

## Letzte Woche

Themen der letzten Vorlesung:

- Scheme
- Listen, Funktionen auf Listen
- Symbole, Quote, Quasiquote
- Continuations, `call/cc`
- SURflets, SURflet-API
- Eingabewidgets als Werte erster Klasse

## wishlist.scm (1)

```
(define-structure surflet surflet-interface
  (open surflets
    scheme-with-scssh)
  (begin
    (define (ask-for-wish)
      (let ((wish-input (make-text-field))
            (submit-button (make-submit-button)))
        (let ((req
              (send-html/suspend
                (lambda (k-url)
                  `(html
                    (body
                     (h1 "Enter your wish")
                     (surflet-form ,k-url
                                   (p "Your wish:"
                                       ,wish-input
                                       ,submit-button))))))))
          (input-field-value wish-input
                              (get-bindings req))))))
```

## Implementierung von send/suspend

Die Implementierung von (`send/suspend proc`):

- Continuation einfangen mit `call/cc`
- Eindeutigen Schlüssel generieren
- Continuation in Tabelle unter Schlüssel ablegen
- Schlüssel zu `k-url` zusammenbasteln:  
Z. B.: `http://127.0.0.1/surflet/;id185*k3-5799`
- Übergebene Prozedur `proc` mit `k-url` aufrufen

Nach Rückkehr von `proc`:

- Rückgabewert in HTML umwandeln und an Client senden
- Ende der Bearbeitung des Requests

## wishlist.scm (2)

```
(define (print-wish wish)
  (send-html/finish
   `(html
     (body
      (h1 "Your (short) wishlist")
      (p ,wish))))))

(define (main req)
  (print-wish (ask-for-wish)))
```

## Session-Daten

Was ist eigentlich mit den Session-Daten passiert?

- Werte von Aufruf zu Aufruf transportieren: Rückgabewert von Funktionen (oder Session-API)
- Werte global für alle Sessions eines SURflets: globale Variable im SURflet-Modul
- Werte global für alle Sessions aller SURflets: globale Variable in einem Extramodul

## wishlist-pro.scm (1)

```
(define-structure surflet surflet-interface
  (open scheme-with-scsh
    surflets
    (subset srfi-1 (filter))
    inspect-exception)
  (begin

    (define *wish-id* 0)

    (define (fresh-wish-id)
      (set! *wish-id* (+ *wish-id* 1))
      *wish-id*)

    (define (make-wish text)
      (cons (fresh-wish-id) text))

    (define wish-id car)
    (define wish-text cdr)
```

## wishlist-pro.scm (2)

```
(define (output-wish-list wish-list delete-button k-url)
  `(ul
    ,@(map
      (lambda (wish)
        `(li ,(wish-text wish) " "
            (url ,(delete-button k-url (wish-id wish))
                "delete wish")))
        wish-list)))
```

## wishlist-pro.scm (3)

```
(define (manage-wish-list wish-list)
  (let ((wish-input (make-text-field))
        (no-more-wishes (make-address))
        (submit-button (make-submit-button))
        (delete-button (make-annotated-address)))

    (let ((req
           (send-html/suspend
            (lambda (k-url)
              `(html
                (body
                 (h1 "Wishlist Manager Pro")
                 (surflet-form
                  ,k-url POST
                  ,(output-wish-list
                     wish-list delete-button k-url)
                  (p "Add wish:" ,wish-input
                    ,submit-button))
                  (p (url ,(no-more-wishes k-url)
                        "Perfectly happy"))))))))))
```

## wishlist-pro.scm (4)

```
(case-returned-via (get-bindings req)
  ((no-more-wishes) wish-list)
  ((delete-button) => (lambda (id)
    (manage-wish-list
      (filter
        (lambda (wish)
          (not (= id (wish-id wish))))
        wish-list))))
  (else
   (manage-wish-list
    (cons (make-wish
      (input-field-value
        wish-input (get-bindings req)))
      wish-list))))))
```

## *Datenrepräsentation mit XML*

## wishlist-pro.scm (5)

```
(define (main req)
  (with-inspecting-handler
   8888
   (lambda (condition)
     (with-current-output-port (current-error-port)
      (display "starting remote handler for condition ")
      (display condition)
      (newline)
      (display "Please connect to port 8888")
      (newline)))
   (lambda ()
     (send-html/finish
      `(html
        (body
         (h2 "Your wishlist")
         (ul
          ,@(map (lambda (wish) `(li ,(wish-text wish))
                 (manage-wish-list '())))))))))
   ))
```

## Ausgabe in Web-Applikationen

Bisher:

- Berechne Ergebnis der Anfrage
- Innerhalb der Web-Applikation: Ergebnis ist eine Datenstruktur
- Übersetze die Datenstruktur in (lesbare) HTML-Seite

Die birgt einige Nachteile:

- Code enthält viele die Präsentation betreffende Details
- Struktur der Daten geht möglicherweise verloren
- Daten schwer von Formatierungsinformation zu trennen
- Andere Applikationen haben es schwer die Daten weiterzuverarbeiten

## Extensible Markup Language

### Extensible Markup Language (XML):

Sprache zur Beschreibung von strukturierten Datenformaten

Vorteile durch Beschreibung:

- Es wird leicht, das Datenformat zu lesen
- Erleichtert Austausch von strukturierten Daten

Beispiel für eine Anwendung:

- Umwandlung von XML-Daten in im Browser darstellbare (X)HTML-Seite

Durch den Browser!

## Daten in XML

Datenrepräsentation in XML:

- Daten werden in Markup-Sprache (ähnlich HTML) repräsentiert
- Struktur durch Definition von Tags und Attributen:  
*Document Type Definition (DTD)*
- Einheitliche Syntax; Ein Parser für alle XML-Datenformate
- Entspricht eine XML-Datei nicht der Beschreibung (DTD), bricht der Parser den Lesevorgang ab.

## Syntax von XML (1)

XML ist eine Markup-Sprache ähnlich HTML. Untermenge von SGML (Standard Generalized Markup Language).

Elemente der Syntax:

Start-Tag	<code>&lt;name attribut*&gt;</code>
End-Tag	<code>&lt;/name&gt;</code>
Leere Tags	<code>&lt;name attribut*&gt;</code>
Entitäten	<code>&amp;name;</code>
Zeichen	<code>&amp;#[0-9]+;</code> oder <code>&amp;#x[0-9a-fA-F]+;</code>
Kommentare	<code>&lt;-- ... --&gt;</code>
CDATA-Abschnitt	<code>&lt;![CDATA[ beliebige Struktur ]]&gt;</code>

## Elemente in XML

Ein *Element* bezeichnet die Folge von Start-Tag, Inhalt und End-Tag. Beispiele:

- `<telefon/>`
- `<telefon>07071/12345</telefon>`
- `<telefon><vorwahl>07071</vorwahl>12345</telefon>`
- `<telefon privat="nein">07071/12345</telefon>`
- `<telefon privat="&nein;">07071/12345</telefon>`
- `<telefon><![CDATA[+49-<bla>07071<<blub/12345]]></telefon>`

## Elemente

Für Elemente gelten die folgenden Regeln:

- Entweder `<name>inhalt</name>` oder `<name/>`
- Elemente müssen überlappungsfrei geschachtelt werden

### Verboten:

```
<tag-1>abc<tag-2>def</tag-1>ghi</tag-2>
```

Namen für Tags (und Attribute und Entities):

- Groß-/Kleinschreibung wird unterschieden
- Gültige Zeichen: Buchstaben, Zahlen, `_`, `-`, `.` und `:`
- Der Doppelpunkt hat allerdings eine spezielle Bedeutung

## Attribute und Entities

Attribute

- Name-Werte-Paare im Start-Tag
- Erlaubte Attribute sind in DTD festgelegt

Entities

- Werden vom Parser durch definierten Text ersetzt
- Viele verschiedene Arten

## Syntax von XML (2)

Weitere Elemente der XML-Syntax:

- Dokument-Typ-Deklaration:  
`<!DOCTYPE name [ Definition ]>`  
`<!DOCTYPE name SYSTEM Pfad >`  
`<!DOCTYPE name PUBLIC URI >`
- Processing Instruction (Verarbeitungsanweisung):  
`<?name anweisung ?>`
- XML-Deklaration:  
`<?xml version="1.0" encoding="..."?>`

## Aufbau einer XML-Datei

Aufbau einer XML-Datei:

- XML-Deklaration (optional)
  - Processing-Instructions und Kommentare (optional)
  - Dokument-Typ-Deklaration (optional)
  - Genau ein *Root-Element* oder *Document-Element*
- ⇒ Erfüllt eine XML-Datei diese Forderungen, ist es *well-formed*
- ⇒ Enthält und entspricht eine well-formed XML-Datei den Regeln der Dokument-Typ-Deklaration ist sie *valid*

## Beispiel: lager.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE lager SYSTEM "lager.dtd">

<lager>
  <eingang>
    <eingangsnummer>452</eingangsnummer>
    <artikelnummer>45-234</artikelnummer>
    <artikelname>Schreibtischstuhl "Chef"</artikelname>
    <artikelmenge>10</artikelmenge>
    <warenklasse>C-III</warenklasse>
  </eingang>
  <ausgang>
    <ausgangsnummer>318</ausgangsnummer>
    <artikelnummer>37-917</artikelnummer>
    <artikelname>Spiegelschrank "Narziss"</artikelname>
    <artikelmenge>3</artikelmenge>
    <beschreibung>F-IV</beschreibung>
  </ausgang>
</lager>
```

## Document-Typ-Deklaration (DTD)

Inhalt einer Document-Typ-Deklaration:

- Listet erlaubte Elemente, Attribute und Entities auf
- Ist selber in XML-Syntax abgefasst
- Nur eine von mehreren Möglichkeiten ein XML-Datenformat zu beschreiben. Alternative: XML Schema

## DTD: Elemente beschreiben

Elemente beschreiben mit:

```
<!ELEMENT name inhalt>
```

Wobei *inhalt* sein kann:

- **EMPTY** falls das Element keinen Inhalt hat
- **ANY** falls das Element beliebigen Inhalt hat
- (**#PCDATA [ |Name]\***) gemischter Inhalt, bestehend aus Zeichendaten **#PCDATA** und den Elementen **Name**
- Regulärer Ausdruck, der gültige Unterelemente beschreibt

## Unterelemente beschreiben

Die gültigen Unterelemente werden durch einen regulären Ausdruck beschrieben:

- Auflistung mit Komma ,
- Alternative mit |
- Beliebige Wiederholung \*
- optionale Elemente ?
- Mindestens einmal vorhanden +

Beispiele:

- `<!ELEMENT eingang (eingangsnummer, artikel, zusatz)>`
- `<!ELEMENT p (#PCDATA|b|i|br)>`

## Attribute beschreiben (1)

Gültige Attribute beschreiben mit:

```
<!ATTLIST name Attdef*>
Attdef = name AttType DefaultDecl
```

Der **AttType** bestimmt, welche Werte das Attribut annehmen darf.  
Mögliche Typen:

- **CDATA** für beliebige Zeichenfolgen
- Ein Spezialtoken wie **ID**, **ENTITY** oder **NMTOKEN**
- Ein Vereinigung aus obigen Typen, gebildet durch |

## Attribute beschreiben (2)

Die **DefaultDecl** beschreibt, wie das Attribut vorkommt:

- **#REQUIRED** falls das Attribut immer angegeben werden muss
- **#IMPLIED** falls das Attribut optional ist
- **( AttValue )** falls das Attribut optional ist. *AttValue* ist der Default-Wert für das Attribut
- **( #FIXED AttValue )** falls das Attribut immer den Defaultwert haben soll

## Attribute beschreiben (3)

Beispiele für Attributsbeschreibungen:

- ```
<!ATTLIST zeit
  änderung_nach_release (ja | nein) "nein">
```
- ```
<!ATTLIST termdef
  id      ID      #REQUIRED
  name    CDATA   #IMPLIED>
```
- ```
<!ATTLIST veranstaltung
  jahr CDATA #IMPLIED
  art  (Vorlesung | Seminar | Praktikum) #REQUIRED
  titel CDATA #REQUIRED>
```