

## Datenspeicherung und Datenaustausch über Textdateien

### 1. In eine Textdatei schreiben:

Mit `open()` wird ein Datei-Handler, hier das Objekt `datei` erstellt. Er ermöglicht den Dateizugriff. Es wird die Textdatei erstellt.

```
1 datei = open("Textdatei.txt", "a")
2 datei.write("Hallo, guten Tag")
3 datei.close()
```

einfacher mit with:

**with** arbeitet ähnlich wie eine Kontrollstruktur, das Objekt für den

```
1 with open('Textdatei.txt', 'w') as inhalt:
2     inhalt.write('Guten Tag')
```

Zugriff wird mit **as** definiert. Wenn die Einrückung beendet ist, wird alles automatisch geschlossen (praktisch, wenn mehrere Dateien geöffnet wurden; hinter `with` kann `open()`, durch Komma getrennt, mehrmals aufgerufen werden).

### Parameter von `open()`:

Modus	Auswirkung auf Datei	Position i. d. Datei
r	öffnet die Datei ausschließlich zum Lesen.	Anfang
w	öffnen zum Schreiben dabei löschen aller Inhalte.	Anfang
a	öffnen, neue Inhalte werden an das Ende geschrieben.	<b>Ende</b>
r+	schreiben und lesen, aktualisiert ab der Position bei Aufruf!, der Rest bleibt erhalten	Anfang
w+	schreiben und lesen (Inhalte werden überschrieben)	Anfang

### Anwendung

Das folgende Programm speichert Temperatureingaben mit Datum und Uhrzeit. Bei einer leeren Eingabe wird das Programm beendet.

*temperaturwerte.py*

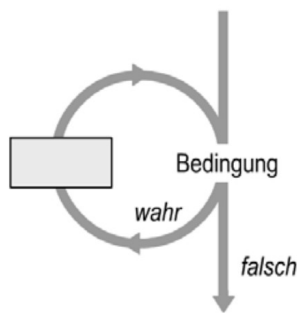
```
1 from time import *
2 with open("Temperaturwerte.txt", "a") as datei:
3     while True:
4         text = input("Temperaturwert: ")
5         if text == "":
6             break
7         datei.write(asctime())
8         datei.write("    "+text+"\n")
```

Anweisungen with - Struktur

Anweisungen while - Schleife

Die while – Schleife wird solange durchlaufen, bis sie mit `break` verlassen wird.

Die Methode *write* erzeugt keinen Zeilenumbruch (im Gegensatz zu *print*), er wird mit `"\n"` hinzugefügt.



### Bedingte Wiederholung – while

Hinter *while* muss eine Bedingung stehen, die wahr oder falsch sein wird,

*while* – Schleifen werden benutzt, wenn die Anzahl der Schleifendurchgänge nicht vorhersehbar ist.

```
>>> x=1
>>> while x < 100:
...     print(x)
...     x = x*2
...
1
2
4
8
16
32
64
```

Im Schleifenrumpf wird der Wert der Schleifenvariable so verändert, dass die Schleifenbedingung schließlich falsch und die Schleife beendet wird. Hier wird der Wert der Variablen *x* immer wieder verdoppelt, solange er noch kleiner als 100 ist.

## 2. Aus einer Textdatei lesen

Datei als Ganzes auslesen:

```
1 with open('Textdatei.txt','r') as datei:
2     print(datei.read())
```

Die Methode *read* liest die gesamte Datei aus und gibt sie zurück – hier wird alles mit *print* ausgegeben.

### Datei zeilenweise auslesen:

```
1 with open('Textdatei.txt','r') as datei:
2     for zeile in datei:
3         print("Inhalt aus Datei:", zeile, end="!")
```

Bei jedem Schleifendurchlauf wird eine Zeile aus der Datei geholt und in die Variable *zeile* geschrieben. Bei der Ausgabe kann es sinnvoll sein, den Zeilenumbruch von *print* zu verhindern, da jede Zeile aus der Datei bereits einen eigenen Zeilenumbruch mitbringt.

### **Aufgabe 1:**

Nachdem im Programm *temperaturwerte.py* alle Werte eingegeben wurden, soll zum Schluß der komplette Inhalt der Datei ausgegeben werde.

Hilfe: Für den Sprung an den Dateianfang dient die Funktion: *seek(0)* ( *datei.seek(0)* ).

## Aufgabe 2:

### Analysieren von Log – Dateien

Beispiel einer Log-Datei: (log.txt) .....

```
Dec 12 13:29:04 vserver pop3d: LOGIN FAILED, user=root, ip=[::ffff:159.120.142.174]
Dec 12 13:29:18 vserver pop3d: LOGIN FAILED, user=root, ip=[::ffff:87.139.68.91]
Dec 12 13:29:18 vserver pop3d: LOGIN FAILED, user=root, ip=[::ffff:59.120.142.74]
Dec 12 13:29:18 vserver pop3d: LOGIN FAILED, user=root, ip=[::ffff:59.120.142.74]
```

a)

Es soll untersucht werden, wie oft versucht wurde, sich in einem bestimmten Monat auf dem Server einzuloggen.

b)

Es soll untersucht werden, wie oft sich eine bestimmte IP-Adresse, **159.120.142.174**, auf dem Server eingeloggt hat. Lassen Sie die Anzahl ausgeben.

Eine Lösung mit count() ist nicht ganz korrekt, weil die beiden Adressen 159.120.142.174 und 59.120.142.17 als gleich angesehen werden würden.

c)

Es sollen alle IP-Adressen **einmal** ausgegeben werden, die sich auf dem Server eingeloggt hatten → die IP-Adresse in jeder Zeile muss dazu separiert werden.

Wenn man sich die Log-Datei genauer anschaut, erkennt man, dass direkt vor der IP-Adresse ein ':' und direkt dahinter ein ']' steht. Ersetzt man ']' durch ':' wird die IP von Doppelpunkten umgeben.

Teilt man nun jede eingelesene Zeile in Teilstrings und nimmt den ':' als Teilungszeichen, ist die IP-Adresse isoliert.

Die Funktion **split()** benötigt:

```
1 daten = "vorname,nachname,alter"
2 einzeldaten = daten.split(",")
3 print(einzeldaten)
```

```
['vorname', 'nachname', 'alter']
```

weitere Möglichkeit die IP-Adresse zu separieren:  
reguläre Ausdrücke

in Python:

**Regular Expression** (RegEx), Modul: re

Suchmuster:

```
:
vier Zahlen durch . getrennt
]
```

### *Lösungsvorschlag Struktogramm zu 2c)*



## Datenaustausch

Für den Datenaustausch zwischen verschiedenen Programmen gibt es semi-strukturierte Datenformate, bei denen die Bedeutung der Information in der Datei mitgeliefert wird.

### 1. Dateiformat **CSV**

– eines der am häufigsten verwendeten Formate zum Darstellen, Importieren und Exportieren von Tabellendaten. CSV wird im Allgemeinen zwischen Programmen verwendet, die normalerweise keine Daten austauschen können. Merkmale von CSV:

- Daten werden schlank und leicht lesbar dargestellt
- jeder Datensatz hat exakt den gleichen Aufbau
- die erste Zeile enthält keine Werte, sondern die **Spaltenbezeichnungen**

#### *Gewachshaus.csv*

```
1 "Anlage", "GwhNr", "Temp1", "Temp2", "Temp3", "luftf"
2 "MA", "2", 23.15, "27.99", "32.22", "52"
3 "HD", "1", "25.15", "21.89", "23.72", "63"
4 "LU", "2", "21.15", "24.17", "31.42", "52"
5 "HD", "3", "25.17", "31.26", "26.19", "60"
```

#### *csv\_auslesen.py*

```
1 import csv
2 with open("Gewachshaus.csv") as datei:
3     csv = csv.DictReader(datei)
4     for zeile in csv:
5         #print(zeile)
6         #print(zeile['Anlage'])
7         print("Neue Zeile:")
8         for schlüssel, wert in zeile.items():
9             print(schlüssel, '->', wert)
```

3:  
Der CSV.DictReader ordnet jedem Wert den passenden Spaltennamen als Schlüssel zu.  
4,5:  
Jeder Datensatz wird mit Schlüsseln und Werten in einem Dictionary (alphanumerischem Array) gespeichert.

6: Die Werte können in der [ ] Klammer angesprochen werden.

8,9: Jede Zeile wird als Schlüssel / Wert- Kombination ausgelesen. *items()* gibt eine Liste der Schlüssel-Wert-Paare zurück.

#### *Ausgabe, Zeile 5,6:*

```
>>> %Run csv_auslesen.py
{'Anlage': 'MA', 'GwhNr': '2', 'Temp1': '23.15', 'Temp2': '27.99', 'Temp3': '32.22', 'luftf': '52'}
```

#### *Ausgabe, Zeile 7,8,9:*

```
Neue Zeile:
Anlage -> HD
GwhNr -> 3
Temp1 -> 25.17
Temp2 -> 31.26
Temp3 -> 26.19
luftf -> 60
```

#### **Aufgabe 3:**

Werten Sie die Datei *Gewachshaus.csv* aus: es soll der Durchschnittswert der Luftfeuchte der Anlagen in Heidelberg (HD) ermittelt werden.

## 2. Datenformat JSON

JavaScript Object Notation. Ähnlich wie CSV ist es ein lesbares Format, mit dem Daten gespeichert werden können. Es lassen sich relativ komplexe Datenstrukturen abbilden, die dabei noch gut lesbar sind.

Alle Informationen liegen als Schlüssel-Wert Paare vor. Der Wert eines Schlüssel-Wert-Paares kann ein einzelner Wert oder wiederum eine Listenart sein.

Syntax:

Allgemein	Objekt	Datentypen
<pre>{   KEY1: VALUE1,   KEY2: VALUE2,   KEY3: VALUE3 }</pre>	<pre>{   "name": "Franz",   "alter": 27,   "hobby": "Python" }</pre>	Werte können folgende Datentypen besitzen: <ul style="list-style-type: none"><li>• int, float, boolean, string</li><li>• Liste mit Werten</li><li>• Objekte</li></ul>
Objekt mit Liste (Array)		Objekt mit verschachteltem Objekt
<pre>{   "name": "Karin",   "alter": 26,   "hobby": ["Python", "Java", "C++"] }</pre>		<pre>{   "name": "Beate",   "alter": 37,   "hobby": "Python"   "kinder": {     "name": "Paul"     "alter": 7   } }</pre>

Exkurs: Liste (numerisches Array)

Dictionary (alphanumerisches Array)

```
1 #List_Dict.py
2 #Aufbau:
3 personal = {'name':'Franz','alter':27}      #Dict
4 pers = ['Franz',27,'Karin', 17, 'Heinz','Franz']  #List
5
6 #Zugriff auf einzelne Werte
7 print(personal['name'])      #Dict
8 print(pers[0])              #List
9
10 #Zugriff auf alle Werte
11 for key, value in personal.items():      #Dict
12     print(key,':',value)
13 for value in pers:                      #List
14     print(value)
```

## JSON – String:

```
1 #json_a.py
2 import json
3 json_str = '''
4 {
5   "name":"Franz",
6   "alter": 27,
7   "hobby":"Python",
8   "kinder": 0
9 }
10 '''
11 json_dict = json.loads(json_str)
12 print(json_dict)
13 print(json_dict["hobby"])
14
15 with open("personal_1.json","w") as datei:
16     datei.write(json.dumps(json_dict, indent=4))
```

11:  
loads() wandelt  
einen JSON-String  
in ein Python-  
Dictionary um.

13:  
Die Felder des  
Dictionarys werden  
nach über die  
Schlüssel des  
JSON-Strings  
angesprochen.

15: dumps()  
wandelt ein  
Python-Objekt  
(Python Dictionary)  
in einen JSON-  
String um.

### Ausgabe

```
>>> %Run json_a.py
{'name': 'Franz', 'alter': 27, 'hobby': 'Python', 'kinder': 0}
Python
```

Die Datei *personal\_1.json* wird eingelesen, alle Schlüssel / Werte werden angezeigt.  
Es wird überprüft, ob der Name ‚Franz‘ in der Datei vorkommt.

```
1 #personal.py
2 import json
3 with open('personal_1.json','r') as datei:
4     json_dict = json.load(datei)
5     for name in json_dict.keys():
6         print(name, json_dict[name])
7         if json_dict[name] == 'Franz':
8             print("Treffer")
```

```
>>> %Run personal.py
name Franz
Treffer
alter 27
hobby Python
kinder 0
```

- 4: load() wandelt eine JSON-Datei in ein Python-Dictionary um.
- 5: in die Variable *name* wird bei jedem Schleifendurchlauf der Key des Python-Dictionary gespeichert.
- 6: der Schlüssel und der entsprechende Wert werden ausgegeben.
- 7,8: es wird überprüft, ob in der JSON-Datei, bzw. im Python-Dictionary, unter dem Schlüssel *name* der Wert *Franz* existiert.

## JSON – String mit 2 Objekten und einer Liste

```
1 #json_b.py
2 import json
3 element = '''
4 [
5 {
6     "name": "Franz",
7     "alter": 27,
8     "hobby": "Python",
9     "kinder": 0
10 },
11 {
12     "name": "Karin",
13     "alter": 25,
14     "hobby": ["Python","Java","C++"]
15 }
16 ]
17 '''
18 json_str = json.loads(element)
19 print(json_str)
20 print(json_str[1]['hobby'][1])
21
22 with open('personal_2.json','w') as datei:
23     datei.write(json.dumps(json_str, indent=4))
```

### Ausgabe

```
>>> %Run json_b.py
[{'name': 'Franz', 'alter': 27, 'hobby': 'Python', 'kinder': 0}, {'name': 'Karin', 'alter': 25, 'hobby': ['Python', 'Java', 'C++']}]
Java
```

Zeile 4 und 16: Da es mehrere Objekt sind, werden sie in einer Liste gespeichert.

Zeile 20: Die Liste mit den zwei Objekten wird angesprochen mit: [0] oder [1]. Die Schlüssel / Werte werden in einem Dict gespeichert und mit z. B. ['hobby'] angesprochen. Gibt es mehr als 1 Hobby, stehen diese in einer Liste.

Die Datei *personal\_2.json* wird eingelesen, es werden alle Namen der Personen ausgegeben, die als Hobby 'Python' haben.



```

1 #personal_2.py
2 import json
3 hobby = []
4 with open('personal_2.json','r') as datei:
5     json_dict = json.load(datei)
6     for i in range(len(json_dict)):
7         hobby.append(json_dict[i]['hobby'])
8         if 'Python' in hobby:
9             print(json_dict[i]['name'])
10    hobby.clear()

```

3:  
Eine leere Liste wird  
angelegt.

7:  
Von jedem Objekt  
werden alle Hobbys in  
diese Liste  
gespeichert.

8,9:  
Wenn in der Liste  
*hobby* der Wert

Python vorkommt, wird der Schlüssel *name* ausgelesen.

10: Nach jedem Schleifendurchlauf werden die Inhalte der Liste *hobby* gelöscht.

JSON – String mit  
verschachteltem Objekt

```

1 #json_c.py
2 import json
3 element = '''
4 [
5 {
6     "name": "Franz",
7     "alter": 27,
8     "hobby": "Python",
9     "kinder": 0
10 },
11 {
12     "name": "Karin",
13     "alter": 25,
14     "hobby": ["Python","Java","C++"],
15     "kinder": 0
16 },
17 {
18     "name": "Beate",
19     "alter": 37,
20     "hobby": "Node-Red",
21     "kinder": {
22         "name": "Paul",
23         "alter": 7
24     }
25 }
26 ]
27 ...
28 json_str = json.loads(element)
29 #print(json_str)
30 print(json_str[2]['kinder']['alter'])

```

↑  
**Liste**

↑  
**Dict**

↑  
**Dict**

*Ausgabe*

```

>>> %Run json_c.py
7

```



## JSON – String mit Liste mit verschachtelten Objekten

```
1 #json_d.py
2 import json
3 element = '''
4 [
5     {
6         "name": "Franz",
7         "alter": 27,
8         "hobby": "Python",
9         "kinder": 0
10    },
11    {
12        "name": "Karin",
13        "alter": 25,
14        "hobby": ["Python","Java","C++"],
15        "kinder": 0
16    },
17    {
18        "name": "Beate",
19        "alter": 37,
20        "hobby": "Node-Red",
21        "kinder":
22            [
23                {
24                    "name": "Paul",
25                    "alter": 7
26                },
27                {
28                    "name": "Kira",
29                    "alter": 4
30                }
31            ]
32    }
33 ]
34 ...
35 json_str = json.loads(element)
36 #print(json_str)
37 print(json_str[2]['kinder'][1]['alter'])
```

Liste

Dict

Liste

Dict

*Ausgabe*

```
>>> %Run json_d.py
4
```

```
1 #personal_4.py
2 import json
3 anzahl = 0
4 with open('personal_4.json','r') as datei:
5     json_dict = json.load(datei)
6     for i in range(len(json_dict)):
7         if json_dict[i]['kinder'] != 0:
8
9             for i in range(len(json_dict[i]['kinder'])):
10                 print(json_dict[i]['name'])
11                 if json_dict[i]['name'] !=0:
12                     anzahl +=1
13 print(anzahl)
```

Die Datei *personal\_4.json* wird eingelesen, es werden alle Namen der Personen ausgegeben, die Kinder haben. Die Anzahl der Kinder wird gezählt.

*Ausgabe*

```
>>> %Run personal_4.py
Franz
Karin
2
```

zeiterfassung.json

```

1  {
2      "UserAccount":
3      {
4          "persid": 34,
5          "vorname": "Max",
6          "nachname": "Musterhaus",
7          "lastUpdate": "2020-12-02 15:12:22",
8          "LogEntries":
9          {
10
11              "LogEntry": [
12                  {
13                      "id": 23,
14                      "Type": "kommt",
15                      "Time": "2022-12-02 7:12:22",
16                      "Host": "10.34.154.65"
17                  },
18                  {
19                      "id": 25,
20                      "Type": "Dienstgang geht",
21                      "Time": "2022-12-02 9:32:22",
22                      "Host": "10.34.154.65"
23                  },
24                  {
25                      "id": 26,
26                      "Type": "Dienstgang kommt",
27                      "Time": "2022-12-02 17:32:22",
28                      "Host": "10.34.154.65"
29                  },
30                  {
31                      "id": 27,
32                      "Type": "geht",
33                      "Time": "2022-12-02 18:37:22",
34                      "Host": "10.34.154.65"
35                  }
36              ]
37          }
38      }
39  }

```

**Aufgabe 4:**

- Lassen Sie sich die Struktur der obigen JSON-Datei in Firefox anzeigen.
- Es soll herausgefunden werden, wann der User Max (Vorname) lt. der JSON-Datei mit seiner Arbeit begonnen hat (erster Log-Eintrag). Ergänzen Sie dazu den unteren Quellcode.

```

1  #zeiterfassung.py
2  import json
3  anzahl = 0
4  with open('zeiterfassung.json', 'r') as datei:
5      json_list = json.load(datei)
6      print(json_list['UserAccount']['LogEntries']['LogEntry'][2]['Host'])
7      if json_list['UserAccount']['vorname'] == 'Max':

```

```

>>> %Run zeiterfassung.py
10.34.154.65

```

## JSON – Dateien über eine API

Daten, die von Servern über API's (i.d.R. REST-API) zur Verfügung gestellt werden, besitzen meist das JSON-Format.

*http\_url\_weatherapi\_abfrage.py*

```
1 import requests
2 import json
3 import pandas
4
5 #URL='https://api.openweathermap.org/data/2.5/weather?lat=49.487457&lon=8.466040&appid=963325ea0e6f';
6 URL = "https://api.open-meteo.com/v1/forecast?latitude=49.49&longitude=8.47&hourly=temperature_2m"
7 # GET Anfrage an Webserver findet statt, alle Informationen werden in
8 # die Variable (Objekt) response gespeichert
9 response = requests.get(URL)
10 # Antwort vom Server wird in String umgewandelt
11 responseText = response.text
12 # String wird in json bzw. dictionary umgewandelt
13 parsedJson = json.loads(responseText)
14 print(parsedJson)
```

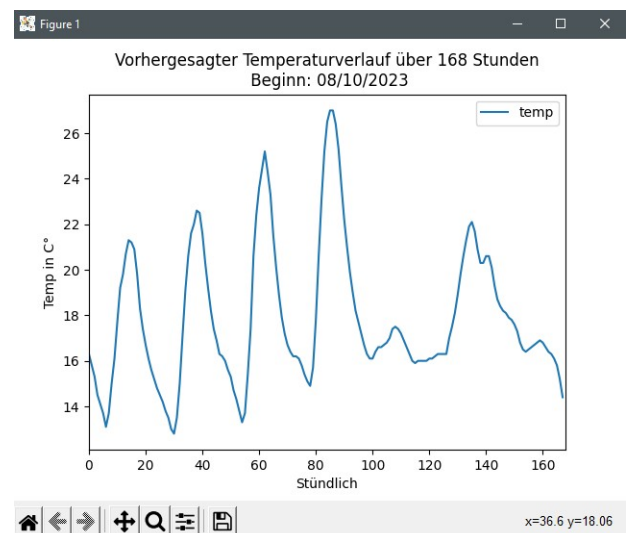
*http\_url\_weatherapi\_abfrage.py*

```
>>> %Run http_url_weatherapi_abfrage.py
      time      temp
0    2023-10-08T00:00  16.3
1    2023-10-08T01:00  15.8
2    2023-10-08T02:00  15.3
3    2023-10-08T03:00  14.5
4    2023-10-08T04:00  14.1
..
163  2023-10-14T19:00  16.3
164  2023-10-14T20:00  16.1
165  2023-10-14T21:00  15.8
166  2023-10-14T22:00  15.2
167  2023-10-14T23:00  14.4

[168 rows x 2 columns]
```

*Darstellung der Daten als Datenframe  
mit Hilfe des Moduls pandas*

*wettervorhersage.py*



*Visualisierung der Daten mit Hilfe  
des Moduls matplotlib*

### Aufgabe 5:

Ermitteln Sie aus der JSON-Datei vom Webservice <https://api.open-meteo.com> die vorhergesagte Temperatur für die nächsten Tag um 13.00 Uhr.

## XML

Auch mit XML kann man Daten zusammen mit der Beschreibung der Daten übertragen, es ist, wie JSON, ein textbasiertes Datenformat, welches aus Tags besteht, die zwischen spitzen Klammern < > stehen.

Beispiel:

```
1 #xml_test.py
2 import xml.etree.ElementTree as ET
3 data = '''
4 <person>
5     <name>Franz</name>
6     <alter>27</alter>
7     <hobby>Python</hobby>
8     <kinder>0</kinder>
9 </person>'''
10
11 tree = ET.fromstring(data)
12 print('Name:', tree.find('name').text)
13 print('Alter:', tree.find('alter').text)
14 print('Hobby:', tree.find('hobby').text)
15 print('Kinder:', tree.find('kinder').text)
```

```
>>> %Run xml_test.py
```

```
Name: Franz
Alter: 27
Hobby: Python
Kinder: 0
```

## Gegenüberstellung JSON – XML-Datei

```
1 [
2   {
3     "name": "Franz",
4     "alter": 27,
5     "hobby": "Python",
6     "kinder": 0
7   },
8   {
9     "name": "Karin",
10    "alter": 25,
11    "hobby": ["Python", "Java", "C++"],
12    "kinder": 0
13  }
14 ]
```

```
<personen>
  <0>
    <name>Franz</name>
    <alter>27</alter>
    <hobby>Python</hobby>
    <kinder>0</kinder>
  </0>
  <1>
    <name>Karin</name>
    <alter>25</alter>
    <hobby>Python</hobby>
    <hobby>Java</hobby>
    <hobby>C++</hobby>
    <kinder>0</kinder>
  </1>
</personen>
```

```

1  [
2  {
3      "name": "Franz",
4      "alter": 27,
5      "hobby": "Python",
6      "kinder": 0
7  },
8  {
9      "name": "Karin",
10     "alter": 25,
11     "hobby": ["Python", "Java", "C++"],
12     "kinder": 0
13 },
14 {
15     "name": "Beate",
16     "alter": 37,
17     "hobby": "Node-Red",
18     "kinder": {
19         "name": "Paul",
20         "alter": 7
21     }
22 }
23 ]

```

```

1  <personen>
2  <0>
3      <name>Franz</name>
4      <alter>27</alter>
5      <hobby>Python</hobby>
6      <kinder>0</kinder>
7  </0>
8  <1>
9      <name>Karin</name>
10     <alter>25</alter>
11     <hobby>Python</hobby>
12     <hobby>Java</hobby>
13     <hobby>C++</hobby>
14     <kinder>0</kinder>
15 </1>
16 <2>
17     <name>Beate</name>
18     <alter>37</alter>
19     <hobby>Node-Red</hobby>
20     <kinder>
21         <name>Paul</name>
22         <alter>7</alter>
23     </kinder>
24 </2>
25 </personen>

```

```

1  [
2  {
3      "name": "Franz",
4      "alter": 27,
5      "hobby": "Python",
6      "kinder": 0
7  },
8  {
9      "name": "Karin",
10     "alter": 25,
11     "hobby": ["Python", "Java", "C++"],
12     "kinder": 0
13 },
14 {
15     "name": "Beate",
16     "alter": 37,
17     "hobby": "Node-Red",
18     "kinder":
19         [
20             {
21                 "name": "Paul",
22                 "alter": 7
23             },
24             {
25                 "name": "Kira",
26                 "alter": 4
27             }
28         ]
29 }
30 ]

```

```

1  <personen>
2  <0>
3      <name>Franz</name>
4      <alter>27</alter>
5      <hobby>Python</hobby>
6      <kinder>0</kinder>
7  </0>
8  <1>
9      <name>Karin</name>
10     <alter>25</alter>
11     <hobby>Python</hobby>
12     <hobby>Java</hobby>
13     <hobby>C++</hobby>
14     <kinder>0</kinder>
15 </1>
16 <2>
17     <name>Beate</name>
18     <alter>37</alter>
19     <hobby>Node-Red</hobby>
20     <kinder>
21         <name>Paul</name>
22         <alter>7</alter>
23     </kinder>
24     <kinder>
25         <name>Kira</name>
26         <alter>4</alter>
27     </kinder>
28 </2>
29 </personen>

```