Labor 2

Inhaltsverzeichnis:

1. XML-Dokumente und DTD

1.1. XML-Dokumente und Namespaces

1. Recherchieren Sie zunächst im Internet nach dem Begriff XML Namespaces und erklären Sie den Einsatzzweck anhand eines konkreten Beispiels.

→ Ein XML-Namespace dient der eindeutigen Identifikation von Elementen und Attributen in XML-Dokumenten, insbesondere wenn Daten aus unterschiedlichen Quellen oder Schemata kombiniert werden. Dies ist essentiell, um Namenskonflikte zu vermeiden, da XML-Elemente und Attribute denselben Namen, aber unterschiedliche Bedeutungen haben können. Durch die Einführung eines Namespaces wird jedes Element oder Attribut einem spezifischen Kontext zugeordnet, was sowohl die Lesbarkeit als auch die Verarbeitbarkeit von XML-Daten verbessert.

Die Hauptziele von XML-Namespaces umfassen:

Eindeutigkeit: Namespaces ermöglichen es, gleiche Namen in unterschiedlichen Datenmodellen für verschiedene Zwecke zu verwenden. Dies verhindert Kollisionen und erleichtert die klare Zuordnung der Bedeutung.

Modularität: Namespaces erlauben die Integration verschiedener XML-Sprachen in einem Dokument. Dadurch können verschiedene Standards oder Schemata in einem einheitlichen Dokument kombiniert werden.

Flexibilität: Namespaces fördern die Wiederverwendbarkeit existierender XML-Standards und erleichtern die Erweiterung von Anwendungen oder Datenmodellen, ohne dass Konflikte entstehen.

Ein praktisches Beispiel verdeutlicht den Einsatz von Namespaces. Angenommen, eine XML-Datei kombiniert Daten aus einer Produktdatenbank und einer Kundendatenbank, wobei beide die gleichen Elementnamen wie <nummer> und <name> verwenden. Durch die Einführung von Namespaces können diese eindeutig voneinander unterschieden werden:

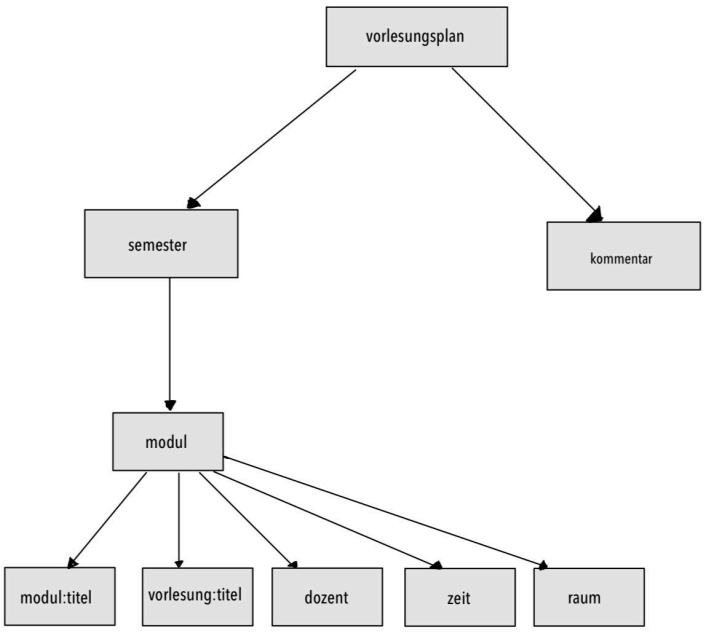
```
<bestellung xmlns:produkt="http://example.com/produkt"
        xmlns:kunde="http://example.com/kunde">
  <produkt:nummer>12345</produkt:nummer>
  <produkt:name>Rasierer</produkt:name>
```

```
  <produkt:preis>49.99</produkt:preis>
  <kunde:nummer>67890</kunde:nummer>
  <kunde:name>Max Mustermann</kunde:name>
  <kunde:adresse>Beispielstraße 1, 12345
   Musterstadt</kunde:adresse>
</bestellung>
```

Hierbei definiert produkt:name den Namespace für die Produkte und kunde:name den Namespace für die Kundendaten. Durch die Präfixe, Produkt und Kunde wird eindeutig, aus welcher Quelle die Elemente stammen. Somit wird verhindert, dass die gleichlautenden Namen, *Nummer* und *Name* zu Verwechslungen führen.

2. Erstellen Sie jetzt ein gültiges XML-Dokument für ihren aktuellen Vorlesungsplan. Die erlaubten Elemente und Attribute definieren Sie in einer externen DTD-Datei. Um den Begriff „Titel“ mehrfach verwenden zu können, setzen Sie Namespaces ein.

a. Entwerfen Sie hierzu grafisch ein baumartiges Datenmodell.
→



b. Nach Fertigstellung und Überprüfung des Datenmodells gegen die fachlichen Anforderungen setzen Sie das Datenmodell in Form einer DTD-Datei um.
→ Eine DTD-Datei (Document Type Definition) definiert die Struktur eines XML-Dokuments und beschreibt, welche Elemente, Attribute und deren Beziehungen erlaubt sind. Sie wird genutzt, um XML-Dokumente validierbar zu machen, indem überprüft wird, ob sie mit den in der DTD definierten Regeln übereinstimmen.

```
<!ELEMENT vorlesungsplan (semester+, kommentar?)>
    <!ATTLIST vorlesungsplan
        xmlns:modul CDATA #REQUIRED
        xmlns:vorlesung CDATA #REQUIRED>
    <!ELEMENT semester (modul+)>
        <!ELEMENT modul (modul:titel, vorlesung:titel, dozent,
        zeit, raum)>
```

```
<!ELEMENT modul:titel (#PCDATA)>
<!ELEMENT vorlesung:titel (#PCDATA)>
<!ELEMENT dozent (#PCDATA)>
<!ELEMENT zeit (#PCDATA)>
<!ELEMENT raum (#PCDATA)>
<!ELEMENT kommentar (#PCDATA)>
```

Strukturdefinition: Die DTD legt fest, welche Elemente und Attribute ein XML-Dokument enthalten darf, in welcher Reihenfolge diese vorkommen und wie sie miteinander verschachtelt sein können.

- vorlsungsplan muss min ein semester enthalten, kann aber einen oder keinen kommentar haben
- Jedes semster-Element muss das Kind modul min. einmal enthalten
- jedes modul-element kann die Kinder modul:titel, vorlesungs:titel, dozent, zeit und raum haben

Datenvalidierung: Ein XML-Dokument kann gegen die DTD geprüft werden, um sicherzustellen, dass es den definierten Regeln entspricht. Dies garantiert, dass die Struktur konsistent ist und Fehler frühzeitig erkannt werden.
Portabilität: Mit einer DTD können XML-Daten leicht zwischen Systemen und Anwendungen ausgetauscht werden, da die Struktur eindeutig dokumentiert ist.
Semantische Eindeutigkeit: Die DTD beschreibt nicht nur die Syntax, sondern auch die Bedeutung der Elemente und Attribute, was die Lesbarkeit und Nachvollziehbarkeit von XML-Dokumenten verbessert.
Datentypen:

- *#PCDATA:* Steht für "Parsed Character Data" und gibt an, dass das Element Text enthalten kann.
- *EMPTY:* Element darf keinen Inhalt haben.
- *ANY:* Element kann beliebigen Inhalt haben (selten verwendet).

<!ATTLIST>-Deklaration: Es definiert Attribute, die einem Element zugeordnet sein können, und spezifiziert deren Eigenschaften wie Datentyp, Standardwerte und Gültigkeitsanforderungen. Die Struktur sieht folgendermaßen aus:
<!ATTLIST Elementname Attributname Attributtyp Standardwertspezifikation>
Elementname: Gibt das Element an, dem die Attribute zugeordnet werden.
Attributname: Definiert den Namen des Attributs, das zum Element gehört.
Attributtyp: Legt den Datentyp oder die zulässigen Werte für das Attribut fest. Gängige Typen sind:

- CDATA: Beliebige Zeichenfolge (Character Data).
- ID: Ein eindeutiger Bezeichner innerhalb des Dokuments.
- IDREF: Verweist auf einen anderen eindeutigen ID-Wert.
- NMTOKEN: Ein gültiger Name gemäß XML-Standards.
- ENUMERATION: Eine Liste zulässiger Werte (z. B. (Monday | Tuesday | Wednesday)).

Standardwertspezifikation: Gibt an, ob das Attribut erforderlich ist und ob ein Standardwert definiert wird:

- #REQUIRED: Das Attribut ist zwingend erforderlich und muss im XML-Dokument angegeben werden.
- #IMPLIED: Das Attribut ist optional und kann weggelassen werden.
- "Wert": Gibt einen Standardwert an, der verwendet wird, wenn kein Wert im XML-Dokument angegeben wird.

c. Final erstellen Sie ihre XML-Datei und übernehmen die Daten aus dem Stundenplan.
→

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE vorlesungsplan SYSTEM
<vorlesungsplan xmlns:modul="http://example.com/modul"
xmlns:vorlesung="http://example.com/vorlesung">
  <semester>
    <modul>
      <modul:titel>Programmierung 1</modul:titel>
      <vorlesung:titel>Grundlagen der
          Programmierung</vorlesung:titel>
      <dozent>Prof. Dr. Müller</dozent>
      <zeit>Montag, 10:00 - 12:00</zeit>
      <raum>Raum A101</raum>
    </modul>
    <modul>
      <modul:titel>Mathematik 1</modul:titel>
      <vorlesung:titel>Lineare Algebra</vorlesung:titel>
      <dozent>Dr. Schmidt</dozent>
      <zeit>Mittwoch, 14:00 - 16:00</zeit>
      <raum>Raum B202</raum>
    </modul>
  </semester>
        <kommentar>Dies ist der Vorlesungsplan für das
        Wintersemester 2025.</kommentar>
</vorlesungsplan>
```

d. Nach Fertigstellung überprüfen Sie ihr XML-Dokument auf Validität. Hierzu können Sie beispielsweise die Webseite https://www.xmlvalidation.com/ verwenden.

→

No errors were found

The following files have been uploaded so far:

XML document:

vorlesungsplan.dtd

e. Laden Sie ihre XML-Datei in einen Browser. Kann die XML-Datei fehlerfrei geladen und wird die von Ihnen entworfene Baumstruktur angezeigt?

→

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```xml
▼<vorlesungsplan xmlns:modul="http://example.com/modul" xmlns:vorlesung="http://example.com/vorlesung">
  ▼<semester>
    ▼<modul>
      <modul:titel>Programmierung 1</modul:titel>
      <vorlesung:titel>Grundlagen der Programmierung</vorlesung:titel>
      <dozent>Prof. Dr. Müller</dozent>
      <zeit>Montag, 10:00 - 12:00</zeit>
      <raum>Raum A101</raum>
    </modul>
    ▼<modul>
      <modul:titel>Mathematik 1</modul:titel>
      <vorlesung:titel>Lineare Algebra</vorlesung:titel>
      <dozent>Dr. Schmidt</dozent>
      <zeit>Mittwoch, 14:00 - 16:00</zeit>
      <raum>Raum B202</raum>
    </modul>
  </semester>
  <kommentar>Dies ist der Vorlesungsplan für das Wintersemester 2025.</kommentar>
</vorlesungsplan>
```

1.2. XML-Dokumente und Textformatierungen

Speichern Sie den folgenden Text inklusive seiner Formatierungen als XML-Dokument ab, sodass die wesentliche Information über die Formatierung des Objektes erhalten bleibt: Überschrift, Leerzeilen, Absätze, Schriftart, Hervorhebungen, …

Edward Snowden

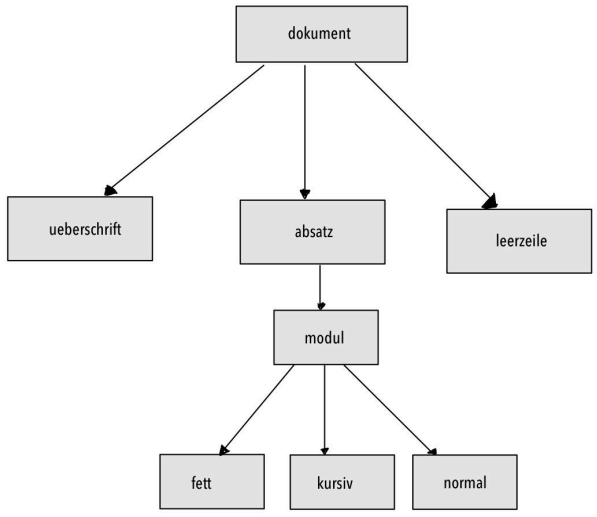
Edward Snowden ist ein *US-amerikanischer Whistleblower* und ehemaliger *CIA-Mitarbeiter*. Seine Enthüllungen gaben Einblicke in das Ausmaß der weltweiten Überwachungs- und Spionagepraktiken von Geheimdiensten – überwiegend jenen der Vereinigten Staaten und Großbritanniens. Diese lösten im Sommer 2013 die NSA-Affäre aus.

Er wurde dafür mehrfach von nichtstaatlichen Organisationen ausgezeichnet und für den Friedensnobelpreis nominiert.

1. Entwerfen Sie wieder ein baumartiges Datenmodell und erstellen dann die zugehörige DTD-Datei. Verwenden Sie beim Design ihres Dokumentes Entitäten, die die Formatierung von Text beschreiben (z.B.: Fettdruck, Schrägruck, …).
   → DTD- Datei:

```
<!ELEMENT dokument (ueberschrift, absatz+, leerzeile?)>
    <!ELEMENT ueberschrift (#PCDATA)>
    <!ELEMENT absatz (#PCDATA|fett|kursiv|normal)*>
    <!ELEMENT leerzeile EMPTY>

    <!ELEMENT fett (#PCDATA)>
    <!ELEMENT kursiv (#PCDATA)>
    <!ELEMENT normal (#PCDATA)>
    <!ENTITY nbsp " ">
```



2. Auf Basis der DTD-Datei speichern sie den obigen Text als XML-Datei. Überprüfen Sie ihr Modell in einem XML-Validator (siehe oben) und zeigen Sie die entworfene Baumstruktur in einem Browser an.

→

No errors were found

The following files have been uploaded so far:

XML document:

textformat.dtd

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<dokument>
   <ueberschrift>Edward Snowden</ueberschrift>
  ▼<absatz>
     <fett>Edward Snowden</fett>
     <normal>ist ein</normal>
     <kursiv>US-amerikanischer Whistleblower</kursiv>
     <normal>und ehemaliger</normal>
     <kursiv>CIA-Mitarbeiter</kursiv>
     <normal>. Seine Enthüllungen gaben Einblicke in das Ausmaß der weltweiten Überwachungs- und Spionagepraktiken von Geheimdiensten – überwiegend jenen der
     Vereinigten Staaten und Großbritanniens. Diese lösten im Sommer 2013 die NSA-Affäre aus.</normal>
   </absatz>
  ▼<absatz>
     <normal>Er wurde dafür mehrfach von nichtstaatlichen Organisationen ausgezeichnet und für den Friedensnobelpreis nominiert.</normal>
   </absatz>
 </dokument>
```

2. Zeichensätze, Code-Tabellen und Character Encoding

2.1. Erweitern der Laborumgebung

→ Installationsprozess

2.2. UTF-8, UTF-16, UTF-32 Encoding von Zeichen

Erstellen Sie jetzt ein Python-Programm, das die folgenden Glyphen aus einer Datei einliest und

1. deren Unicode Code Point,
2. deren Zeichen-Encoding für UTF-8, UTF-16be und UTF32-be,
3. deren Namen in Unicode bestimmt.

Die Programmausgabe sollte in Form einer Tabelle erfolgen, die die folgenden Spalten enthält

Encoding	Codepoint	Encoded Codepoint	Name of Character	Glyph

4. Vergleichen Sie die Ausgabe für die verschiedenen Unicode-Encodings. Welches Encoding ist optional für europäische Sprachen und welches Encoding ist optimal für asiatische Sprachen.
   → Unicode ist ein Standard, der es ermöglicht, Schriftzeichen nahezu aller Sprachen zu kodieren. Dabei stehen unterschiedliche Encodings (Zeichenkodierungen) zur Verfügung, die sich in ihrer Speicherplatznutzung und Effizienz unterscheiden:
   UTF-8 (8-Bit Unicode Transformation Format):
   UTF-8 ist das am weitesten verbreitete Encoding und wird insbesondere für europäische Sprachen und andere lateinische Schriftsysteme bevorzugt. Es ist rückwärtskompatibel mit ASCII, was bedeutet, dass Zeichen des ASCII-Bereichs (z. B. Buchstaben und Zahlen) dieselbe Kodierung verwenden. Dies führt zu einer effizienten Speicherplatznutzung für Texte, die hauptsächlich aus diesen Zeichen bestehen. UTF-8 verwendet eine variable Byte-Länge (1 bis 4 Bytes pro Zeichen), was Speicherplatz spart, aber die Verarbeitung komplizierter machen kann.
   UTF-16BE (16-Bit Unicode Transformation Format, Big Endian):
   Dieses Encoding ist effizienter für Sprachen mit vielen Zeichen außerhalb des ASCII-Bereichs, wie asiatische Sprachen (z.B. Chinesisch, Japanisch, Koreanisch). Während UTF-8 für solche Zeichen oft 3 bis 4 Bytes benötigt, verwendet UTF-16BE für viele dieser Zeichen nur 2 Bytes. Dies macht es zu einer guten Wahl für Anwendungen, die diese Sprachen primär unterstützen müssen. Allerdings ist UTF-16BE weniger speicherplatzsparend für Texte mit vielen ASCII-Zeichen.
   UTF-32BE (32-Bit Unicode Transformation Format, Big Endian):
   UTF-32 kodiert jedes Zeichen mit einer festen Länge von 4 Bytes. Dies vereinfacht die Verarbeitung von Texten, da jedes Zeichen gleich groß ist. Allerdings ist der Speicherplatzbedarf deutlich höher als bei UTF-8 oder UTF-16BE, weshalb UTF-32 selten in der Praxis verwendet wird. Es ist eher für spezielle Anwendungen nützlich, die eine konstante Zeichenlänge erfordern.
   Zusammenfassend kann gesagt werden, dass UTF-8 optimal für europäische Sprachen ist, da es ASCII-kompatibel und speicherplatzsparend ist. UTF-16BE

eignet sich hingegen besser für asiatische Sprachen, da es komplexe Schriftzeichen effizienter speichert.

5. Erklären Sie den Begriff der Glyphe anhand von Beispielen.
   → Eine Glyphe bezeichnet die konkrete visuelle Darstellung eines Zeichens in einer Schriftart. Während ein Unicode-Zeichen eine abstrakte Definition ist, beschreibt die Glyphe, wie dieses Zeichen aussieht, wenn es gerendert wird. Glyphen sind somit eng mit der Schriftart verbunden, da verschiedene Schriftarten dasselbe Zeichen unterschiedlich darstellen können.
   Beispiele:
   - Das Unicode-Zeichen U+0041 repräsentiert den Großbuchstaben „A“. Je nach Schriftart (z. B. Arial, Times New Roman) sieht die Glyphe unterschiedlich aus.
   - Das Zeichen U+306B (に) aus dem japanischen Hiragana-Alphabet wird in verschiedenen japanischen Schriftarten (z. B. Mincho oder Gothic) unterschiedlich visualisiert.

   Glyphen spielen auch eine Rolle bei der Unterscheidung von Symbolen. Beispielsweise können dekorative oder kalligrafische Variationen eines Zeichens als verschiedene Glyphen eines einzigen Unicode-Zeichens betrachtet werden.

6. Können Sie die obigen Unicode-Zeichen in einem Terminal ausgeben? Erklären Sie ihre Beobachtung
   → Unicode-Zeichen können in modernen Terminals problemlos ausgegeben werden, sofern folgende Bedingungen erfüllt sind:
   - Schriftart-Unterstützung: Das Terminal muss eine Schriftart verwenden, die die Glyphen für die entsprechenden Unicode-Zeichen enthält. Beispiele für solche Schriftarten sind „DejaVu Sans“ oder „Noto Sans“.
   - Zeichenkodierung: Das Terminal muss UTF-8 als Zeichencodierung verwenden, da diese mittlerweile der Standard für die Darstellung von Unicode-Zeichen ist.

   Wenn die obigen Unicode-Zeichen im Terminal ausgegeben werden, ist Folgendes zu beachten:
   - Werden die Glyphen nicht unterstützt (z. B. weil die Schriftart sie nicht enthält), erscheinen Platzhalter wie „?“, Rechtecke oder leere Kästchen.
   - Bei einer korrekten Konfiguration (UTF-8 und unterstützende Schriftart) werden die Zeichen wie erwartet dargestellt.

   Die Darstellung im Terminal kann getestet werden, indem der Python-Code ausgeführt wird. Falls es zu Problemen kommt, liegt dies meist an den Terminal-Einstellungen oder der Auswahl der Schriftart. Durch Anpassung dieser Parameter kann sichergestellt werden, dass alle Unicode-Zeichen korrekt gerendert werden.

Encoding	Codepoint	Encoded Codepoint	Name of Character	Glyph
UTF-8	U+00C8	C388	LATIN CAPITAL LETTER E WITH GRAVE	È
UTF-16BE	U+00C8	00C8	LATIN CAPITAL LETTER E WITH GRAVE	È
UTF-32BE	U+00C8	000000C8	LATIN CAPITAL LETTER E WITH GRAVE	È
UTF-8	U+00FC	C3BC	LATIN SMALL LETTER U WITH DIAERESIS	ü
UTF-16BE	U+00FC	00FC	LATIN SMALL LETTER U WITH DIAERESIS	ü
UTF-32BE	U+00FC	000000FC	LATIN SMALL LETTER U WITH DIAERESIS	ü
UTF-8	U+00A9	C2A9	COPYRIGHT SIGN	©
UTF-16BE	U+00A9	00A9	COPYRIGHT SIGN	©
UTF-32BE	U+00A9	000000A9	COPYRIGHT SIGN	©
UTF-8	U+263A	E298BA	WHITE SMILING FACE	☺
UTF-16BE	U+263A	263A	WHITE SMILING FACE	☺
UTF-32BE	U+263A	0000263A	WHITE SMILING FACE	☺
UTF-8	U+2728	E29CA8	SPARKLES	✨
UTF-16BE	U+2728	2728	SPARKLES	✨
UTF-32BE	U+2728	00002728	SPARKLES	✨
UTF-8	U+058E	D68E	LEFT-FACING ARMENIAN ETERNITY SIGN	֎
UTF-16BE	U+058E	058E	LEFT-FACING ARMENIAN ETERNITY SIGN	֎
UTF-32BE	U+058E	0000058E	LEFT-FACING ARMENIAN ETERNITY SIGN	֎