Datenstrukturen

Grundlagen der Programmierung

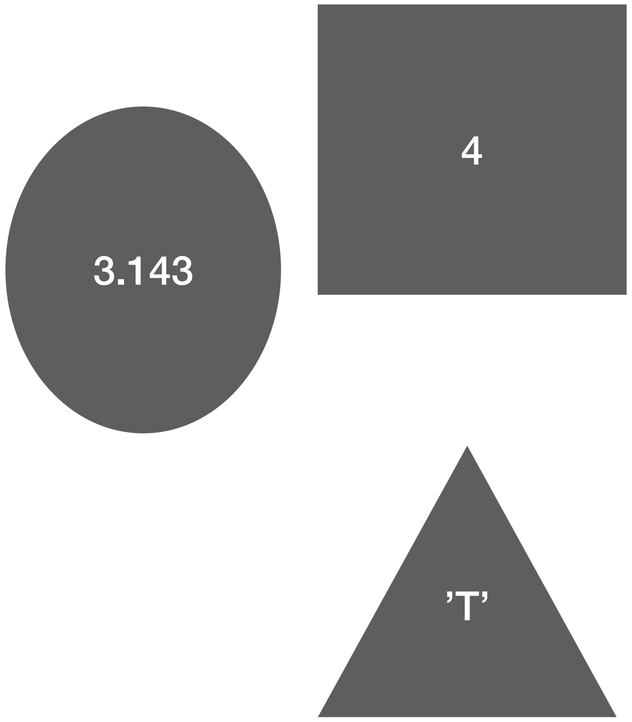
Wirtschaft
Hauptcampus

HOCHSCHULE TRIER

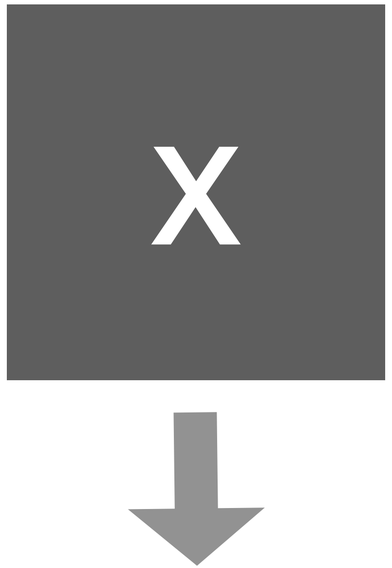
Fabian Roth, B. Sc.

Rückblick

Datentypen

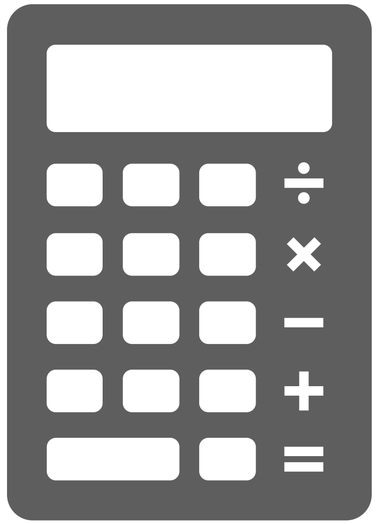


Variablen




```
'Grundlagen der
Programmierung’
```

Arithmetische Operatoren



Werte sammeln

Fabian Roth, B. Sc.

Problemstellung

- Werte können nur einzeln gespeichert werden
- Eingabe von großen Datenmengen ist aufwendig
- Regeln innerhalb der Daten fehlen

Beispiel Einkaufswagen

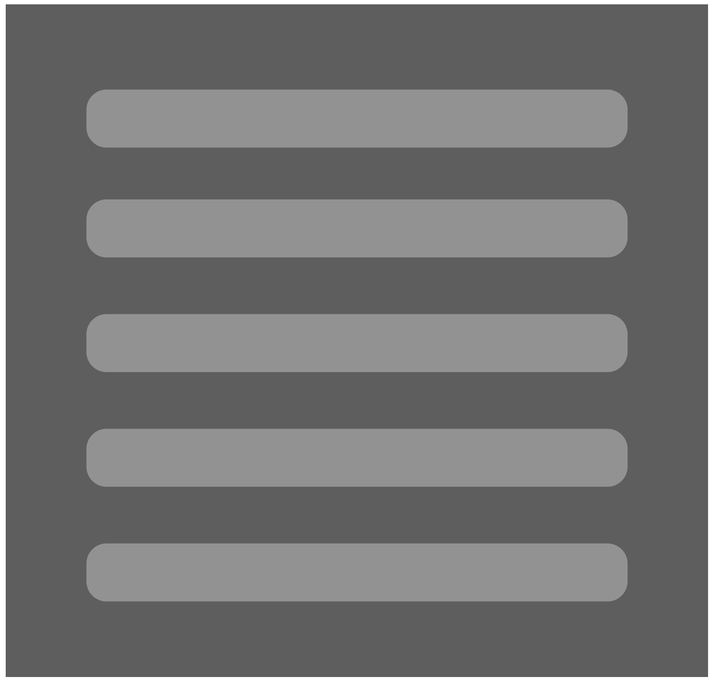
- Ein ganz normaler Einkauf
- Ab 11 Artikeln wird es monoton
- Namen werden zum Problem
- Je eine Variable für einen Artikel
  - Sehr aufwendig

```
artikel1 = 'Apfel'
artikel2 = 'Banane'
artikel3 = 'Tomaten'
artikel4 = 'Apfel'
artikel5 = 'Birne'
artikel6 = 'Milch'
artikel7 = 'Käse'
artikel8 = 'Butter'
artikel9 = 'Pudding'
artikel10 = 'Schokolade'
artikel11 = 'Eis'
artikel12 = 'Orangen'
```

Datenstrukturen

- Statt einzelner Werte, gibt es Gruppen von Werten
- Diese bringen eigene Regeln für Ihre Mitglieder mit

Metapher der Datenstrukturen

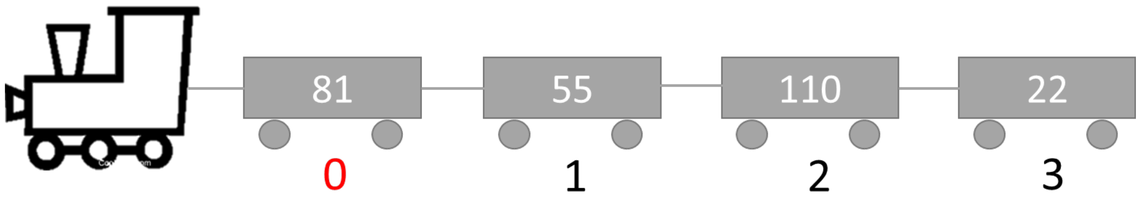


- Ähnlich einem Regal
- Alle Werte im Regal nennt man Gegenstände (***items***)
- Dieses Regal kann als eine Variable gespeichert werden

Listen

Key Facts

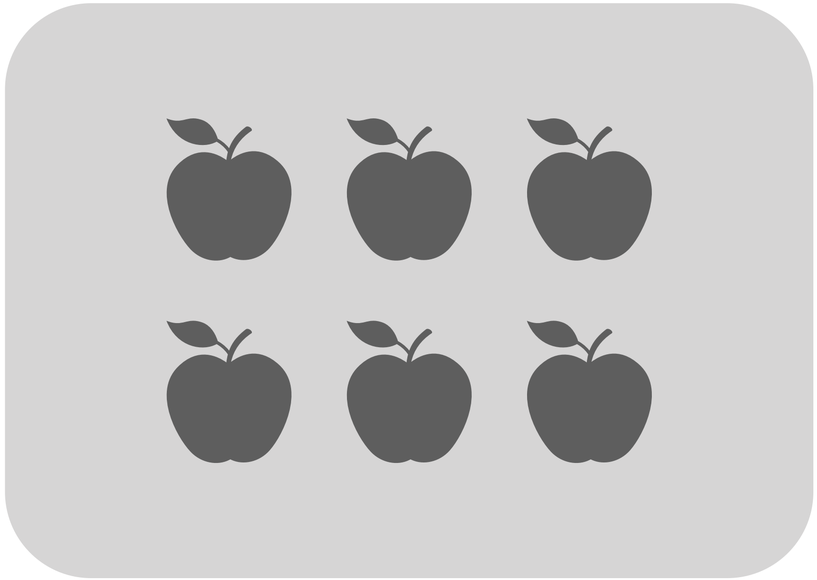
- *Eine Liste kann man sich vorstellen wie ein Zug mit Waggons.*
- *Alles mögliche* kann in jedem Waggon stehen, daher *ungeordnet*



Waggonnummer = Position der Liste (Index)

Waggon = Zugewiesene Element

Gründe für Listen



- Wenn man **gleichartige** Daten übergeben möchte
- Zugriff durch **Iteration** (verändern des Index)

Einkaufswagen als Liste

- Kein Problem mit den Namen mehr
- Wichtig ist, dass die ***eckigen Klammern*** verwendet werden, da sonst keine Liste erkannt wird
- Die Listeneinträge werden durch ein Komma getrennt

```
1 cart = ['Apfel',
2         'Banane',
3         'Tomaten',
4         'Apfel',
5         'Birne']
```

Daten über Index erfragen

- Umgang und Zugriff
  - Index folgt in eckigen Klammern
  - Änderung ebenfalls möglich

Die Liste beginnt immer mit einem Index von 0!

```
#Zugriff auf ein Element
a = [81,55,110,22]

print(a[0])
print(a[1])
print(a[2])
print(a[3])
```

Daten überschreiben

- Der Liste a wird an der Stelle 0 ein anderer Wert zugewiesen

```
#Überschreiben eines Elementes
a = [81,55,110,22]

print(a[0])
a[0] = 5
print(a[0])
print(a[1])
print(a[2])
print(a[3])
```

Daten mit *append* hinzufügen

- Fügt Gegenstand der Liste hinzu
- Dafür muss eine Stelle im Index angegeben werden

```
a = [81,55,110,22]
print(a)
#Ausgabe: [81,55,110,22]

#Der Index wird jetzt erweitert auf den
Wert 4
a.append(35)
print(a)
#Ausgabe: [81,55,110,22,35]
```

Daten mit *del* löschen

- Löscht Gegenstand aus der Liste
- Dafür muss eine Stelle im Index angegeben werden

```
a = [81,55,110,22]
del(a[2])

#Ausgabe: [81,55,22]
```

Listen - Länge bestimmen

- Für bestimmte Bedingungen ist die Länge von Nöten, deswegen gibt es die in Python vordefinierte Funktion len().

```
a = [81,55,110,22]
print(len(a))

a.append(len(a))
print(len(a))

Ausgabe:
4
5
```

Listen

- Listen von Listen
- Die Verschachtelung von Listen ist auch möglich. Jedoch kann dies sehr schnell unübersichtlich werden. Daher aufpassen!

```
temp = [ [ 1 , 2 , 3 ] , [ 4 , 5 , 6 ] ]

print ( temp [ 0 ] [ 0 ] )
print ( temp [ 1 ] [ 0 ] )

#Ausgabe: ?
```

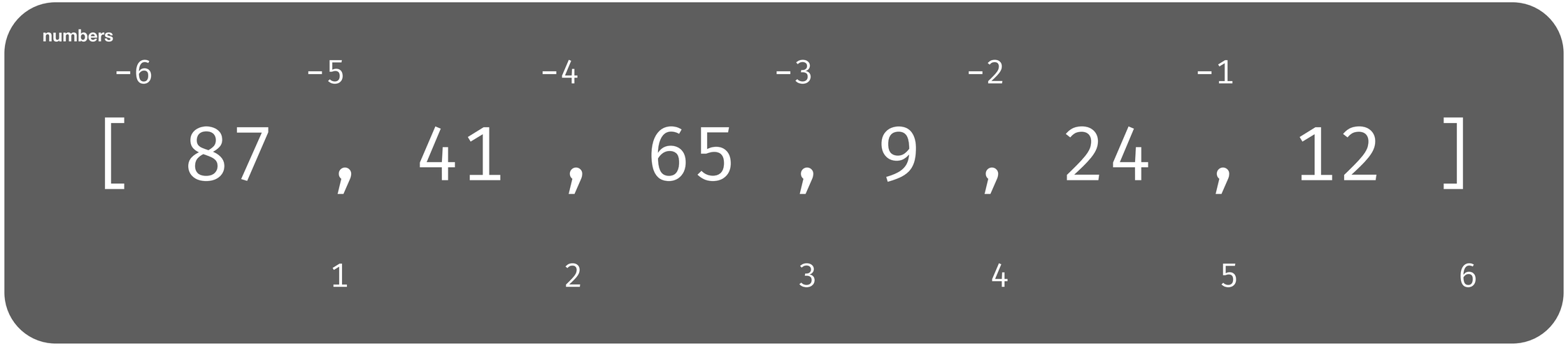
```
temp = [[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]]
(Sehr unübersichtlich)
```

Slices einer Liste erfragen

- Ein Stück (Slice) kann aus einer Liste erfragt werden
- Auch hier kann verändert werden

```
1 cars = ['bmw', 'audi', 'vw']
2 twoCars = cars[0:2]
3 print(twoCars)
4 # ['bmw', 'audi']
5
6 cars[0:2] = 'toyota', 'mercedes'
7 print(cars)
8 # ['toyota ', 'mercedes', 'vw']
```

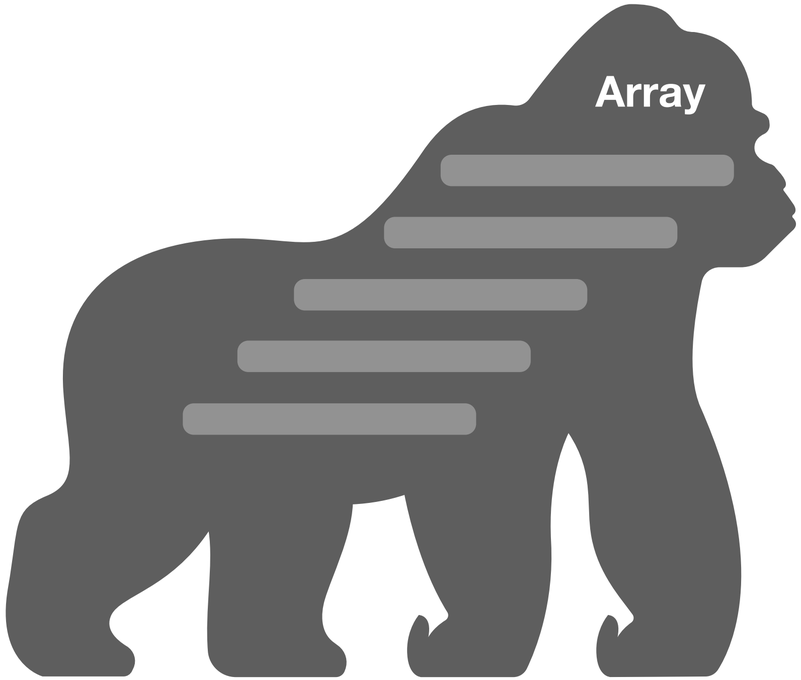
Slices zählen in Python




```
numbers[2:4] = [65,9]
numbers[7: ] = [ ]
numbers[3: ] = [9,24,12]
numbers[ :3] = [87,41,65]
```

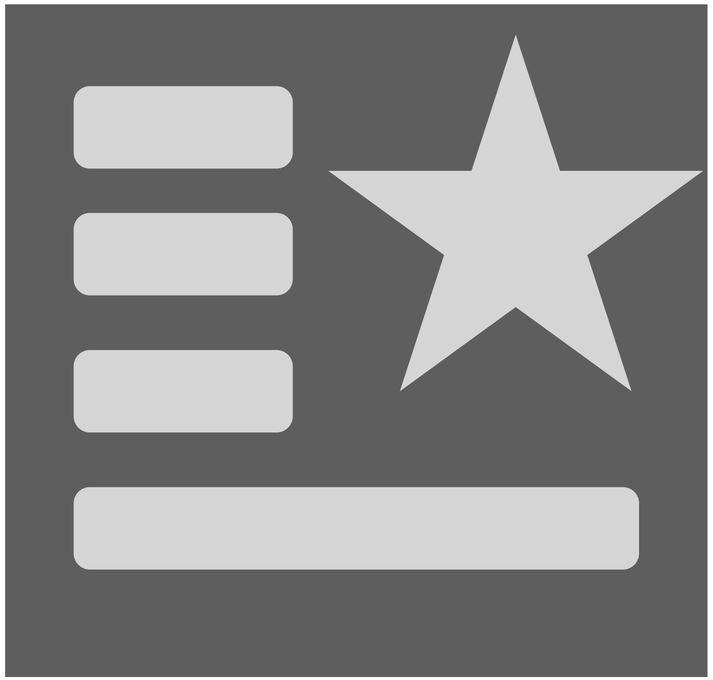
```
numbers[-5:-1] = [41,65,9,24]
numbers[-3:-6] = [ ]
numbers[-4:-2] = [65,9]
numbers[  :-3] = [87,41,65]
```

Warnung an Polyglotts



- Python bietet kein richtiges Äquivalent zu dem weit verbreiteten Array
- Eigentlich sind Arrays die grundlegendste Daten Struktur und ziemlich primitiv
- Namen und Eigenschaften von Daten Strukturen unterscheiden sich je nach Sprache mehr oder weniger
- Zweck von Daten Strukturen ist hingegen konstant

Erweiterte Problemstellung

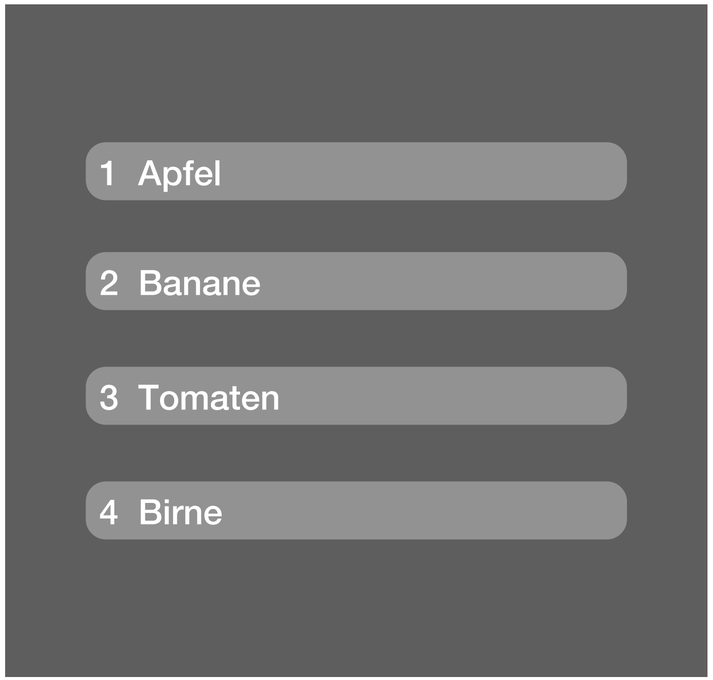


- Was wäre wenn …
  - keine Artikel doppelt vorkommen dürfen
  - Zahlen als Index ungeeignet sind
  - der Einkaufswagen nicht verändert werden darf

Das Set

Fabian Roth, B. Sc.

Das Set



- Liste ohne Duplikate
- Ungeordnet
- Statt einfach nur Werte unter einem Namen zusammen zu fassen, gibt es zusätzlich Regeln für die Daten

Einkaufswagen als Set

- Gleichen Eigenschaften wie die Liste
- Keine Duplikate möglich
- Werte stehen in **geschweiften Klammern**

```
>>> cart = {'Apfel',
            'Banane',
            'Tomaten',
            'Apfel',
            'Birne'}
>>> cart
{'Apfel', 'Banane', 'Tomaten', 'Birne'}
```

Daten ~~über Index~~ erfragen

- Änderung sind nicht möglich
  - Achtung: Das Set ist ‚*unindexed*‘
  - *Zugriff über Index ist also nicht möglich!*

```
#Zugriff auf ein Element
a = {81,55,110,22}

print(a[0])
print(a[1])
print(a[2])
print(a[3])


print(81 in a)
#Ausgabe: True

for x in a:
    print(x)
```

Daten mit *add* hinzufügen

- Fügt Gegenstand einem Set hinzu
- *Aber Achtung: Duplikate wie z.B. die Zahl 81 kann nicht nochmals hinzugefügt werden!*

```
a = {81,55,110,22}
print(a)
#Ausgabe: {81,55,110,22}

a.add(35)
print(a)
#Ausgabe: {81,55,110,22,35}
```

Daten mit *update* hinzufügen

- Fügt mehrere Gegenstände einem Set hinzu

```
a = {81,55,110,22}
print(a)
#Ausgabe: { 81,55,110,22 }

a.update([35,36,37,40])
print(a)
#Ausgabe: { 81,55,110,22,35,36,37,40 }
```

Daten mit *remove* löschen

- Löscht Gegenstand aus dem Set

```
a = {81,55,110,22}
print(a)
#Ausgabe: { 81,55,110,22 }

a.remove(35)
print(a)
#Ausgabe: { 81,55,110,22,36,37,40 }
```

Sets - Länge bestimmen

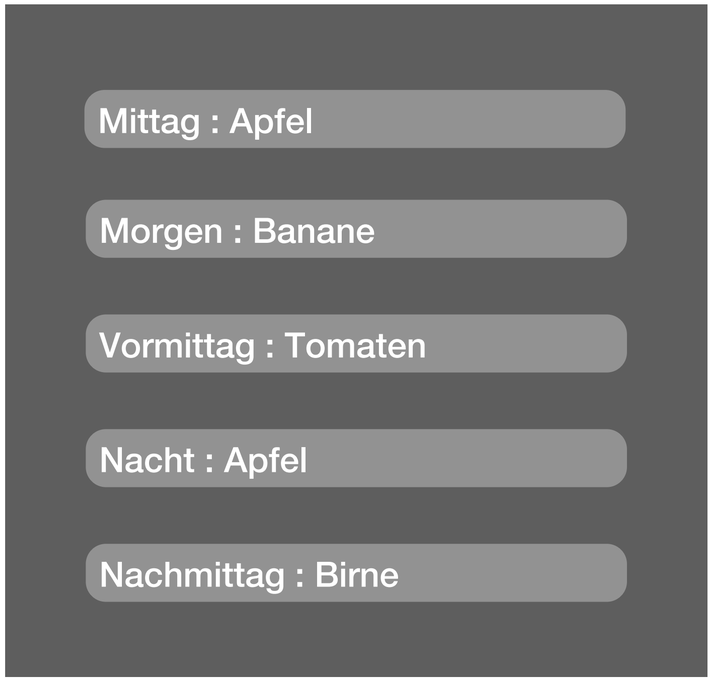
- Für bestimmte Bedingungen ist die Länge von Nöten, deswegen gibt es die in Python vordefinierte Funktion len().

```
a = {81,55,110,22}
print(len(a))

a.add(len(a))
print(len(a))

Ausgabe:
4
5
```

Das Dictionary

Fabian Roth, B. Sc.

Das Dictionary



- zu Deutsch Wörterbuch
- Nutzt Schlüssel (Keys) statt Zahlen als Index
- Es ist also ein Regal mit Namensschildern

Einkaufswagen als Dictionary

- Gleiche Eigenschaften wie die Liste
- Aber Schlüssel statt Zahlen als Index
- Werte sind sogenannte **Schlüssel Werte Paare** und werden durch den Doppelpunkt getrennt
- Umschlossen von **geschweiften Klammern**

```
1 cart = {'mittag':'Apfel',
2         'morgen':'Banane',
3         'vormittag':'Tomaten',
4         'nacht':'Apfel',
5         'nachmittag':'Birne'}
```

Daten über Index erfragen

- Änderung ebenfalls möglich

```
cars = { "brand": "Ford",
         "model": "Mustang",
         "year": 1964 }

print(cars)
#Ausgabe: {'brand': 'Ford‘,
 'model': 'Mustang', 'year': 1964}

print(cars["brand"])
#Ausgabe: Ford
```

Daten hinzufügen

- Fügt Gegenstand einem Dictionary hinzu
- Index ist hier das Schlüsselwort

```
cars = { "brand": "Ford",
         "model": "Mustang",
         "year": 1964 }


cars["color"] = „green"


print(cars)
#Ausgabe: {'brand': 'Ford‘,
          'model': 'Mustang', 'year': 1964,
          color‘: green }
```

Daten mit *del* löschen

- Löscht Gegenstand aus dem Dictionary

```
cars = { "brand": "Ford",
         "model": "Mustang",
         "year": 1964 }

del(cars["color"])


print(cars)
#Ausgabe: {'brand': 'Ford‘,
          'model': 'Mustang', 'year': 1964}
```

Dictionary - Länge

- Für bestimmte Bedingungen ist die Länge von Nöten, deswegen gibt es die in Python vordefinierte Funktion len().

```
cars = { "brand": "Ford",
         "model": "Mustang",
         "year": 1964 }


print(len(cars))
#Ausgabe: 3
```

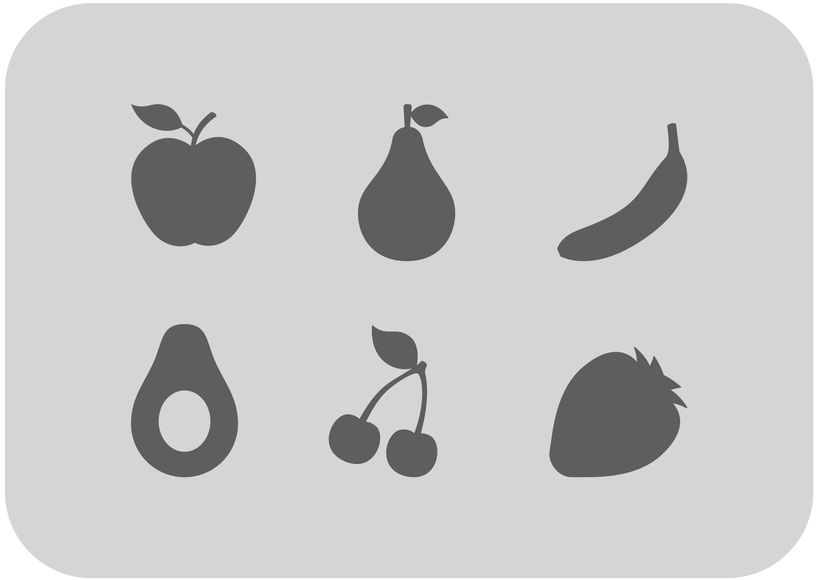
Das Tuple

Fabian Roth, B. Sc.

Das Tuple

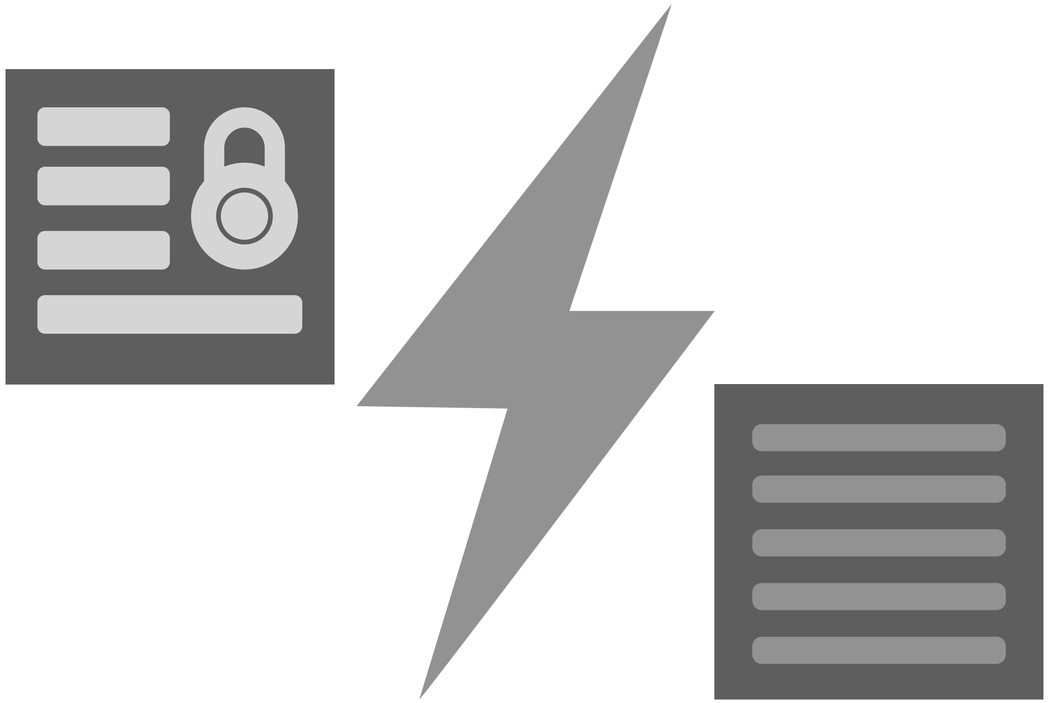
- Eine Liste, aber ohne die Möglichkeit etwas zu verändern
- Ungeordnet
- Also kein Regal, sondern ein Schaufenster

Gründe für Tuple



- Wenn man **vielfältige** Daten als ein Paket übergeben möchte

Tuple vs Liste



- Tuple sind unveränderlich (immutable)
- Listen sind veränderlich (mutable)
- Verwendungszweck ist daher abzuwägen

Einkaufswagen als Tuple

- unveränderlich
- Zu erkennen an den **runden Klammern**
- Definition aber auch ohne Klammern möglich, falls es eindeutig ist

```
1 cart = ('Apfel',
2         'Banane',
3         'Tomaten',
4         'Apfel',
5         'Birne')
```

Daten über Index erfragen

- Änderung sind nicht möglich!

```
cart = ('Apfel', 'Banane', 'Tomaten',
'Apfel', 'Birne')


print(cart)
#('Apfel', 'Banane', 'Tomaten`,
 'Apfel', 'Birne')

print(cart[1])
#Ausgabe: Banane
```

Weitere Befehle

- *Hinzufügen, löschen oder ändern sind in einem Tuple nicht möglich!*
  - Abhilfe schafft die Konvertierung zu einer Liste

```
cart = ('Apfel', 'Banane', 'Tomaten',
'Apfel', 'Birne‘)

cart_list = list(cart)
cart_list[0] = 'Kiwi‘

cart = tuple(cart_list)

print(cart)
```