LFCX:Datenaustauschformate

[→ ZP:Sheet:2]:

Die Frage nach Daten-(Datei)-Austauschformaten taucht in der Ausbildung zur Fachinformatikerin an verschiedenen Stelle auf. Es geht dabei um Erfassung strukturierter Daten in Dateien - insbesondere im CSV-, INI-, JSON-, XML- und YAML-Format.

Die erläutern wir anhand derselben Referenzdaten:

1.) Referenzdaten *([→ ZP:Sheet:3])*

Messdatum:	2025-03-01	2025-03-02	2025-03-03	2025-03-04
Systole	182	158	179	170
Puls	66	75	67	78

2.) CSV *([→ ZP:Sheet:4])*

= Comma-separated values

Charakteristika:

- Zeilen bilden einen Datensatz = Messung.
- Ein Datensatz besteht aus Datenfeldern.
- Datenwerte können - abgesehen von den Sonderzeichen - beliebige Zeichen enthalten.
- Der erste Datensatz **kann** ein Kopfdatensatz sein, der die Spaltennamen definiert
- Gut geeignet zur Erfassung von sich wiederholenden Messdatensätzen

Sonderzeichen:

1. ***Datensatz****trenner* = meistens `LF` (LNX) bzw. `CR LF` (WIN)
2. ***Datenfeld****trenner* = oft `,` (andere sind möglich, etwa [`;`, `:`, `TAB`, , …])
3. ***Datenfeld****begrenzer* = Anführungszeichen vorne und hinten ‘

Beispiel:

```
Datum,Systole,Puls
2025-03-01, 182, 66
2025-03-02, 158, 75
2025-03-03, 179, 67
2025-03-04, 170, 78
```

vgl.

- https://de.wikipedia.org/wiki/CSV_(Dateiformat)
- https://en.wikipedia.org/wiki/Comma-separated_values

3.) INI *([→ ZP:Sheet:5])*

= Initialisierungsdatei

- von Microsoft für die Windowskonfiguration eingeführt
- von vielen anderen ‘Systemen’ zu vielen anderen Zwecken übernommen - gelegentlich mit veränderter Syntax

Charakteristika:

- Jede Zeile ist
  - ein Schlüsselwertpaar oder
  - eine Sektionsüberschrift
- Ein Schlüssel ist ein String ohne innere Leerzeichen (oder andere Whitespacezeichen)
- Ein Wert ist String ohne Blanks oder ein mit Anführungszeichen begrenzter String mit Blanks
- Gut geeignet zur Erfassung / Strukturierung von einem Datensatz zur Konfiguration
- Nur mit Verrenkungen geeignet für sich wiederholende Messungen

Sonderzeichen:

1. ***Konfigurations****trenner* = meistens `LF` (LNX) bzw. `CR LF` (WIN) (könnte auch ein Blank sein)
2. `[` und `]` zur Markierung einer Sektionsüberschrift
3. ***Wer****begrenzer* = Anführungszeichen vorne und hinten
4. `;` als Kommentareinleiter

Semantische Regeln:

- Sektionsbezeichner müssen unique sein.
- In einer Sektion müssen die Schlüssel unique sein.
- Kommentare dürfen nur auf separaten Zeilen
- Groß- und Kleinschreibung nicht unterschieden

Beispiel:

```
[ Blutdruck ]
Systole=mmHg
Diastole=mmHg
; 'mmHg' = Millimeter Quecksilbersäule
[ Puls]
Frequenz=bpm
; 'bpm' = Nits per Minute
```

versus

```
[2025-03-01]
Systole=182
Puls=66
[2025-03-02]
Systole=158
Puls=75
[2025-03-03]
Systole=179
Puls=67
[2025-03-04]
Systole=170
Puls=78
```

Hinweis: Sektionen dürfen üblicherweise NICHT verschachtelt sein.

vgl. https://de.wikipedia.org/wiki/Initialisierungsdatei

4.) JSON *([→ ZP:Sheet:6])*

= JavaScript Object Notation

Charakteristika:

- kennt die Elemente
  - **`null`** (Nullwerte)
  - **`true`** bzw. **`false`** (Boolsche Werte)
  - [+,-] [0-9] [0-9]* (`.` [0-9] [0-9]*)
  - (Unicode) *Zeichenkette* mit oder ohne Escapesequenzen und eingefasst mit `"`
  - *Array* beginnend mit `[` und endend mit `]`.
  - *Object* beginnend mit `{` und endend mit `}`.
- besteht aus einem Schlüsselwertpaar mit innerem Doppelpunkt: `Schlüssel : Wert`
  - *Schlüssel* ist eine Zeichenkette
  - *Wert* ist ein beliebiges Element

- Aufeinander folgende Schüsselwertpaare werden durch Kommata abgetrennt.
- Whitespaces habe keine Bedeutung und können beliebig oft eingeschoben werden.

Sonderzeichen: `:`, `"`, \, [und], { und }

Semantische Regeln:

- Schlüssel müssen unique sein

Beispiel:

```
[
   { "2025-03-01":{
     "Systole": 182,
     "Puls": 66
     } } ,
   { "2025-03-02":{
     "Systole": 158,
     "Puls": 75
     } } ,
   { "2025-03-03":{
     "Systole": 179,
     "Puls": 67
     } } ,
   { "2025-03-04":{
     "Systole": 170,
     "Puls": 78
    } }
]
```

- vgl. https://de.wikipedia.org/wiki/JSON

Disclaimer: ***([→ ZP:Sheet:7])***

- *JSON* ist in der 2. Version aus dem Dezember 2017 als `JSON data interchange syntax` von der ECMA (= *international industry association for standardization of information and communication systems*) unter der Nummer 404 als `JSON data interchange syntax` standardisiert. Das zugehörige Standardisierungsdokument ist als PDF frei herunterladbar.
  - https://ecma-international.org/publications-and-standards/standards/ecma-404/
  - https://ecma-international.org/wp-content/uploads/ECMA-404_2nd_edition_december_2017.pdf
  - https://ecma-international.org/
- Die JSON-Homepage bietet selbst eine darauf aufbauende Syntaxspezifikation in Form einer visualisierten Backus-Naur-Form
  - https://www.json.org/json-de.html

  - https://de.wikipedia.org/wiki/Backus-Naur-Form
  - https://de.wikipedia.org/wiki/Erweiterte_Backus-Naur-Form

- Hier eine kondensierte Version als EBNF-Spezifikation:

```
(R1) json = element ;
(R2) element = ws value ws ;
(R3) value = object | array | string | number | 'true' | 'false' | '
   null';
(R4) object = '{' ws '}' | '{' ws members ws '}';
(R5) members = member | member ',' members;
(R6) member = ws string ws ':' element;
(R7) array = '[' ws ']' | '[' elements ']';
(R8) ws = '' | '0020' ws | '000A' ws | '000D' ws | '0009' ws;
```

Aus R1 - R3 folgt, dass jede Datei, die mit einem korrekt notierten Object, Array etc. beginnt, eine korrekte JSON-Datei ist.

Gelegentlich hört man trotzdem, eine JSON-Datei müsse mit einer geschweiften Klammer beginnen. Das ist falsch.

ÜBUNG **CRX:Datenaustauschformate:01****

☐ Welche der folgenden JSON-Dateien ist nicht 'wohlgeformt' und warum?

1. { }
2. [ ]
3. "stimme": "hoch"
4. true

Lösung:

- [1] folgt aus R1 + R2 + R3 + R4 + R8
- [2] folgt aus R1 + R2 + R3 + R7 + R8
- [3] keine Ableitung
- [4] folgt aus R1 + R2 + R3

ÜBUNG **CRX:Datenaustauschformate:02****

- ☐ Die ‘kondensierte’ EBNF auf [→ ZP:Sheet:7] ist unvollständig. Für welches zentrale Syntagma habe ich die Ersetzungsregel “vergessen”?
- ☐ Sehen Sie unter https://www.json.org/json-de.html nach, wie die Regel heißen müsste.

Lösung:

- Für das Syntagma *elements* in `array = '['ws ']'| '['elements ']';` gibt es keine Ersetzungsregel.
- Dort findet man die Regel `elements = element | element ','elements`

5.) XML *([→ ZP:Sheet:8])*

= Extensible Markup Language

Charakteristika:

- Jeder Wert wird durch ein öffnendes und ein schließendes Tag ausgezeichnet
  - Jedes Tag beginnt mit einem < und endet mit einem >
  - Beim schließenden Tag folgt auf das öffnende < ein Backslash /
  - Der eigentliche Text innerhalb eines Tags beginnt mit einem Zeichen, gefolgt von beliebig vielen Zeichen und Zahlen.
  - Der eigentliche Text innerhalb eines Tags enthält keine WhiteSpace-Zeichen.
  - Der eigentliche Text innerhalb des öffnenden und schließendes Tags muss gleich sein.
  - Jede XML-Datei beginnt mit <?`xml version="1.1"encoding="UTF9"`>.
  - Danach folgen beliebig viele getaggte und verschachtelte Werte
  - Vor und nach einem Tag können beliebig viele Whitespaces folgen.
  - Whitespaces zwischen den Tags gehören zum Wert. Es ist gute Tradition, Strings trotzdem mit Hochkommata zusammenzufassen.

Sonderzeichen: * & * < * > * "

Beispiel:

```
<?xml version="1.1" encoding="UTF9">
<DemoMessReihe>
   <D20250301>
     <Systole>182</Systole>
     <Puls>66</Puls>
   </D20250301>
   <D20250302>
```

```
      <Systole>158</Systole>
      <Puls>75</Puls>
    </D20250302>
    <D20250303>
      <Systole>179</Systole>
      <Puls>67</Puls>
    </D20250303>
    <D20250304>
      <Systole>170</Systole>
      <Puls>78</Puls>
    </D20250304>
</DemoMessReihe>
```

Anmerkung: Welche Tags und welche Typen der Werte in einem XML-Dateityp erlaubt sind, werden durch externe 'Doc-Type-Definition'-Dateien (DTD) oder xml-Schema-Dateien bestimmt.

zZ den vielen anderen Detailanforderungen/Möglichkeiten vgl.

- https://de.wikipedia.org/wiki/Extensible_Markup_Language
- Robert Eckstein: XML Pocket Reference, O'Reily, 2001
- u. viele andere neuere Werke

6. YAML *([→ ZP:Sheet:9])*

steht für *YAML Ain't Markup Language* (ursprünglich: *Yet Another Markup Language*))

Charakteristika:

- besteht aus Blöcken, jeweils abgetrennt durch ---
- Hat die Datei nur einen Block, werden die Trenner weggelassen
- Ein Block kann 1-n Schlüssel-Wert-Paare enthalten:
  - Schlüssel und Wert sind durch ':' getrennt
  - Ein Block kann in einer Zeile geschrieben werden:
  - Diese Zeile beginnt mit { und endet mit }
  - Die Schlüsselwert-Paare werden dann durch Kommata getrennt.
- Ein Block kann eine assoziative Liste enthalten

Sonderzeichen:

vgl. https://de.wikipedia.org/wiki/YAML

Beispiel:

```yaml

- D20250301: {Systole: 182, Puls: 66}
- D20250302:
   Systole: 158
   Puls: 75
---
Messreihe-2:
  - D20250303: {Systole: 179, Puls: 67}
  - D20250304:
      Systole: 170
      Puls: 78
---
```

Hinweis: * YAML-Dateien werden oft als Konfigurationsdazteien im Cloud-Kontext verwendet. * Waren gedacht als Vereinfachung der JSON-Syntax * Ist aber durch die Variantenvielfalt schwer einheitlich / projektübergreifend zu nutzen.

vgl. https://de.wikipedia.org/wiki/YAML#Kritik