

## 3. Weiterführende Python Konzepte

---

Eingebaute Funktionen, Bedingte Ausdrücke, List und Dict Comprehension, File IO, Fehlerbehandlung

# Eingebaute Funktionen: Aufzählungen mit Index

Manchmal möchte man durch Listen iterieren, inklusive Index für jedes Element. Dies geht mit `enumerate( ... )`

```
data = [ 'Spam', 'Eggs', 'Ham' ]
```

```
for index, value in enumerate(data):  
    print(index, ":", value)
```

Output:

```
0 : Spam  
1 : Eggs  
2 : Ham
```

# Eingebaute Funktionen: Listen kombinieren

Es gibt eine einfache Möglichkeit, Listen element-weise zusammenzuführen (wie ein Reissverschluss!): `zip( ... )`

```
places = [ 'Zurich', 'Basel', 'Bern' ]  
plz = [ 8000, 4000, 3000, ]
```

```
list(zip(places, plz))  
# [('Zurich', 8000), ('Basel', 4000), ('Bern', 3000)]
```

```
dict(zip(places, plz))  
# {'Zurich': 8000, 'Basel': 4000, 'Bern': 3000}
```

# Bedingte Ausdrücke

In Python kann der Wert eines Ausdrucks von einer Bedingung abhängen (als Teil des Ausdrucks!)

Beispiel: Collatz Folge

```
while a != 1:  
    a = a // 2 if a % 2 == 0 else a * 3 + 1
```

Beispiel: Textformatierung

```
print('I see', n, 'mouse' if n == 1 else 'mice')
```

# List Comprehension

- Python bietet eine sehr angenehme Möglichkeit an, Listen deklarativ zu erstellen
- Ähnliche Denkweise wie bei 'map' und 'filter' bei funktionalen Sprachen

Beispiel: Eine Sequenz von Zahlen einlesen

```
line = input('Enter some numbers: ')
s_list = line.split()
n_list = [ int(x) for x in s_list ]
```

Das selbe kombiniert in einem Ausdruck

```
n_list = [ int(x) for x in input('Enter some numbers: ').split() ]
```

# List Comprehension

Beispiel: Leerschläge vorne und hinten eliminieren

```
line = [ ' some eggs ', ' slice of ham ', ' a lot of spam ' ]  
cleaned = [ item.strip() for item in line ]  
  
# cleaned == [ 'some eggs', 'slice of ham', 'a lot of spam' ]
```

# Dict Comprehension

- Wie bei Listen, aber mit key/value Paaren

Beispiel: Daten extrahieren aus einem Dict

```
data = {  
    'Spam' : { 'Amount' : 12, 'Price': 0.45 },  
    'Eggs' : { 'Price': 0.8 },  
    'Ham'  : { 'Amount': 5,  'Price': 1.20 }  
}
```

```
total_prices = { item : record['Amount'] * record['Price']  
    for item, record in data.items()  
    if 'Amount' in record }
```

```
# total_prices == {'Spam': 5.4, 'Ham': 6.0}
```

# File IO

- Files können mit dem Befehl **open** geöffnet werden.
- Damit Files automatisch wieder geschlossen werden, muss das innerhalb eines **with** Blocks geschehen.

Beispiel: CSV File einlesen

```
import csv

with open('times.csv', mode='r') as csv_file:
    csv_lines = csv.reader(csv_file)
    for line in csv_lines:
        # do something for each record
```

Schreiben geht ähnlich. Siehe Python Doku.



# Ausnahmebehandlung

Gegeben folgender Code:

```
x = int(input('A number please: '))
```

Wenn keine Zahl eingegeben wird, stürzt das Programm ab:

Traceback (most recent call last):

```
File "main.py", line 1, in <module>
```

```
    x = int(input('A number please: '))
```

```
ValueError: invalid literal for int() with base 10: 'a'
```

Wir können diesen Fehler auffangen und entsprechend reagieren.

# Ausnahmebehandlung

```
try:  
    x = int(input('A number please: '))  
except ValueError:  
    print('Oh boy, that was no number...')  
    x = 0  
print('x:', x)
```

Ausgabe, wenn 'spam' eingegeben wird statt einer Zahl:

```
Oh boy, that was no number...  
x: 0
```