



# Programmieren für Einsteiger

## Python cheat sheet - Tag 3

### Namensräume

Namen werden immer zuerst im aktuellen Namensraum gesucht. Wird ein Name im aktuellen Namensraum nicht gefunden, dann sucht der Pythoninterpreter im nächst höheren Namensraum.

```
def tolle_funktion(z):
    x = 1
    print(x + y + z)

x = 1000
y = 2

tolle_funktion(3) -> 6
```

#### built-in Namespace

print, len, min, range, ...

#### global Namespace

tolle\_funktion = function  
x = 1000  
y = 2

#### local Namespace

z = 3  
x = 1

### Strings Typfunktionen von String

Text in Großbuchstaben umwandeln: `x.upper()` -> 'SOME TEXT'

Text in Kleinbuchstaben umwandeln: `x.lower()` -> 'some text'

Sind alle Symbole Zahlen? `x.isdigit()` -> False

Sind alle Symbole Buchstaben? `x.isalpha()` -> False

`'abc'.isalpha()` -> True

Zeichen an Beginn & Ende entfernen: `x.strip('sxt')` -> 'ome Te'

String in Liste von Teilstrings zerlegen: `x.split(' ')` -> ['some', 'Text']

### Dateien lesen/ schreiben

Dateien werden mit **open()** geöffnet, d.h. der Inhalt der Datei wird in den Arbeitsspeicher geladen. Mit **close()** wird die Datei gespeichert und wieder geschlossen. Erzeugt man ein Dateiojekt im Schreibmodus ('w'), wird unter dem angegebenen Pfad eine leere Datei mit diesem Namen angelegt. Achtung: existiert bereits eine Datei mit diesem Namen wird diese Datei überschrieben.

```
y = open('C:\Users\Emanuel\neue_textdatei.txt', 'w') <- # Datei im Schreib-Modus öffnen
y.close()

x = open('C:\Users\Emanuel\neue_textdatei.txt', 'r') <- # Datei im Lese-Modus öffnen
x.close()
```

Öffnet man ein Datei-Objekt mit der Anweisung **with as**, so wird das Datei-Objekt automatisch geschlossen, wenn der **with** Anweisungsblock verlassen wird.

```
with open('/home/python/wichtige_daten.dat', 'w') as output:
    output.write('Datei-Objekt wird geschlossen.')
output.write('schon geschlossen') -> ValueError: I/O operation on closed file.
```

### f-strings

Mit f-strings (format strings) lassen sich String und nicht-String Argumente mit Hilfe von Platzhalten in einen String zusammenfassen.

```
print(f'Liste: {l} 1tes Element: {l[0]} Summe: {l[0] + l[1] + l[2]}')
```

### json

Das Format json kann verwendet werden Daten in komplexeren Datenstrukturen (also nicht als Strings) zu speichern

```
import json
wichtigeDaten = {'Formel': 'E=m*c**2', 'Messwerte': [5.19, 11.74], 'Zahl': 8}
json.dump(wichtigeDaten, open('/home/Documents/wichtigeDaten.json', 'w'))
wichtigeDaten = json.load(open('/home/Documents/wichtigeDaten.json', 'w'))
```

Lesemodus ('r')

Schreibmodus ('w')  
Erweiterungsmodus ('a')

Typfunktion	Erklärung
<code>read()</code>	Der gesamte Inhalt der Datei, wird als String zurückgegeben.
<code>readline()</code>	Die nächste Zeile (bis zum nächsten Zeilenbruch) der Datei wird als String zurückgegeben.
<code>readlines()</code>	Der gesamte Inhalt der Datei, wird zeilenweise eingelesen und als eine Liste von Strings zurückgegeben.
<code>close()</code>	Das Datei-Objekt wird geschlossen.

Typfunktion	Erklärung
<code>write(x)</code>	Das Argument x wird als String in das Datei-Objekt geschrieben (aber noch nicht auf die Datei auf der Festplatte).
<code>flush()</code>	Alle Änderungen am Inhalt des Datei-Objekts werden auf die Datei auf der Festplatte übertragen. Das Datei-Objekt wird <u>nicht</u> geschlossen.
<code>close()</code>	Das Datei-Objekt wird geschlossen und alle Änderungen an dessen Inhalt werden auf die Datei auf der Festplatte übertragen.

### Module importieren

Mit der **import** Anweisung import man sich den Inhalt eines Moduls in den *global Namespace* seines Programms

```
import random
random.randint(0,10)
```

```
from random import randint
randint(0,10)
```

Externe Module lassen sich über das Verwaltungsprogramm pip herunterladen und installieren.

```
pip install PACKETNAME
```