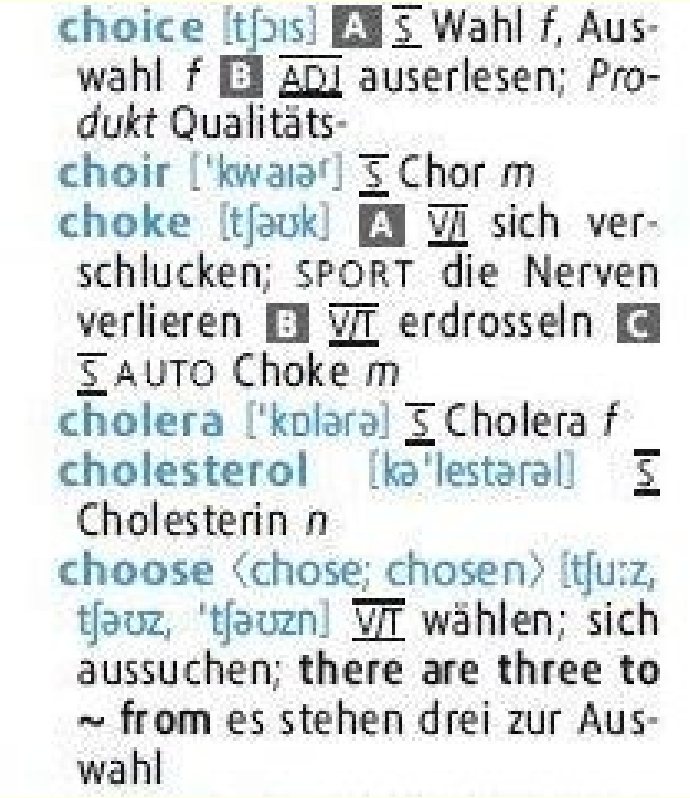
# Dictionaries

# Dictionaries

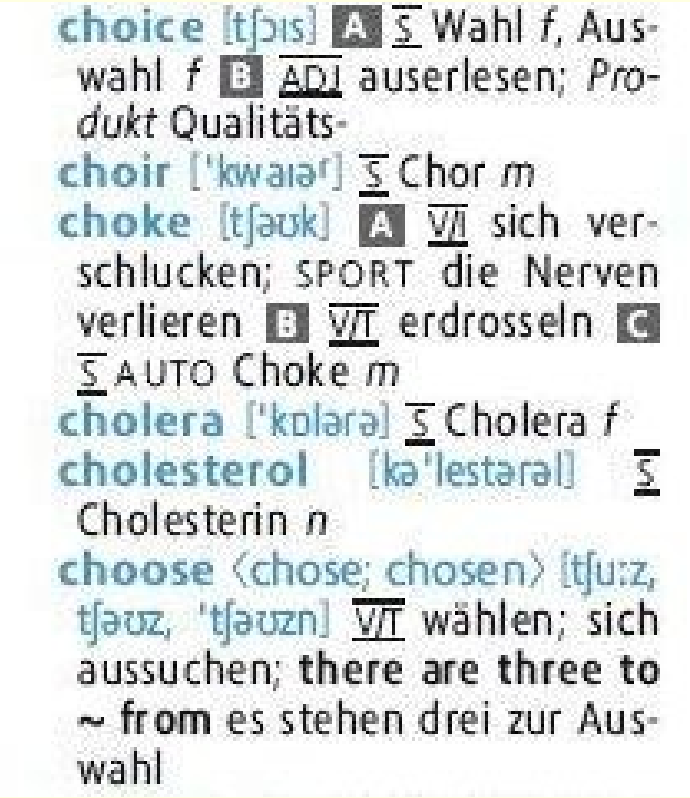
- Einträge zweiteilig
- Zugriff über erste Komponente



**choice** [tʃɔɪs] A S Wahl *f*, Auswahl *f* B ADJ auserlesen; *Produkt* Qualitäts-
**choir** ['kwaɪəʳ] S Chor *m*
**choke** [tʃəʊk] A V/I sich verschlucken; SPORT die Nerven verlieren B V/T erdrosseln C S AUTO Choke *m*
**cholera** ['kɒlərə] S Cholera *f*
**cholesterol** [kə'lestərəl] S Cholesterin *n*
**choose** ‹chose; chosen› [tʃu:z, tʃəʊz, 'tʃəʊzn] V/T wählen; sich aussuchen; **there are three to ~ from** es stehen drei zur Auswahl

# Dictionaries

- Einträge zweiteilig
- Zugriff über erste Komponente



**choice** [tʃɔɪs] A S Wahl *f*, Auswahl *f* B ADJ auserlesen; *Produkt* Qualitäts- **choir** ['kwaɪəʳ] S Chor *m* **choke** [tʃəʊk] A V/I sich verschlucken; SPORT die Nerven verlieren B V/T erdrosseln C S AUTO Choke *m* **cholera** ['kɒlərə] S Cholera *f* **cholesterol** [kə'lestərəl] S Cholesterin *n* **choose** ‹chose; chosen› [tʃu:z, tʃəʊz, 'tʃəʊzn] V/T wählen; sich aussuchen; **there are three to ~ from** es stehen drei zur Auswahl

- in Python nicht: geordnet

  nach erster Komponente

# Dictionaries

Erzeugung mit

- geschweiften Klammern
- Komponenten getrennt durch Doppelpunkt
- Einträge getrennt durch Komma

```
di = {
        'parentheses' : 'runde Klammern',
        'brackets' : 'eckige Klammern',
        'curly braces' : 'geschweifte Klammern'
      }
```

# Dictionaries

Zugriff über erste Komponente

- Schlüssel / key
- muss eindeutig sein

```
di = {
        'parentheses' : 'runde Klammern',
        'brackets' : 'eckige Klammern',
        'curly braces' : 'geschweifte Klammern'
     }

>>> di['brackets']
'eckige Klammern'
```

# Dictionaries

Mehrfaches Vorkommen eines Schlüssels in Definition möglich

Nur letzter Eintrag wird übernommen,

erster überschrieben

kein Fehler

```
>>> { 'eins' : 1,
       'eins' : 'keins',
       'zwei' : 2}

{'eins': 'keins', 'zwei': 2}
```

# Dictionaries

Für beide Komponenten der Einträge Mischung von Datentypen erlaubt

- Schlüssel nur immutabel
- Wert mutabel oder

  immutabel

```
mapp = {
        0           : 1,
        "abc"       : 0.5,
        1.2e22      : [1,2,3,4],
        (1,3,3,7) : "def"
}
```

# Dictionaries

Dictionary selbst ist

- mutabel
- iterabel

# Dictionaries

## Iterabel

```
>>> di = { 'parentheses' : 'runde Klammern', 'brackets' : 'eckige
Klammern', 'curly braces' : 'geschweifte Klammern' }

>>> for wert in di: print( wert )
...
parentheses
brackets
curly braces
```

# Dictionaries

## Iterabel

```
>>> di = { 'parentheses' : 'runde Klammern', 'brackets' : 'eckige
Klammern', 'curly braces' : 'geschweifte Klammern' }

>>> for wert in di: print( wert )
...
parentheses
brackets
curly braces

>>> for wert in di: print( di[ wert ] )
...
runde Klammern
eckige Klammern
geschweifte Klammern
```

# Dictionaries

Iterabel für Schlüssel und Wert

- Methode items()

```
>>> di = { 'parentheses' : 'runde Klammern', 'brackets' : 'eckige
Klammern', 'curly braces' : 'geschweifte Klammern' }

>>> for key, wert in di.items() : print( key, " / ", wert )
...
parentheses  /  runde Klammern
brackets  /  eckige Klammern
curly braces  /  geschweifte Klammern
```

# Dictionaries

Iteration direkt durch Werte

- Methode values()

```
>>> di = { 'parentheses' : 'runde Klammern', 'brackets' : 'eckige
Klammern', 'curly braces' : 'geschweifte Klammern' }

>>> for wert in di.values() : print( wert )
...
runde Klammern
eckige Klammern
geschweifte Klammern
```

# Dictionaries

## Operationen

- Zugriff und len() wie Liste

```
>>> len(di)
3

>>> di[ 'brackets' ]
'eckige Klammern'
```

# Dictionaries

## Operationen

- Zugriff und len() wie Liste

```
>>> len(di)
3

>>> di[ 'brackets' ]
'eckige Klammern'

>>> di[ 'brackets' ] = 123

>>> di[ 'brackets' ]
123
```

# Dictionaries

Operationen

- schreibender Zugriff auf nicht existierenden Schlüssel

  ==>  neuer Eintrag

```
>>> di[ 'neu' ] = 'neuwert'

>>> len(di)
4
```

# Dictionaries

## Operationen

- Löschen nur im Paar ( Schlüssel/Wert )
- über Schlüssel

```
>>> del di['neu']

>>> len(di)
3

>>> di['neu']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'neu'
```

# Dictionaries

Schreiben Sie eine Funktion neues_dict(l1, l2), die

- zwei Listen gleicher Länge als Argumente nimmt
- ein Dictionary zurückliefert, bei dem
  - die Schlüssel aus der ersten Liste stammen
  - die Werte aus der zweiten Liste stammen

```
l1 = [ 1, 2, 3 ]

l2 = [ 'A', 'B', 'C' ]
```

```
{  1: 'A',
   2: 'B',
   3: 'C'  }
```

# Dictionaries

zip() kombiniert die korrespondierenden Werte von mehreren iterables in Tupeln

- Ergebnis iterable vom Typ *zip*

```
[  ( 1, 'A' ),
   ( 2, 'B' ),
   ( 3, 'C' ) }
```

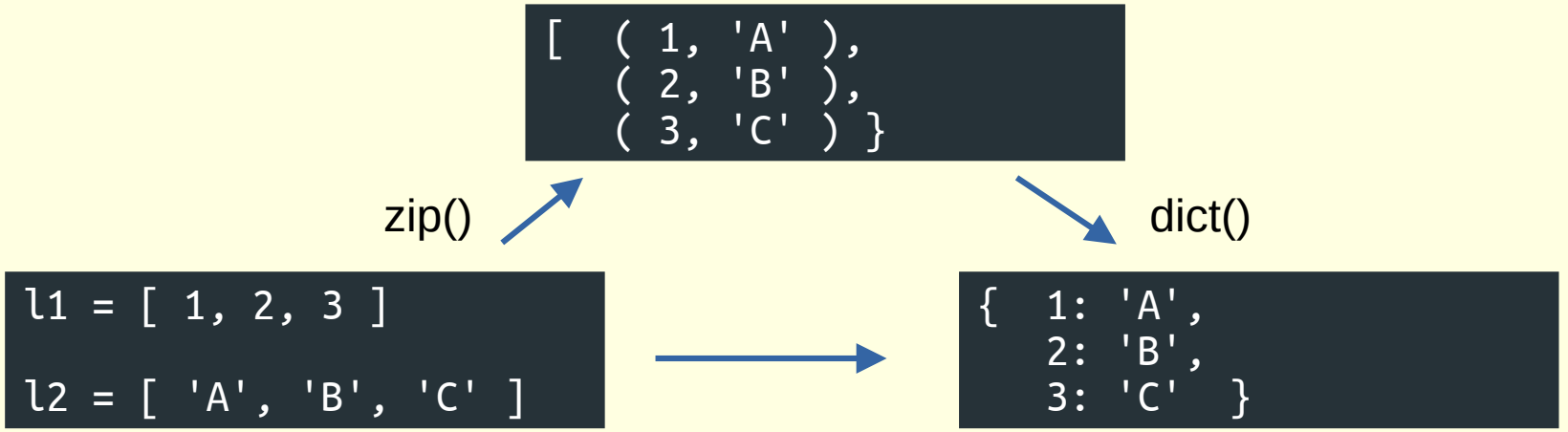
```
l1 = [ 1, 2, 3 ]

l2 = [ 'A', 'B', 'C' ]
```

```
{  1: 'A',
   2: 'B',
   3: 'C'  }
```

# Dictionaries

Konvertierungsfunktion dict() kann Iterable von Paaren konvertieren

dict ( zip ( l1, l2 ) )



# Dictionaries

## Testen auf Schlüssel

- in und not in

```
>>> di = { 'parentheses' : 'runde Klammern', 'brackets' : 'eckige
Klammern', 'curly braces' : 'geschweifte Klammern' }

>>> 'brackets' in di
True

>>> 'brackets' not in di
False
```

# Dictionaries

## Verschmelzen zweier Dictionaries

- Operator |

```
>>> di1 = { 1 : "a", 2 : 'b' }

>>> di2 = { 3 : "c" }

>>> di1 | di2
{1: 'a', 2: 'b', 3: 'c'}
```

# Dictionaries

## Verschmelzen zweier Dictionaries

- Operator |

```
>>> di1 = { 1 : "a", 2 : 'b' }

>>> di2 = { 3 : "c" }

>>> di1 | di2
{1: 'a', 2: 'b', 3: 'c'}

>>> di1 |= di2

>>> di1
{1: 'a', 2: 'b', 3: 'c'}
```

# Dictionaries

dict Comprehension wie list Comp.

- { } statt [ ]
- vorne : weil Paar

```
>>> { a : a+1 for a in range(3) }

{ 0: 1,
  1: 2,
  2: 3 }
```

# Dictionaries

**Aufgabe**

Konvertieren Sie eine Liste von paarweise unterschiedlichen Strings in ein Dictionary mit

- den Strings als Schlüsseln
- den Längen der Strings als Werten.
- Verwenden Sie eine Dict Comprehension

# Dictionaries

## Methoden

di.clear()

- leert das Dictionary

```
>>> di = { 1 : "a", 2 : 'b' }

>>> di.clear()

>>> di
{}
```

# Dictionaries

## Methoden

di.copy()

- liefert eine Kopie von Dictionary

```
>>> di = { 1 : "a", 2 : 'b' }

>>> di2 = di.copy()

>>> di2
{1: 'a', 2: 'b'}
```

# Dictionaries

Methoden

di.update(di2)

- fügt Einträge von di2 zu di hinzu
- Was passiert, wenn ein Schlüssel in beiden vorhanden ist?