Beispiele für Texte mit Auszeichnungen

"CSV" steht für "Character Separated Values" (nicht "Comma Separated Values"!).

Beispiel-01: Daten zu einer Person in einem CSV-Format (Trennzeichen: Komma):

```
1 Jonas, Karl, 182, 85, 714430
```

Beschrieben wird hier Karl Jonas, der 182 cm groß ist, 85 kg wiegt und sich an der BHT unter der Nr. 714430 immatrikuliert hat.

Curl-Beispiele aus http://en.wikipedia.org/wiki/Curl_(programming_language)

Beispiel-02: Ein Applet in der Auszeichnungs- und Programmiersprache Curl:

```
1 {Curl 5.0 applet}
2 {applet license="development"}
3 {text
4    color = "blue",
5    font-size = 16pt,
6    Hello World}
```

Beispiel-03: Noch ein Curl-Text/Programm

```
1 {paragraph
2    paragraph-left-indent=0.5in,
3    {text color = "red", font-size = 12pt,
4      Hello}
5    {text color = "green", font-size = 12pt,
6      World}}
```

Anmerkung: Curl ist eine *homoikonische* Programmiersprache. D.h. jedes Curl-*Programm* ist ein *Wert* eines bestimmten Curl-Datentyps. Somit ist Curl eine Programmiersprache und gleichzeitig ein Datenformat. Lisp war die erste homoikonische Programmiersprache.

YAML-Beispiele aus http://en.wikipedia.org/wiki/Yaml

Grundkonzepte von YAML: *Listen*, *assoziative Listen* und *Einzelwerte*. `---` markiert (innerhalb eines YAML-Dokuments) den Beginn eines neuen Dokuments, mit # beginnt ein Kommentar bis zum Zeilenende, mit `-` beginnen die Elemente einer Liste, `:` trennt in einer assoz. Liste einen Schlüssel von seinem Wert, `|` bedeutet: Zeilenumbrüche beibehalten, `>` bedeutet: Zeilenumbrüche nicht beibehalten.

Beispiel-04: Zwei Listen in YAML

```
1 --- # Favorite movies, block format
2 - Casablanca
3 - Spellbound
4 - Notorious
5 --- # Shopping list, inline format
6 [milk, bread, eggs]
```

Beispiel-05: Zwei *Abbildungen* (engl. maps or hashes) in YAML:

```
1 --- # Block
2 name: John Smith
3 age: 33
4 --- # Inline
5 {name: John Smith, age: 33}
```

Beispiel-06: Ein größeres YAML-Beispiel
(eine *Abbildung* mit 7 Schlüsseln `receipt:` bis `specialDelivery:`):

```
 1 ---
 2 receipt:     Oz-Ware Purchase Invoice
 3 date:        2007-08-06
 4 customer:
 5     given:   Dorothy
 6     family:  Gale
 7
 8 items:
 9     - part_no:   A4786
10       descrip:   Water Bucket (Filled)
11       price:     1.47
12       quantity:  4
13
14     - part_no:   E1628
15       descrip:   High Heeled "Ruby" Slippers
16       price:     100.27
17       quantity:  1
18
19 bill-to:  &id001
20     street: |
21             123 Tornado Alley
22             Suite 16
23     city:   East Westville
24     state:  KS
25
26 ship-to:  *id001
27
28 specialDelivery:  >
29     Follow the Yellow Brick Road to the Emerald City.
30     Pay no attention to the man behind the curtain.
31 ...
```

Der Wert des Schlüssels `items` ist eine *Liste* mit zwei Elementen, die *Abbildungen* mit je 4 Schlüsseln sind. In Zeile 19 wird der Bezeichner `id001` definiert und in Zeile 26 angewendet (statt die Zeilen 20 bis 24 zu kopieren).

YAML passt besonders gut zu den Programmiersprachen Perl, Python, PHP, Ruby, Java und C#, die mit ganz ähnlichen Datenstrukturen (Reihungen und Abbildungen) arbeiten. YAML ist eine *Obermenge* von JSON (jedes wohlgeformte JSON-Dokument ist auch ein wohlgeformtes YAML-Dokument).

Die Sprache JSON (Java Script Object Notation) wurde zusammen mit der Sprache JavaScript entwickelt, um Objekte (von objektorientierten Programmiersprachen) kompakt und doch lesbar darzustellen. Sie kann auch ganz unabhängig von JavaScript als allgemeine Auszeichnungssprache benutzt werden. Ist heute ein wichtiger Bestandteil von JavaFX.

JSON-Beispiel aus http://de.wikipedia.org/wiki/JSON

Beispiel-07: Ein Objekt, in JSON dargestellt:

```
 1 {
 2   "Kreditkarte"   : "Xema",
 3   "Nummer"        : "1234-5678-9012-3456",
 4   "Inhaber"       : {
 5     "Name"        : "Reich",
 6     "Vorname"     : "Rainer",
 7     "Geschlecht"  : "\"männlich\"",
 8     "Vorlieben"   : [
 9       "Reiten",
10       "Schwimmen",
11       "Lesen"
12     ],
13     "Alter"       : null
14   },
15   "Deckung"       : 1e+6,
16   "Währung"       : "EURO"
17 }
```

Ein Objekt `{ ... }` kann eine *ungeordnete* Liste von *Eigenschaften* enthalten.
Eine Reihung `[ ... ]` kann eine *geordnete* Liste von *Werten* enthalten.
Eine Eigenschaft `... : ....` besteht aus einem Schlüssel (Zeichenkette) und einem Wert.
Ein Wert ist eins von: Objekt, Reihung, Zeichenkette, Zahl, true, false, null.

Beispiel-08: Struktur einer HTML-Seite

```
1 <html>
2     <head>
3         <title>Titel der Webseite</title>
4         <!-- Evtl. weitere Kopfinformationen -->
5     </head>
6     <body>
7         Inhalt der Webseite
8     </body>
9 </html>
```

Beispiel-09: Ein besonders kleines und einfaches XML-Dokument:

```
1 <name>
2   <vor_name>Anna</vor_name>
3   <nach_name>Blume</nach_name>
4 </name>
```

Dieses XML-Dokument enthält ein `name`-Element, welches ein `vor_name`- und ein `nach_name`-Element enthält.

Beispiel-10: Ein etwas komplizierteres XML-Dokument:

```
 1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
 2
 3 <person>
 4   <name>
 5     <vor_name> Alan  </vor_name>
 6     <nach_name>Turing</nach_name>
 7   </name>
 8   <beruf>Informatiker</beruf>
 9   <beruf>Mathematiker</beruf>
10   <beruf land="England" zeit="WK2">Kryptograf</beruf>
11 </person>
```

Dieses XML-Dokument enthält eine XML-Deklaration (in Zeile 1) und ein `person`-Element (in den Zeilen 3 bis 11), welches ein `name`-Element und drei `beruf`-Elemente enthält. Das dritte `beruf`-Element (in Zeile 10) hat zwei Attribute namens `land` und `zeit`.

Beispiel-11: Ein XML-Dokument mit Hinweisen auf eine `.dtd`- und eine `.xsd`-Datei:

```
 1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
 2 <!DOCTYPE name SYSTEM "personA.dtd">
 3
 4 <person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 5     xsi:noNamespaceSchemaLocation="personA.xsd">
 6   <name>
 7     <vor_name>Alan</vor_name>
 8     <nach_name>Turing</nach_name>
 9   </name>
10   <beruf>Informatiker</beruf>
11   &hi;
12   <beruf>Mathematiker</beruf>
13   <beruf land="England" zeit="WK2">Kryptograf</beruf>
14 </person>
```

In Zeile 2 wird auf eine `.dtd`-Datei (`personA.dtd`) und in Zeile 5 auf eine `.xsd`-Datei (`personA.xsd`) verwiesen. In Zeile 11 steht eine Entity-Referenz `&hi` (d.h. eine Abkürzung). Was sie bedeuten soll, muss in der Datei `personA.dtd` definiert werden (siehe Beispiel-12, Zeile 16).

Beispiel-12: Eine `.dtd`-Datei:

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 3    Datei personA.dtd:
 4    Document Type Definition fuer die Dateien daten02x.xml
 5 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  -->
 6 <!ELEMENT person    (name, beruf+)>
 7 <!ELEMENT name      (vor_name, nach_name)>
 8 <!ELEMENT vor_name  (#PCDATA)>
 9 <!ELEMENT nach_name (#PCDATA)>
10 <!ELEMENT beruf     (#PCDATA)>
11
12 <!ATTLIST beruf land CDATA #IMPLIED
13                 zeit CDATA #IMPLIED
14 >
15
16 <!ENTITY hi "Hi Alan, how are you?">
```

Beispiel-13: Eine `.xsd`-Datei

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 3    Datei personA.xsd
 4    XML-Schema-Datei fuer die Dateien daten02x.xml
 5    Hier werden alle Typen "so lokal wie moeglich" definiert
 6 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  -->
 7 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 8     elementFormDefault="qualified">
 9
10   <xsd:element name="person">
11     <xsd:complexType mixed="true">
12       <xsd:sequence>
13
14         <xsd:element name="name">
15           <xsd:complexType>
16             <xsd:sequence>
17               <xsd:element name="vor_name"  type="xsd:string"/>
18               <xsd:element name="nach_name" type="xsd:string"/>
19             </xsd:sequence>
20           </xsd:complexType>
21         </xsd:element>
22
23         <xsd:element name="beruf" minOccurs="1" maxOccurs="5">
24           <xsd:complexType>
25             <xsd:simpleContent>
26               <xsd:extension base="xsd:string">
27                 <xsd:attribute name="land"  type="xsd:string"/>
28                 <xsd:attribute name="zeit"  type="xsd:string"/>
29               </xsd:extension>
30             </xsd:simpleContent>
31           </xsd:complexType>
32         </xsd:element>
33
34       </xsd:sequence>
35     </xsd:complexType>
36   </xsd:element> <!-- person -->
37
38 </xsd:schema>
```

In Zeile 23 wird festgelegt, dass (innerhalb eines `person`-Elements) mindestens 1 `beruf`-Element vorkommen muss und höchstens 5 solche Elemente vorkommen dürfen. Eine so genaue Festlegung von Unter- und Obergrenzen ist in einer DTD nicht möglich.