

Einführung in die Programmierung mit Python

Lucerne University of
Applied Sciences and Arts

**HOCHSCHULE
LUZERN**

Syllabus

1. Python installieren; Jupyter Notebooks
2. Grundlagen: Arithmethik, Variablen und Datentypen
3. **Methoden; Listen; Bedingte Anweisungen**
4. Schleifen
5. Benutzerdefinierte Funktionen
6. Datenaufbereitung und Grafische Darstellungen

Methoden

Neben Funktionen (wie `len` und `print`) gibt es in Python auch *Methoden*. Sie funktionieren fast wie Funktionen, werden aber aufgerufen, indem der Methodenname an den Namen eines Objekts (hier eine Zeichenkette) angehängt wird:

```
In [1]: s1 = "Simon"
```

```
In [2]: s1.upper()
```

```
Out[2]: 'SIMON'
```

Objekte verschiedener Typen unterstützen unterschiedliche Operationen (Methoden). Hier ist eine Liste für Zeichenketten:

```
In [3]: print(', '.join(filter(lambda m: callable(getattr(s1, m)) and not m.startswith
```

```
    capitalize, casefold, center, count, encode, endswith, expandtabs, find, format, format_map, index, isalnum, isalpha, isascii, isdecimal, isdigit, isidentifier, islower, isnumeric, isprintable, isspace, istitle, isupper, join, ljust, lower, lstrip, maketrans, partition, removeprefix, removesuffix, replace, rfind, rindex, rjust, rpartition, rsplit, rstrip, split, splitlines, startswith, strip, swapcase, title, translate, upper, zfill
```

Der Schlüssel im obigen Befehl ist `dir(s1)` (probieren Sie es aus). Der Rest dient nur der schönen Ausgabe.

Eine weitere Möglichkeit, herauszufinden, welche Methoden von einem Objekt eines bestimmten Typs unterstützt werden, ist die Verwendung von *Autovervollständigung*:

```
In [4]: # Entkommentieren Sie die nächste Zeile  
# s1. # Versuchen Sie, die Tab-Taste nach dem Punkt zu drücken
```

Sie können herausfinden, was eine Methode tut, indem Sie die `help`-Funktion verwenden:

```
In [5]: help(s1.upper)
```

Help on built-in function upper:

upper() method of builtins.str instance

 Return a copy of the string converted to uppercase.

Übung

Die folgenden Methoden für Zeichenketten werden in den Hausaufgaben benötigt. Versuchen Sie herauszufinden, was sie tun, indem Sie eine Mischung aus `help` und Ausprobieren verwenden: `lower` , `upper` , `capitalize` , `startswith` , `endswith` , `index` und `find` .

In []:

Listen

Listen sind indizierbare Sammlungen von beliebigen (meist jedoch homogenen) Dingen:

```
In [6]: list1 = [1, 2., 'hi']; print(list1)  
[1, 2.0, 'hi']
```

Wie bei Zeichenketten gibt die Funktion `len` die Länge einer Liste (oder jeder anderen Sequenz) zurück:

```
In [7]: len(list1)  
Out[7]: 3
```

Wie Zeichenketten unterstützen sie die Indizierung, aber im Gegensatz zu Zeichenketten sind sie *veränderbar*, d.h., Elemente der Liste können geändert werden.

```
In [8]: list1 = [1, 2., 'hi']  
list1[2] = 42  
print(list1)  
[1, 2.0, 42]
```

Die Funktionen `sum`, `min` und `max` berechnen jeweils die Summe, das Minimum und das Maximum einer Liste, sofern dies in Bezug auf die Elemente der Liste sinnvoll ist:

```
In [9]: list1 = [1, 2., 42]
print(sum(list1))
print(min(list1))
print(max(list1))
```

```
45.0
1
42
```

Wie Zeichenketten unterstützen Listen eine Reihe von Methoden:

```
In [10]: print(', '.join(filter(lambda m: callable(getattr(list1, m)) and not m.startswith('__'), dir(list1))))
```

append, clear, copy, count, extend, index, insert, pop, remove, reverse, sort

Übung

Die Methoden `append`, `insert`, `index`, `remove`, `reverse` und `count` werden für die Hausaufgaben benötigt. Finden Sie heraus, was sie tun.

```
In [ ]:
```

Tupel

- Ein Tupel ist ähnlich wie eine Liste, aber *unveränderbar*. Es wird mit runden Klammern erstellt:

```
In [11]: (1, 2., 'hi')
```

```
Out[11]: (1, 2.0, 'hi')
```

Übung

1. Erstellen Sie eine Liste mit den Namen "Simon", "Carl" und "Lucy" als Zeichenketten und speichern Sie sie in einer Variable.
2. Ändern Sie das zweite Element der Liste in "Karl".
3. Wiederholen Sie dies, aber verwenden Sie diesmal ein Tupel anstelle einer Liste.
Warum schlägt dies fehl?

In []:

Kontrollfluss

Bedingungen

Bedingte Anweisungen werden verwendet, wenn Code nur ausgeführt werden soll, wenn eine bestimmte Bedingung erfüllt ist. Beispiel:

```
In [12]: x = int(input("Geben Sie eine positive Zahl ein: "))
if x < 0:
    print("Sie haben eine negative Zahl eingegeben.")
```

Hinweise:

1. Codeblöcke werden durch Doppelpunkte eingeleitet und müssen eingerückt werden.
2. Der `if`-Block wird nur ausgeführt, wenn die Bedingung `True` ist.

Was, wenn wir etwas tun wollen, falls die Bedingung `False` ist? `else` hilft weiter.

```
In [13]: x = int(input("Geben Sie eine positive Zahl ein: "))
if x < 0:
    print("Sie haben eine negative Zahl eingegeben.")
else:
    print("Danke.")
```

```
Sie haben eine negative Zahl eingegeben.
```

Offenbar wird der `else`-Block ausgeführt, wenn die Bedingung `False` ist.

Was, wenn wir mehrere Bedingungen haben? Zum Beispiel wollen wir eine Zahl zwischen 0 und 9? Lösung: `elif` (lies: else if).

```
In [14]: x = int(input("Geben Sie eine Zahl zwischen 0 und 9 ein: "))
if x < 0:
    print("Sie haben eine negative Zahl eingegeben.")
elif x > 9:
    print("Ihre Zahl ist größer als 9.")
else:
    print("Danke.")
```

Danke.

Hinweise:

1. Der `elif` -Block wird ausgeführt, wenn die `if` -Bedingung falsch ist, aber die `elif` -Bedingung wahr ist.
2. Genau einer der drei Blöcke wird ausgeführt. Der `else` -Block dient als Auffanglösung, wenn keiner der anderen ausgelöst wird.
3. Es kann mehr als einen `elif` -Block geben, aber nur einen `if` - und einen `else` -Block.

Übung

Der Body-Mass-Index (BMI) wird definiert als das Gewicht einer Person (in kg) geteilt durch die quadrierte Körpergröße in Metern. Die folgenden Schwellenwerte existieren (Quelle: [NHS](#)):

- unter 18.5 – Untergewicht
- 18.5 bis 25 - gesund
- 25 bis 30 - Übergewicht
- über 30 - fettleibig.

Schreiben Sie ein Programm, das den Benutzer nach seinem Gewicht und seiner Größe fragt, den BMI berechnet und eine Nachricht ausgibt, die dem Benutzer mitteilt: **Sie sind**

Weitere eingebaute Datentypen

- Weitere eingebaute Datentypen umfassen `set s` (ungeordnete Sammlungen) und `dict s` (Sammlungen von Schlüssel-Wert-Paaren). Mehr dazu später.

Hausaufgabe

- Listen, Methoden: Anfängerübungen 6-7, 9-10, 17, 18 von <https://holypython.com/beginner-python-exercises/>
- Bedingungen:
 - Mittelschwere (Intermediate) Übungen 7a, 7b von <https://holypython.com/intermediate-python-exercises/>
 - Übungen 2, 31, 33 und 40 von <https://www.w3resource.com/python-exercises/python-conditional-statements-and-loop-exercises.php>

Hinweis für die Musterlösung von Übung 33: Sie können wie folgt prüfen, ob eine Liste oder ein Tupel einen bestimmten Wert enthält:

```
In [15]: 3 in [1, 2, 3]
```

```
Out[15]: True
```