Der Datentyp dict

Ein Key Value Speicher

Dict steht für Dictionary

Die zusammengesetzten Datentypen, die wir bisher behandelt haben — Strings und Listen — verwenden **Ganzzahlen als Indizes.**

Der Datentyp **Dictionary** ist den anderen zusammengesetzten Datentypen ähnlich.

Ein Unterschied besteht darin, dass man beliebige unveränderbare Datentypen für die Indizes verwenden kann.

Das Dict ist mutable (veränderbar) und ungeordnet (unsorted), im Gegensatz zur Liste, die zwar veränderbar aber geordnet ist (Indexiert).

Wozu Dictionary? Wir haben doch schon Liste!

Der Datentyp Dict eignet sich hervorragend, um sachlich zusammenhänge Daten an einem Ort zu verwalten.

Die Schlüssel können angesprochen werden. Eine Zugriff über index ist nicht nötig.

Beispiel:

Zum Aufzeichnen von Meßdaten (ein Wert) eignen sich Listen, zum Verwalten von Personen eignen sich dicts.

Es ist durchaus üblich, Listen von dicts zu verwalten.

Hash Map

Das Dictionary ist als Hash Map implementiert und ermöglicht über seinen Key konstante Zugriffszeit auf das entsprechende Element.

Key-Value Datentypen (wie Dict) spielen in der Informationsverarbeitung eine große Rolle.

zb.

DNS (IP => Domainname)

Finanzverarbeitung (Transaktion => Transaktionsdaten)

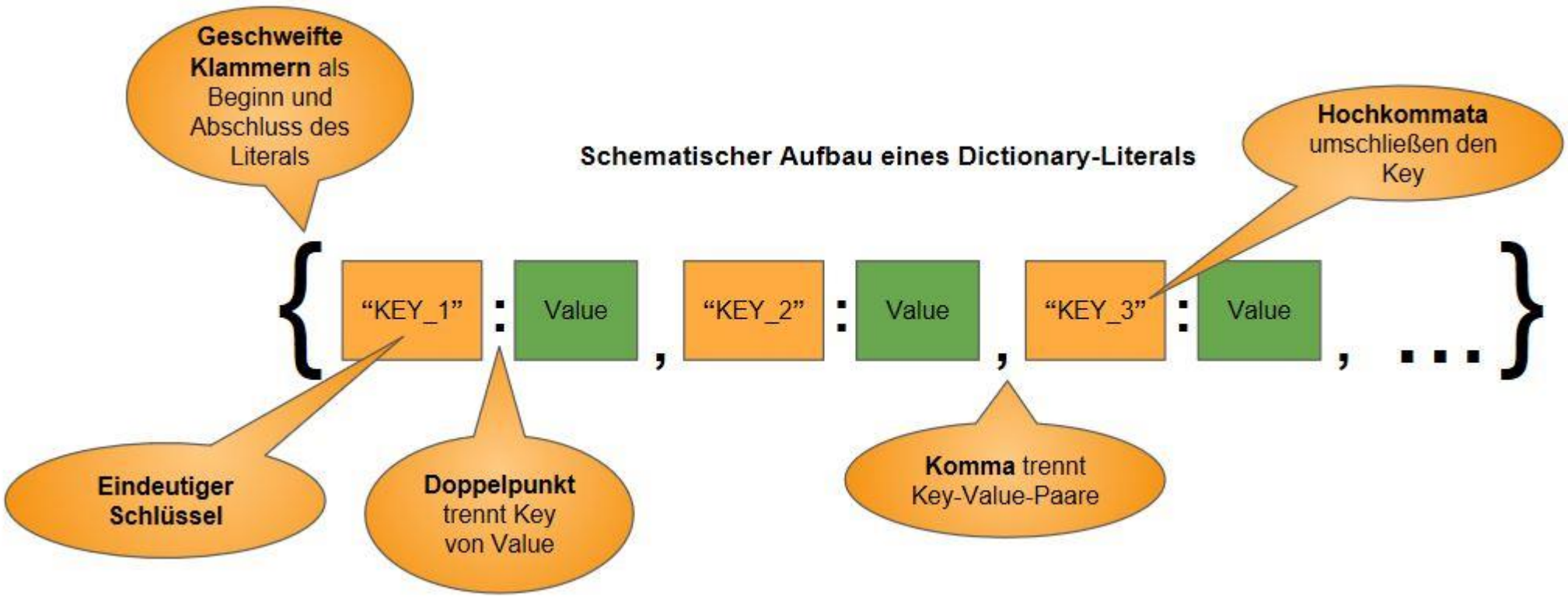
Telefonbuch (Name => Nummer)

Schlüssel-Wert Paare

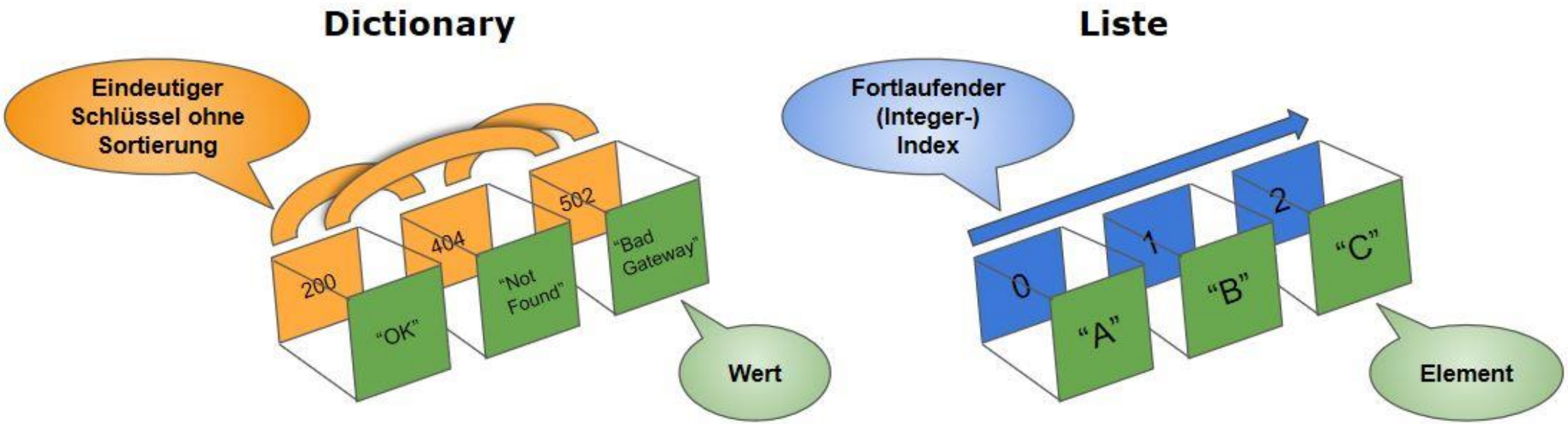
In einem Dictionary nennt man die Indizes **Schlüssel** und daher die Elemente **Schlüssel-Wert Paare**.

```
player = {'life_points': 430, 'power': 1, 'speed': 2}
```

Aufbau eines Dict



Unterschied zur Liste



Dictionary erzeugen

Ein Möglichkeit, ein dict zu erzeugen, ist mit einem leeren Dictionary-Literal zu beginnen und dann Elemente hinzuzufügen.

```
player = {}
player['life_points'] = 300
player['power'] = 2
player['name'] = 'Roger'
Player['coords'] = (4, 4)
```

Schlüssel adressieren

Dictionaries sind Datenstrukturen, auf die über einen Schlüssel zugegriffen wird.

```
a = {
    'x': 1,
    'y': ["das", "ist", "eine", "liste"]
}

value_x = a['x']
value_y = a['y']
```

Oder mit der get-Methode (Vorteil: falls Index nicht existiert, kommt es zu keinem Fehler)

```
value_x = a.get('x')
```

verschachtelte Dictionaries

Dictionaries lassen sich ähnlich wie Listen unbegrenzt verschachteln:

```
cities = {
   "dortmund": {
      "info": {
                  "population": 400000,
                  "state": 'North Rhine-Westphalia"
        },
      },
    "bielefeld": False,
}
```

Cities besitzt zwei Schlüssel (dortmund und bielefeld). Der Schlüssel dortmund besitzt einen Schlüssel info und dieser ist ein dict mit zwei Schlüsseln population und state.

Tupel als Schlüssel

[0,0,0,1,0,0,0],
[0,0,0,0,0,0,0],
[0,1,1,1,0,0,0],
[0,0,0,0,0,0,0],
[1,0,0,0,0,0,1]

m = {(0,3): 1, (2, 1): 1, (2, 2): 1, (2, 3): 1, (4, 0): 1, (4, 6): 1}

Die Dict-Methode get bietet Zugriff auf die Schlüssel. Optional kann ein Default-Wert (0) angegeben werden.

m.get((0,3), 0) (liefert 1)

Dictionary erstellen mit dict()

Aus einer zweidimensionalen Liste lässt sich mit der builtin Funktion dict() ein Dictionary erstellen.

```
alphabet_liste = [
    ["A", "aleph"],
    ["B", "beth"]
]
alphabet = dict(alphabet_liste)
```

Das erste Element einer Sub-Liste fungiert dabei immer als Key, das zweite als Value.

```
alphabet = {
    "A": "aleph", "B": "beth"
}
```

Das funktioniert nur, wenn die Listen in alphabet_liste genau zwei Elementen haben, die von dict() als key und value interpretiert werden können.

Dictionary erstellen mit zip

Zwei Listen lassen sich via der builtin-Funktion zip() und dict() zu einem Dict zusammenführen.

```
countries = ["Deutschland", "Türkei", "USA", "Schweden"]
food = ["Currywurst", "Pilav", "Mais", "Hering"]
world_food = zip(countries, food)
```

zip() erzeugt ein Zip-Objekt (Tupel mit zwei Werten), welches sich via dict() zu einem Dictionary umwandeln lässt. Der erste Parameter dient als Schlüssel, der zweite als Value.

```
world_food = dict(world_food).
```

Kurze Schreibweise für die Aufgabe wäre:

```
world_food = dict(zip(countries, food))
```