Beispiel Kontakte (1)

- In diesem Beispiel soll eine Kontaktliste abgespeichert werden.
- Definition eines (vereinfachten) Kontakts und Beispiel eines Kontakts in Python:

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
    }
```

Beispiel Kontakte (2)

- Erstellen der Kontaktliste, Sortieren nach Namen, Durchlaufen der Kontakte und Ausgabe der Namen:

```
kontakte = [ ]
kontakte.append(person)
kontakte.sort(key=lambda x:x["name"])

for p in kontakte:
    if "name" in p:
        print(p["name"])
```

- Abspeichern der Kontaktliste als JSON-Datei mit der Funktion **dump**:

```
import json
# Erstellen und Fuellen von kontakte (siehe oben)

with open("contacts.json", "w") as fh:
    json.dump(kontakte, fh, indent=4)
```

Beispiel Kontakte (3)

- Vergleichen Sie den Inhalt der JSON-Datei contacts.json mit der Konsolausgabe print(kontakte).
  - Die einzigen Unterschiede sind die Formatierung und die Schreibweise true statt True.
- Man kann die Elemente eines Kontakts auch nach den Schlüsseln sortieren:

```
json.dump(kontakte, fh, indent=4, sort_keys=True)
```

- Im Prinzip kann man auch zuerst einen JSON-String mit der Funktion **dumps** erstellen und diesen in eine Textdatei abspeichern:

```
import json
# Erstellen und Fuellen von kontakte (siehe oben)

kontakte_str = json.dumps(kontakte, indent=4)

with open("contacts.json", "w") as fh:
    print(kontakte_str, file=fh)
```

Beispiel Kontakte (4)

- Genauso einfach ist das Laden einer JSON-Datei mit der Funktion **load**:

```
# Erstellen und Fuellen von kontakte und Dumpen in contacts.json (siehe oben)

input("Fuegen Sie einen weiteren Kontakt in die Datei ein.\nDruecken Sie danach die Eingabetaste.")

with open("contacts.json", "r") as fh:
    kontakte = json.load(fh)

for p in kontakte:
    if "name" in p:
        print(p["name"])
```

Beispiel Kontakte (5)

- Natürlich kann man Datenstrukturen auch schachteln. Nehmen wir an, dass first name eine Liste von Vornamen ist:

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

# Erstellen und Fuellen von kontakte und Dumpen in contacts.json (siehe oben)
# Oeffen von contacts.json in einem Texteditor
```

Beispiel Kontakte (6)

- Das einzige Problem ist die Verwendung von Tupel. Sie werden beim Abspeichern in eine Liste umgewandelt. Nehmen wir an, street ist ein Tupel aus Straße und Hausnummer:

```
person = {
        "name": "Mustermann",
        "first name": [ "Max", "Ludwig" ]
        "appellation": "Herr",
        "street": ("Esplanade", 10),
        "code": 85049,
        "city": "Ingolstadt",
        "active": True
    }

# Erstellen und Fuellen von kontakte und Dumpen in contacts.json (siehe oben)
# Oeffen von contacts.json in einem Texteditor
```

Beispiel Kontakte (7)

- Funktionen des Moduls json:

  def dump(obj, fp, indent=None, sort_keys=False)
  Serialize obj as a JSON formatted stream to fp (a .write()-supporting file-like object).
  If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.
  If *sort_keys* is true (default: False), then the output of dictionaries will be sorted by key.

  def dumps(obj, indent=None, sort_keys=False) Serialize obj to a JSON formatted str.

  def load(fp)
  Deserialize fp (a .read()-supporting file-like object containing a JSON document) to a Python object.

  def loads(s)
  Deserialize s (a str, bytes or bytearray instance containing a JSON document) to a Python object.

  Die Funktionen haben weitere optionale Parameter, die hier nicht angegeben sind.

Beispiel Kontakte (8)

Aufgaben

In den beiden folgenden Aufgaben sollen in den Kontakten die Vornamen ("first name") listen und die Straßen ("street") Tupel sein, wie oben definiert.

1. Erstellen Sie eine Funktion, die den ersten Kontakt zurückgibt, der den als Parameter übergebenen Vornamen enthält.

   Dazu muss die Kontaktliste durchsucht werden. Dabei muss für jeden Kontakt die Liste der Vornamen durchsucht werden. Wenn dabei der Eintrag in der Liste der Vornamen gleich dem Parameter fname ist, bricht die Funktion ab und gibt eine Referenz auf den Kontakt zurück.

```
def find_fname(contacts, fname):

Parameter:      list contacts : Kontaktliste
                str fname : Vorname

Rückgabewert:   dict : erster Kontakt in contacts, der fname als Vornamen enthält
```

Beispiel Kontakte (9)

Aufgaben

2. Bei Abspeichern in eine JSON-Datei werden Tupel in Listen umgewandelt. D.h. nach dem Laden hat man Listen statt Tupel.

   Erstellen Sie eine Funktion, die nach dem Laden der Kontakte die Tupel aus Straße und Hausnummer wieder herstellt. Die Kontaktliste wird dabei geändert.

```
def restore_street_tuple(contacts):
Parameter:      list contacts : Kontaktliste
Rückgabewert:   keiner
```

Beispiel:

in kontakte:	"street": ("Esplanade", 10)
in contacts.json nach dem Speichern:	"street": ["Esplanade", 10]
in kontakte nach dem Laden:	"street": ["Esplanade", 10]
in kontakte nach dem Aufruf der Funktion:	"street": ("Esplanade", 10)