oder Zahlen herum gereicht werden.

```
def summe(lauf_a, lauf_b):
    return lauf_a + lauf_b

def minimum(lauf_a, lauf_b):
    if lauf_a < lauf_b:
        lauf_a
    else:
        lauf_b

def schluss_zeit(lauf_a, lauf_b, berechnungsfunktion):
    return berechnungsfunktion(lauf_a, lauf_b)

zeit_qualifying = schluss_zeit(lauf_a, lauf_b, minimum)

zeit_rennen = schlusszeit(lauf_a, lauf_b, summe)
```

Man sagt, Funktionen seinen in Python *first class citizens*, also Bürger erster Klasse. Wie andere Werte auch (Text-Strings, Zahlen, ...) können sie Funktionen übergeben, in Variablen abgespeichert werden, und so weiter.

Listen

Leere Liste erstellen:

```
leere_liste = []
```

Liste mit Inhalt erstellen:

```
ziffern = [0, 1, 2, 3, 4, 5, 6, 7, 8]
```

Listen können wachsen, in dem man ihnen am Ende etwas anhängt (engl. `append`):

```
ziffern.append(9)
print(ziffern)   # -> gibt [1, 2, 3, 4, 5, 6, 7, 8, 9] aus
```

Listen können gemischte Daten enthalten:

```
ziffern_komisch = ["eins", 2, "drei", 4, 5.0, "seven", "八", 9]
```

Länge von Listen (`len`)

Die Funktion `len` berechnet die Länge einer Liste:

```
anzahl_ziffern = len(ziffern)
print(f"Es gibt {anzahl_ziffern} verschiedene Ziffern")
```

Text-Strings haben auch eine Länge:

```
message = "Hallo Leute!"
print("Länge der Mitteilung:", len(message))
```

Text aufsplitten und wieder verbinden

Ein Text-String kann in eine Liste von Wörtern zerlegt (engl. `split`) werden:

```
>>> "Es irrt der Mensch\nsolang' er strebt".split()
['Es', 'irrt', 'der', 'Mensch', "solang'", 'er', 'strebt']
```

Standardmässig wird der Text an den Leerzeichen *und* Zeilenumbrüchen aufgetrennt.

Es kann auch explizit ein Trenn-String angegeben werden:

```
>>> "192.168.10.177".split(".")
['192', '168', '10', '177']
>>> "AF::46::4A::97::0E::01".split("::")
['AF', '46', '4A', '97', '0E', '01']
```

Eine Liste von Strings kann auch wieder zu einem einzigen String zusammen gefügt werden:

```
>>> werte = ["Mr.", "Joaquin", "Phoenix", "Actor"]
```

Auf Elemente einer Liste zugreifen

Mit eckigen Klammern und einer Ganzzahl als Index, kann auf die einzelnen Elemente einer Liste zugegriffen werden:

```
>>> liste = ['a', 'b', 'c', 'd', 'e']
>>> liste[0]
'a'
>>> liste[2]
'c'
```

Der verwendete Index kann auch in einer Variable definiert sein:

```
>>> index = 1
>>> liste[index]
'b'
```

Auf Elemente am Ende einer Liste zugreifen

Mit negativen Indizes wird vom Ende der Liste her gezählt:

```
>>> liste = ['a', 'b', 'c', 'd', 'e']
>>> liste[-1]
'e'
>>> liste[-2]
'd'
>>> liste[-5]
'a'
```

Praktisch ist vor allem der Zugriff auf das letzte Element einer Liste:

```
>>> liste[-1]
'e'
```

For-Schleife

Über Listen und listenähnliche Objekte kann iteriert werden:

```
liste = ['a', 'b', 'c']

for buchstabe in liste:     # über den Inhalt von Liste iterieren
    print(f"Der Buchstabe ist: {buchstabe}")
```

Ausgabe:

```
Der Buchstabe ist: a
Der Buchstabe ist: b
Der Buchstabe ist: c
```

Die Schleife wird für jedes einzelne Element der Liste einmal durchgegangen. Der Variabelnnamen für das Element des aktuellen Durchlaufs der Schleife ist frei wählbar. Sie heisst `buchstabe` im obigen Fall.

Durchnummerierte For-Schleife

Bei der `for`-Schleife gibt es standartmässig keine Indexvariabel. Mit `enumerate` kann aber eine Liste durchnummeriert werden:

```
liste = ['a', 'b', 'c']

for index, buchstabe in enumerate(liste):
    print(f"Index {index}: {buchstabe}")
```

Ausgabe:

```
Index 0: a
Index 1: b
Index 2: c
```

Bei `enumerate` kann zusätzlich der Startwert der Indexvariabel übergeben werden:

```
for index, buchstabe in enumerate(liste, 1):
    print(f"Buchstabe Nummer {index} ist {buchstabe}")
```

While-Schleife

Die `while`-Schlaufe wird ausgeführt bis die angegebene Bedingung nicht mehr erfüllt ist:

```
zahl = 16

while zahl >= 1:
    print(zahl)
    zahl = zahl / 2
```

Ausgabe:

```
16
8.0
4.0
2.0
1.0
```

Eine Endlosschleife:

```
while True:
    print("Ein Sprung in der Schallplatte!")
```

Vollständiges Beispiel: Zahl als Wort

Verwendung möglichst vieler gesehener Konstrukte:

- `for`-Loop
- `while`-Loop
- `list.append`
- `str.split`
- `str.join`

Ziel als Test mit `assert` formuliert:

```
assert zahlen_als_woerter(123) == 'eins-zwei-drei'
```

Weitere Hilfsmittel:

- Ganzzahldivision: `12 // 5` ⇨ `2`
- Liste umkehren: `reversed([1, 2, 3])` ⇨ `[3, 2, 1]`

Vollständiges Beispiel: Zahl als Wort

Code:

```
def ziffern_einer_zahl(zahl):
    ziffern = []
    while zahl > 0:
        rest = zahl % 10
        ziffern.append(rest)
        zahl = zahl // 10  # Ganzzahldivision
    return reversed(ziffern)

namen_von_ziffern = 'null eins zwei drei vier fünf sechs sieben acht
neun'.split()

def zahl_in_woertern(zahl):
    ziffern = ziffern_einer_zahl(zahl)
    woerter = []
    for ziffer in ziffern:
        woerter.append(namen_von_ziffern[ziffer])
    return '-'.join(woerter)
```

Anwendung:

```
>>> zahl_in_woertern(117)
'eins-eins-sieben'
```