HTML

```
<!doctype html>       <!-- minimalbsp-->
<html>
 <head>
  <meta charset=“utf-8"/>
  <link rel="stylesheet" href=“style.css” type=“text/css”/>
  <script src="myscript.js"></script>
  <title>A first HTML-Example</title>
 </head>
 <body onload="someFunctionFromMyScriptJS()">
   <article>
      <p>Hello <span id="type">class</span>!</p>
      <a href="http://example.com">A link!</a>
      <form action="tables.html" method="get">
         Fach:<br/>
         <input type="text" name="fach" value="Web-Prog"/>
         <br/><br/>
         <input type="submit" value="Submit"7>
      </form>
    </article>
 </body>
</html>
```

- Listen: ul,ol Listenelemente: li
- sematisch: main, article, aside, header, footer,section
- <!- - - -> Kommentar
- div, span (span ist inline)
- img, audio, video (Quell-Url: src)
- form: (action, method, target, autocomplete, novalidate)
  input type: text, password, submit, radio, checkbox, button, color
- <script src=URL/> laden von Javascript-Files, interpretation beim Einlesen

Sonderzeichen: <:< >: > & : &

Box-Model

Margin, **Border** Padding, Content(height,width)

CSS

```
Selector [, Selector2, ..., SelectorN]
{
  Property1: Value1; /* Comment */
  ...
  PropertyN: ValueN;
}
```

Selectors

- *: alle elemente
- p: alle <p> elemente
- h1, p: alle <h1> und alle <p> elemente
- div p: alle <p> elemente in elementen
- .example: alle elemente mit class=” example”
- #id: alle elemente mit id=”id”
- div > p: alle <p> elemente mit als parent
- div + p: <p> elemente, die direkt auf elemente folgen
- h1~p: alle <p> elemente die auf <h1> elemente folgen
- [href] : elemente mit href-Attribut
- [href=/url.tst] : elemente mit href="/url.tst"

pseudo-tags

- :active geklickt
- :hover mausschwebend
- :visited besuchter link

browser-präfixe: -ms-, -moz-, -webkit-

Media-Query

```
@media not|only mediatype and (expressions) {  CSS-Code;}

z.B.
@media screen and (min-width: 480px) {
  body {
    background-color: lightgreen;
  }
}
```

JavaScript

es gibt nur referenz-Typen
undefined, null, infinity,NaN
[1, 2, 3] // Array initializer
Vergleiche mit === oder !==
Objektvergleiche vergleichen die Referenz,
Primitive Typen(z.B. Strings) werden Wertverglichen
Strikter Modus: "use strict"; // z.B. für Vergleiche ohne Konvertierung
Foreach schleife: for (var p in o){ ... }

funktionen

hoisting: variablendeklarationen werden an Funktionsanfang gezogen
scope: global scope and function scope, kein Block-level-scope
Zugriff auf aufrufkontext: this
innere Funktionen können auf umgebende Properties Zugreifen (außer this);
=> closure: funktionen merken sich umgebungsvariablen in einer scope chain z.B.

```
function outer () {
   var x = 2;
   return function (){ return 2*x;};
}
outer()() // gibt 4 zurück;
```

indirekte funktionsaufrufe: apply(context, [args]) bzw. call(context, arg1, arg2, ...)

Objektorientierung (Prototypen)

Objekte sind sammlungen von Properties (name / value pairs)
{x:1, y:2} // Object initializer
Konstruktoraufruf: var x = new MyClass();
MyVec.prototype.addTo = function (v) { //sth; }

```
--Vererben:
function Kind(x) {
   Eltern.apply(this, x);   }// Elternkonstruktor
--Kapselung:
function klasse(x) { // Constructor
   var _x = x;
   this.getX = function() { return _x; };    }
```

DOM

Erlaubt Manipulation des Dokumentes zur laufzeit;
Darstellung der Elemente durch Nodes (Inhalt in TextNode)
Nodes sind in Baumstruktur gegliedert, incl. Geschwisterliste
zugriff: getElementById (): liefert HTML-Element
getElementsByClassName(), getElementsByName(): liefern liste von HTML-Elementen
Attribute auf HTML-Elementen:
innerHTML, setAttribute(name, value), style.property;
nodeName (Tag), nodeValue (Text in TextNodes)
Events:
Im HTML: <h1 onclick="this.innerHTML='Foo!'»Click me!</h1>
Im JS: element.addEventListener(event, function); (event ist ohne "on")
Event-weitergabe (Defaultverhalten) stoppen: event.preventDefault(); event.stopPropagation();
Baum-navigation: parentNode, childNodes[idx], firstChild,lastChild, nextSibling, previousSibling
Elemente Einfügen (Bsp)

```
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);
var element = document.getElementById("div1");
element.appendChild(para); // analog dazu insertBefore(), ...
                           // ... replaceChild(), removeChild()
```

URL codieren: encodeURIComponent(*url*);

AJAX

Serverkommunikation ohne neuladen der Seite

```
var req = new XMLHttpRequest();
var response = null;
req.open("GET", URL, true);
req.onreadystatechange = function () {
  if (req.readyState == 4) {
    response = req.responseText; // responseXML when transferring XML
    // do something with the response
  }
};
req.send(null)
```

BOM

Interaktion mit dem Browser,kein standard aber in allen Browsern ähnlich window ist der Tab, bei browsern = globales Objekt, jeder Tab eigener Interpreter

Objekte

- document das HTML document als DOM object
- screen Bildschirminformationen (width, height, ...)
- location aktuelle URL, ermöglicht redirects Properties:
  - hash: teil nach #
  - hostname
  - href: ganze URL
  - pathname:
  - port
- history eingeschränkter zugriff auf Browserverlauf
- navigator Browserinformationen
  nicht standardisiert, weit unterstützt (appName, geolocation ...);

Methoden

- alert(msg): Popup mit Text und OK-Button
- atob()/btoa():Base-64 encoding / decoding von strings
- open(url) Öffnet URL in neuem Fenster (Popup)
- close() Schließt das Fenster
- setInterval(function,milliseconds,parameters) / clearInterval()
  Ruft funktion wiederholt in Intervall auf.
- setTimeout() / clearTimeout() Ruft funktion nach Ablauf von zeit auf

Local Storage

```
if (typeof(Storage) !== "undefined") {
  // Store
  localStorage.setItem("name", "Smith");
  // Retrieve
  document.getElementById("result").innerHTML
                  = localStorage.getItem("name");
}
```

Drag& Drop

```
<script>
function allowDrop(ev) {
  ev.preventDefault();
}
function drag(ev) {
  ev.dataTransfer.setData("text", ev.target.id);
}
function drop(ev) {
  ev.preventDefault();
  var data = ev.dataTransfer.getData("text");
  ev.target.appendChild(document.getElementById(data));
}
</script>
...
<div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)"/>
<p id="drag1" draggable="true" ondragstart="drag(event)"/>
```

PHP
Wird auf Server ausgeführt, ausgabe wird versendet => Browser weiss davon nichts.
Einbetten in HTML: <?php ... ?>

Variablen:
$myInt = 10;
isset($myInt); //True wenn definiert
unset($myInt); //Löschen

Superglobale variablen Immer direkt aufrufbar

- $GLOBALS - alle Variablen im Globalen scope
- $_SERVER - Informationen über Server und ausführungsumgebung
- $_GET - HTTP GET variablen
- $_POST - HTTP POST variablen
- $_FILES - HTTP datei upload variablen
- $_COOKIE - HTTP cookies
- $_SESSION - Session variablen
- $_REQUEST - HTTP request variablen
- $_ENV - Umgebungs variablen

Referenzen

```
$a =& $b; // $a and $b zeigen auf gleiches objekt
function foo(&$var) {  //übergabe einer Referenz
   $var++;
}
class Foo {
   public $value = 42;
   public function &getValue() {
      return $this->value;
   }
}
```

Strings
Kein Unicode! verketten mit .
$helloStr = 'Hello Nr. $myInt'; // Nr. $myInt
$helloStr2 = "Hello Nr. $myInt"; // Nr. 10
$helloStr3 = "Hello Nr. {$myInt}eger // Nr. 10eger

Arrays
key-value!

```
$a1 = array("a", "b", "c");
$a2 = array( "name" => "hans",
   "alter" => 5,
   10 => 25,
   100 => 78)

echo $a1[1];
echo $a2["foo"];
echo $a2[100];
```

count($a2); // höchster integer arraykey (in diesem Fall 100)

objekte

```
class Foo {
  public $var = 'default value';
  protected $bar;
  public function __construct() { // Konstruktor
    $this->bar = 42;
  }
  public function displayVar() {  echo $this->var;  }
}
$f = new Foo();
$f->displayVar();  // methodenaufruf
//----vererbung
class Bar extends Foo {
  public function __construct() {
    parent::__construct();    // Konstruktor vaterklasse
    $this->bar = 24;
  }
}
$b = new Bar();
$b->displayVar();  //Memberaufruf
```

.

Kontrollstrukturen

Einbinden: include, require, include_once, require_once (require löst fehler aus)
if, elseif, else
while, do-while,for analog zu C

```
foreach ($a as $value) {
//bzw.
foreach ($a as $key => $value) {
   echo $key . PHP_EOL;
   echo $value . " == " . $a[$key] . PHP_EOL;
```

funktionen

```
$gvar = 'sth';      // globale variable
function foo($arg1, $arg2, ..., $argN) {
   global $gvar;    // aufruf globale Variable
   return $gvar;
}
foo();           // aufruf
$bar = 'foo';    // dynamischer aufruf
$bar();
$bar = function () { ... }; // anonyme funktion
```

CGI

Aufruf von Kommandozeilenbefehlen durch Web-Server:
server setzt umgebungsvariablen (z.B. QUERY_ STRING, REQUEST_ URI, REQUEST_ METHOD, HTTP_ ACCEPT, ...)
und ruft ein Skript auf(mit PUT/POST als stdin), stdout wird als antwort versendet
Nachteile: Nach Angriff u.U. beliebiger Code direkt auf Betriebssystem einschleusbar, Erzeugt Prozess für jeden Aufruf

Verbesserungen

Compilierte Anwendungen statt Skripte
FastCGI: Länger Laufender Interpreter führt mehrere Anfragen nacheinenader aus
=> weniger Prozesse
HTTP-Server modules: Interpreter als Modul des Webservers

http

generisch, plain text, zustandslos;

request-line oder status-line
*(message-header + Leerzeile) //optional
Leerzeile (CRLF)
Nachrichtenteil
request-line : METHOD URL HTTP/1.1
Methoden: GET, PUT, POST, DELETE... response line : HTTP/1.1 STATUS-CODE STATUS-MESSAGE

Status-code:

- 1XX - Informational (request received, continues processing)
- 2XX - Success
- 3XX - Redirection (further action required in order to fulfill the request)
- 4XX - Client error (request contains bad syntax or cannot be fulfilled)
- 5XX - Server error (server failed to fulfill an otherwise valid request)

Sessions

Http ist zustandslos, Sitzungen durch austausch von Session-Identifiern bei Login.
SessionID muss eindeutig sein.
Client schickt SessionID bei jeder Anfrage, z.B. per Cookie, URL
Cookie: key-value-paare als string, JavaScript-zugriff möglich

https

TLS = Transport Layer Security zwischen TCP und http => schutz vor Man-in-the-middle, authentifikation des Servers und Verschlüsselung der Verbindung
Attacken: wenn URI erratbar: Known-plaintext attacken, Traffic-analysen

Web-Service

Interaktion von Rechnern über das Netzwerk per Remote Procedure Call
RESTful: Verwendet Standart-Http Methoden(GET, PUT, POST, DELETE, ...) & URI

Sicherheit

- Injection: interpretation von Daten durch Compiler => Einschleusen von Befehlen
- Session Management: Abgreifen der Session ID => Session Hijacking
- Cross Site Scripting: Ausführen von Angriffscode durch Browser,
  z.B. aus Datenbank, reflexion aus Input
  (Cross Site: Angriff zwischen 2 Aufrufen der Seite)
- insecure direct object references/missing function level access control: privilegierte Objekte/Funktionen werden nur verborgen (presentation layer access control), nicht über authentifizierung geprüft
- (server) misconfiguration
- Sensitive Data Exposure: Leak z.B. in Logs
- Cross Site Request Forgery: Angreifer bringt Browser dazu, Anfrage an Funktion zu stellen (z.B. Bank);
  wenn User eingeloggt wird Session ID mit übertragen => Profit
  Gegenmaßnahme: Erzeugen und mitschicken von starken Tokens bei kritischen anfragen
- Vulnerable Components: fehler in libraries/modulen
- Unvalidated Redirects: z.B auf Phising-Sites oder um Überprüfungen herum

URL

$$\underbrace{http}_{schema}:\!/\!/\underbrace{user:password@}_{credentials}\underbrace{example.org}_{host}\underbrace{/demo/example.cgi}_{path}\underbrace{?a=foo\&b=bar}_{query}\underbrace{\#foobar}_{fragment}$$