# 03_logical_functions

October 20, 2020

## 1 Logische Strukturen (if-else, for, while), Lambda-Ausdrücke, Funktionen (in Jupyter Notebook)

Übersicht Python zur Veranschaulichung der Syntax

### 1.1 Datentypen

#### 1.1.1 Zahlen

[7]:
```
1 + 1
```

[7]:
```
2
```

[8]:
```
1 * 3
```

[8]:
```
3
```

[9]:
```
1 / 2
```

[9]:
```
0.5
```

[10]:
```
2 ** 4
```

[10]:
```
16
```

[11]:
```
4 % 2
```

[11]:
```
0
```

[12]:
```
5 % 2
```

[12]:
```
1
```

[13]:
```
(2 + 3) * (5 + 5)
```

[13]:
```
50
```

### 1.1.2 Variablenzuordnung

[14]:
```
# Kann nicht mit Zahl oder Sonderzeichen beginnen
# Konvention: verständliche Variablennamen in Kleinbuchstaben mit _ separiert
name_der_var = 2
```

[15]:
```
x = 2
y = 3
```

[16]:
```
z = x + y
```

[17]:
```
z
```

[17]: 5

### 1.1.3 Strings

[18]:
```
'single quotes'
```

[18]: 'single quotes'

[19]:
```
"double quotes"
```

[19]: 'double quotes'

[20]:
```
# wenn ' in String vorkommt, dann double quotes verwenden
"I don't know"
```

[20]: "I don't know"

### 1.1.4 Printing

[21]:
```
x = 'moin'
```

[22]:
```
x
```

[22]: 'moin'

[23]:
```
print(x)
```

```
moin
```

[24]:
```
num = 12
name = 'Sam'
```

[25]:
```
# sehr nützlich und wird sehr oft verwendet
print('Meine Zahl ist: {one}, und mein Name ist: {two}'.format(one=num,two=name))
```

```
Meine Zahl ist: 12, und mein Name ist: Sam
```

[26]:
```
print('Meine Zahl ist: {}, und mein Name ist: {}'.format(num,name))
```

```
Meine Zahl ist: 12, und mein Name ist: Sam
```

### 1.1.5 Listen

[27]:
```
[1,2,3]
```

[27]:
```
[1, 2, 3]
```

[28]:
```
['hi',1,[1,2]]
```

[28]:
```
['hi', 1, [1, 2]]
```

[29]:
```
meine_liste = ['a','b','c']
```

[30]:
```
meine_liste.append('d')
```

[31]:
```
meine_liste
```

[31]:
```
['a', 'b', 'c', 'd']
```

[32]:
```
meine_liste[0]
```

[32]:
```
'a'
```

[33]:
```
meine_liste[1]
```

[33]:
```
'b'
```

[34]:
```
meine_liste[1:]
```

[34]:
```
['b', 'c', 'd']
```

[35]:
```
meine_liste[:1]
```

[35]:
```
['a']
```

[36]:
```
meine_liste[0] = 'NEU'
```

[37]:
```
meine_liste
```

[37]:
```
['NEU', 'b', 'c', 'd']
```

[38]:
```
# geschachtelte Listen bzw. mehrdimensionale Listen
nest = [1,2,3,[4,5,['target']]]
```

[39]:
```
nest[3]
```

[39]:
```
[4, 5, ['target']]
```

[40]:
```
nest[3][2]
```

[40]:
```
['target']
```

[41]:
```
nest[3][2][0]
```

[41]:
```
'target'
```

### 1.1.6 Dictionaries

[42]:
```
# sogenannte key-value pairs und haben keine Reihenfolge (Unterschied zu Listen)
d = {'key1':'item1','key2':'item2'}
```

[43]:
```
d
```

[43]:
```
{'key1': 'item1', 'key2': 'item2'}
```

[44]:
```
d['key1']
```

[44]:
```
'item1'
```

[45]:
```
# auch hier mehrdimensional möglich
d_nest = {'k1': {'innerkey':[1,2,3]}}
```

[46]:
```
d_nest['k1']['innerkey'][2]
```

[46]:
```
3
```

### 1.1.7 Booleans

[47]:
```
True
```

[47]:
```
True
```

[48]:
```
False
```

[48]: False

### 1.1.8 Tuple

[49]:
```
# ähnlich zu Listen aber () werden verwendet
t = (1,2,3)
```

[50]:
```
t[0]
```

[50]: 1

[51]:
```
# führt zu einem Error, da Tuple sich nicht verändern lassen (Unterschied zu␣
 ↪Listen)
# nützlich wenn man nicht möchte, das ein Benutzer die Werte ändern kann
t[0] = 'NEU'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-51-c071ac6a331b> in <module>
      1 # führt zu einem Error, da Tuple sich nicht verändern lassen (Unterschie␣
 ↪zu Listen)
      2 # nützlich wenn man nicht möchte, das ein Benutzer die Werte ändern kan
----> 3 t[0] = 'NEU'

TypeError: 'tuple' object does not support item assignment
```

### 1.1.9 Mengen

[ ]:
```
{1,2,3}
```

[ ]:
```
# jedes Element, kann nur einmal vorkommen (Eigenschaft einer Menge)
# dadurch wwerden hier die Elemente reduziert
{1,2,3,1,2,1,2,3,3,3,3,2,2,2,1,1,2}
```

## 1.2 Vergleichsoperatoren

[ ]:
```
1 > 2
```

[ ]:
```
1 < 2
```

[ ]:
```
1 >= 1
```

[ ]:
```
1 <= 4
```

[ ]:
```
1 == 1
```

[ ]:
```
'hi' == 'bye'
```

### 1.3 logische Operatoren

[ ]:
```
(1 > 2) and (2 < 3)
```

[ ]:
```
(1 > 2) or (2 < 3)
```

[ ]:
```
(1 == 2) or (2 == 3) or (4 == 4)
```

### 1.4 if,elif, else Ausdrücke

[ ]:
```
if 1 < 2:
    print('Yep!') # Abstand wichtig (Python verwendet keine {}); wird von␣
 ↪Jupyter und den meisten IDEs automatisch eingefügt
```

[ ]:
```
if 1 < 2:
    print('yep!')
```

[ ]:
```
if 1 < 2:
    print('first')
else:
    print('last')
```

[ ]:
```
if 1 > 2:
    print('first')
else:
    print('last')
```

[ ]:
```
# man kann soviele elif Anweisungen verwenden wie man möchte, es wird immer der␣
 ↪Block der ersten gültigen Anweisung ausgeführt
if 1 == 2:
    print('first')
elif 3 == 3:
    print('middle')
else:
    print('Last')
```

### 1.5 for Schleifen

[ ]:
```
seq = [1,2,3,4,5]
```

[ ]:
```
# erlaubt einem durch eine Liste/Sequenz zu iterieren
for item in seq:
    print(item)
```

[ ]:
```
for item in seq:
    print('Yep')
```

[ ]:
```
for num in seq:
    print(num+num)
```

### 1.6 while Schleife

[ ]:
```
i = 1
while i < 5:
    print('i is: {}'.format(i))
    i = i+1 # diese Zeile ist sehr wichtig, da wir sonst eine Endlosschleife␣
 ↪erzeugen
```

### 1.7 range()

[ ]:
```
range(5)
```

[ ]:
```
for i in range(5):
    print(i)
```

[ ]:
```
list(range(5))
```

### 1.8 list comprehension

[ ]:
```
x = [1,2,3,4]
```

[ ]:
```
out = []
for item in x:
    out.append(item**2)
print(out)
```

[ ]:
```
# in Python gibt es die Möglichkeit den Code der vorangegangenen Zelle kürzer zu␣
 ↪fassen
# sehr nützliches Konzept
```

```
[item**2 for item in x]
```

### 1.9 Funktionen

[ ]:
```
def my_func(param1='default'):  # hier wird durch die Zuweisung ='default' ein default Parameter gesetzt
    """
    Docstring hier
    mehrere Zeilen möglich
    """
    print(param1)
```

Docstring einer Funktion kann durch Shift+Tab abgefragt werden, wenn der Cursor an das Ende der ausgeschrieben Funktion gesetzt ist!

[ ]:
```
my_func
```

[ ]:
```
# führt die Funktion mit dem default Parameter aus. Geht nur wenn ein default Parameter gesetzt ist, oder die Funktion
# kein Parameter besitzt
my_func()
```

[ ]:
```
my_func('new param')
```

[ ]:
```
my_func(param1='new param')
```

[ ]:
```
def square(x):
    return x**2
```

[ ]:
```
out = square(2)
```

[ ]:
```
print(out)
```

### 1.10 lambda Ausdruck

[2]:
```
def times2(var):
    return var*2
```

[3]:
```
times2(2)
```

[3]:
```
4
```

[4]:
```
lambda var: var*2
```

[4]:
```
<function __main__.<lambda>(var)>
```

### 1.11 map und filter

[ ]:
```
seq = [1,2,3,4,5]
```

[ ]:
```
map(times2,seq)
```

[ ]:
```
list(map(times2,seq))
```

[ ]:
```
list(map(lambda var: var*2,seq))
```

[ ]:
```
filter(lambda item: item%2 == 0,seq)
```

[ ]:
```
list(filter(lambda item: item%2 == 0,seq))
```

### 1.12 Methoden

[ ]:
```
st = 'hello my name is Sam'
```

[ ]:
```
st.lower()
```

[ ]:
```
st.upper()
```

[ ]:
```
st.split()
```

[ ]:
```
tweet = 'Go Sports! #Sports'
```

[ ]:
```
tweet.split('#')
```

[ ]:
```
tweet.split('#')[1]
```

[ ]:
```
d
```

[ ]:
```
d.keys()
```

[ ]:
```
d.items()
```

[ ]:
```
lst = [1,2,3]
```

[ ]:
```
lst.pop()
```

[ ]:
```
lst
```

[ ]:
```
'x' in [1,2,3]
```

[ ]:
```
'x' in ['x','y','z']
```

## 2 Übungsblatt folgt!