Sequenzen (Folgen)

Sequenzen sind:

- Zeichenketten (Strings)
- Listen
  In einer Liste kann eine Folge beliebiger Objekte gespeichert werden, also zum Beispiel Strings, Integers, Float-Zahlen aber auch Listen und Tupel selbst. Eine Liste kann jederzeit während des Programmablaufs wieder geändert werden.
- Tupel
  In einem Tupel können wie in einer Liste eine Folge beliebiger Objekte gespeichert werden, aber ein Tupel kann dann während des weiteren Programmverlaufs nicht mehr verändert werden.
- Binärdaten
  Eine Sequenz von Binärdaten, die unveränderlich sein kann beim Typ "bytes" oder veränderlich beim Typ "bytearray"

Entnommen von: http://www.python-kurs.eu/python3_sequentielle_datentypen.php

Listen sind mutable (dt. veränderliche) Datentypen. Im Gegensatz sind Tupel (s.u.) immutable (dt. unveränderlichen) Datentypen.

Zu mutable vs. immutable siehe auch hier: http://python.haas.homelinux.net/python_kapitel_07_003.htm

Listen

Hier ein paar Beispiele:

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

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

```
['cat', 'bat', 'rat', 'elephant']
```

```
['cat', 'bat', 'rat', 'elephant']
```

```
['cat', 3.1415, True, None, 42]
```

```
['cat', 3.1415, True, None, 42]
```

```
spam = ['cat', 'bat', 'rat', 'elephant']
```

```
spam
```

```
['cat', 'bat', 'rat', 'elephant']
```

Elemente aufrufen

```
spam[0]
```

```
'cat'
```

```
spam[1]
```

```
'bat'
```

```
spam[2]
```

```
'rat'
```

```
spam[3]
```

```
'elephant'
```

```
spam[4]
```

```
---------------------------------------------------------------------------

IndexError                                Traceback (most recent call last)

<ipython-input-10-fe61278173ab> in <module>()
----> 1 spam[4]


IndexError: list index out of range
```

```
spam[1.0]
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-11-bbdea97a156f> in <module>()
----> 1 spam[1.0]


TypeError: list indices must be integers or slices, not float
```

Zu slices (dt. Abschnitte) siehe unten.

```
spam[int(1.0)]
```

```
'bat'
```

Listen in der Liste:

```
spam = [['cat', 'bat'], [10, 20, 30, 40, 50]]
```

```
spam[0]
```

```
['cat', 'bat']
spam[0][1]
'bat'
spam[1][2]
30
```

Negative Indizes

```
spam = ['cat', 'bat', 'rat', 'elephant']
spam[-1]
'elephant'
spam[-3]
'bat'
```

Teillisten mit Slices

```
spam[0:4]
['cat', 'bat', 'rat', 'elephant']
spam[1:3]
['bat', 'rat']
spam[0:-2]
['cat', 'bat']
spam[:2]
['cat', 'bat']
spam[1:]
['bat', 'rat', 'elephant']
spam[:]
['cat', 'bat', 'rat', 'elephant']
```

Länge einer Liste

```
len(spam)
4
```

Ändern einer Liste

```
spam
['cat', 'bat', 'rat', 'elephant']
spam[1] = 'bird'
spam
['cat', 'bird', 'rat', 'elephant']
spam[2] = spam[1]
spam
['cat', 'bird', 'bird', 'elephant']
spam[-1] = 123456
spam
['cat', 'bird', 'bird', 123456]
```

Listenverkettung

```
[1, 2, 3] + ['A', 'B', 'C', 'D']
[1, 2, 3, 'A', 'B', 'C', 'D']
['X', 'Y', 'Z'] * 3
['X', 'Y', 'Z', 'X', 'Y', 'Z', 'X', 'Y', 'Z']
spam = [1, 2, 3]
spam = spam + ['A', 'B', 'C', 'D']
spam
[1, 2, 3, 'A', 'B', 'C', 'D']
```

Entfernen von Elementen

```
spam = ['cat', 'bat', 'rat', 'elephant']
del spam[2]
spam
['cat', 'bat', 'elephant']
del spam[2]
spam
['cat', 'bat']
```

Beispiel

```
catNames = []
while True:
    print('Bitte geben Sie den Namen der Katze ' + str(len(catNames) + 1) + ' Oder enter zum
    name = input()
    if name == '':
        break
    catNames = catNames + [name]  # list concatenation
print('Die Katzen heißen:')
for name in catNames:
    print(' ' + name)
```

```
Bitte geben Sie den Namen der Katze 1 Oder enter zum Beenden
michl
Bitte geben Sie den Namen der Katze 2 Oder enter zum Beenden
daniel
Bitte geben Sie den Namen der Katze 3 Oder enter zum Beenden
mausi
Bitte geben Sie den Namen der Katze 4 Oder enter zum Beenden

Die Katzen heißen:
 michl
 daniel
 mausi
```

For-Schleifen mit Listen

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

```
0
1
2
3
4
```

```
for i in [0, 1, 2, 3, 4]:
    print(i)
```

```
0
1
2
3
4
```

```
supplies = ['pens', 'staplers', 'flame-thrower', 'binders']
for i in range(len(supplies)):
```

```
    print('Index ' + str(i) + ' in supplies ist: ' + supplies[i])
```

```
Index 0 in supplies ist: pens
Index 1 in supplies ist: staplers
Index 2 in supplies ist: flame-thrower
Index 3 in supplies ist: binders
```

Operatoren `in` und `not in`

```
'howdy' in ['Hallo', 'hi', 'howdy', 'heyas']
```

```
True
```

```
spam = ['Hallo', 'hi', 'howdy', 'heyas']
'cat' in spam
```

```
False
```

```
'howdy' in spam
```

```
True
```

```
'cat' not in spam
```

```
True
```

```
'howdy' not in spam
```

```
False
```

Mehrfachzuweisung

```
cat = ['fat', 'black', 'loud']
size = cat[0]
color = cat[1]
disposition = cat[2]
print(size)
print(color)
print(disposition)
```

```
fat
black
loud
```

```
cat = ['fat', 'black', 'loud']
size, color, disposition = cat
print(size)
print(color)
print(disposition)
```

```
fat
black
loud
```

```
cat = ['fat', 'black', 'loud']
size, color, disposition, name = cat
print(size)
print(color)
print(disposition)
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-63-26ea76513fef> in <module>()
      1 cat = ['fat', 'black', 'loud']
----> 2 size, color, disposition, name = cat
      3 print(size)
      4 print(color)
      5 print(disposition)


ValueError: not enough values to unpack (expected 4, got 3)
```

Zuweisungsanweisung	Entsprechende Anweisung mit erweitertem Zuweisungsoperator
spam = spam + 1	spam += 1
spam = spam - 1	spam -= 1
spam = spam * 1	spam *= 1
spam = spam / 1	spam /= 1
spam = spam % 1	spam %= 1

Methoden von Listen

Siehe auch hier: http://www.python-kurs.eu/python3_listen.php

Elemente finden mit index()

```
spam = ['Hallo', 'hi', 'howdy', 'heyas']
spam.index('Hallo')
```

```
0
```

```
spam.index('heyas')
```

```
3
```

```
spam = ['Hallo', 'hi', 'howdy', 'heyas', 'Hallo']
spam.index('Hallo')
```

```
0
```

Elemente hinzufügen

```
spam = ['Hallo', 'hi', 'howdy']
spam.append('Hola')
spam
```

```
['Hallo', 'hi', 'howdy', 'Hola']
```

```
spam.insert(1, 'Hurray')
spam
```

```
['Hallo', 'Hurray', 'hi', 'howdy', 'Hola']
```

insert() und append() haben den Rückgabewert None. Diese Listen-Methoden **ändern** die zugehörige Liste.

Achtung insert() und append() funktionieren nicht auf Strings oder Integers.

Der String ansich ist immutable.

```
spam = 'Hallo'
spam.append(' Welt')
```

```
---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-71-0bc0d7f5af26> in <module>()
      1 spam = 'Hallo'
----> 2 spam.append(' Welt')


AttributeError: 'str' object has no attribute 'append'
```

```
spam = 42
spam.insert(1, Welt)
```

```
---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-72-dd7235097e60> in <module>()
      1 spam = 42
----> 2 spam.insert(1, Welt)


AttributeError: 'int' object has no attribute 'insert'
```

```
import time

n= 100000

start_time = time.time()
l = []
for i in range(n):
    l = l + [i * 2]
print("Zeit mit l = l + [i * 2]: {0:9.4f}".format(time.time() - start_time))

start_time = time.time()
l = []
for i in range(n):
    l += [i * 2]
print("Zeit mit l += [i * 2]: {0:12.4f}".format(time.time() - start_time))

start_time = time.time()
l = []
for i in range(n):
    l.append(i * 2)
print("Zeit mit l.append(i * 2): {0:9.4f}".format(time.time() - start_time))
```

```
Zeit mit l = l + [i * 2]:   35.8227
Zeit mit l += [i * 2]:       0.0306
Zeit mit l.append(i * 2):    0.0271
```

Elemente löschen

```
spam = ['Hallo', 'hi', 'howdy', 'heyas', 'Hallo']
spam.remove('Hallo')
spam
```

```
['hi', 'howdy', 'heyas', 'Hallo']
```

```
spam.remove('Hola')
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-75-8caad95758bd> in <module>()
----> 1 spam.remove('Hola')


ValueError: list.remove(x): x not in list
```

Elemente sortieren

```
spam = [2, 5, 3.14, 1, -7]
spam.sort()
spam
```

```
[-7, 1, 2, 3.14, 5]
```

```
spam = ['Hallo', 'hi', 'howdy', 'heyas', 'Hallo']
spam.sort(reverse=True)
spam
```

```
['howdy', 'hi', 'heyas', 'Hallo', 'Hallo']
```

```
# String und Integer geht nicht
spam = [2, 5, 3.14, 1, -7, 'Hallo', 'hi', 'howdy', 'heyas', 'Hallo']
spam.sort()
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-79-9f2ffbd30d10> in <module>()
      1 # String und Integer geht nicht
    2 spam = [2, 5, 3.14, 1, -7, 'Hallo', 'hi', 'howdy', 'heyas', 'Hallo']
----> 3 spam.sort()


TypeError: unorderable types: str() < int()
```

Sortiert wird nach ASCII

```
spam = ['Alice', 'ants', 'Bob', 'badgers', 'Carol', 'cats']
spam.sort()
spam
```

```
['Alice', 'Bob', 'Carol', 'ants', 'badgers', 'cats']
```

```
spam.sort(key=str.lower)
spam
```

```
['Alice', 'ants', 'badgers', 'Bob', 'Carol', 'cats']
```

Arbeitsauftrag Implementieren Sie mit Hilfe von Listen einen Magic 8 Ball.

- https://de.wikipedia.org/wiki/Magic_8_Ball
- http://www.m8ball.com/de/

Listenähnliche Typen: Strings und Tupel

```
name = 'Sophie'
name[0]
```

```
'S'
```

```
name[-2]
```

```
'i'
```

```
name[0:4]
```

```
'Soph'
```

```
'So' in name
```

```
True
```

```
's' in name
```

```
False
```

```
'p' not in name
```

```
False
```

```
for i in name:
    print(i)
```

```
S
o
p
h
i
e
```

```
type(name)
```

```
str
```

Veränderbare und unveränderbare Datentypen

```
name = 'Sophie eine Katze'
name[7] = 'die'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

<ipython-input-1-1bdc148e1df1> in <module>()
      1 name = 'Sophie eine Katze'
----> 2 name[7] = 'die'


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

```
name = 'Sophie eine Katze'
newName = name[0:7] + 'die' + name[11:17]
newName
```

```
'Sophie die Katze'
```

Tupel

Tuppel sind, wie Strings, nicht veränderbare Typen.

Benötigt man eine sortierte unveränderliche Reihenfolge, so verwendet man Tupel.

Nicht veränderbare Typen können optimiert werden.

```
eggs = ('hallo', 42, 0.5)
eggs[0]
```

```
'hallo'
```

```
eggs[1:3]
```

```
(42, 0.5)
```

```
len(eggs)
```

```
3
```

```
eggs[1]
```

```
42
```

```
eggs[2]
```

```
0.5
```

```
eggs[1] = 99
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

<ipython-input-100-4c1b72b4295d> in <module>()
----> 1 eggs[1] = 99


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

```
type(eggs)
```

```
tuple
```

```
type('Hallo')
```

```
str
```

```
eggs = ('hallo',)
```

```
eggs
```

```
('hallo',)
```

```
type(eggs)
```

```
tuple
```

```
type(('hallo',))
```

```
tuple
```

```
type(('hallo'))
```

```
str
```

Umwandlen von Tupel und Listen

```
spam = ['Alice', 'ants', 'badgers', 'Bob', 'Carol', 'cats']
spamTuple = tuple(spam)
type(spamTuple)
```

```
tuple
```

```
list(spamTuple)
```

```
['Alice', 'ants', 'badgers', 'Bob', 'Carol', 'cats']
```

```
type(list(spamTuple))
```

```
list
```

```
list('hallo')
```

```
['h', 'a', 'l', 'l', 'o']
```

Verweise

```
spam = 42
id(spam)
```

```
140663981683168
```

```
cheese = spam
id(cheese)
```

```
140663981683168
```

```
spam = 100
spam
```

```
100
```

```
cheese
```

```
42
```

```
id(spam)
```

```
140663981685024
```

```
id(cheese)
```

```
140663981683168
```

Jetzt sollte man dies wirklich lesen: http://python.haas.homelinux.net/python_kapitel_07_003.htm

```
spam = [0, 1, 2, 3, 4, 5]
id(spam)
```

```
140663684979720
```

```
cheese = spam
id(cheese)
```

```
140663684979720
```

```
cheese[1] = 'Hallo!'
spam
```

```
[0, 'Hallo!', 2, 3, 4, 5]
```

```
id(spam)
```

```
140663684979720
```

```
id(cheese)
```

```
140663684979720
```

Verweise übergeben

```
def eggs(einParameter):
    einParameter.append('Hallo!')

spam = [1, 2, 3]
eggs(spam)
spam
```

```
[1, 2, 3, 'Hallo!']
```

copy() und deepcopy()

```
import copy
spam = [0, 1, 2, 3, 4, 5]
cheese = copy.copy(spam)
cheese[1] = 'Hallo!'
spam
```

```
[0, 1, 2, 3, 4, 5]
```

```
cheese
```

```
[0, 'Hallo!', 2, 3, 4, 5]
```

Enthält eine Liste eine List, so verwendet man deepcopy() um alles zu kopieren.