

- Wir haben **Dictionaries** (dict) bereits im Zusammenhang mit JSON kennengelernt. Dictionaries gehören zu den assoziativen Datenstrukturen.
- Ein in einer Liste gespeicherter Wert wird durch den Index, eine „nichtssagende“ Zahl, bezeichnet. Dagegen wird in einem Dictionary einem **Wert** ein „sprechender“ **Schlüssel**, z.B. ein String zugeordnet. Wenn man den Schlüssel kennt, kann man ohne Suche auf den Wert zugreifen. Das ist ein riesiger Vorteil.
- Beispiel dict person (aus einer früheren Folie) und zum Vergleich eine Liste lperson:

Schlüssel:	Wert:
name	str
first name	str
appellation	str
street	str
code	int
city	str
active	boolean

```
person = {  
    "name": "Mustermann",  
    "first name": "Max",  
    "appellation": "Herr",  
    "street": "Esplanade",  
    "code": 85049,  
    "city": "Ingolstadt",  
    "active": True  
}
```

Index:	Wert:
0	str
1	str
2	str
3	str
4	int
5	str
6	boolean

```
lperson = [  
    "Mustermann",  
    "Max",  
    "Herr",  
    "Esplanade",  
    85049,  
    "Ingolstadt",  
    True  
]
```

- Dictionaries werden durch geschweifte Klammern gekennzeichnet. Beispiele:

```
termin = {"wann": "07.01.2020", "was": "Vorlesung Programmierung 1"}  
e_dict = { }
```

- Dictionaries können geprintet werden. Beispiel:

```
print(termin)      # Ausgabe: {'wann': '07.01.2020', 'was': 'Vorlesung Programmieren 1'}
```

- Der Zugriff auf einzelne Werte geschieht wieder durch eckige Klammern. Beispiel:

```
person["name"]
```

- Bei Zugriff auf ein Listenelement muss man aufpassen, dass der Index zwischen 0 und Länge – 1 ist. Einen Index-Range wie bei Listen gibt es bei Dictionaries nicht. Deshalb vor dem Zugriff immer prüfen, ob der Schlüssel vorhanden ist. Beispiel:

```
if "name" in person:  
    print(person["name"])
```

- Als Schlüssel werden häufig Strings verwendet. Es sind aber alle unveränderlichen Typen erlaubt.

- Dictionaries selbst sind veränderlich wie Listen! Beispiel:

```
person["name"] = "Musterfrau"  
print(person["name"])
```

```
person["farbe"] = "rot"           # Achtung: farbe war bisher kein Schlüssel. Deshalb wird er neu eingetragen.  
print(person["farbe"])
```

- Durchlaufen kann man ein Dictionary wie im folgenden Beispiel gezeigt:

```
for k, v in person.items():  
    print(k, v)  
# oder  
for k in list(person):  
    print(k, person[k])
```

■ Ausgewählte Funktionen auf Dictionaries:


<code>clear()</code>	# Alle Einträge löschen, z.B. <code>person.clear()</code>
<code>copy()</code>	# Flache Kopie erstellen, z.B. <code>person2 = person.copy()</code>
<code>pop(...)</code>	# Eintrag entfernen und zurückgeben, z.B. <code>f = person2.pop("farbe")</code>
<code>update(...)</code>	# Erweitern, z.B. <code>person.update({"farbe": "rot", "groesse": 180})</code>
<code>items()</code>	# Schlüssel-Werte-Paare zurückgeben, z.B. <code>kv = person.items()</code>
<code>dict.fromkeys(...)</code>	# Ein Dictionary erstellen, z.B. <code>d = dict.fromkeys(["a", "b", "c"], 0)</code>
<code>list(...)</code>	# Eine Liste der Schlüssel zurückgeben, z.B. <code>k = list(person)</code>
<code>len(...)</code>	# Gibt die Länge eines Dictionaries zurück, z.B. <code>n = len(person)</code>
<code>zip(...)</code>	# Ein Dictionary aus zwei Listen erstellen, z.B. <code>d = dict(zip(["a", "b", "c"], [0, 2, 4]))</code> # Die zip-Funktion liefert ein Objekt, das mit dict in ein Dictionary umgewandelt werden kann.

- Aus einem zip-Objekt kann mit list auch eine Liste von Tupeln erstellt werden. Beispiel:

```
t = list(zip(["a", "b", "c"], [0, 2, 4]))
```

Nach einer Umwandlung ist das zip-Objekt allerdings „verbraucht“. Folgendes funktioniert nicht:

```
z = zip(["a", "b", "c"], [0, 2, 4])  
d = dict(z)  
t = list(z)
```

- Wir haben auch bei anderen Datenstrukturen gesehen, dass es Funktionen gibt, bei denen das Objekt vorangestellt wird, z.B. `person.copy()`, und solche, bei denen das Objekt der Parameter ist, z.B. `len(person)`. Das ist ein prinzipieller Schwachpunkt von Python.
-  Kapitel 6, 13.5 in (Klein 2018)

1. Gegeben sei eine Liste von Dictionaries von Personen. Erstellen Sie eine Funktion, die herausfindet, wie viele Personen in der Liste aktiv bzw. nicht aktiv sind.

```
def actives(persons):
```

Parameter: dict persons : Liste von Dictionaries

Rückgabewert: dict : Dictionary mit den Schlüsseln "active", "non active". Die Werte sind ganze Zahlen.

Beispiel:

```
p_list = [ {"name": "Mustermann", "first name": ["Max", "Ludwig"], "appellation": "Herr",  
            "street": ("Esplanade", 10), "code": 85049, "city": "Ingolstadt", "active": True},  
            {"name": "Castro", "first name": ["Enrique", "Max"], "appellation": "Presidente",  
            "street": ("Plaza Major", 1), "code": 8411, "city": "Havanna", "active": False},  
            {"name": "Lopez", "first name": ["Jennifer"],  
            "street": ("Ocean Drive", 6), "city": "Miami", "active": True}]
```

Rückgabewert: {'active': 2, 'non active': 1}

2. „Überlegen Sie sich eine Darstellung des Schachbretts als Dictionary. Die Schlüssel dieses Dictionaries sollen die Felder des Schachbretts sein und die Werte die Information, welche Figur sich auf einem Feld befindet. Das Ergebnis könnte also beispielsweise von der Form ("König", "schwarz") sein. Ein Schachbrett besteht bekanntlich aus 64 Feldern, die in 8 Zeilen und 8 Spalten angeordnet sind. Ein Feld des Schachbretts kann mit einem Tupel bezeichnet werden. Die erste Komponente dieses Tupels entspricht den Spalten des Spielfeldes. Diese werden mit Buchstaben zwischen "a" und "h" von links nach rechts gekennzeichnet. Die zweite Komponente stellt eine Zahl zwischen 1 und 8 dar.“ (Klein 2018, S. 56)

