WWW und Multimedia

Release 2015

Peter Meerwald

20 March 2015

Diese Materialien basieren weitgehend auf dem Buch Web Development (http://web-development.github.io) das ursprünglich von Brigitte Jellinek (http://brigitte-jellinek.at/) an der FH Salzburg (http://www.fh-salzburg.ac.at) zur Verwendung im Studiengang MultiMediaTechnology (http://multimediatechnology.at) geschrieben, und 2012 unter der CC-NC-SA (http://creativecommons.org/licenses/) Lizenz auf github (http://github.com) publiziert wurde.

Diese Version ist eine Überarbeitung von Peter Meerwald (http://pmeerw.net) für die Lehrveranstaltung WWW und Multimedia im Sommersemester 2014 an der Universität Salzburg (http://uni-salzburg.at), Fachbereich Computerwissenschaften (http://www.cosy.sbg.ac.at).

KAPITEL 1

Das Web - eine Einführung

Das erste Kapitel bietet einen theoretischen und praktischen Einstieg in das World Wide Web.

Was Sie wissen sollten

- Ich weiss, wer das World Wide Web erfunden hat, wer die Standards des Webs definiert
- Ich verstehe wie die drei Standards HTTP, URL und HTML zusammenspielen und das Web definieren.
- Ich verstehe wie eine HTTP Anfrage aufgebaut ist: Request und Response
- Ich weiss wie ein Dokument im HTML aufgebaut ist: head + body, tags, attribute, character entities
- Ich kenne die HTML-Tags für Überschriften, Absätze, Links, Bilder
- Ich verstehe, dass es mehrere Methoden des Uploads es gibt

Was Sie können sollten

- Ich kann ein einfaches Dokument im HTML erstellen, oder einen vorgegebenen Text in HTML umwandeln
- Ich kann die Korrektheit des HTMLCodes mit dem Validator des World Wide Web Consortium prüfen, und Fehler ausbessern bis der Code valide ist
- Ich kann die Dateien einer Webseite auf einen Webserver hochladen

Quellen und empfohlene Literatur

- Berners-Lee, Tim and Fischetti, Mark (1999): Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web. HarperBusiness. ISBN-10: 006251587X.
- Berners-Lee, Tim and Fischetti, Mark (1999): Der Web Report. Econ. ISBN-10: 3430114683.
- Hefner, Katie and Lyon, Matthew (1998): Where Wizards Stay Up Late: The Origins Of The Internet. Simon & Schuster. ISBN-10: 0684832674.
- Hefner, Katie and Lyon, Matthew (2008): ARPA Kadabra oder Die Anfänge des Internet. 3.Auflage. dpunkt Verlag. ISBN-10: 3898645517.
- Fielding, Roy T.; Gettys, James; Mogul, Jeffrey C.; Nielsen, Henrik Frystyk; Masinter, Larry; Leach, Paul J.; Berners-Lee (1999). RFC 2616: Hypertext Transfer Protocol – HTTP/1.1.

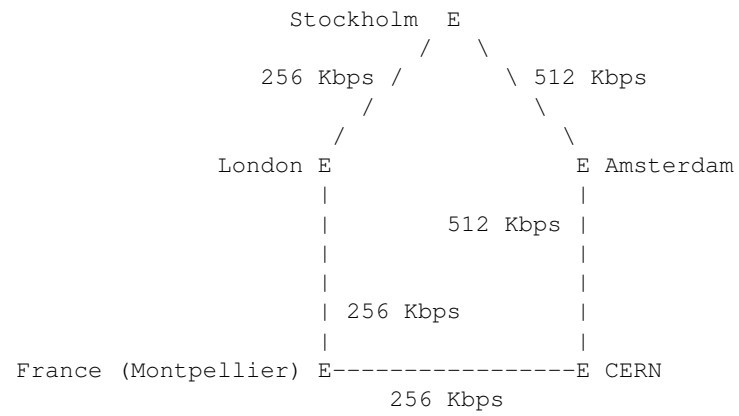
Eine Kurze Geschichte des World Wide Web

Viele moderne Erfindungen sind das Ergebnis von Teamarbeit und langen Planungsprozessen. Nicht so das World Wide Web: es geht auf eine einzelne Person zurück. **Tim Berners-Lee**, gebürtiger Engländer und studierter Physiker, hat es in den Jahren 1989, 1990, 1991 als EDV-Mitarbeiter am CERN in der Schweiz entwickelt.

Zu dieser Zeit – Anfang der 90er Jahre – war das Internet ein rein akademisches Projekt mit sehr geringen Bandbreite, wie folgendes „fact sheet“ aus der Zeit dokumentiert:

```
Date: Fri, 20 Mar 1992 13:50:09
From: dekker@rare.nl (Marieke G. Dekker)
Subject: ebone fact sheet, for your information

Ebone will operate a core backbone between London, Stockholm,
Amsterdam, Geneva, and France (Montpellier).
```



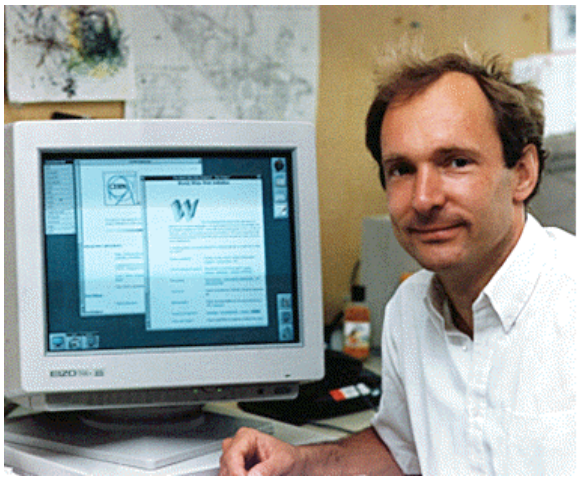

```
Intercontinental links to the United States are provided from London,
Stockholm and Geneva. The European links operate initially at speeds
between 256 and 512 kilobits per second.
```

Neben E-Mail und FTP wurden USEnet Newsgroups – ähnlich den heutigen Diskussionsforen – als wichtigste Form des Datenaustausches verwendet, meist im Textmodus ohne grafische Oberfläche.

Berners-Lee definierte als Eckpunkte die technischen Standards des Web (siehe nächstes Kapitel) und programmierte – gemeinsam mit KollegInnen – die erste Software. In der *Abbildung* ist links Tim Berners-Lee mit dem ersten grafischen Webbrowser abgebildet. Dieser Browser ist gemeinsam mit dem Betriebssystem NeXT verschwunden. Rechts ist ein Screenshot des Line Mode Browsers zu sehen, der die damalige Homepage des CERN anzeigt. Der Line Mode Browser funktionierte ohne Grafik. Links konnten mit Nummern aufgerufen werden.

Dieses Lehrbuch handelt hauptsächlich von der **Technik** des Web. Verlieren Sie dabei nicht aus den Augen, wie wichtig das Web ist, und wie sehr es unsere Gesellschaft schon verändert hat. Das Web ist heute überall:

Und das Web ist noch nicht “fertig erfunden”.



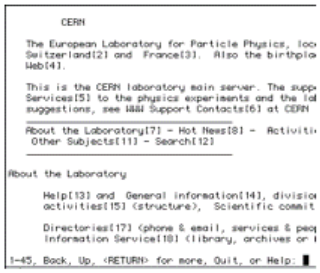


Abbildung 2.1: Abbildung: Der erste grafische Webbrowser, Tim Berners-Lee, die erste Webseite des CERN im Line mode browser

Abbildung 2.2: Abbildung: Das Web ist überall: vom ältesten zum neusten Gerät, in Bildung und Arbeit, 3. und 1. Welt.

Die Erfindung der maschinellen Massen-Produktion von Büchern durch Johannes Gutenberg im 15. Jahrhundert war nur der Auslöser, schaffte nur die Voraussetzung für weitere Erfindungen und große gesellschaftliche Veränderungen: Massenhafte Verbreitung der Bibel, Reformation, politisches Pamphlet, Revolution, wissenschaftliche Publikation, ...

Die Erfindung des Web um 1990 herum ist wahrscheinlich eine ähnlich fruchtbare Erfindung, die die Voraussetzung für viele Folge-Erfindungen schafft. Die gesellschaftlichen Auswirkungen können wir noch nicht absehen, wir sind mittendrin:

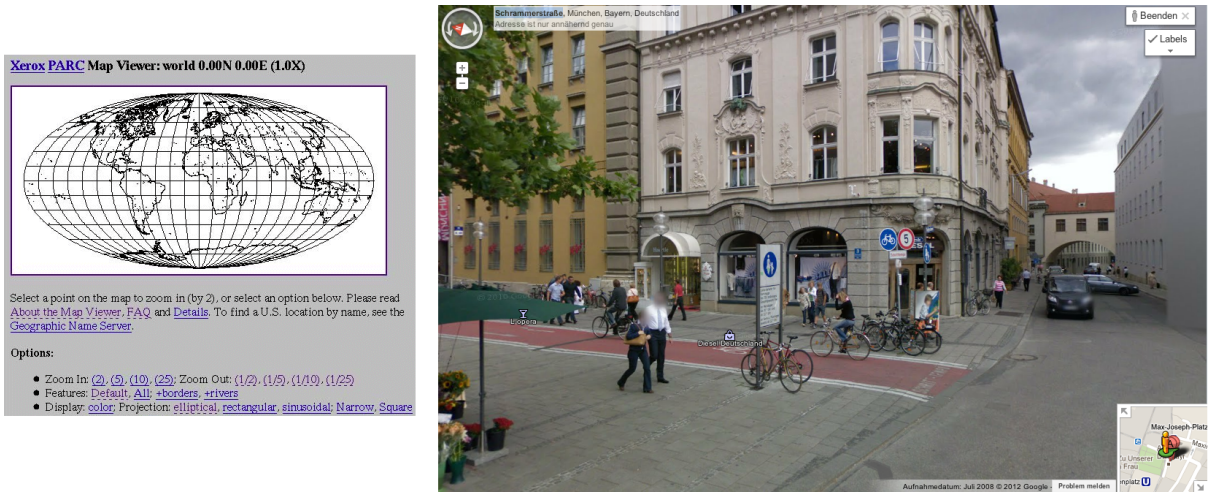


Abbildung 2.3: Abbildung: Interaktive Landkarten

Die interaktive Landkarte war eine sehr frühe Anwendung des Web: der “Xeroc Parc Map Viewer” ging 1993 online (https://en.wikipedia.org/wiki/Xerox_PARC_Map_Viewer). Heute ist eine Reise ohne Online-Karten kaum noch vorstellbar. Was kommt nach Streetview?

Abbildung 2.4: Abbildung: Buchhandeln im Internet: Amazon 1995 und Amazon 2012

Bücher über das Web zu verkaufen erscheint im Nachhinein als ganz einfache Idee. Es hat den Buchmarkt weltweit verändert, und wird es weiter tun. Wann werden E-Books die Papier-Bücher ablösen? Wer braucht noch eine Buchhandlung in der Kleinstadt, wenn Amazon doch überallhin liefert?

Im Jahr 2000 war eine Enzyklopädie noch das teuer bezahlte Werk von wenigen ExpertInnen. 2001 wurde Wikipedia gestartet. Die Erfindung der Wikipedia hat nicht nur den Markt für Enzyklopädien zerstört, sondern auch unsere Vorstellung verändert wie Wissen gesammelt werden kann.

Abbildung 2.5: Abbildung: Wikipedia 2001 und 2012

Abbildung 2.6: Abbildung: Facebook 2004 und 2010

Dass ich am Web speichern kann wer meine FreundInnen sind, um dann mit diesen FreundInnen kurze “Status-Meldungen” auszutausche - das wurde erst im 21. Jahrhundert erfunden. Wie geht es weiter?

KAPITEL 3

Drei Standards definieren das Web

Für diese Lehrbuch ist eine technische Sicht auf das World Wide Web die relevante. Das World Wide Web ist also ein verteiltes (Client/Server) Informationssystem, das durch folgende drei Standards in der jeweils aktuellen Form definiert wird:

- URL (Uniform Resource Locators) nach RFC 1738 (http://www.w3.org/Addressing/rfc1738.txt)
- HTTP (Hypertext Transfer Protocol) nach RFC 2616 (http://www.w3.org/Protocols/rfc2616/rfc2616.html)
- HTML (Hypertext Markup Language)

Wir befinden uns gerade im Übergang auf HTML 5. Noch ist der HTML 5 Standard nicht ganz fertig und ganz offiziell - aber er wird schon von den gängigen Browsern umgetzt.

Rund um diese drei Standards ordnen sich weitere wichtige Begriffe an:

3.1 Client/Server

Das Client / Server Prinzip ist ein allgemeines Prinzip wie Dienste in einem Computernetzwerk aufgebaut sein können: Ein Server ist ein Computer der einen bestimmten Dienst anbietet, ein Client ist der „Kunde“, also der Computer der den Dienst in Anspruch nimmt. Nach diesem Prinzip funktionieren Web, E-Mail, FTP:

Dienst	Client	Server
Web	Webbrowser - lädt Webseiten vom Server und stellt sie dar	Webserver – liefert auf Anfrage die Webseiten
E-Mail	E-Mail Programm – lädt E-Mails vom Server, zeigt sie an, kann neue E-Mails an einen Server schicken der sie zustellt	Mailserver – speichert E-Mail in verschiedenen Postfächern, leitet E- Mail weiter (an den Server der EmpfängerIn)
FTP	FTP-Client – lädt Dateien von einem Server herunter oder auf einen Server hinauf	FTP-Server – speichert Dateien und hält diese zum Übertragen bereit

Eine Alternative zu Client/Server ist Peer-to-Peer. Dabei sind alle beteiligten Computer gleichberechtigt, es gibt keine verschiedenen Rollen. Nach diesem Prinzip funktionieren Datei-Tauschbörsen.

3.2 Webbrowser

Ein Webbrowser, oder kurz Browser, ist ein Programm, das bei Eingabe einer URL über HTTP eine HTML-Webseite laden und anzeigen kann, es ist also der Client zum World Wide Web. Es gibt sehr viele verschiedene Webbrowser. Die folgende Abbildung zeigt vier davon: den Browser „Mosaic“ , der im Jahre 1993 als zweiter Webbrowser mit grafischer Oberflächen stark zur Verbreitung des World Wide Web beigetragen hat, und die Browser Opera, Mozilla und Chrome (in Versionen aus verschiedenen Jahren).

Alle eben erwähnten Browser haben gemeinsam, dass sie auf einem typischen „Büro- Computer“ eingesetzt werden, einem Computer mit grafischer Oberfläche und einem Farb- Monitor. Es gibt aber auch „exotischere“ Browser. Die nächste Abbildung zeigt den Browser lynx, der nur Text darstellt, aber keine Bilder. Daneben sehen Sie

Abbildung 3.1: Abbildung: Webbrowser: Mosaic (1993), Opera (2004) und Mozilla (2004), Chrome (2011)

eine „Braille Ausgabezeile“, ein Gerät, das eine Zeile Text in eine Zeile Blindenschrift übersetzt. Mit diesem Webbrowser und diesem Ausgabegerät können Blinde im Web surfen.

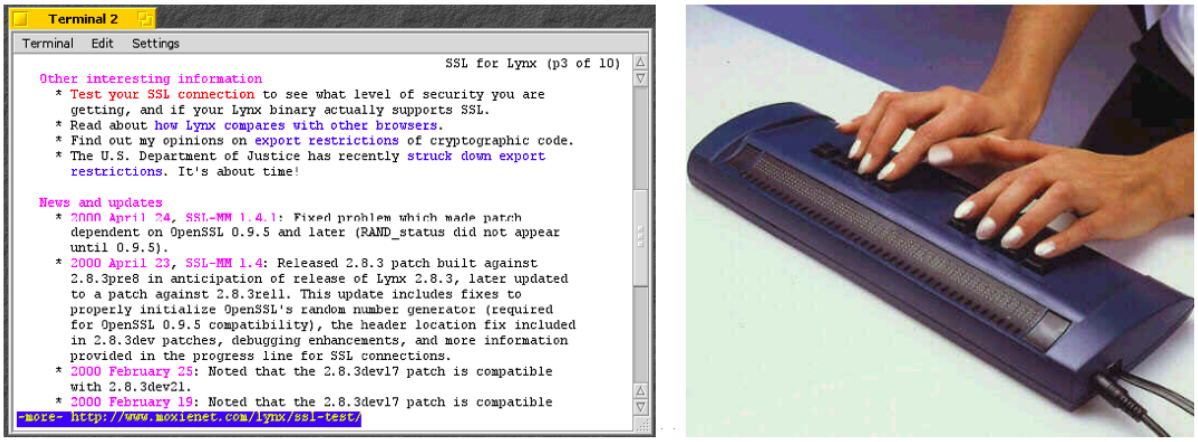


Abbildung 3.2: Abbildung: Text-Only Browser “links” und Braille Ausgabegerät

In den letzten Jahren haben mobile Geräte mit Internetzugang stark an Bedeutzung zugenommen: Smartphones und Tablets. Die nächste Abbildung zeigt Browser auf verschiedenen mobilen Geräten:

3.3 Webseite

Die Dokumente, die im Webbrowser dargestellt werden nennt man Webseiten. Eine Webseite ist – technisch gesehen – ein Dokument im HTML-Format.

Eine Webseite kann – im Gegensatz zu einer Seite in einem Buch – beliebig lang sein. Ist die Seite zu groß / zu lang für das Browser-Fenster, dann erscheint ein Scrollbalken mit dem man den Rest der Seite erreichen kann, wie in der nächsten Abbildung gezeigt.

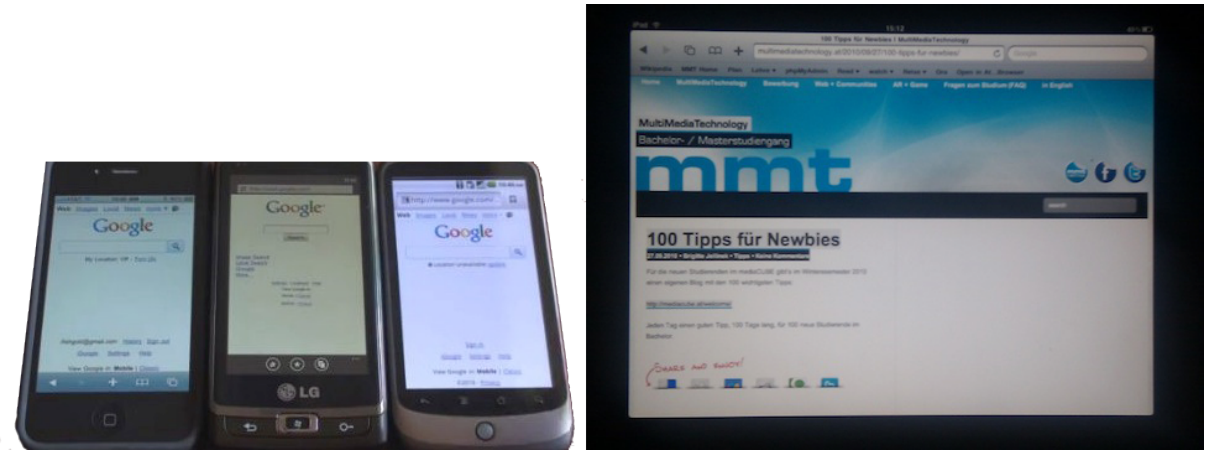


Abbildung 3.3: Abbildung: Browser auf mobilen Geräten: Handys und iPad, 2010

3.4 Website

Als Webseite wird also ein Dokument bezeichnet. Verwechseln Sie diesen Begriff nicht mit dem englischen Wort Website. Eine Website besteht aus mehreren Webseiten, die zusammen gehören und untereinander verlinkt sind. Achtung: es gibt kein Wort 'Webside'.

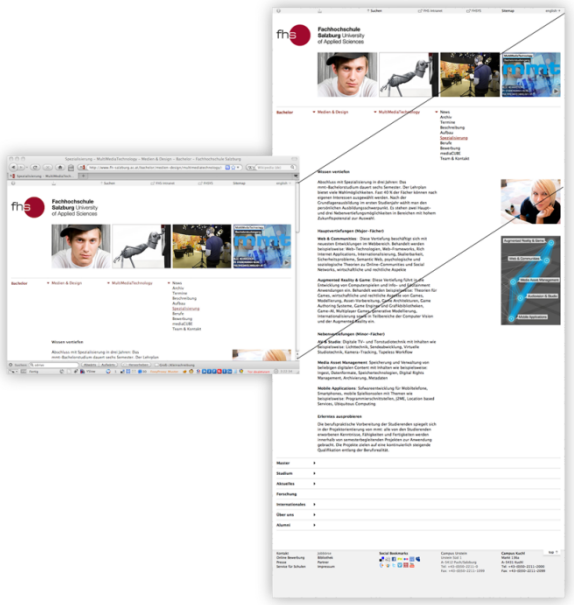


Abbildung 3.4: Abbildung: Ein Browser zeigt eine lange Webseite an

3.5 URL

Eine URL ist die Adresse eines Dokuments am Web oder in einem anderen Online-Informationssystem. Ein Beispiel:

http://uni-salzburg.at:80/index.php#top

Diese URL zerfällt in 5 Teile:

http das Übertragungsprotokoll

uni-salzburg.at Domain Name des Webservers

80 Port am Server. Wenn es Port 80 ist (default) kann man :80 weglassen.

/index.php Wird vom Webserver interpretiert, meist eine Pfad-Angabe. In diesem Fall aber nicht, da die Seite von einem PHP-Script erstellt wird.

#top Textmarke innerhalb des Dokuments – wird vom Browser interpretiert wenn das Dokument dargestellt wird um den zur Marke gehörenden Teil der Seite anzuzeigen

Im Zusammenhang mit Web-Formularen werden wir oft mit URLs zu tun haben die Parameter enthalten:

> http://www.google.com/search?q=schokolade&ie=utf-8&oe=utf-8

Mit den Fragezeichen, dem kaufmännischen Und und dem Ist-Gleich-Zeichen werden hier Parameter an die URL angefügt.

Parameter	Wert
q	schokolade
ie	utf-8
oe	utf-8

Das war nur eine informelle Darstellung der Syntax einer URL. Die ganze Wahrheit finden wir im Dokument RFC 1738 (http://www.w3.org/Addressing/rfc1738.txt). Dort wird die Syntax in Backus-Naur-Form (http://de.wikipedia.org/wiki/Backus-Naur-Form) beschrieben.

3.6 Backus-Naur-Form

Die Backus-Naur-Form sollten Sie auf jeden Fall lesen können. Ein kurzes BNF-Beispiel mit vier Ableitungsregeln:

```
studiengang = "MMA" | "MMT"
jahrgang  = studiengang "-" boderm jahr
boderm    = "B" |  "M"
jahr      = ziffer ziffer ziffer ziffer
ziffer    = "0" |  "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

Eine letzte Ableitungsregel liest man so: eine *ziffer* ist entweder eine *0* oder eine *1* oder ... oder eine *9*. Ein *jahr* ist vier ziffern hintereinander.

Leichter zu lesen ist das sogenannte “Railroad Diagram”:

Hier eine gekürzte Darstellung der http URL aus dem RFC1738:

```
httpurl        = "http://" login [ "/" hpath [ "?" search ]]

login          = [ user [ ":" password ] "@" ] hostport
hostport       = host [ ":" port ]
host           = hostname | hostnumber
hpath          = hsegment *[ "/" hsegment ]

user           = *[ uchar | ";" | "?" | "&" | "=" ]
password       = *[ uchar | ";" | "?" | "&" | "=" ]
port           = digits
hsegment       = *[ uchar | ";" | ":" | "@" | "&" | "=" ]
search         = *[ uchar | ";" | ":" | "@" | "&" | "=" ]
alphadigit     = alpha | digit

unreserved     = alpha | digit | safe | extra
uchar          = unreserved | escape
xchar          = unreserved | reserved | escape
alpha          = lowalpha | hialpha
digits         = 1*digit
```

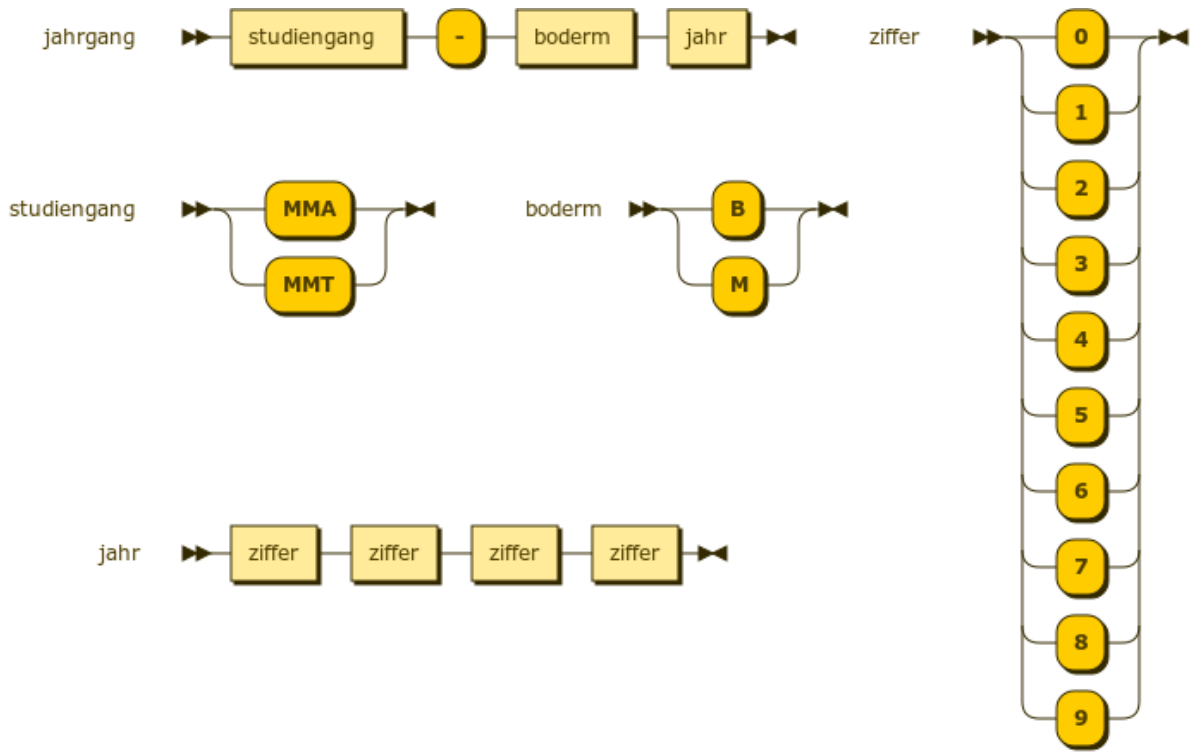


Abbildung 3.5: Abbildung: Railroad Diagramm einer BNF

```
safe            = "$" | "-" | "_" | "." | "+"
extra           = "!" | "*" | "'" | "(" | ")" | ","
national        = "{" | "}" | "|" | "\" | "^" | "~" | "[" | "]" | "`"
punctuation     = "<" | ">" | "#" | "%" | <">
reserved        = ";" | "/" | "?" | ":" | "@" | "&" | "="
hex             = digit | "A" | "B" | "C" | "D" | "E" | "F"
                        | "a" | "b" | "c" | "d" | "e" | "f"

escape          = "%" hex hex

lowalpha        = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" |
                  "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" |
                  "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" |
                  "y" | "z"
hialpha         = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" |
                  "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" |
                  "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"

digit           = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

HTTP-URLs sind nicht die einzigen URLs, ein Beispiel mit einem anderen Übertragungsprotokoll:

mailto:pmeerw@cosy.sbg.ac.at

3.7 HTTP

Das Hypertext Transfer Protocol ist ein relativ simples Protokoll, das immer TCP-Verbindungen verwendet. Alle Aktivität wird vom Client (= Browser) initiiert. In der einfachsten Form sieht HTTP so aus (hier 9 Schritte am Beispiel der URL http://uni-salzburg.at/index.php?id=39957#top):

1. Eine URL wird in den Browser eingetippt, oder ein Link wird im Browser angeklickt

2. Der Browser analysiert die URL, stellt fest dass es sich um HTTP handelt und ermittelt daraus den Domain Namen des Webservers (*uni-salzburg.at*), löst diese über DNS zur IP-Adresse auf, das Ergebnis lautet *141.201.80.15*
3. Der Browser baut eine TCP-Verbindung zu *141.201.80.15*, Port 80 auf
4. Er sendet einen HTTP-Request: *GET /index.php HTTP/1.0\n\n*
5. Der Webserver nimmt die Anfrage entgegen und analysiert sie. Meistens interpretiert er sie als Aufforderung, eine bestimmte Datei von der Platte zu lesen. In diesem Fall aber wird ein PHP Programm gestartet, dass Daten aus einer Datenbank (einem Content-Management-System, CMS) abfragt und als HTML aufbereitet.
6. Der Webserver schickt einen HTTP-Response an den Browser, diese enthält einen Statuscode, z. B. *200 OK*, einige Zusatzinformationen und dann die eigentlichen Daten des Dokuments (den erzeugten HTML-Code)
7. Der Browser nimmt das Dokument in Empfang und stellt es dar
8. Der Browser scrollt das Dokument bis zur Textmarke *top*
9. Der Browser beendet die TCP-Verbindung

Die nächste Anfrage des Clients kann sich an einen anderen Server, oder wieder an denselben Server richten. Die nächste Anfrage, die der Server beantwortet, kann vom selben Client kommen, oder von einem anderen Client. Keiner der beiden (Client und Server) muß speichern mit wem er gerade Daten ausgetauscht hat, um die nächste Anfrage durchführen/beantworten zu können. Ein Protokoll mit dieser Eigenschafft nennt man “zustandslos” (“stateless”). Dadurch ist es sehr einfach einen Server zu programmieren.

Das war ein sehr einfaches Beispiel, wie das Protokoll ablaufen kann. Einen tieferen Einblick in HTTP erhalten Sie im Kapitel [http](/http/).

KAPITEL 4

HTML Grundkurs

Wir werden HTML in der Version 5 verwendet. Die wichtigsten HTML-Tags (Links, Bilder, Tabellen, Formulare) werden Sie bald auswendig können. Alle Details können Sie Internet nachschlagen.

HTML hat sich in den ca. 20 Jahren seines Bestehens weit entwickelt. Die Version 5, die wir verwenden, ist ein relativ neuer Standard, der aber in den aktuellen Versionen der gängigen Browsern bereits umgesetzt ist.

In den letzten 10 Jahren wurde auch XHTML neben HTML verwendet. Das X von XHTML steht für die Kompatibilität mit XML. Die Dateien haben dabei weiterhin die Endung .htm oder .html (nur sehr selten oder .xhtm, .xhtml). Im Unterschied zu HTML war XHTML strenger in der Schreibweise. Am Web finden Sie sowohl Tutorials zu HTML als auch zu XHTML – lassen Sie sich davon nicht verwirren!

4.1 Arbeitsablauf

Der Arbeitsablauf beim Erstellen von HTML ist eine (hoffentlich nicht) endlose Schleife:

1. Code im Editor eintippen
2. Abspeichern
3. Zum Browser wechseln
4. Neu Laden, das Ergebnis betrachten
5. meistens: nicht zufrieden mit dem Ergebnis sein, zurück zu 1.

Es gibt verschiedene Online-Tools um HTMl auszuprobieren:

- Slowparse (http://toolness.github.com/slowparse/demo/) erklärt die Struktur von HTML, hilft Fehler zu finden

4.2 Zeichensatz

HTML-Dateien bestehen aus reinem Text, in unserem Falle entweder aus Text im Format ISO 8859-1 (Latin-1) oder im Format UTF-8. Mit Latin-1 ist man auf das lateinische Alphabet mit westeuropäischen Sonderzeichen beschränkt - man kann im selben Dokument nicht auch grieschische, hebräische, arabische, japanische Zeichen darstellen. Mit UTF-8 hat man den gesamten Zeichensatz der Menschheit zur Verfügung. Mein Empfehlung lautet: immer UTF-8 verwenden.

Diese Abbildung zeigt wie die character sets in Windows- und Mac-Programmen dargestellt werden. Links im Bild, in Notepad, wird ISO 8859-1 als ANSI bezeichnet. Rechts im Bild, im Mac-Programm Textwrangler, ist die Bezeichnung klarer.

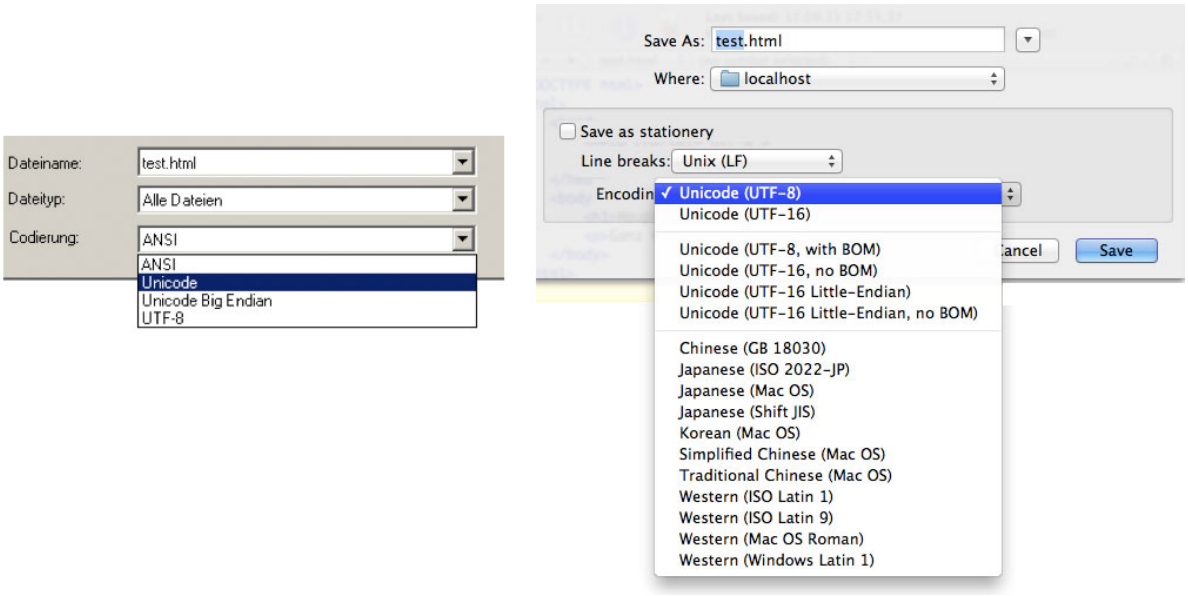


Abbildung 4.1: Abbildung: Auswahl des Charactersets „Unicode“

4.3 Sonderzeichen

Sonderzeichen in HTML nennt man „character entities“. Sie haben die From *&entityname;*.

Folgende Character Entities **müssen** Sie verwenden wenn Sie das betreffende Zeichen in HTML darstellen wollen:

Gewünschtes Zeichen	Character Entity
<	<
>	>
&	&
“	"
‘	'

Folgende Character Entities können Sie vermeiden, indem Sie das Zeichen selbst im Code verwenden:

Gewünschtes Zeichen	Character Entity
Ä	Ä
ä	ä
ß	ß
...	...

4.4 HTML-Tags

Die gewünschte Formatierung bzw. Strukturierung des Textes wird mit HTML-Tags angegeben.

1. Tags sind zwischen spitzen Klammern eingeschlossen (kleiner-gleich und größer-gleich Zeichen).
2. Zu (fast) jedem “Anfangs-Tag” gibt es einen “End-Tag”, der sich nur durch den Schrägstrich vom Anfangs-Tag unterscheidet. Z. B. * hier der Absatz *.
3. Nur in XHTML müssen “alleinstehende” Tags mit einem Schrägstrich am Ende geschrieben werden: *
* In HTML5 schreibt man diesen Tag (wieder) als *
*
4. In XHTML werden Tags immer klein geschrieben, bei HTML ist die Groß- oder Kleinschreibung egal.

4.5 Whitespace

Sogenannter "whitespace" - das sind mehrere Leerzeichen, Tabulatoren und Zeilenumbrüche hintereinander – wird vom Browser völlig ignoriert. Ob Sie also in Ihrer HTML-Datei ein Leerzeichen oder 7 Leerzeilen einfügen macht keinen Unterschied. (Gar kein Leerzeichen oder zumindest ein Leerzeichen macht schon einen Unterschied!) Die folgenden beiden Dokumente sind also äquivalent:

```
<p>Halli
Hallo</p>
```

bzw.:

```
<p>
Halli
Hallo
</p>
```

4.6 Zeilenumbruch

Nur die Tags beeinflussen die Darstellung der Webseite. Sie müssen den Tag <*br*> verwenden um einen Zeilenumbruch auf der Webseite zu erzwingen – dies ist aber nur selten sinnvoll, da der Browser einen automatischen Zeilenumbruch durchführt, um den Text im vorhandenen Platz optimal darzustellen.

4.7 Attribute

Manche Tags können Attribute enthalten. Ein Beispiel ist der Tag <*img*> der ein Bild in die Seite einfügt (Englisch: Image). Die wichtigsten Attribute von <*img*> sind *src* (von Source = Quelle) und *alt* (Alternative Darstellung, Ersatztext).

```
<img src="neu.gif" alt="Das ist neu!">
```

Es ist egal, in welcher Reihenfolge Sie die Attribute schreiben:

```
<img alt="Das ist neu!" src="neu.gif">
```

bzw.

```
<img alt="Das ist Neu!"
    bli="bla, blo"
    src="neu.gif"
>
```

4.8 Unbekannte Tags und Attribute

Das Attribut *bli*, welches nicht zu HTML gehört, also kein Browser kennt, wird wie andere unbekannte Attribute einfach ignoriert.

Der Wert eines Attributes muß in XHTML immer in Anführungszeichen geschrieben werden, in HTML kann man die Anführungszeichen weglasse, dann endet der Wert beim nächsten Leerzeichen.

```
<img alt="Das ist Neu!" src="neu.gif"
    width=50 height=15>
```

Warnung: Achtung, ein häufiger Fehler ist es, das zweite Anführungszeichen zu vergessen!

Falsch ist etwa

```
<img alt="Das ist neu!" src="neu.gif >
```

oder

```
<img alt="Das ist neu! src=neu.gif" >
```

4.9 Kompatibilität

Jeder Browser (egal ob Chrome, Firefox, Safari, Microsoft Internet Explorer, Opera, ...) kann jedes HTML-Dokument darstellen. Mit jedem Editor (egal ob Notepade, vim, Dreamweaver, Eclipse, ...) kann jedes HTML-Dokument bearbeitet werden.

So soll das Web funktionieren. (Die schweren Fälle, wo es manchmal doch nicht funktioniert, verschieben wir auf etwas später.)

Neue Tags, die z. B. ein Browser nicht erkennt, sind zu ignorieren – es gibt **keine Fehlermeldungen**!

Wenn das W3C in HTML Version 21 beispielsweise den neuen Tag <*jump*> (hüpfenden Text) einführen würde, können Sie diesen Tag auf Ihrer Webseite verwenden. Der Text würde jedoch nur in den neuesten Versionen der Browser (die schon die Version 21 kennen) hüpfend dargestellt. In älteren Browsern wird der Tag <*jump*> ignoriert.

```
<p>Dies ist meine erste
<jump>supercoole</jump>
Webseite!!!!</p>
```

Bevor Sie also einen neuen Tag einsetzen, sollten Sie sich darüber klar sein, welche Versionen der Browser diesen Tag darstellen und wie viel Prozent Ihres Zielpublikums schon eine passende Browser-Version benutzen. Meist können Sie eine Seite so gestalten, dass auch mit älteren Browsern der gesamte Inhalt lesbar ist.

In älteren Browser geht die Information hier verloren:

```
<p>Zu den hüpfenden Terminen
sind noch Plätze im Kurs frei:
</p>
<ul>
<li><jump>Montag</jump></li>
<li>Dienstag</li>
<ul>
```

Jedoch kann in allen Browser die Information wie folgt dargestellt werden (durch zusätzliche ***)

```
<p>Zu den markierten Terminen sind
noch Plätze im Kurs frei</p>
<ul>
<li><jump>Montag ***</jump></li>
<li>Dienstag</li>
</ul>
```

Diese Herangehensweise an Neuerungen nennt man „graceful degradation“ – davon werden Sie noch viel hören.

Bemerkung: Hinweis: Der <*jump*>–Tag ist ein Scherz, den gibt es nicht wirklich, und wird es hoffentlich nie geben.

4.10 Text formatieren

Wir unterscheiden zwischen HTML-Tags die Blöcke definieren, und solchen die das nicht tun. Blockbildende Tags beanspruchen immer einen rechteckigen Bereich bei der Ausgabe, nicht blockbildende Tags tun das nicht.

Der *em*-Tag ist nicht-blockbildend und wird im zweiten Absatz auf zwei Zeilen umgebrochen.

Abbildung 4.2: Abbildung: Darstellung von blockbildenden (grau hinterlegten) und nicht-blockbildenden (rot hinterlegten) tags

4.10.1 Blockbildende Tags für Text

Normaler Fließtext:

```
<p>Text text text, text text. Text
text text, text text. Text text
text, text text. Text und text text,
text text. </p>
```

In folgendem Beispiel wird schon ein bisschen CSS verwendet, hier mit Hilfe des Attributes *style*. CSS ist für die Darstellung von HTML zuständig, hier verwenden wir es um den Text im Absatz zu zentrieren.

Zentrierter Text:

```
<p style="text-align:center;">Text
text text, text text. Text text
text, text text. Text text text,
text text. Text und text text, text
text. </p>
```

Überschriften werden für die Strukturierung des Textes verwendet, mehr als 3 Ebenen braucht man selten:

```
<h1>Überschriftstext</h1>
<h2>Überschriftstext</h2>
<h3>Überschriftstext</h3>
```

Neu in HTML5 ist der *nav* Tag zur Auszeichnung von Navigations-Menüs. Achtung: dieser Tag hat erst mal keine sichtbare Wirkung.

Bereich mit Navigations-Menü, Links:

```
<nav>
   <a href="index.html">Home</a>
   ....
</nav>
```

Auch die folgenden drei Tags haben keine sichtbare Wirkung, sondern dienen erst mal nur zu Strukturierung des Dokuments.

```
<header> ... </header>
<footer> ... </footer>
<aside> ... </aside>
```

Für umfangreiche Zitate gibt es einen blockbildenden Tag (eingerückt):

```
<blockquote>Alle meine Entchen<br>
schwimmen auf dem See</blockquote>
```

4.10.2 Nicht-Blockbildende Tags für Text

Diese müssen Sie innerhalb eines Blocks verwenden. Diese Formate werden unterschieden in logische und physische Elemente.

Die logischen Tags geben die genaue Darstellung nicht vor.

Gewünschte Darstellung	Code
Sehr stark betonter Text (meist fett)	Eine <*strong*> **wichtige** </*strong*> Sache
Betonter Text (meist kursiv)	und eine <*em*> *interessante* </*em*> Sache

Physische Tags geben die genaue Darstellung vor. Das sind eigentlich „altmodische Tags“, besonders der *font*-Tag wurde schon um das Jahr 2000 herum durch Stylesheets vollständig ersetzt, und sollte möglichst nicht mehr verwendet werden. Sie werden diese Tag in „alten“ Dokumenten aber noch finden:

Veraltete Tags zum Formatieren von Text”:

```
Eine <b>fette</b> Sache
Eine <i>schräge</i> Sache
<font face="Arial">Text</font>
<font color="red">rot</font>
```

Heute verwendet man CSS im *style*-Attribut statt dessen, aktuelle Tags zum Formatieren von Text”:

```
<span style="font-family:Arial;">Text</span>
<span style="color:red;">rot</span>
```

4.11 Bilder

Bilder werden in separaten Dateien gespeichert, in der HTML-Datei erfolgt nur ein Verweis auf die Datei des Bildes. Als Attribut *src* im *img*-Tag können Sie eine absolute oder relative URL angeben:

```
<img src="http://www.google.at/intl/de_at/images/logo.gif" alt="Google">
<img src="neu.gif">
<img src="10prozent.gif" alt="jetzt 10% verbilligt!">
(mit Ersatztext, weil das Bild wichtige Information enthält)
<img src="zierleiste.gif" alt="">
(ohne Ersatztext, weil das Bild zur zur Dekoration dient)
<img src="neu.gif" style="float:left;" alt="neu">
(Bild nach links, Text fließt rechts vorbei)
```

Für Blinde, Suchmaschinen, Browser die keine Bilder darstellen können, usw. gibt man für jedes Bild einen Alternativtext an. Mit dem Firefox AddOn „Web Developer“ kann man testen wie die Seite mit *alt* statt Bildern aussieht. Die Abbildung zeigt die Verwendung dieses Features am Beispiel eines Wetterberichts.

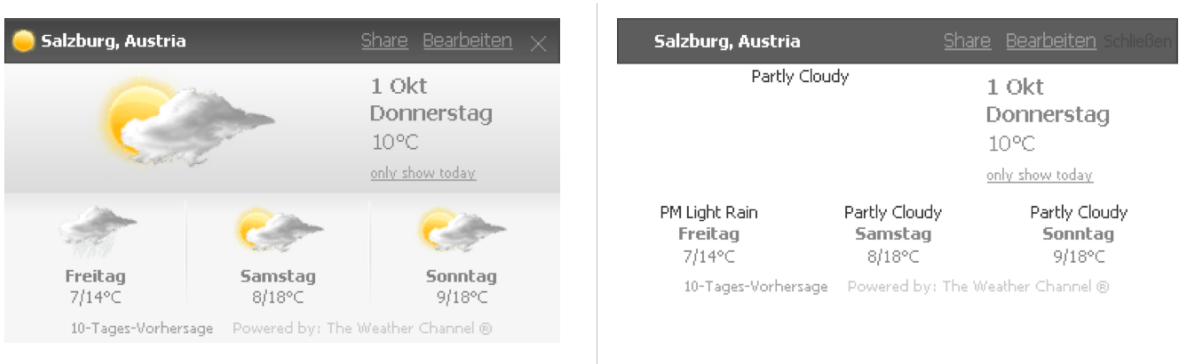


Abbildung 4.3: Abbildung: Wetterbericht mit Bildern und ohne Bilder (nur *alt*-Texte)

Als Datenformate für *<img>*-Bilder werden drei Pixel-Formate von vielen Browsern unterstützt, erst seit kurzem auch das Vektor-Format SVG:

GIF Palette von 255 Farben + 1 Transparenz-Farbe (kein Alpha). „animiertes Gif“ enthält mehrere Bilder, die der Reihe nach angezeigt werden (Daumenkino). Besonders geeignet für Bilder mit klaren Kanten, einfärbigen Flächen, wenigen Farben, z. B. Comics, Logos.

JPEG, JPG Millionen von Farben, verlustbehaftete Kompression, keine Transparenz. Besonders geeignet für Bilder mit Farbverläufen, z. B. Photos.

PNG Verlustfreie Kompression, mit Alpha-Transparenz.

SVG Vektor-Format, das Bild kann beliebig groß oder klein dargestellt (‘skaliert’) werden.

Ideal wäre, das Format auszuwählen, in dem das Bild nicht an Qualität verliert, und die Dateigröße möglichst gering ist.

Weitere Arten Bilder zu erstellen werden wir später genauer betrachten: Mit ‘responsive images’ kann man verschieden große Bilder für verschiedene Ausgabegeräte anbieten. Mit dem *canvas*-Tag und Javascript kann man Bilder zeichnen.

4.12 Links

Als Attribut *href* des Link kann eine absolute oder relative URL angeben werden:

```
Link zu Webseite (absolute URL)
<a href="http://cnn.com">zu CNN</a>
Link zu Webseite im selben Ordner (relative URL)
<a href="seite2.htm">mehr</a>
Link zu E-Mail Adresse
<a href="mailto:pmeerw@gmail.com">Mail</a>
Bild als Link
<a href="seite2.htm"><img src="mehr.gif"alt="zur Seite 2"></a>
```

4.13 Gesamt-Struktur einer Webseite

Eine HTML Seite hat ein Grundgerüst aus *DOCTYPE*, *html*, *head* und *body*. Der erste Tag ist der *DOCTYPE*, er gibt die Version von HTML an, die verwendet wird. Es folgt der *html*-Tag. Innerhalb des *html*-Tags gibt es erst einen *head* und dann einen *body*-Tag – nicht mehr und nicht weniger. Hier am Beispiel von HTML5:

Grundgerüst eines HTML5 Dokuments:

```
<!DOCTYPE HTML>
<html>
<head>
    <meta charset="utf-8">
    <title>Eine Webseite</title>
</head>
<body>
    Hier die Tags, die Sie schon kennen
</body>
</html>
```

Bei XHTML ist der DOCTYPE sehr viel komplizierter. Der *meta*-Tag für die Auswahl des Charactersets ist ebenfalls komplizierter.

Grundgerüst eines XHTML Dokuments”:

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>Eine Webseite</title>
</head>
<body>
     Hier die Tags, die Sie schon kennen.
</body>
</html>
```

Warnung: Achtung: es kann nur einen *head* und einen *body* geben, keine Wiederholungen!

4.14 Listen

Listen werden mit zwei verschachtelten Tags gebaut: dem *li*-Tag für einen einzelnen Listen-Punkt, und dem *ol*- oder *ul*-Tag für die gesamte geordnete bzw. ungeordnete Liste.

Ungeordnete Liste (mit 'bullet points'):

```
<ul>
    <li>punkti</li>
    <li>punkti</li>
</ul>
```

Geordnete Liste (mit Nummerierung):

```
<ol>
  <li>eins</li>
  <li>zwei</li>
  <li>drei</li>
</ol>
```

4.15 Tabellen

Um eine Tabelle zu erzeugen müssen Sie die Tags ´table´, ´tr´ (Table Row = Tabellenzeile) und ´td´ (Table Data) richtig ineinander verschachteln.

```
<table style="border-collapse:collapse;">
  <tr>
    <td>Freitag</td>
    <td>Samstag</td>
    <td>Sonntag</td>
  </tr>
  <tr>
    <td>lernen</td>
    <td>lernen</td>
    <td>lange schlafen</td>
  </tr>
</table>
```

Der Rahmen der Tabelle ist normalerweise unsichtbar, mit einer Zeile CSS wird er hier auf sichtbar geschalten:

```
<style>td {  border:2px #ddd solid; padding: 5px; }</style>
```

HTML-Tabellen wurden früher in Webseiten für das Layout der Seite verwendet, diese Tabellenlayouts sind noch auf älteren Webseiten zu finden. Dazu noch ein Literaturhinweis: http://shouldiusetablesforlayout.com/

Moderne Webseiten werden mit CSS-Layouts gestaltet. Heute verwendet man Tabellen wirklich nur noch zur Darstellung von Tabellen.

4.16 Weitere Quellen

- Characters, Symbols and the Unicode Miracle, YouTube Video von Computerphile http://www.youtube.com/watch?v=MijmeoH9LT4
- UTF-8 in der deutschen Wikipedia https://de.wikipedia.org/wiki/UTF-8

Validator und Uplaod

Der HTML-Code ist fertig, aber damit ist die Arbeit noch lange nicht vorbei.

5.1 Validator

Um Fehler im HTML-Code zu finden reicht ein Webbrowser nicht aus. Wie oben beschrieben sind Webbrowser sehr tolerant, was kaputten Code betrifft. Für eine strenge Prüfung des HTML-Codes kann man den Validator des W3C verwenden:

http://validator.w3.org/

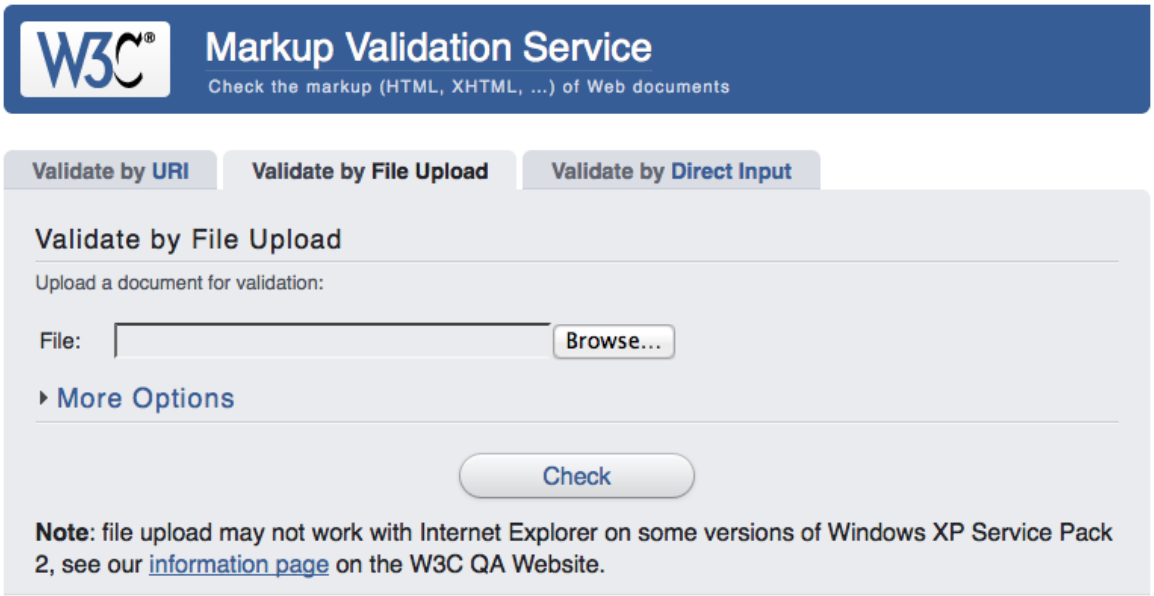


Abbildung 5.1: Abbildung: Der HTML-Validator des World Wide Web Consortiums

Die Fehlermeldungen des Validators muss man sorgfältig lesen, von oben nach unten durcharbeiten und reparieren. Mit etwas Glück repariert man das erste Problem und viele Folge-Fehler fallen damit auch weg.

Die Fehlermeldung in der Abbildung geht auf einen Tippfehler zurück: der schließende Tag wurde falsch geschrieben.

Wenn der Code dem HTML-Standard entspricht erscheint eine Erfolgsmeldung:

Bei der Verwendung von HTML5 erscheint aber immer noch ein Warning, dass der Validator experimentell sei. Das kann man ignorieren.

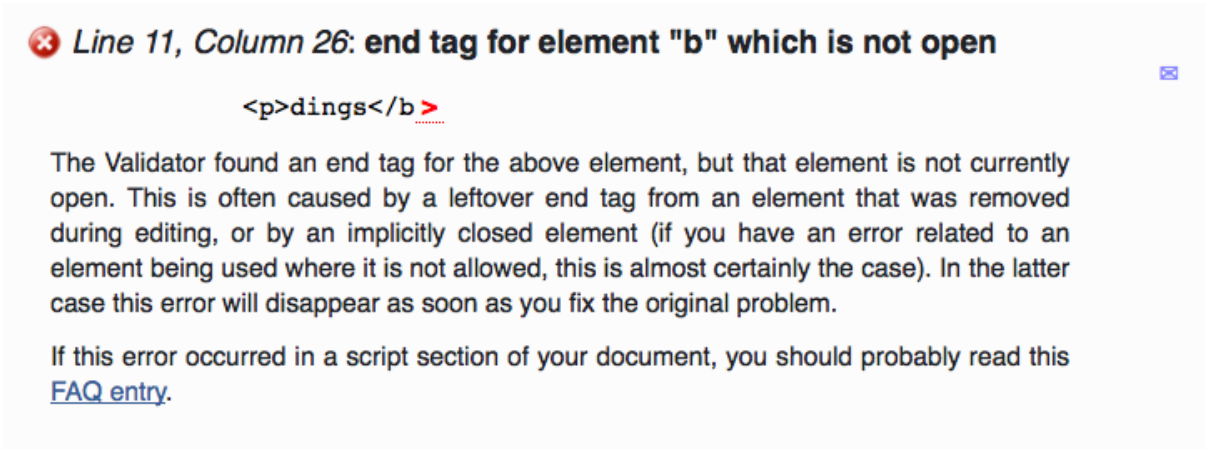


Abbildung 5.2: Abbildung: Fehlermeldung des Validators

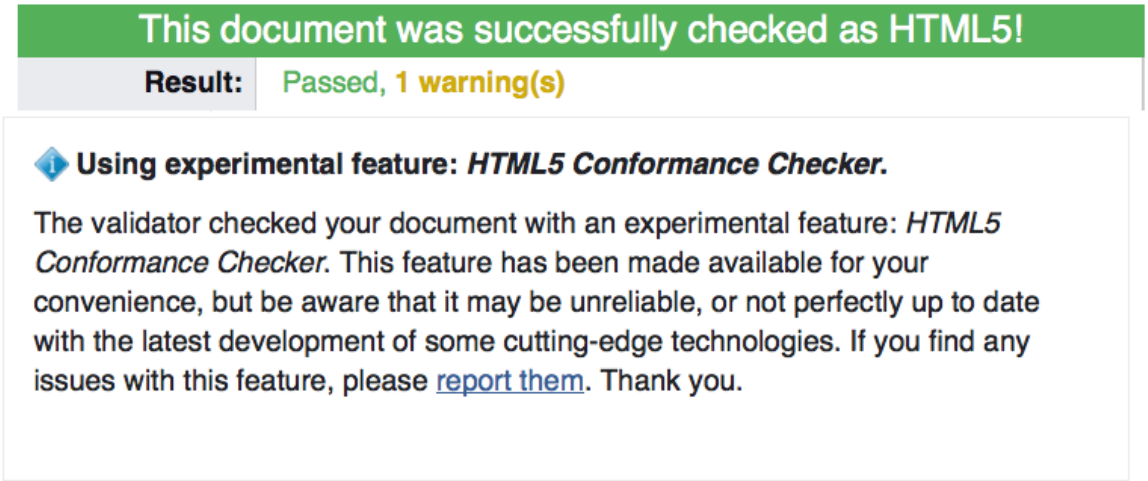


Abbildung 5.3: Abbildung: Erfolgsmeldung des Validators

5.2 Upload

Die fertige Webseite muss auf einem Webserver veröffentlicht werden. Um die Daten vom eigenen Computer auf den Webserver zu laden gibt es verschiedene Methoden, die wichtigsten sind FTP, WebDAV und SFTP.

Bei jeder Upload-Methode brauchen Sie folgende Informationen: den Namen des Servers, Usernamen, Passwort, in welchen Ordner Sie die Daten speichern, unter welcher URL die Daten im Web sichtbar sind.

Ein hypothetisches Beispiel: Auf dem Server meinhoster.at ist auch ihre Domain meinefirma.at untergebracht. Die Konfiguration ist wahrscheinlich so ähnlich:

- Name des SFTP-Servers: meinhoster.at
- Port für SFTP 4711
- Username + Passwort
- Ordner bei Upload public_html/
- URL http://meinefirma.at

Ein paar Programme zum Upload:

- Secure File Transfer Client (http://winscp.net): nur für SFTP, nur auf Windows.
- FireFTP (https://addons.mozilla.org/en-US/firefox/addon/fireftp/), ein AddOn von Firefox, Windows und Mac.
- Dreamweaver (http://www.adobe.com/products/dreamweaver.html): Upload ist nach der Definition einer „Site“ möglich, auf Windows und Mac.

Achten Sie beim Upload darauf, dass die Ordnerstruktur und die relative Position der Dateien beibehalten wird; nur dann funktionieren die relativen Links!

Dreamweaver und FireFTP bieten Hilfe beim Erhalt der Struktur. In der folgenden Abbildung wurde rechts (am lokalen Computer) eine Datei ausgewählt und dann der Button „Datei bereitstellen“ gedrückt. Dreamweaver beachtet, dass die Datei lokal im Ordner html-ue1 liegt, und lädt Sie in den entsprechenden Ordner am Webserver hoch.

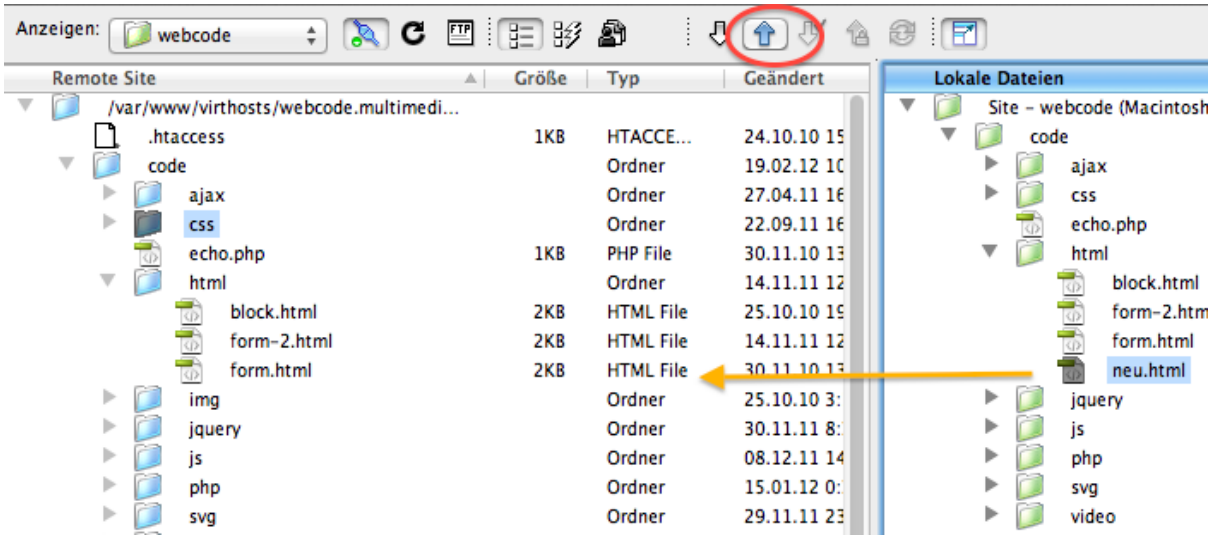


Abbildung 5.4: Abbildung: Site-Fenster von Dreamweaver – Upload einer Datei automatisch in den richtigen Ordner

Diese Abbildung zeigt, was Sie nicht tun sollten: die Datei mit Drag-and-Drop in den falschen Ordner am Webserver laden. Dann funktionieren die relativen Links nicht mehr.

Die anderen erwähnten Programme funktionieren sehr ähnlich, bieten aber etwas weniger Automatik.

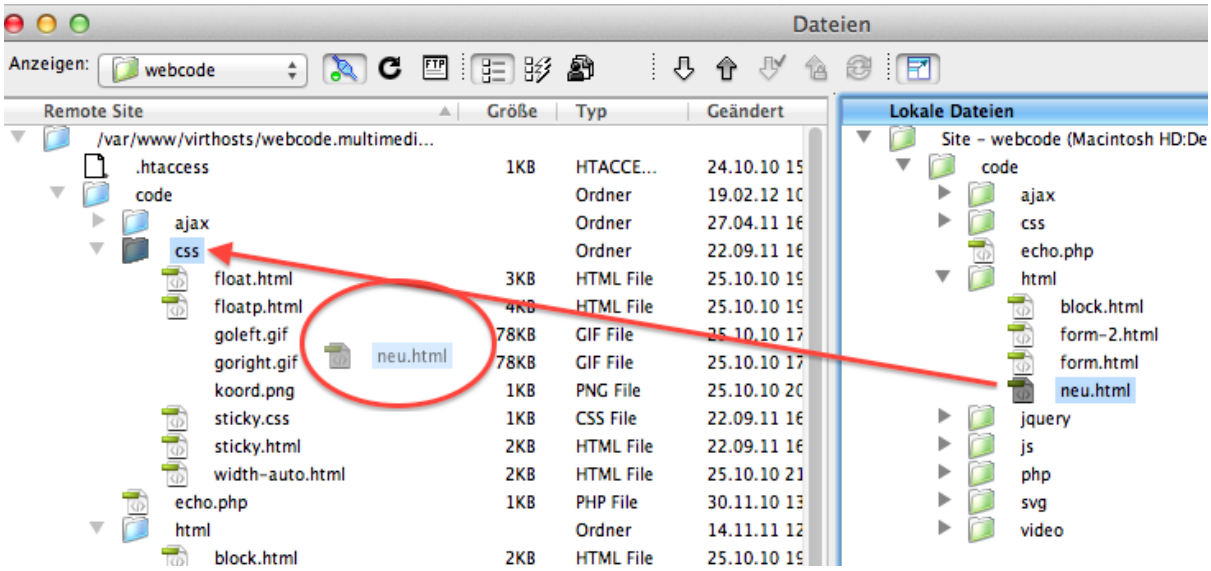


Abbildung 5.5: Abbildung: Upload mit Dreamweaver in den falschen Ordner

Immer gilt: mit dem Browser nachprüfen, ob die Webseite nach dem Upload wirklich unter der richtigen URL erreichbar ist.

KAPITEL 6

URLs - die Adressen des World Wide Webs

Was Sie wissen sollten

- Ich kenne die absolute und relative Schreibweise von URLs, kann dieses lesen und hin- und her-übersetzen
- Ich kenne die Rolle des Webservers in Bezug auf URLs

Was Sie können sollten * Ich kann in einem Apache Webserver die Anzeige einer Default-Datei für einen Ordner konfigurieren, oder die Auflistung des Ordners. * Ich halte bei meinen Web-Projekten eine Konvention zu Ordner- und Dateinamen ein, und kann die Vor- und Nachteile meiner Konvention erläutern.

Absolute und relative URLs

Verwendung

URLs werden in CSS und HTML an vielen Stellen verwendet: bei Links und Bilder, beim Verweis auf externe Stylesheets, externe Javascript-Dateien, etc. An all diesen Stellen können Sie URLs in den hier beschriebenen Schreibweisen verwenden.

Die „absolute“ und „relative“ Schreibweise wurde von den Pfadangaben im UNIX-Dateisystem übernommen; auch die Pfadangaben von Windows funktionieren ähnlich.

Beispiel

In der Beispiel-Datei (http://pmeerw.net/www-mm14/images/urls.html) unten

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Beispiel-Seite für relative und absolute URLs</title>
</head>
<body>
    <h1>Relative und absolute URLs</h1>

    <h2>absolut</h2>
    <p><kbd>http://pmeerw.net/www-mm14/images/you_are_here.jpg</kbd></p>
    <p><img src="http://pmeerw.net/www-mm14/images/you_are_here.jpg"></p>

    <h2>relativ zum Webserver</h2>
    <p><kbd>/www-mm14/images/you_are_here.jpg</kbd></p>
    <p><img src="/www-mm14/images/you_are_here.jpg"></p>

    <h2>relativ zum HTML-Dokument</h2>
    <p><kbd>you_are_here.jpg</kbd></p>
    <p><img src="you_are_here.jpg"></p>

    <p><a href="http://commons.wikimedia.org/wiki/File:You_are_here_-_T-shirt.jpg">Image 'You are
    <a href="http://www.flickr.com/photos/geekgirly/">geekygirly</a> used under
    <a href="http://creativecommons.org/licenses/by/2.0/deed.en">CC-BY</a></p>
</body>
</html>
```

sind folgende drei URLs enthalten:

absolut beginnt mit Protokoll *http://pmeerw.net/www-mm14/images/you-are-here.jpg*

relativ zum Webserver beginnt mit einem Schrägstrich (Slash) */www-mm14/images/you-are-here.jpg*

relativ zum Dokument kein besonderes Kennzeichen *you-are-here.jpg*

7.1 Relative URLs

Bei den relativen URLs muss man die URL des enthaltenden Dokuments und die relative URLs "zusammen—addieren", um zu einer absoluten URL zu kommen.

Wenn die relative URL mit einem Schrägstrich (Slash) beginnt ersetzt sie den gesamten "Pfad" in der URL des enthaltenden Dokuments:

> **http://pmeerw.net/www-mm14/images/urls.html** + /www-mm14/images/you_are_here.jpg =
>
> http://pmeerw.net/www-mm14/images/you_are_here.jpg

Wenn die relative URL nicht mit einem Schrägstrich beginnt wird sie zum Pfad des aktuellen Ordners dazugerechnet:

> **http://pmeerw.net/www-mm14/images/urls.html** + you_are_here.jpg =
>
> http://pmeerw.net/www-mm14/images/you_are_here.jpg

Relative URLs an anderer Stelle

Wird das Dokument an eine andere Stelle (auf einen anderen Server) verschoben, und änderst sich damit die Ausgangs-URL, dann ist das Ergebnis anders (und würde nicht funktionieren):

> **http://multimediatechnology.at/2011/06/19/ruby** + /images/you_are_here.jpg =
>
> http://multimediatechnology.at/images/you_are_here.jpg

und

> **http://multimediatechnology.at/2011/06/19/ruby** + you_are_here.jpg =
>
> http://multimediatechnology.at/2011/06/19/you_are_here.jpg

Relative URLs mit Ordnernamen

Relativen URLs können selbst wieder Ordnernamen enthalten:

> **http://multimediatechnology.at/2011/** + 12/15/screenshot/ =
>
> http://multimediatechnology.at/2011/12/15/screenshot/

Relative URLs mit Punkt-Punkt

Zwei Punkte als Ordnernamen bedeuten dabei: „raus aus dem aktuellen Ordner":

> **http://multimediatechnology.at/web-communities/barcamp/** + ../ =
>
> http://multimediatechnology.at/web-communities/

Dieser Schreibweise erlaubt auch sehr umständliche Formulierungen (mit Redundanz):

> **http://multimediatechnology.at/2011/06/19/ruby** + ../../2008/05/../04/17/rattenscharfes-computergame =
>
> http://multimediatechnology.at/2008/04/17/rattenscharfes-computergame

Alles nur im Client

All diese Berechnungen finden im Webbrowser statt, und sind völlig unabhängig davon, wie der Webserver die URL interpretiert. So könnte z.B. die letzten URLs „in Wirklichkeit" gar nicht auf Ordner mit Namen 2008, 04, 17 verweisen, sondern eine Datenbanksuche nach „2008-04-17" nach sich ziehen – das ist Sache des Servers.

KAPITEL 8

Konfiguration Webserver

Die Konfiguration des Webservers wird hier am Beispiel von Apache (http://httpd.apache.org/) gezeigt. Andere Webserver, z.B: Internet Information Server (IIS) von Microsoft oder nginx (http://nginx.org) (ausgesprochen "engine x") verfügen auch über ähnliche Fähigkeiten, werden aber anders konfiguriert.

Betriebssystem des Webservers

Viele Webserver werden auf einem UNIX-Betriebssystem betrieben. Die Dateisysteme von UNIX-Systemen unterscheiden bei Datei- und Ordnernamen zwischen Groß- und Kleinschreibung, sind also „case-sensitive". Der Rechner auf dem Sie Webseiten erstellen läuft wahrscheinlich unter Windows oder MacOS; dort sind die Dateisysteme „case-insensitive".

Ein Link auf die Datei *bild.JPG* funktioniert unter Windows oder MacOS auch dann, wenn der Link (falsch) als *bild.jpg* geschreiben wird.

Liegt die Datei aber dann am (UNIX-)Webserver, so funktioniert der Link nicht mehr! *bild.jpg* und *bild.JPG* sind zwei unterschiedliche Dateinamen!

Webspace und Ordner

Im einfachsten Fall wird im Webserver ein Ordner spezifiziert, der als Ausgangspunkt für den Webspace dient.

- URL *http://pmeerw.net/www-mm14/images/you_are_here.jpg*
- Pfad im Dateisystem */var/www/hosts/pmeerw.net/www-mm14/images/you_are_here.jpg*

Die Apache-Konfiguration dazu könnte so aussehen:

```
<VirtualHost pmeerw.net>
DocumentRoot /var/www/hosts/pmeerw.net/
</VirtualHost>
```

Standard-Dokument

Endet eine URL auf einen Schrägstrich, dann verweist sie eigentlich auf einen ganzen Ordner, nicht auf eine spezielle Datei. Für diesen Fall kann im Webserver ein Standard-Dokument definiert werden. Auf vielen Webservern ist dies die Datei *index.html*. d.h. wenn eine URL auf einen Ordner verweist, und in diesem Ordner eine Datei mit Namen *index.html* existiert, dann wird diese Datei geliefert.

URL	Pfad im Dateisystem
http://pmeerw.net/	/var/www/hosts/pmeerw.net/index.html

Die Apache-Konfigurationsanweisung dazu lautet:

```
DirectoryIndex index.html
```

Achtung: die Konfiguration ist am Internet Information Server (IIS) normalerweise anders, dort wird die Datei *default.htm* verwendet!

Directory Index Datei existiert nicht

Was passiert wenn keine Datei für den Verzeichnis-Index mit dem richtigen Namen vorhanden ist? Entweder wird eine Auflistung der Dateien im Ordner oder eine Fehlermeldung geliefert.

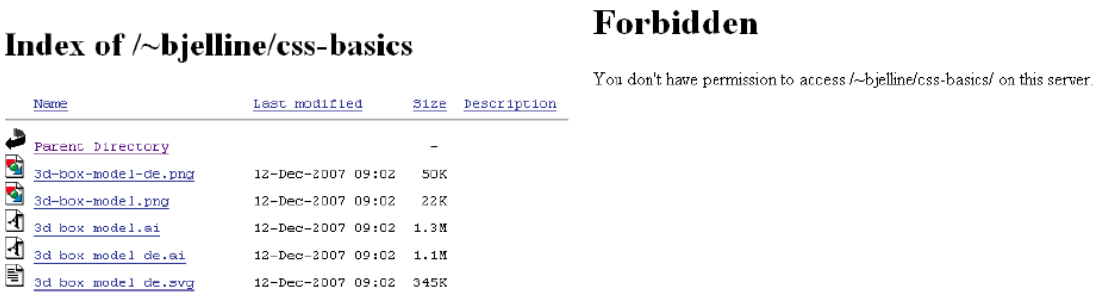


Abbildung 8.1: Abbildung: Zugriff auf einen Ordner ohne Standard-Dokument (index.html): Auflistung oder Fehlermeldung

Die entsprechenden Apache-Konfigurationsanweisung dazu sind:

```
Options +Indexes
Options -Indexes
```

Automatischer Webspace für Alle

Auf UNIX-Servern mit vielen Accounts ist es üblich, dass für jeden Account automatisch ein Webspace existiert. Dieser Webspace befindet sich innerhalb des Home-Verzeichnisses des jeweiligen Accounts. Z.B. könnte ein Benutzer den Usernamen *pmeerw* habe und das Home-Verzeichnis */home/pmeerw/*. Der zugehörige Webspace befindet sich dann im Unter-Ordner *public_html*:

URL	Pfad im Dateisystem
http://pmeerw.net/~pmeerw/test.html	/home/pmeerw/public_html/test.html
http://pmeerw.net/~hugo/	/home/hugo/public_html/index.html

Die Apache Konfigurationsanweisung lautet:

```
UserDir public_html
```

Beachten Sie: für das Formulieren von relativen URLs sind wirklich nur die URLs relevant, nicht die Position der Dokumente im Dateisystem! Der Ordnername *public_html* wird also nie in einer URL vorkommen.

Tipps zur Ordnerstruktur

Groß- und Kleinschreibung

Hintergrund: Viele Webserver werden auf UNIX betreiben, dort sind Dateinamen *case-sensitive*.

Tipp: ich schreibe alle Ordner- und Dateinamen sowie alle URLs immer durchgängig **klein**.

Leerzeichen

Hintergrund: Leerzeichen in URLs müssen als %20 geschrieben werden.

Tipp: ich verwende **keine Leerzeichen** in Ordner- und Dateinamen meiner Web-Projekte.

Umlaute

Hintergrund: URLs sind global sichtbar. Eine URL, die ich erschaffe, muss vielleicht einmal auf einer Tastatur eingetippt werden, die keine deutschen Umlaute hat.

Tipp: ich **vermeide Umlaute** in Ordner- und Dateinamen meiner Web-Projekte und beschränke mich auf den englischen Zeichensatz.

Ordnerstruktur

Hintergrund: Nicht nur der Code meiner Webseiten ist für das Funktionieren der Seiten wichtig, sondern auch die Dateinamen und Ordnernamen, bzw. die Ordnerstruktur. Liegt eine Datei im falschen Ordner wird sie nicht mehr gefunden.

Tipp: ich überlege mir die **Ordnerstruktur** meines Webspaces sorgfältig. Wenn ich auf mehreren Rechnern entwickle, dann baue ich auf jedem dieser Rechner die Ordnerstruktur wieder auf. Ich nutze ein Programm für den Upload und Download das die Ordnerstruktur beibehält.

KAPITEL 10

HTTP - Das Protokoll des Web

Was Sie alle wissen sollten

- Das Computer am Internet über IP-Adressen oder Domain Names adressierbar sind
- Wie http-Request und http-Response prinzipiell aufgebaut sind,
- dass GET und POST die wichtigsten Request-Methoden sind,
- dass 200 und 404 die wichtigsten Status-Codes der Responds sind,
- welche http-Header für Umleitung und Authentisierung notwendig sind

Was Sie können sollten

- Mit dem Firefox-AddOn Live http Headers oder mit Firebug HTTP abhören.

KAPITEL 11

TCP/IP und DNS

Um das Protokoll des Web, HTTP, zu verstehen sind erst ein paar grundsätzliche Informationen zur Funktionsweise des Internet notwendig. Genaueres erfahren Sie in einer separaten Lehrveranstaltung zum Thema Computer-Netzwerke.

11.1 Das Internet

Das Internet ist ein weltweites Computernetzwerk, oder besser: ein Netzwerk von Netzwerken. Es sind **verschiedene Computer** daran angeschlossen: PCs mit Betriebssystem Windows oder Linux, Macs, UNIX-Workstations, Smartphones und Tablets, Drucker und Fernseher, und noch viele mehr. Die einzelnen **Netze** sind sehr unterschiedlich: Kupferleitungen, Glasfaserleitungen, Satelliten-Verbindungen, Ethernet, Funkstrecken, verschiedene Handy-Netze. Die **Besitzverhältnisse** sind kompliziert: die Leitungen und Computer gehören verschiedenen Firmen, Universitäten, Schulen, Vereinen, Privatpersonen.

Was hält das Internet dann zusammen? Das **Internet Protocol** (IP). Aufbauend auf die grundlegende Übertragungsnetze (z.B. Ethernet) muss jeder Computer am Internet (genannt „Host“) diese Protokoll-Familie implementieren. Zwischen den Netzen vermitteln **Router** die Pakete von einem Netz zum nächsten.

Derzeit ist IP in der Version 4 allgegenwärtig. Allerdings sind die etwa 4 Milliarden Adressen (32-bit) nunmehr aufgebraucht und viele hosts können nur durch Network Address Translation (NAT) das Internet erreichen. Der Nachfolger IP Version 6 bietet einen 128-bit Adressraum, setzt sich aber nur langsam durch.

IP-Adressen

Die eindeutigen Adressen für Hosts am Internet werden zentral verwaltet. Die Internet Assigned Numbers Authority (IANA) hat diese Aufgabe an Organisationen auf den verschiedenen Kontinenten verteilt, in Europa an das Réseaux IP Européens Network Coordination Centre (RIPE NCC). RIPE vergibt die Adressen an die Internet-Provider in Europa. In der Whois-Datenbank (http://www.ripe.net/data-tools/stats/ris/riswhois) von RIPE kann man die „Besitzer“ von IP-Adressen herausfinden.

IP – das Internetprotokoll

Der Host teilt die zu sendenden Daten in einzelne Pakete und versieht jedes Pakt mit der Absender- und Zieladresse (IP-Adressen, jeweils 4 Byte). Der Host selbst kennt nur sein Standard-Gateway (= der nächste Router) und das eigene Netzwerk. Über das eigene Netzwerk schickt er das Paket an das Standard-Gateway.

Der Router nimmt das Paket auf dem einen Netzwerk entgegen und entscheidet auf Grund der Adressen auf welchem Netzwerk und an welchen Router er das Paket weiterleitet. Beim Ziel-Host langen die Pakete ein – es gibt aber keine Garantie, dass alle ankommen oder dass sie in der richtigen Reihenfolge ankommen.

TCP - Transmission Control Protocol

TCP bietet zusätzlich zur Datenübertragung die Sicherheit, dass Pakete nicht unerkannt verloren gehen und dass sie – falls sie ankommen – in der richtigen Reihenfolge ankommen.

Dazu wird der Datenstrom wieder in IP-Pakete zerlegt, diese werden aber nummeriert bevor sie abgesendet werden. Die Adressierung erfolgt über IP-Adresse plus **Port-Nummer**. Der Ziel-Host prüft die Reihenfolge der Pakete und meldet zurück falls Pakete fehlen.

Aus Programmier-Sicht erhält man also entweder die Daten in richtiger Reihenfolge oder eine Fehlermeldung.

DNS – Domain Name System

Das Domain Name System ist eine verteilte Datenbank die „hübsche Namen“ für Hosts speichert. Z.B. ist dort zu *pmeerw.net* die IP-Adresse *87.118.82.44* gespeichert. Viele Namen verweisen übrigens auf die gleiche IP-Adresse. D.h. mehrere Domains werden von einer IP-Adresse bedient. Auch der umgekehrte Fall ist möglich, zu der Domain gehören mehrere IP-Adressen - dies sorgt für Redundanz und Ausfallsicherheit.

Auf jedem Host ist die IP-Adresse des nächsten Domain Name Servers gespeichert. So kann der Host einen „domain name lookup“ machen: er fragt seinen DNS-Server „was ist die IP-Adresse von x.y.z“ und erhält als Antwort die IP-Adresse.

Der DNS-Server übernimmt dabei die Arbeit, eventuell bei mehreren anderen DNS-Servern nachzufragen. Auch die Top-Level-Domains (.com, .at, .de, . . .) werden von IANA verwaltet und können über Whois abgefragt werden. Für die Domain .at ist die Firma nic.at (http://www.nic.at) zuständig.

KAPITEL 12

HTTP

HTTP in der aktuell gültigen Version 1.1 ist in RFC 2616 (https://www.ietf.org/rfc/rfc2616.txt) definiert. HTTP baut auf TCP auf, d.h. die hier dargestellten Daten werden über eine TCP-Verbindung zwischen Client und Server übertragen. Im [Kapitel „Drei Standards definieren das Web“](/das-web-und-html/standards/) wurde HTTP schon einmal grob vorgestellt; nun werden wir HTTP genauer betrachten.

12.1 HTTP im Überblick

Egal ob der Vorgang durch das Eintippen einer URL oder durch das Anklicken eines Links gestartet wird — das Laden einer Webseite über HTTP funktioniert immer gleich.

1. Der **Browser** analysiert die **URL**: falls Sie eine IP-Adresse enthält geht’s weiter zum nächsten Schritt. Falls die URL einen Domain Namen enthält wird dieser mittels DNS-Lookup in die entsprechende IP-Adresse übersetzt.
2. Der Browser baut eine **TCP-Verbindung** zum Server auf, falls in der URL nicht anders angegeben, wird dabei Port 80 angesteuert.
3. Der Browser sendet über die TCP-Verbindung einen **HTTP-Request**; dieser besteht aus einer ersten Zeile (Request-Line), mehreren Header-Zeilen und manchmal einem Body.
4. Der **Webserver** nimmt den Request entgegen und analysiert ihn. Der Webserver entscheidet, ob er zur Beantwortung der Anfrage nur eine bestimmte Datei aus dem Dateisystem zu liest, oder ein Programm aufruft.
5. Der Webserver schickt über die TCP-Verbindung einen **HTTP-Response** an den Browser, dieser besteht aus einer ersten Zeile (Response-Line) mit Statuscode, z. B. „200 OK“, mehreren Header-Zeilen und der angeforderten Ressource.
6. Der Browser nimmt das Dokument in Empfang, **stellt es dar**, und beendet die TCP-Verbindung.

Dieser einfache Ablauf kann durch die Verwendung von Proxies und Caches sowie durch das wiederholte Abrufen von Dokumenten vom selben Server komplizierter werden — das ignorieren wir aber erst einmal.

12.2 Aufbau von HTTP Request und Response

Jede Anfrage des Clients und jede Antwort des Servers besteht aus einer ersten Zeile mit besonderer Bedeutung, einem Header und einem Body. Header und Body funktionieren ähnlich wie bei einer E-Mail, es kann viele Header-Zeilen geben. Der Body ist beim Request meist leer.

Hier ein Beispiel für einen Request:

```
GET /rezensionen/list.php3?no=20 HTTP/1.1
Host: www.biblio.at
User-Agent: Mozilla/5.0 (Win98; de-AT) Gecko/20020311
Accept: text/html;q=0.9,text/plain;q=0.8,*/*;q=0.1
```

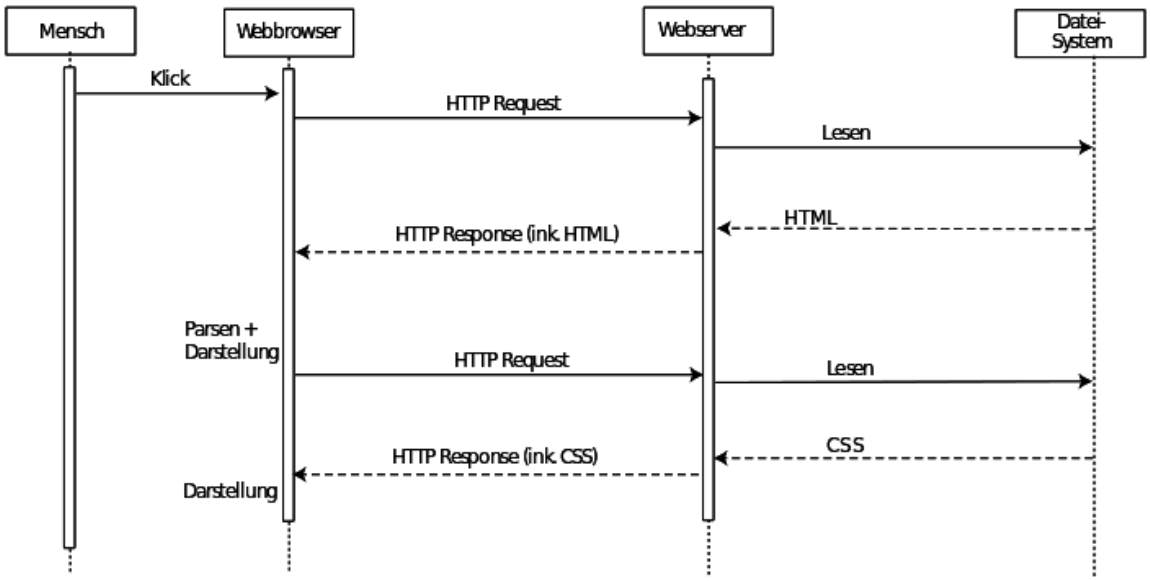


Abbildung 12.1: Abbildung: Zwei HTTP Requests

```
Accept-Language: de-at, de;q=0.66, en-us;q=0.33
Accept-Encoding: gzip, deflate, compress;q=0.9
Accept-Charset: ISO-8859-15, utf-8;q=0.66, *;q=0.66
```

Die erste Zeile besteht aus: Methode, URL-Fragment und HTTP-Versionsnummer. Die meist-verwendete Methode ist GET. Bei Web-Formularen muss man die Methode angeben, mit der die Daten an den Server übertragen werden sollen: GET oder POST.

Hier ein Beispiel für eine Server-Antwort (Response):

```
HTTP/1.0 200 OK
Date: Sat, 27 Apr 2002 05:52:57 GMT
Server: Apache/1.3.9 (Unix) Debian/GNU
Content-Type: text/html

<!DOCTYPE html>
<html><head><title>Rezensionsdatenbank</title>
<link rel="stylesheet" href="rezensionen.css">
</head><body>nix</body></html>
```

Die erste Zeile der Server-Antwort besteht aus der HTTP-Versionsnummer, dem Statuscode und einem erklärenden Text zum Statuscode, der aber nicht standardisiert ist.

Die wichtigsten Statuscodes sind *200 (OK)*, *404 (Not found)*, *403 (Forbidden)*. Siehe http://httpstatus.es/.

12.3 HTTP Header

Header-Zeilen gibt es sehr viele; relativ wenige davon werden von Clients und Servern wirklich beachtet.

Host

Im Request oben *Host: www.biblio.at*. Wichtig wenn der Server unter mehreren Domain Names (aber nur einer IP-Adresse) erreichbar ist. Das ist fast immer der Fall, dieser Header ist also fast immer notwendig.

User-Agent

Im Request oben:

```
User-Agent: Mozilla/5.0 (Win98; de-AT) Gecko/20020311
```

oder

```
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98; Win 9x 4.90)
```

Das ist die Selbstoffenbarung des Clients: welcher Browser, welche Version. Die meisten Clients lügen, und behaupten sie wären Mozilla, erst in der Klammer folgt die richtige Angabe.

Siehe dazu History of the browser user-agent string (http://webaim.org/blog/user-agent-string-history/).

Referer

Im Request oben: *Referer: http://my.app.at/form.html* Welche Seite hat der Client zuvor angezeigt? URL der vorigen Seite — falls von dort ein Link hierher verfolgt wurde oder ein Formular gesendet wurde. Dieser Header kann im Browser deaktiviert werden!

Date, Server

In der Response oben:

```
Date: Sat, 27 Apr 2002 05:52:57 GMT
Server: Apache/1.3.9 (Unix) Debian/GNU
```

Zeigt Datum und Uhrzeit am Server, bzw. die verwendete Webserver-Software. Wenn Sie Statistiken über den Marktanteil der verschiedenen Server sehen, dann basieren diese auf dieser Angabe.

Content-Type

In der Response:

```
Content-Type: text/html
```

MIME-Type des im Body gelieferten Dokuments. Andere Werte die sie hier häufig antreffen sind:

- text/css
- application/javascript
- image/png
- image/jpg
- image/gif
- image/svg+xml

12.4 HTTP abhören

Wie können Sie HTTP beobachten? Entweder mit einem allgemeinen **Netzwerk-Sniffer** wie Wireshark (http://www.wireshark.org/) oder mit der Firefox-Extension Live HTTP Headers (https://addons.mozilla.org/en-US/firefox/addon/live-http-headers/) oder Firebug (https://addons.mozilla.org/en-US/firefox/addon/firebug/?src=search).

Die folgenden Anwendungsbeispiele wurden mit Live HTTP Headers mitgeschnitten.

12.5 Seite laden oder Formulardaten senden mit GET

Die Methode GET wird bei den meisten HTTP-Anfragen verwendet - sowohl bei normalen Links als auch beim Senden von Formulardaten mit GET. Die URL kann dabei ein Fragezeichen gefolgt von Parametern und Werten enthalten.

HTTP Request:

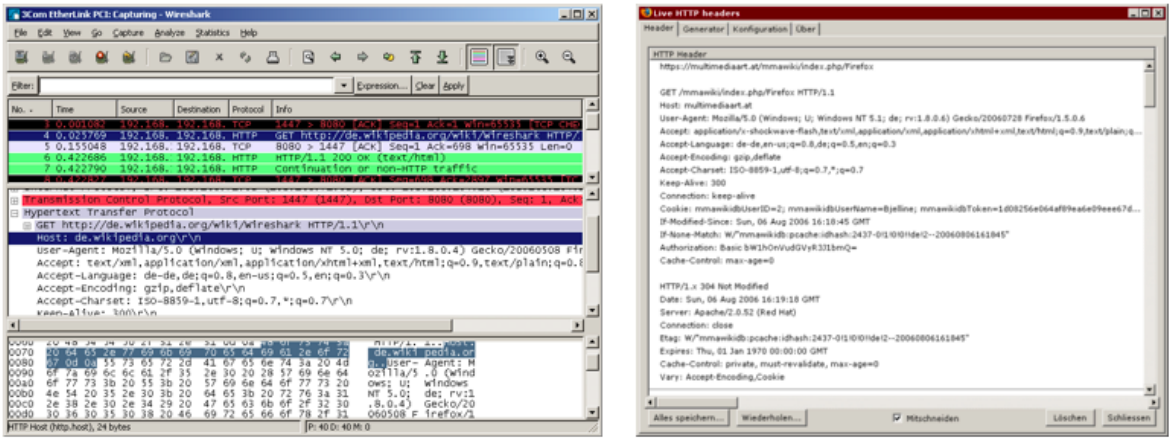


Abbildung 12.2: Abbildung: HTTP abhören mit Wireshark (links) und Live HTTP Headers (rechts)

```
GET /rezensionen/list.php3?no=20 HTTP/1.1
Host: www.biblio.at
User-Agent: Mozilla/5.0 (Win98; de-AT) Gecko/20020311
Accept: text/html;q=0.9,text/plain;q=0.8,*/*;q=0.1
Accept-Language: de-at, de;q=0.66, en-us;q=0.33
Accept-Encoding: gzip, deflate, compress;q=0.9
Accept-Charset: ISO-8859-15, utf-8;q=0.66, *;q=0.66
```

HTTP Response:

```
HTTP/1.0 200 OK
Date: Sat, 27 Apr 2002 05:52:57 GMT
Server: Apache/1.3.9 (Unix) Debian/GNU
Content-Type: text/html

<!DOCTYPE html>
<html>
<head>
   <title>Rezensionsdatenbank</title>
   <link rel="stylesheet" href="rezensionen.css">
</head>
```

Die Länge der übertragenen Daten aus dem Formular ist hier begrenzt durch die Länge der URL. Für größere Datenmengen (z.B. beim Upload von Dateien) gibt es die Methode POST.

Die Header, die mit Accept beginnen, können (laut Standard) dem Aushandeln von Sprache, Datentyp, Encoding dienen; werden aber von Servern und Clients nur teilweise beachtet. Beispiel:

```
Accept: text/html;q=0.9,text/plain;q=0.8,*/*;q=0.1
```

Dies bedeutet laut Standard, dass der Client das Dokument lieber als HTML als als Plain Text erhalten würde. Im realen Web wird aber unter einer URL immer nur ein Dokumententyp angeboten. Wenn man eine PDF-Version der gleichen Information anbietet, dann geschieht dies unter einer anderen URL.

Accept-Language würde dem Aushandeln der Sprache dienen. Dazu müssten die UserInnen aber im Browser die Sprach-Präferenz konfigurieren:

Da aber kaum jemand diese Konfiguration vornimmt wird die Sprach-Aushandlung kaum verwendet. Einziges mir bekanntes Beispiel einer Webseite die unter der gleichen URL in verschiedenen Sprachen erhältlich ist ist die Homepage von Debian (http://debian.org):

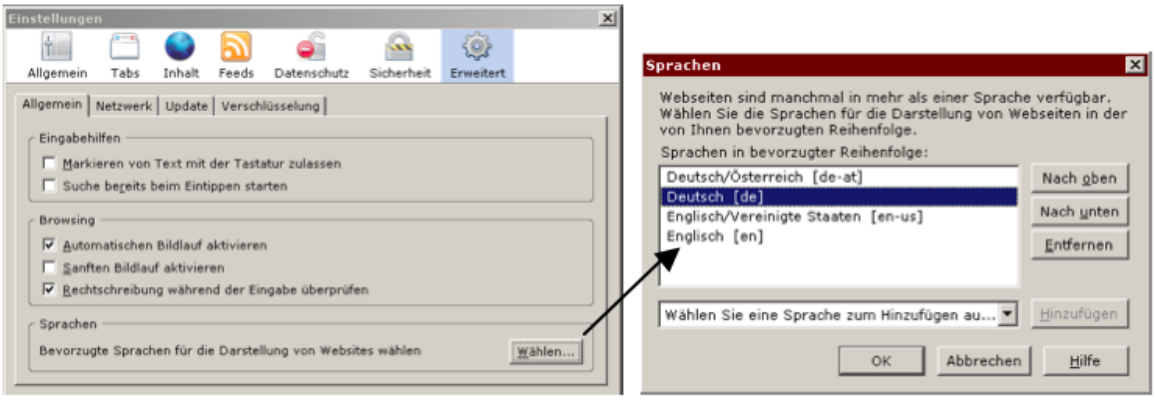


Abbildung 12.3: Abbildung: Festlegen der Sprach-Präferenz im Browser Firefox

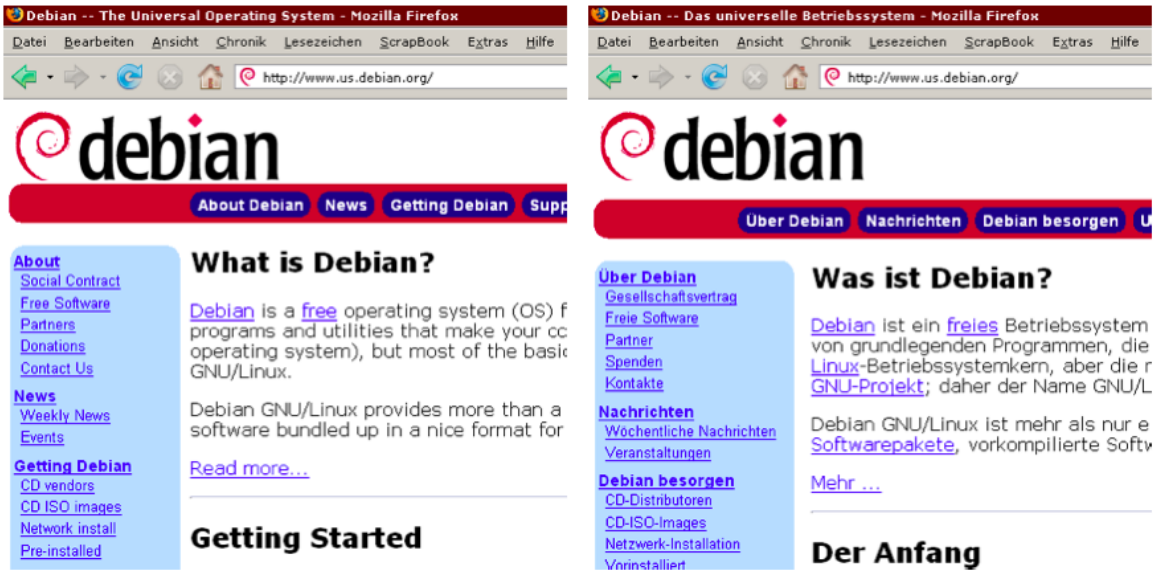


Abbildung 12.4: Abbildung: Homepage von Debian, verschiedene Sprachen bei gleicher URL

12.6 Senden von Formulardaten mit POST

Bei POST werden die Daten aus dem Formular nicht in der URL, sondern im HTTP-Body der Anfrage übertragen. Die Codierung (kaufmännisches-Und zwischen den *namen=wert*-Paaren, + statt Leerzeichen, %-Schreibweise für Sonderzeichen) bleibt gleich. Hier gibt es aber keine Beschränkung der Länge.

HTTP Request:

```
POST /rezensionen/list.php3 HTTP/1.1
Host: www.biblio.at
User-Agent: Mozilla/5.0 (Win98; de-AT) Gecko/20020311
Referer: http://www.biblio.at/rezensionen/formular.htm
Content-Type: application/x-www-form-urlencoded
Content-Length: 129

no=20&limit=1&katalog=all&isbn=&nachname=Jellinek&vornam
e=&titel=&schlagwort1=&schlagwort2=&Bool=AND&verl=&von=&
bis=&submit=SUCHE
```

Die Antwort des Servers unterscheidet sich nicht zwischen GET und POST.

12.7 Umleiten einer URL

Mit dem Statuscode 301 kann der Server anzeigen, dass die Seite an eine neue URL übersiedelt ist. Der Webbrowser schickt dann sofort eine Anfrage an die neue URL, die LeserIn bemerkt so eine Weiterleitung meist gar nicht.

HTTP Request:

```
GET / HTTP/1.1
Host: www.rezensionen.at
User-Agent: Mozilla/5.0 (Win98; de-AT) Gecko/20020311
Accept: text/html;q=0.9,text/plain;q=0.8,*/*;q=0.1
Accept-Language: de-at, de;q=0.66, en-us;q=0.33
Accept-Encoding: gzip, deflate, compress;q=0.9
Accept-Charset: ISO-8859-15, utf-8;q=0.66, *;q=0.66
```

HTTP Response:

```
HTTP/1.0 301 Moved Permanently
Date: Sat, 27 Apr 2002 05:52:26 GMT
Server: Apache/1.3.9 (Unix) Debian/GNU
Location: http://www.biblio.at/rezensionen/
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>301 Moved Permanently</TITLE>
</HEAD><BODY>
<H1>Moved Permanently</H1>
The document has moved <A HREF="http://www.biblio.at/rezensionen/">here</A>.<P>
</BODY></HTML>
```

Neuerlicher HTTP Request:

```
GET /rezensionen/ HTTP/1.1
Host: www.biblo.at
User-Agent: Mozilla/5.0 (Win98; de-AT) Gecko/20020311
Accept: text/html;q=0.9,text/plain;q=0.8,*/*;q=0.1
Accept-Language: de-at, de;q=0.66, en-us;q=0.33
Accept-Encoding: gzip, deflate, compress;q=0.9
Accept-Charset: ISO-8859-15, utf-8;q=0.66, *;q=0.66
```

Location gibt die neue URL an.

12.8 Authentisierung nach RFC 2617

Der Webserver kann so konfiguriert werden, dass er Dokumente nur nach Eingabe von Usernamen und Passwort liefert. Der Browser zeigt dafür ein Eingabefenster an:

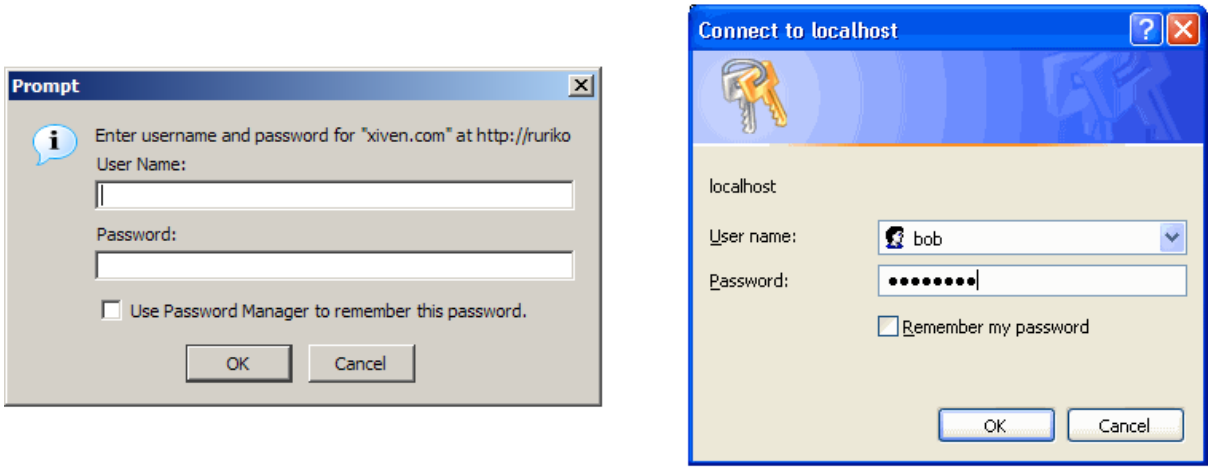


Abbildung 12.5: Abbildung: Eingabefenster für HTTP Authentisierung in verschiedenen Browsern

Auf Ebene des HTTP-Protokolls betrachtet funktioniert diese Authentisierung wie folgt: bei der ersten Anfrage des Clients schickt der Server einen Status-Code 401 (Nicht autorisiert).

HTTP Request:

```
GET /pr/ HTTP/1.1
Host: www.sbg.ac.at
User-Agent: Mozilla/5.0 (Win98; de-AT) Gecko/20020311
Accept: text/html;q=0.9,text/plain;q=0.8,*/*;q=0.1
Accept-Language: de-at, de;q=0.66, en-us;q=0.33
Accept-Encoding: gzip, deflate, compress;q=0.9
Accept-Charset: ISO-8859-15, utf-8;q=0.66, *;q=0.66
```

HTTP Response bei fehlerder Authorisierung:

```
HTTP/1.0 401 Unauthorized
Date: Sat, 27 Apr 2002 06:05:08 GMT
Server: Apache/1.3.22 (Unix)
WWW-Authenticate: Basic realm="unineu"
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>401 Authorization Required</TITLE>
```

Daraufhin zeigt der Browser das Passwort-Eingabefenster an. Nach Eingabe von Usernamen und Passwort schickt der Browser die gleiche Anfrage erneut, diesmal aber mit der zusätzlichen Header-Zeile *Authorization*. In dieser Zeile werden Username und Passwort (leicht verschlüsselt) mitgeschickt.

Wenn Username und Passwort stimmen, schickt der Server eine positive Antwort und das Dokument. Der Browser wird bei allen weiteren Anfragen an diesen Server ebenfalls die Authorization-Zeile mitschicken.

HTTP Request mit Authentisierung:

```
GET /pr/ HTTP/1.1
Host: www.sbg.ac.at
User-Agent: Mozilla/5.0 (Win98; de-AT) Gecko/20020311
Accept: text/html;q=0.9,text/plain;q=0.8,*/*;q=0.1
Accept-Language: de-at, de;q=0.66, en-us;q=0.33
Accept-Encoding: gzip, deflate, compress;q=0.9
Accept-Charset: ISO-8859-15, utf-8;q=0.66, *;q=0.66
Authorization: Basic dHI6cHJyMDBy
```

HTTP Request nach erfolgreicher Authentisierung:

```
HTTP/1.0 200 OK
Date: Sat, 27 Apr 2002 06:05:11 GMT
Server: Apache/1.3.22 (Unix)
Content-Type: text/html; charset=iso-8859-1

<html lang="de">
<head>
<title>Universit&auml;t Salzburg - B&uuml;ro f&uuml;r Public Relations</title>
```

12.9 HTTPS

HTTPS ist HTTP über Secure Socket Layer (SSL) — d.h. auf Ebene der TCP-Verbindung werden alle übertragenen Daten verschlüsselt. Außerdem bietet SSL die Möglichkeit, dass sich der Server und der Client mit einem Zertifikat ausweisen.

Ob HTTPS oder HTTP verwendet wird, hängt von der Konfiguration des Server ab. Wenn Sie nur Webspace in „Untermiete“ benutzen, können Sie HTTPS nicht aktivieren.

Die Verschlüsselung und Entschlüsselung des gesamten Verkehrs braucht CPU-Zeit — der Webserver kann also weniger Anfragen bedienen als mit HTTP. Eine Möglichkeit diese Belastung vom Webserver „fernzuhalten“ ist die Terminierung von SSL auf einem anderen Rechner.

Die Electronic Frontier Foundation (http://eff.org) hat eine Kampagne gestartet um KonsumentInnen und WebsiteBetreiberInnen zu überzeugen, dass HTTPS immer verwendet werden sollte: „HTTPS Everywhere“.

Damit wäre der gesamte Web-Traffic nicht mehr abhörbar.

12.10 Proxy Server

Das HTTP-Protokoll sieht die Möglichkeit von Proxies vor. Ein Proxy ist eine „Zwischenstation“ die HTTP-Verkehr weitergibt, der Proxy agiert also auf der einen Seite als HTTP-Server, auf der anderen Seite als HTTP-Client. Im Browser kann ein Proxy konfiguriert werden:

Ist ein Proxy konfiguriert dann baut der Browser die HTTP-Verbindung nicht direkt zum Zielrechner auf, sondern zum Proxy, und verändert die Form der ersten Zeile der HTTP-Anfrage: die vollständige URL wird angegeben:

```
GET http://www.sbg.ac.at/pr/ HTTP/1.1
```

Proxies können gleichzeitig als Cache fungieren: Anfragen und Antworten werden gespeichert; erfolgt die gleiche Anfrage noch einmal, kann die gespeicherte Antwort verwendet werden.

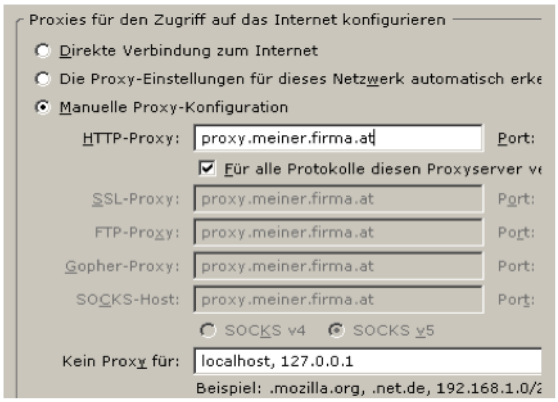


Abbildung 12.6: Abbildung: Proxy-Konfiguration in Firefox

KAPITEL 13

CSS Einstieg

Sie lernen mit Cascading Stylesheets (CSS) einfache Formatierungen an der Webseite vorzunehmen.

Was Sie wissen sollten

- Ich weiss, wie HTML und CSS zusammen hängen. Wie CSS Selektoren aufgebaut sind. Welche Schreibweisen und Maßeinheiten es in CSS für Farben, URLs und Größenangaben gibt.
- Ich weiss, wie das CSS Box Model funktioniert, wie der Zusammenhang zwischen *margin*, *border* und *padding* ist.
- Ich weiss, welche Möglichkeiten zur Darstellung von Schrift, Absatz, Rahmen, Hintergrundfarbe und Hintergrundbilder CSS bietet. Ich kann die genaue Schreibweise der dafür notwendigen CSS-Deklarationen nachschlagen.

Was Sie können sollten

- Ich kann ein einfaches Stylesheet für ein HTML-Dokument erstellen.
- Ich kann anhand einer Vorgabe einer bestimmten visuellen Darstellung und eines HTML-Dokuments ein geeignetes Stylesheet erstellen, das zur gewünschten Darstellung führt.
- Ich kann mit Hilfe von Firebug oder durch Lesen des Codes herausfinden welche CSS-Regeln zu einer bestimmten Darstellung in einer (fremden) Webseite führen.
- Ich kann die Korrektheit des CSS-Codes mit dem CSS-Validator des World Wide Web Consortium überprüfen.
- Ich kann Fehler in einem CSS-Dokument ausbessern bis es valide ist.

KAPITEL 14

Kurzvorstellung von Stylesheets

Von Anfang an sollte eine HTML-Datei keine Information darüber enthalten, wie der Text dargestellt werden soll - keine Schriftart oder Schriftgröße. HTML sollte nur strukturelle Information enthalten („das ist eine Überschrift“ aber nicht „Helvetica 24pt“). Die Formatierungs-Information sollte in sogenannten „Stylesheets“ gespeichert werden. Im Jahre 1993 sah eine typische Webseite so aus:

```
<html><body>
<h1>Das Studium</h1>
<p>Studienziel ist es, breit gefächerte technische und kreative Kompetenzen...
</body></html>
```

Da es keine Stylesheets gab „erfand“ Netscape ab 1994 zusätzliche HTML-Tags, die die Darstellung der Webseite festlegen. Andere Browser übernahmen diese Tags von Netscape.

```
<html><body>
<h1><font face="Arial" color="blue">Das Studium</font></h1>
<p><font size="+1">S</font>tudienziel ist es, breit gefächerte technische und kreative Kompetenzen
<p><font size="+1">P</font>rojekte und Praxissemster stellen schon während des Studiums die Verbi
</body></html>
```

Durch das Hinzufügen der Tags für Formatierung wurde der HTML-Code komplizierter und unübersichtlicher. Erst ab 1995 wurde endlich an den Standards für Stylesheets gearbeitet, seit den frühen 2000ern ist die Unterstützung in den gängigen Browsern vorhanden. So sieht nun eine HTML-Seite mit separaten Stylesheet aus:

```
<html><head>
<link rel="stylesheet" type="text/css" href="design.css">
</head><body>
<h1>Das Studium</h1>
<p>Studienziel ist es, breit gefächerte technische und wirtschaftliche Kompetenzen...
</body></html>
```

Die HTML-Datei enthält die Information und HTML-Tags für die Struktur sowie einen Verweis auf die CSS-Datei.

```
h1 {
   font-family: Arial;
   color: blue;
}
p:first-letter {
   font-size: large;
}
```

Mit der Trennung von Stylesheet und HTML wurde HTML wieder einfacher und übersichtlicher. Besonders das nachträgliche Verändern der Gestaltung wurde vereinfacht. Wichtige Argumente für den Einsatz von Stylesheets sind:

- Zusätzliche Gestaltungsmöglichkeiten
- Einheitliche Gestaltung von mehreren Webseiten

- Arbeitsteilung zwischen DesignerInnen (die CSS erstellen) und RedakteuerInnen (die Inhalte erstellen)

Aber Achtung: CSS ist nicht die Lösung jedes Problems:

- Mit Stylesheets können Sie nur Elemente verändern, die schon in der HTML-Datei vorhanden sind. Es können keine neuen Elemente eingefügt werden.
- Um ein Stylesheet zu erstellen, muss die DesignerIn die HTML-Tags kennen, z.B. wissen, dass *h1*, *h2*, *h3* für die Überschriften stehen.
- Damit das Stylesheet wirkt, muss die RedakteurIn, die das HTML-Dokument erstellt, die richtigen HTML-Tags verwenden, z.B. Überschriften wirklich mit *h1*, *h2*, *h3* formatieren, und nicht mit *b*, *i*, *font*.

14.1 Beispiel

Das Stylesheet definiert für die einzelnen HTML-Tags wie sie dargestellt werden sollen. In folgendem Code werden Formatierungen für die Tags *body*, *p*, *h1* und *h2* vorgenommen, der gezeigte <*style*>-Tag wird im *head* des HTML-Dokument eingebunden:

```
<style type="text/css">
   p {
     font-family: Calibri, Helvetica, Arial, sans-serif;
     font-size: 13px;
   }
   h1,h2 {
     font-family: "Trebuchet MS", Verdana, Arial, sans-serif;
   }
   h1 { font-size: 24px; }
   h2 { font-size: 20px; }
   body { background-color: green; }
</style>
```

An Hand dieses Beispiels werden nun die ersten Stylesheet-Befehle erklärt. Die Syntax von Stylesheets ist völlig anders als die von HTML.

14.2 Interpretation

Zuerst werden die Schriften im Dokument festlegt. Achtung: Welche Schriften auf dem Endgerät zur Verfügung stehen ist nicht bekannt, deswegen kann man mehrere Schriften angegeben. Diese Liste wird vom Browser von links nach rechts abgearbeitet, die erste Schrift die gefunden wird, wird verwendet.

In Zeile 2 bis 5 werden zwei Anweisungen für Fließtext gegeben (HTML-Tag *p*): die Schriftfamilie in Zeile 3 und die Schriftgröße in Zeile 4.

Vergleichen Sie Zeile 3 mit Zeile 7: Schriftnamen die ein Leerzeichen enthalten müssen in Anführungszeichen gesetzt werden, wie “Trebuchet MS” in Zeile 6.

Das letzte Wort in Zeile 7 „sans-serif“ ist ein CSS-Kürzel für „irgendeine serifenlose Schrift“. Es empfiehlt sich am Ende einer Schriftliste ein solches Kürzel als Standardwert „wenn alle Stricke reißen“ anzugeben, dabei sind folgende Werte möglich:

- *serif*,
- *sans-serif*,
- *monospace*.

14.3 CSS erforschen mit Firebug

Sie haben nun einen kurzen Einblick in die Schreibweise und die Möglichkeiten von Stylesheets. Genug um Stylesheets von Webseiten zu lesen um neue Möglichkeiten kennen zu lernen. Beim Lesen und Verstehen von CSS hilft das Firefox Add-On „Firebug“.

Abbildung 14.1: Abbildung: Firefox Add-On Firebug

KAPITEL 15

Syntax von CSS

Ein Beispiel

```
h1,h2 {
  font-family: Arial, Helvetica, sans-serif;
  color: lightblue;
}
body {
  margin-left: 150px;
  background-color: white;
}
p {
  text-align: justify;
}
```

Eine Stylesheet-Regel („Rule“) besteht aus einem **Selektor** gefolgt von einer geschwungenen Klammer die eine oder mehrere Deklarationen enthalten kann.

Der einfachste Selektor besteht aus dem Namen eines einzelnen HTML-Tags. Sie können auch mehrere Elemente durch ein Komma trennen. In diesem Falle werden für alle Elemente die selben Style-Angaben definiert, siehe *h1*,‘h2‘ in obigem Beispiel.

Sie können beliebig Zeilenumbrüche und Whitespace einfügen, beides wird ignoriert.

Style für mehrere Seiten

Die Seiten einer gesamten Website haben meist ein einheitliches Aussehen. Dies können Sie erreichen, wenn Sie für jede Webseite dieselbe CSS Datei verwenden. Dazu müssen Sie in jede HTML-Datei die CSS Datei mittels folgender Anweisung einbinden:

```
<link rel="stylesheet" href="mystyle.css">
```

Diese Zeile sollte innerhalb des *head* Bereiches der HTML-Datei stehen (da sonst die Darstellung der Seite verzögert werden kann). *mysyle.css* könnte etwa so aussehen wie im vorigen Beispiel.

Style für eine Seite

Wenn Sie einen Style nur auf einer einzigen Webseite verwenden, können Sie die *style*-Angaben direkt in die HTML-Datei schreiben, und zwar innerhalb des *head* Bereiches.

```
<head>
<style>
  h1,h2 {
    font-family: Arial, Helvetica, sans-serif;
    color: lightblue
  }
</style>
</head>
```

Style für einen Tag

Style-Angaben können auch direkt für einen einzelnen HTML-Tag geschrieben werden mit Hilfe des *style* Attributs. In diesem Fall gilt die Angabe nur für diesen ganz speziellen Tag.

```
<h1 style="color:red; text-align:center;">Rote, zentrierte Überschrift</h1>
```

Gültigkeitsbereich einer Style-Angabe

Sie können innerhalb einer HTML-Datei sowohl eine externe Stylesheet-Datei verwenden (eingebunden durch eine entsprechende Anweisung im *head* Bereich) als auch eine lokale Definition im *head* Bereich angeben, als auch spezielle Angaben für einzelne Tags erstellen.

Was passiert nun, wenn die verschiedenen Style-Deklarationen sich widersprechen? Die Angaben bei einem einzelnen Tag haben immer Vorrang. Danach folgen die lokalen Angaben im *head* Bereich und erst zum Schluß die externe Datei. „Je näher beim Tag desto stärker wirkt es."

15.1 Klassen, id, span und div

Wenn Sie mit den Style-Angaben den *p*-Tag umformatieren, betrifft das alle *p*-Tags in der Webseite. Oft möchten Sie aber ein oder zwei Absätze anders formatieren als die restlichen Absätze. Zu diesem Zweck können Sie sich im Stylesheet noch weitere Formatvorlagen - sogenannte "Klassen" - definieren:

```
.wichtig { color: red; }
```

Diesen Klassen können Sie eigene Namen geben (hier „wichtig"), vor dem Klassennamen steht immer ein Punkt. Jedem normalen HTML-Tag können Sie nun diese Klasse zuweisen. Dies geschieht mit dem HTML-Attribut *class*.

```
<p>Eine <b class="wichtig">ganz wichtige</b> Meldung</p>
<p class="wichtig">Ein ganz wichtiger Absatz</p>
<p>Ein ganz normaler Absatz</p>
```

Zur „normalen" Wirkung des HTML-Tags kommt nun die Formatierung durch die Klasse hinzu: der Text „ganz wichtige" und „Ein ganz wichtiger Absatz" ist in diesem Beispiel also rot. Eine Klasse kann also mehrmals in einem Dokument verwendet werden. Ein Tag kann mehrere Klassen erhalten, diese werden durch Leerzeichen getrennt im *class*-Attribut angeführt.

```
<p class="wichtig eilmeldung">Ein wichtiger Absatz als Eilmeldung</p>
```

Zur eindeutigen Kennzeichnung von Tags wird das Attribut *id* verwendet:

```
<p>Eine <b class="wichtig">ganz wichtige</b> Meldung</p>
<p class="wichtig">Ein ganz wichtiger Absatz</p>
<p id="impressum">Das einzige Impressum dieser Seite</p>
```

Auf diese eindeutigen 'id's kann in CSS mit der Raute (englisch: „hash sign") referenziert werden:

```
#impressum { background-color: #DDD; }
```

Sowohl Klassen als auch 'id's können mit Tags kombiniert werden um einen komplexen Selektor zu bilden, aber das macht nur bei Klassen wirklich Sinn:

```
.wichtig { font-size: 20px; }   /* alle Tags mit der Klasse wichtig     */
p.wichtig { color: red;    }    /* nur der Tag p mit der Klasse wichtig */

b.wichtig { color: yellow; }    /* nur der Tag b mit der Klasse wichtig */

#impressum { background-color:#ddd; }   /* nur der Tag mit der id #impressum */
p#impressum { background-color:#ddd; } /* == nur der p-Tag mit der id #impressum */

b#impressum { background-color:#ddd; }   /* nix! */
```

Es wird öfter vorkommen, dass Sie einem Bereich eine bestimmte Klasse zuweisen möchten, ohne dass ein passender Tag vorhanden ist. Hier können Sie die beiden Tags *span* und *div* verwenden, die beide selber kaum Eigenschaft aufweisen. *span* eignet sich für die Verwendung in Fließtext, *div* ist ein blockbildender Tag.

```
<p>Es gibt hier ganz <span class="wichtig">besonders interessante</span> Meldungen.</p>
```

15.2 Maßeinheiten in Stylesheets

Für Längen- und Größen-Angaben gibt es mehrere Maßeinheiten:

vw	Hundertstel der Viewport-Breite
vh	Hunderstel der Viewport-Höhe
px	Pixel
em	Breite des Buchstaben M
ex	Höhe des Buchstaben X
%	Prozent
cm	Zentimeter
in	Inch

Einige davon sind relativ (*em* = relativ zur Schriftgröße, *px* = relativ zur Pixel-Größe am aktuellen Ausgabemedium), andere absolut (Zentimeter, Inch). Die absoluten Angaben kann man derzeit nur bei der Ausgabe auf Papier sinnvoll verwenden.

Farbangaben können auf mehrere Arten erfolgen: mit einigen Farbwörtern (*red*, *green*, ...) oder mit der Angabe von rot-, grün- und blau-Anteil (jeweils Werte von 0 bis 255) in verschiedenen Schreibweisen: Dezimal *rgb(16,0,255)*, hexadezimal *#10F* oder zweistellig hexadezimal *#1000FF*. Mit CSS3 ist auch die Angabe eines Alpha-Wertes möglich: Hier ein Braun-Ton der nur zu 20% deckend ist, und zu 80% den darunter liegenden Content bzw. das Hintergrundbild durchscheinen lässt: *rgba(153, 134, 117, 0.2);*

Tool-Tipp: Mit dem Firefox AddOn Colorzilla kann man die Farben einer Webseite auslesen und in verschiedenen Schreibweisen kopieren:

Abbildung 15.1: Abbildung: Colorzilla Pipette zum Auslesen einer Farbe und Colorzilla Menü zum Kopieren des Codes

Wird in einem Stylesheet auf eine URL verwiesen (z.B. auf die URL eines Hintergrundbildes), dann kommt die Schreibweise *url(http://absolute.com/bild.gif)* oder *url(relativ/bild.gif)* zum Einsatz. Achtung: die relative URL bezieht sich auf das Stylesheet (nicht die HTML-Datei in der es verwendet wird).

Wichtige CSS Properties

Schrift

Dieses Beispiel zeigt weitere wichtige Beispiele für Properties:

```
h1,h2 {
  font-family: "Trebuchet MS", Verdana, Arial, sans-serif;
  font-size: 18px;
  letter-spacing: 0.4em
  font-style: italic;
  font-variant: small-caps;
  font-weight: bold;
  text-decoration: underline;
  text-transform: uppercase;
  text-shadow: orange 0 -2px;
}
```

Webfonts

Als Schriften kann man einerseits Schriftarten verwenden, die am Client schon installiert sind, und andererseits kann man auf Schriften verweisen, die im Web gespeichert sind. Eine praktische Möglichkeit Webfonts aus den Web zu laden bietet http://www.google.com/webfonts.

Absätze

Wie immer in HTML erfolgt der Zeilenumbruch automatisch. Mit CSS können Sie verschiedene Aspekte des Absatz-Layouts steuern, einige davon sind in der Abbildung unten gezeigt:

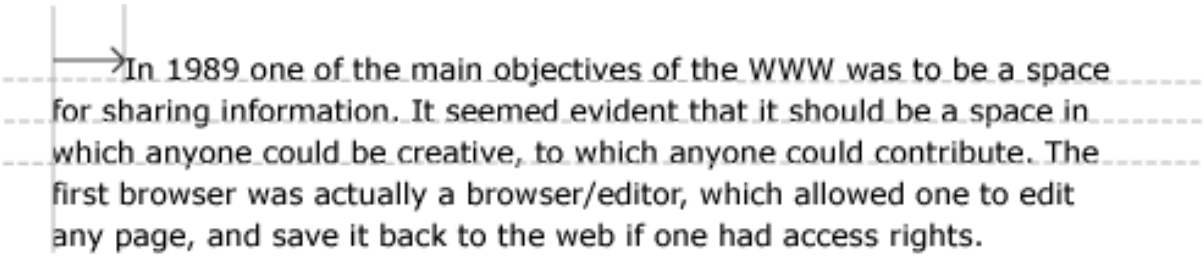


Abbildung 16.1: Abbildung: Absatz mit CSS Formatanweisungen: *text-indent* und *line-height*](/images/image066.png)

Mit *text-align* können Sie die Ausrichtung des Texts im Absatz festlegen: *left*, *right*, *center* oder *justify* (Blocksatz). Blocksatz wurde im Web bis jetzt wenig verwendet, da die Browser lange keine Silbentrennung durchführten. Dadurch entstanden bei Blocksatz oft häßliche Löcher im Text. Seit dem Jahr 2011 unterstützen erste Browser die Silbentrennung, damit wird *justify* besser verwendbar.

Die erste Zeile des Absatzes kann einen separaten Einzug haben, den Sie mit *text-indent* festlegen. Die Zeilenhöhe wird mit *line-height* festgelegt. Hier ist es sinnvoll, für längere Texte einen etwas erhöhten Wert festzulegen (z.B. *1.5em* – d.h. 1,5 Mal die Breite des Buchstaben M in dieser Schrift). Die Standard-Darstellung der Browser ist etwas zu eng um gut lesbar zu sein.

```
p {
   text-align: justify;
   text-indent: 4em;
   line-height: 1.2;
}
```

16.1 Box Model

Jeder blockbildende Tag (z.B. *h1*, *h2*, *p*, *blockquote*, *div*, ...) hat einen Rahmen, Innen- und Außenabstand. Diese werden mit den Deklarationen *border*, *padding* und *margin* festgelegt. Ein Hintergrundbild und/oder eine Hintergrundfarbe des Tags reicht immer bis zum Rahmen, aber nicht darüber hinaus.

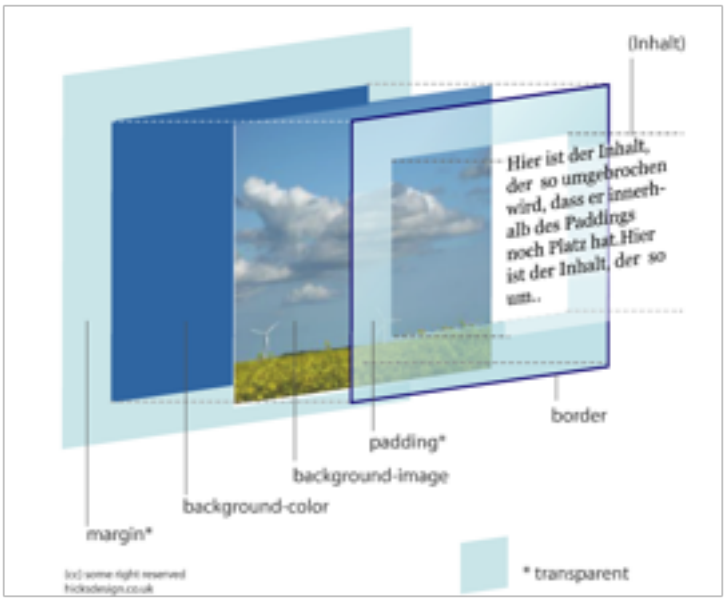


Abbildung 16.2: Abbildung: Darstellung des Box Models von http://hicksdesign.co.uk

Die Ausdehnung von *margin*, *padding* und *border* kann man besonders gut mit der Firefox-Erweiterung Firebug erforschen wie in der Abbildung unten gezeigt.

Dabei wird direkt in der Webseite der Außenabstand (*margin*) gelb und der Innenabstand (*padding*) dunkelviolett hinterlegt.

Achtung: Im traditionelle Box Model bezieht sich die Breite (*width*) auf den Inhalt, *padding*, *border* und *margin* muss man erst dazu zählen, um den Gesamt-Platzbedarf zu errechnen. Mit der Deklaration *box-sizing: border-box* kann man auf ein besseres Box-Model umschalten: dann gibt *width* die Gesamt-Breite an.

```
p {
  padding: 5px;
  margin-top: 5px;
  margin-right: 10px;
  margin-bottom: 5px;
  margin-left: 10px;
  border-width: 0px;
  border-right-width: 1px;
  border-bottom-width: 1px;
```

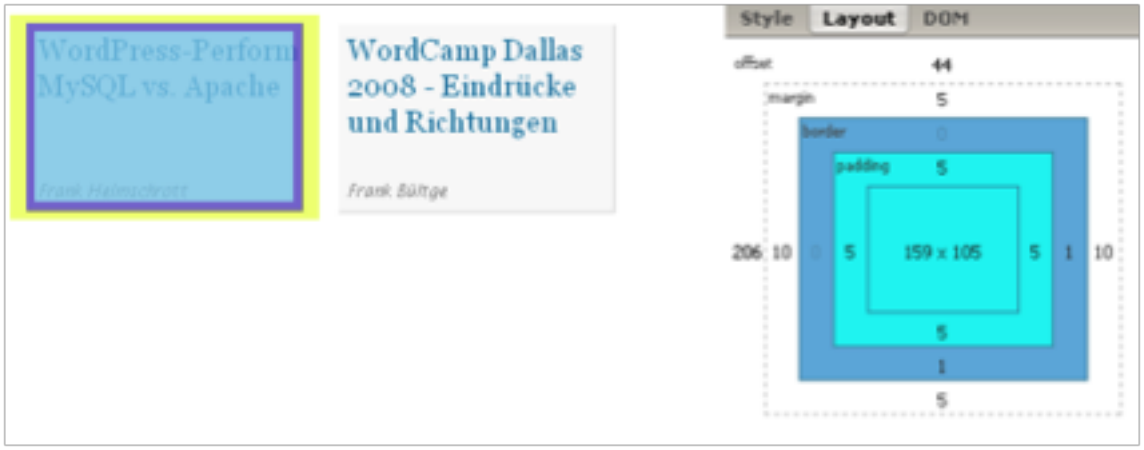


Abbildung 16.3: Abbildung: *margin*, *border*, *padding* in Firebug

```
  background-color: #DDD;
}
```

Mit CSS3 sind zusätzliche Effekte zum Box-Model dazu gekommen: abgerundete Ecken und Schatten:

```
button {
   color: white;
   text-shadow: 0 1px 1px black;
   padding: 5px 30px;
   background-color: red;
   border: 1px solid maroon;
   border-radius: 4px;
   box-shadow: inset 0 1px 3px pink, inset 0 -5px 15px maroon, 0 2px 1px black;
}
```

So sieht's aus:

Abbildung 16.4: Abbildung: Beispiel wie der Button dargestellt wird

Das ist ein Beispiel aus Beautiful UI styling with CSS3 (http://dev.opera.com/articles/view/beautiful-ui-styling-with-css3-text-shadow-box-shadow-and-border-radius/), dort finden sich noch viele ausführlichere Beispiele.

Absatz mit *p*

Die Standard-Darstellung von Absätzen erklärt sich über den *margin-top* und *margin-bottom* des *p*-Tags:

16.2 Farben, Hintergrundfarben, Hintergrundbilder

Die Farbe des Textes wird mit *color*, die Hintergrundfarbe mit *background-color* gesetzt.

Jeder Tag kann mittels CSS ein oder mehrere Hintergrundbilder erhalten (*background-image*). Als „Hintergrundbild" in einer Webseite kann jedes Bild in einem Web-geeigneten Dateiformat (*GIF*, *JPG*, *PNG*) dienen. Das Bild wird einfach dargestellt oder „gekachelt" – horizontal und vertikal so oft wiederholt, bis es die ganze Fläche des Tags ausfüllt (*background-repeat*).

Abbildung 16.5: Abbildung: Standard-Darstellung von Absätzen (*p*) im Browser

Bitte beachten Sie: Der Inhalt einer Seite sollte trotz Hintergrundbild immer noch lesbar sein!

CSS Selektoren

Um CSS Selektoren zu verstehen muss man das Document Object Model (DOM) betrachten, die Darstellung des HTML-Dokuments als Baum:

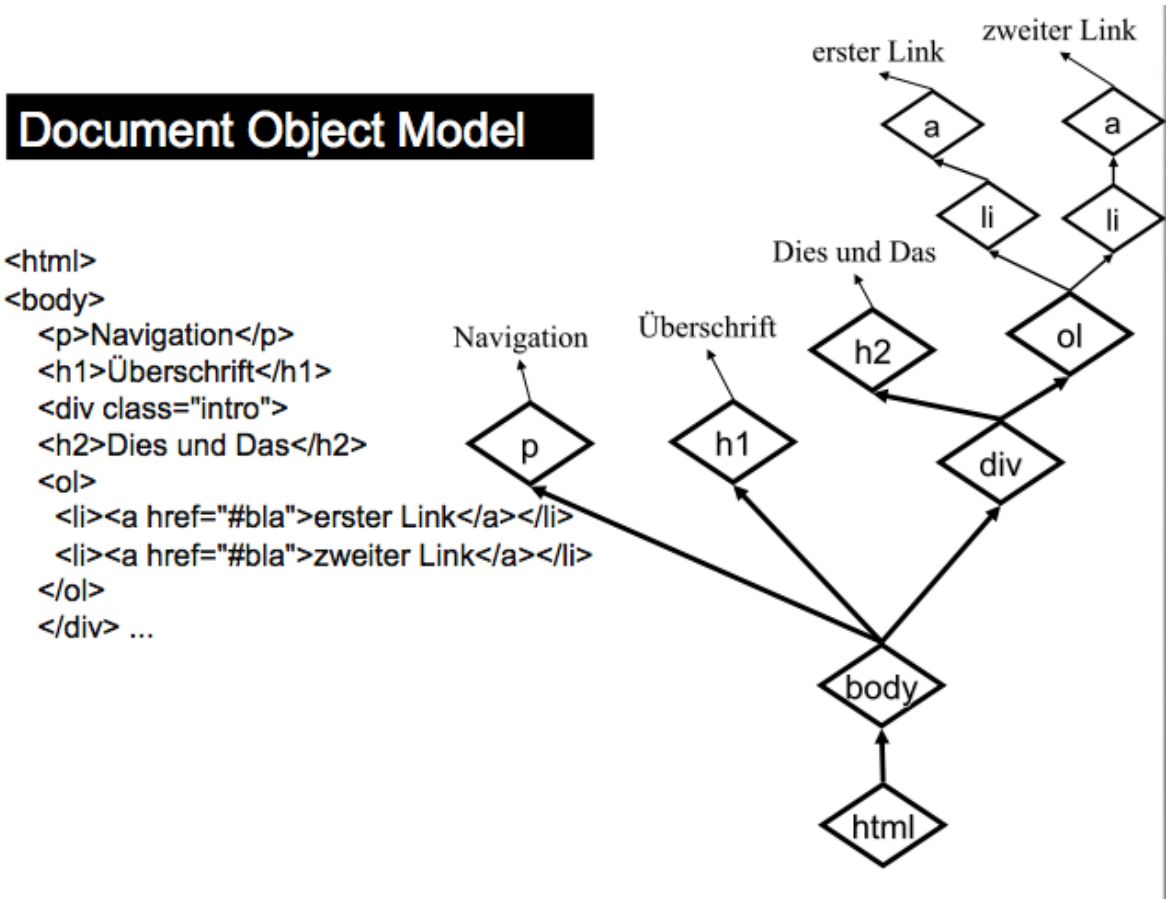


Abbildung 17.1: Abbildung: Document Object Model (DOM)

Bei der Beschreibung des Baumes verwendet man folgende Fachbegriffe:

- Elemente,
- Texte,
- Nodes
- Eltern-Element,
- Kinder

- Vorfahren,
- Nachkommen

17.1 Universal Selector

Der Selektor * wählt alle Elemente des Baums aus:

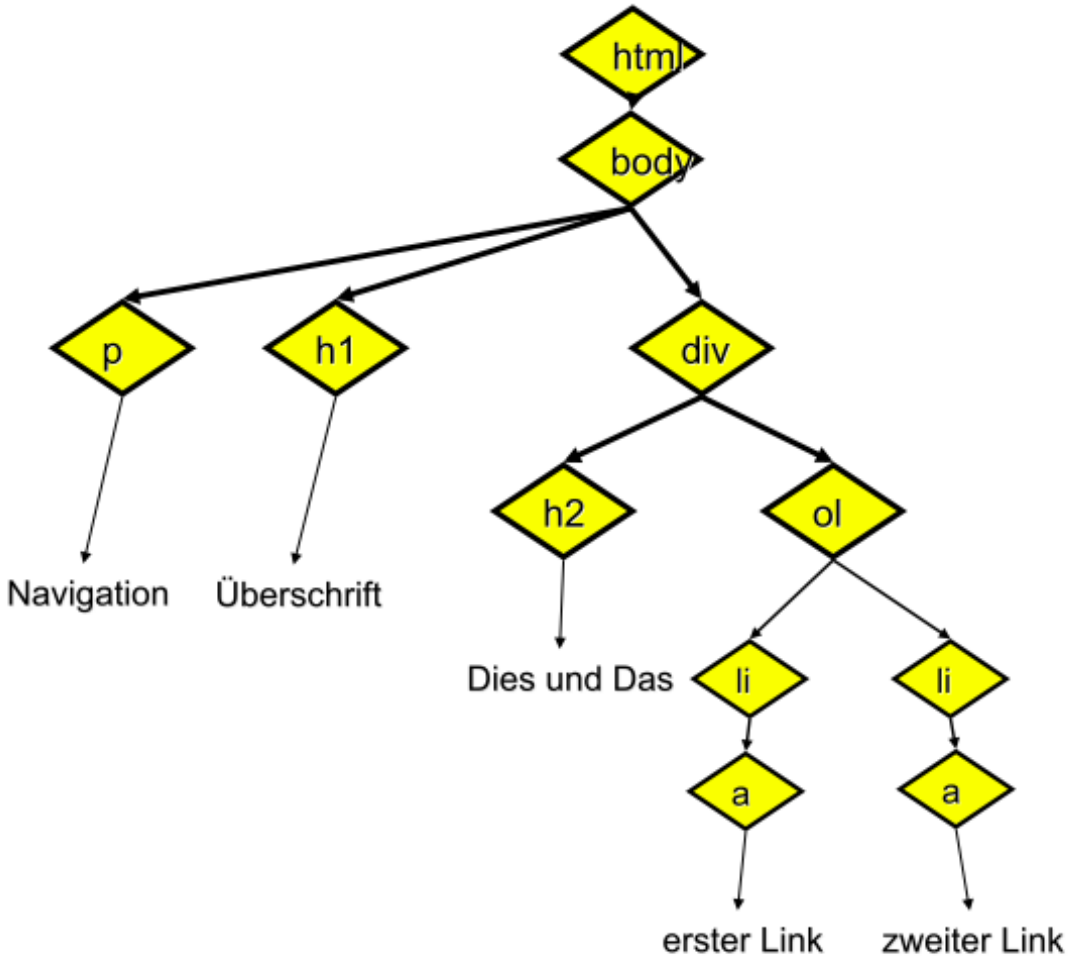


Abbildung 17.2: Abbildung: Document Object Model und Universal Selector

17.2 Type Selector

Über den Namen des HTML-Tags wählt man alle Elemente dieses Typs aus, zum Beispiel wählt *li* alle Listen-Elemente aus:

```
li { color: red; }
```

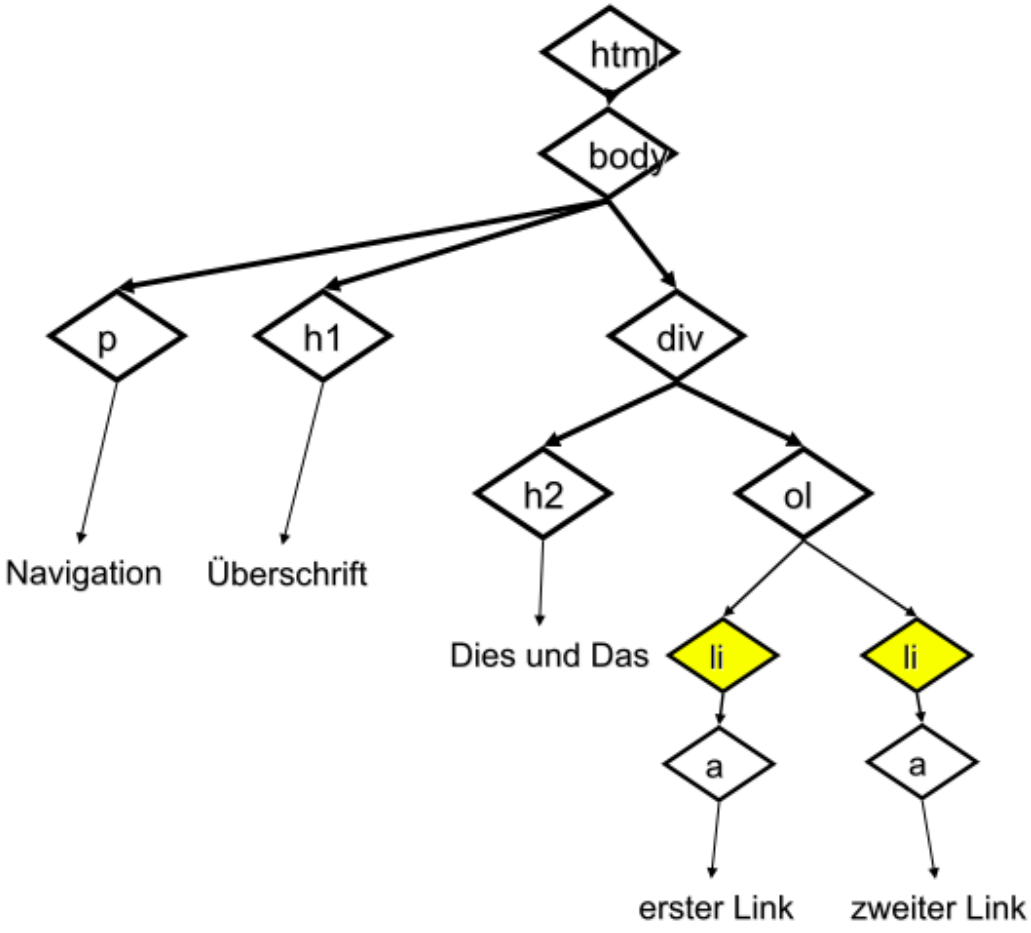


Abbildung 17.3: Abbildung: Document Object Model und Type Selector

17.3 Group Selector

Mehrere Selektoren können mit Kommas zu einem neuen Selektor gruppiert werden. Das Komma entspricht einem "Oder": selektiert werden Tags die entweder *h1* sind, oder *h2*, oder *a*:

```
h1,h2,a { color: red; }
```

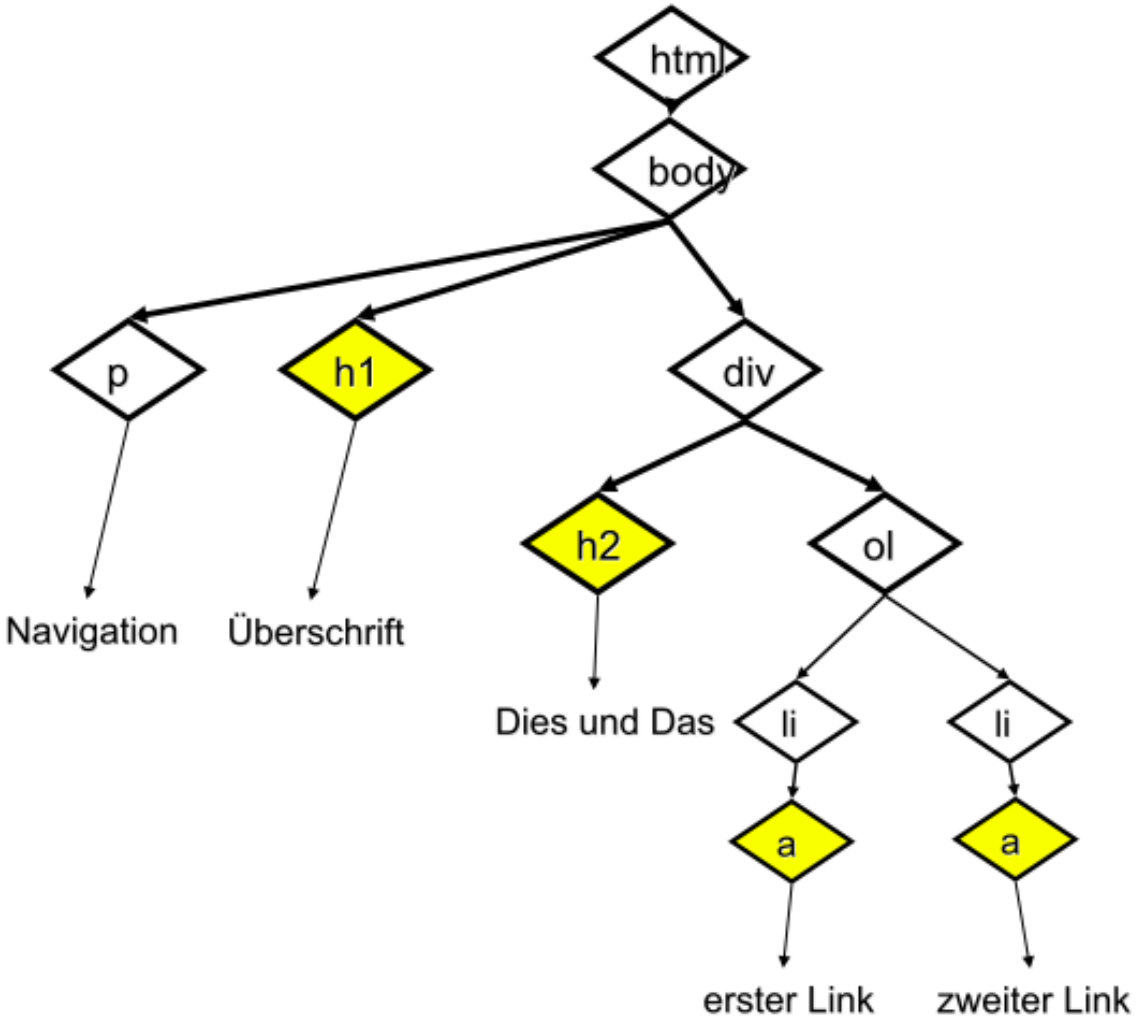


Abbildung 17.4: Abbildung: Document Object Model und Group Selector

(Achtung Falle! Für Links mit dem *a* Tag gelten noch zusätzliche Regeln, siehe weiter unten.)

17.4 Descendant Selector

Hier wird ein Element ausgewählt, das Nachkomme eines anderen Elements ist.

Achtung: *div* wird nur zur Auswahl benützt, wird aber selber nicht ausgewählt!

```
div a { color: red; }
```

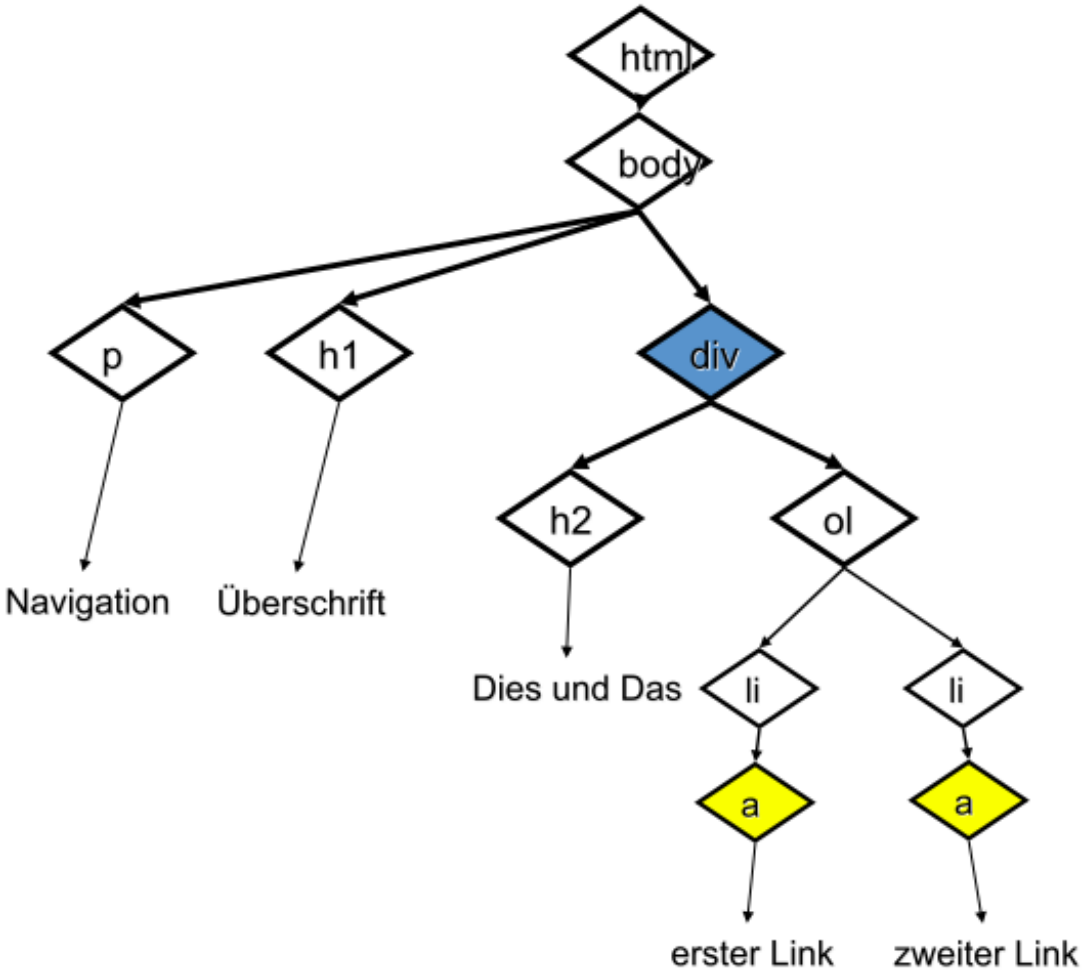


Abbildung 17.5: Abbildung: Document Object Model und Descendant Selector

17.5 Links formatieren

Der *a*-Tag wird in HTML für zwei sehr unterschiedliche Dinge verwendet: zum Setzen von Links und zum Setzen von Textmarken (auch ‚Anker' genannt):

```
<h1><a name="unis"></a>Universitäten</h1>
<a href="http://www.uni-salzburg.at/">Uni Salzburg</a>
```

Textmarken sind normalerweise unsichtbar, Links sind normalerweise blau oder violett und unterstrichen.

Um die Darstellung von Links zu verändern muss man in CSS als Selektor *:link* oder *:visited* verwenden. Der Browser unterscheidet dabei zwischen Links die schon einmal besucht wurden ('visited') und neuen Links.

```
a:link, a:visited { text-decoration: none; }
a:link    {color:blue}
a:visited {color:#FF00FF }
```

Im folgenden Beispiel werden die Links mit einem Icon markiert:

```
a:link, a:visited {
   background-image:    url(icon-link.gif);
   background-position: center right;
   background-repeat:   no-repeat;
   padding-right:       9px;
}
```

KAPITEL 18

Graceful Degradation

Der englische Begriff ‚graceful degradation' könnte man als „allmähliche Funktionsminderung" übersetzen, er bedeutet ungefähr: „funktioniert auch ohne … gut."

Das dahinter stehende Prinzip lautet: für die verschiedenen Ausgabemedien sollen nicht verschiedene Versionen einer Webseite erstellt werden, sondern alle Ausgabemedien werden mit einem Dokument bedient.

Wenn der Browser Javascript, Flash, CSS unterstützt, dann soll die Website besonders schön, interaktiv, praktisch sein. Wenn der Browser etwas nicht unterstützt, dann soll die Website immer noch lesbar und benutzbar bleiben.

Da CSS von allen "klassischen" Browsern am Desktop und den Browsern am Smartphone gut unterstützt wird sind solche Überlegungen immer mehr in den Hintergrund getreten. Bei der Verwendung von CSS3 Features ist das Thema aber ganz aktuell:

Siehe HTML5 & CSS3 Support (http://fmbip.com/litmus#css3-selectors).

Image Replacement

Der “Image Replacement Trick” wurde vor der Einführung von Webfonts verwendet um exotische Schriften darzustellen.

Heute spielt er noch eine Rolle bei der Arbeitsteilung zwischen Development und Design, z.B. beim Erstellen von Wordpress Themes.

Auf jeder Seite im Blog soll das Logo angezeigt werden, das ist gleichzeitig auch die Überschrift des Blogs.

```
<img src="logo-400.png">
```

Das sieht zwar wie eine Überschrift aus – wenn das Bild geladen wird – die Information ist aber für eine Suchmaschine oder ein Braille-Ausgabegerät nicht lesbar.

Besser wäre:

```
<h1><img src="logo-400.png" alt="Company Logo"></h1>
```

In manchen Situation hat man aber nur folgenden Code im HTML, und will das Bild nur über CSS darstellen:

```
<h1>Company</h1>
```

Ein Beispiel für so eine Situation ist z.B: ein Wordpress Theme. Mit *h1* wird der Titel des Blogs angezeit, die Gestaltung der Darstellung erfolgt nur mit CSS.

Der Text wird normal im HTML-Code eingegeben. Bei CSS-fähigen Browsern wird der Text versteckt und das Bild angezeigt. Browser ohne CSS und Suchmaschinen verwenden einfach den Text:

```
h1#bildStattText {
   /* schiebt den "echten text" extrem weit nach links */
   text-indent:-10000px;
   overflow:hidden;
   background: url(logo-400.png);

   /* hoehe und breite der grafik angeben! */
   height:140px;
   width:400px;
}

<h1 id="bildStattText">Company</h1>
```

Das Bild kann ausgetauscht werden indem man das Stylesheet ändert, ohne Eingriff in den HTML-Code.

KAPITEL 20

Formulare

Mit Formularen gestalten Sie ihre Webseite interaktiver.

Was Sie wissen sollten

- Ich verstehe wie Web-Formulare mit URLs zusammen hängen.
- Ich weiss, dass die URL (bzw. der HTTP Request) und nicht das Formular die eigentliche Schnittstelle zum Programm am Webserver ist.
- Ich kenne die HTML-Tags die ein Formular und seine Eingabeelemente definieren.
- Ich bin mir bewusst welche Interaktions-Möglichkeiten und welche Interaktions-Einschränkungen mein Formular den UserInnen bietet.
- Ich kenne das Konzept “Ereignisgesteuerte Programmierung” und kann mehrere Javascript-Events nennen.

Was Sie können sollten

- Ich kann ein Web-Formular erstellen oder verändern.
- Ich kann für einen Anwendungsfall verschiedene Formular-Varianten entwerfen und kann die Vor- und Nachteile der Varianten einschätzen.
- Ich kann ein alternatives Web-Formular für ein bestehendes Formular / Programm definieren.

KAPITEL 21

Eingabefelder

Mit Ihren bisherigen Kenntnissen können Sie schon Webseiten mit einfachen Interaktions-Möglichkeiten gestalten: mit Links ermöglichen Sie der LeserIn die Navigation durch das Web. Formulare ermöglichen mehr Interaktion – aber immer noch in einem sehr strengen, sehr strukturierten Rahmen. Typische Anwendungsgebiete für Web-Formulare sind das Eingabeformular der Suchmaschine Google oder ein Kontaktformular.

Tags für Formulare

Mit den HTML-Tags *form*, *input*, *textarea*, *option*, *select* werden Formulare aufgebaut. Hier der Code für ein einfaches Beispiel:

```
<form method="get" action="bestellung.cgi">
  Bitte schicken Sie den Newsletter an die E-Mail Adresse:
  <input type="text"   name="email">
  <input type="submit" value="Ja, ich will!">
</form>
```

Der *form*-Tag ist „unsichtbar“ und dient nur dazu, die anderen Eingabefelder zu bündeln. Im *action*-Attribut des *form*-tag wird angegeben, an welche URL die Daten zur Verarbeitung geschickt werden.

Im Browser sieht das oben gezeigte Formular so aus:

Abbildung 21.1: Abbildung: Darstellung eines Web-Formulars im Browser

Eingabefelder

Innerhalb des *form*-Tag bauen Sie das Formular aus verschiedenen Eingabeelementen und den „normalen“ HTML-Tags auf.

Text-Eingabefelder

Verschiedenen Arten von Text-Eingabefeldern.

Textfeld	<input type=”text” name=”kommentar”>
Textfeld (E-Mail)	<input type=”email” name=”mailaddr”>
Passwort-Feld	<input type=”password” name=”meinpasswort”>
mehrzeiliges Textfeld	<textarea name=”zitat”>Quod licet jovi...</textarea>

Achtung: das Passwort-Feld schützt nur vor neugierigen Blicken auf den Monitor. Die eingegebenen Daten werden dann genau so übertragen wie auch alle anderen Eingabefeldern - normalerweise unverschlüsselt.

Attribute für Text-Eingabefelder

mit Default-Wert	<input type=”text” name=”vorname” value=”Lara”>
muss eingegeben werden	<input type=”email” name=”nachname” required>
mit Platzhalter	<input placeholder=”ich@some.net (ich@some.net)“>
mit Eingabeprüfung	<input pattern=”.*@sbg.ac.at”>

Die Eingabeprüfung anhand von Muster-Ausdrücken (‘regular expressions’) ist eine neue Eigenschaft von HTML5, siehe dazu auch http://html5pattern.com/.

Ja/Nein Frage

Für einzelne Fragen die mit Ja oder Nein zu beantworten sind wird das Eingabe-Element *checkbox* verwendet.

```
<label><input type="checkbox" name="schlag"> mit Schlagobers</label>
```

Abbildung 21.2: Abbildung: *Checkbox* (Ja/Nein-Auswahl) im Browser

Auswahl

Für Fragen, bei denen eine von mehreren vorgegebenen Antworten möglich sein soll gibt es verschiedene Eingabeelemente.

Radiobuttons (die zusammen gehören) müssen das selbe Attribut *name* haben. Hier ist es wichtig die Beschriftung die zum jeweiligen Button gehört mit dem *label* Tag zu markieren - das erleichtert auch die Eingabe.

```
<label><input type="radio" name="size" value="XL">XL</label>
<label><input type="radio" name="size" value="L">L</label>
<label><input type="radio" name="size" value="M">M</label>
<label><input type="radio" name="size" value="S" checked>S</label>
```

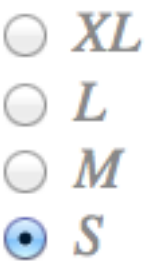


Abbildung 21.3: Abbildung: *Radiobutton* (Auswahl) im Browser

Ein Dropdown-Menü ist platzsparender:

```
<select name="size2">
  <option>XL</option>
  <option selected>L</option>
  <option>M</option>
  <option>S</option>
</select>
```

Die Mehrfach-Auswahl erfolgt durch Drücken der Steuerungs-Taste:

Spezialisierte Eingabefelder

Mit HTML5 wurden neue Eingeabefelder eingeführt, die aber noch nicht von allen Browsern unterstützt werden. (Mit Hilfe von Polyfills (https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills) kann man sie aber auf jeden Fall schon benutzen.)

Zum Beispiel gibt es Felder für Datum, Wertebereiche, Zahlen, Farben, etc.

Abbildung 21.4: Abbildung: *Dropdown*-Menü im Browser

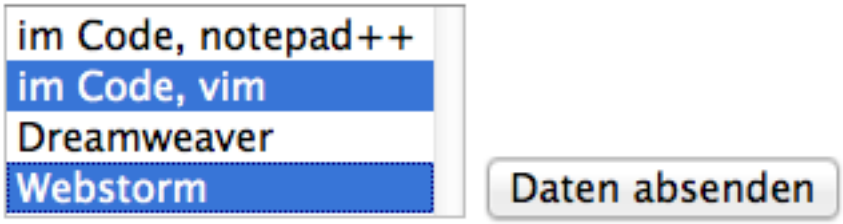


Abbildung 21.5: Abbildung: Mehrfachauswahl im Browser

Datum	<input type=”date” min=”2010-08-01” max=”2011-08-31” value=”2010-08-14”>
Range	<input type=”range” min=”0” max=”50” value=”10”>
Number	<input type=”number” step=”1” min=”-5” max=”10” value=”0”>

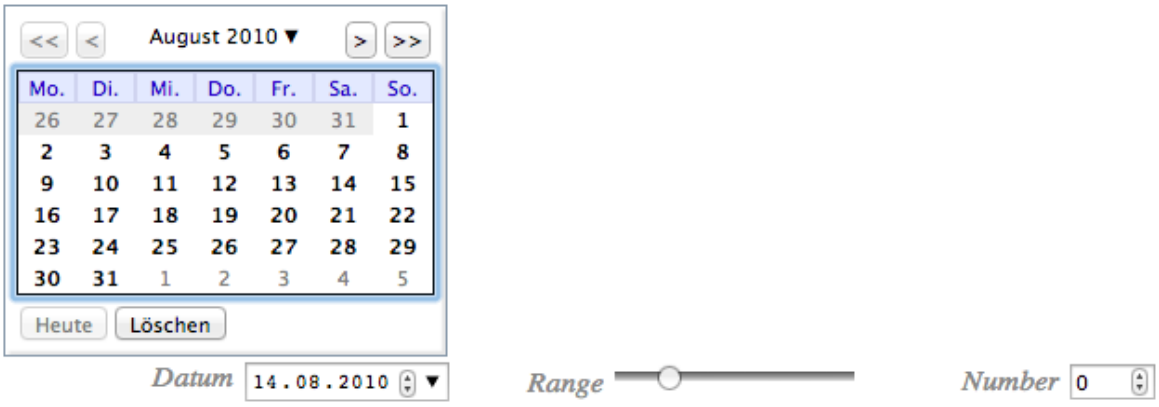


Abbildung 21.6: Abbildung: Spezielle Eingabefelder in HTML5

Absenden

Jedes Formular braucht einen Absende-Button:

Neben dem Absende-Button gibt es noch andere Methoden wie ein Formular „abgesendet“ werden kann: Falls das Formular nur ein einziges Text-Eingabefeld hat, kann man in diesem Feld einfach auf *Enter* drücken um das Formular zu senden. Wenn das Formular ein Bild-Feld enthält (siehe unten) veranlasst ein Klick auf das Bild das Einsenden.

Zurücksetzen? Nein Danke!

Den ‘Zurücksetzen’-Button sollten Sie nur sehr selten einsetzen. Warum? Denken Sie an Ihre eigene Erfahrung mit Web-Formularen zurück: Wie oft haben Sie auf einen ‘Zurücksetzen’-Button gedrückt und dann gedacht „super,

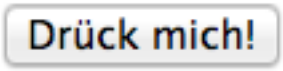


Abbildung 21.7: Abbildung: Absende-Button

Abbildung 21.8: Abbildung: Zurücksetzen-Button (nicht verwenden)

alles gelöscht, genau das wollte ich“, und wie oft haben Sie auf einen Zurücksetzen Button gedrückt und dann gedacht „Mist, das war ja gar nicht der Absende-Button, jetzt muss ich alles noch mal tippen.“

Label

Bisher haben wir nur die Eingabefelder selbst betrachtet. „Rundherum“ verwendet man alle bisher gelernten HTML-Elemente. So wäre es z.B. möglich, die Beschriftung einfach als Text neben das Eingabefeld zu stellen:

```
E-Mail <input type="text" name="mail" placeholder="ihre e-mail">
```

Damit ist aber nicht erkennbar, welche Beschriftung (vorher, nachher, weiter oben, weiter unten) zu welchem Eingabefeld gehört. Damit dieser Zusammenhang klar ist verwendet man den *label*-Tag:

```
<label>E-Mail <input type="text" name="mail" placeholder="ihre e-mail"></label>
```

Falls der Beschriftungs-Tag weiter entfernt ist kann der label auf die ID des Eingabefeldes verweisen:

```
<label for="mail">E-Mail</label>
...
<input type="text" name="mail" id="mail" placeholder="e-mail adresse">
```

Weitere Eingabefelder

Noch mehr Eingabefelder, die eher selten gebraucht werden.

Unsichtbares = Verstecktes Feld	<input type=”hidden” name=”source” value=”wikipedia”>
Button ohne Auftrag, für Javascript	<input type=”button” value=”Extrafenster” onclick=”...”>
Bildfeld, liefert X/Y Koordinaten	<input type=”image” name=”position” src=”austria.gif”>
Datei-Upload	<input type=”file” name=”bilddatei”>

Der Datei-Upload funktioniert nur wenn die Formular-Daten mit Methode POST und speziellem *enctype* an den Webserver geschickt werden.

Ordnung

```
<fieldset>
   <legend>Kreditkarte</legend>
   <input name="nr" placeholder="0000 0000 0000 0000">
   <label><input type="radio" name="kk" value="Master"> Master</label>
   <label><input type="radio" name="kk" value="Visa"> Visa </label>
</fieldset>
```

Abbildung 21.9: Abbildung: Zusammenfassen mehrerer Felder

Formular als Interaktion

Bei der Verwendung von Checkboxen, Radiobuttons, Menüs und Listen geben Sie genau vor, welche Möglichkeiten die LeserIn hat. Wenn Sie beim Entwurf des Formulars eine Möglichkeit vergessen, kann die LeserIn nichts mehr daran ändern. Deswegen ist hier besondere Sorgfalt geboten. Dazu ein paar Beispiele.

Keine Null-Antwort

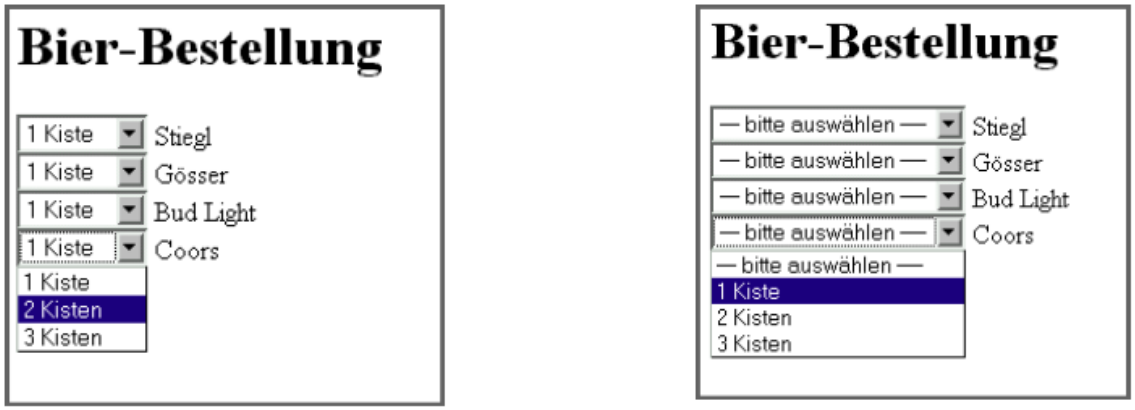


Abbildung 22.1: Abbildung: Eingabe immer erforderlich

Der linke Entwurf oben läßt keine Null-Bestellung zu. Die „Mindestbestellmenge“ für jede Sorte ist jeweils eine Kiste. Durch das Einfügen einer „Leer“-Option in die Menüs wird diese Einschränkung aufgehoben.

Hotel-Reservierung

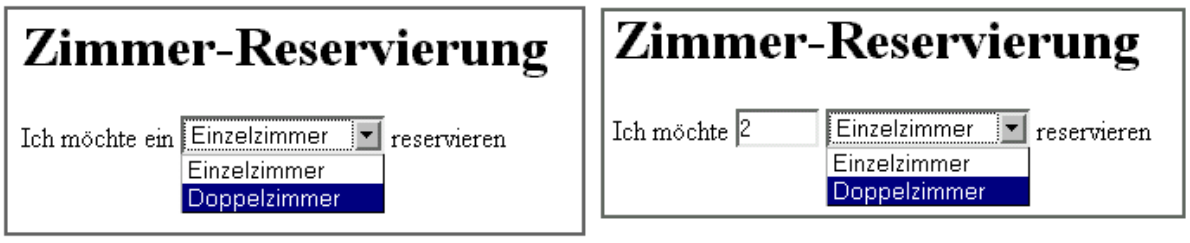


Abbildung 22.2: Abbildung: Zwei Entwürfe für ein Formular zur Zimmer-Reservierung

Version 1: Ein Menü ermöglicht die Auswahl aus vorgegebenen Elementen. Mit diesem Formular kann man nur ein, nicht aber zwei oder mehrere Zimmer reservieren. Version 2: Ein Text-Eingabefeld ermöglicht hier die Eingabe der Anzahl der Zimmer. Es ist aber nicht möglich ein Einzel- plus ein Doppel-Zimmer zu reservieren.

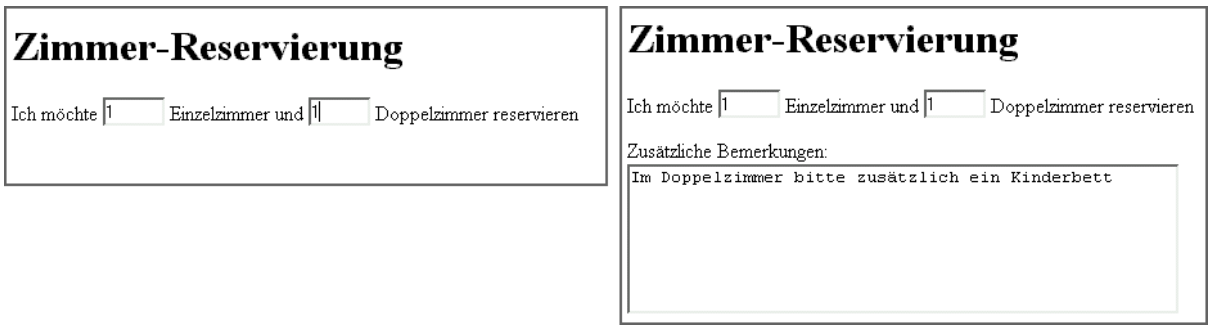


Abbildung 22.3: Abbildung: Zwei weitere Entwürfe für ein Formular zur Zimmer-Reservierung

Version 3 ermöglicht durch die Eingabe der Anzahl die Reservierung jeder Kombination von Einzel- und Doppelzimmern. Version 4: Durch ein zusätzliches Textfeld können alle weiteren Probleme abgefangen werden

Redundanz und Überfrachtung

Achtung: wenn Sie zu viele Eingabemöglichkeiten bieten entsteht Redundanz bzw. zu viel Information, dann können Sie die Bedeutung der Eingabe nicht mehr eindeutig erkennen.

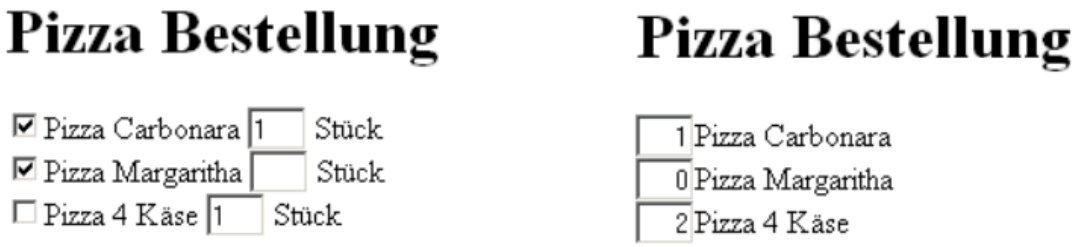


Abbildung 22.4: Abbildung: Redundanz im Formular

Im linken Formular wird die Anzahl der bestellten Pizzas auf zwei Arten festgelegt: durch eine Checkbox und eine Text-Eingabe. In der obersten Zeile stimmen diese beiden Eingabefelder in Ihrer Aussage überein: Ja, 1 Stück Carbonara. Aber es ist auch möglich eine in sich widersprüchliche Bestellung abzugeben: Ja, Null Stück Margaritha bzw. Nein, 1 Stück 4-Käse.

Im rechten Formular wird diese Redundanz vermieden.

Vertiefung

Wir haben bisher Formulare betrachtet, die nur HTML, nicht aber Javascript verwenden. Erst mit Javascript kann man Formulare wirklich flexibel bauen. Und erst mit serverseitiger Programmierung (z.B. in Python oder Ruby) kann man die eingegebenen Daten wirklich verarbeiten.

Weiterführende Literatur

- Evolving E-Commerce Checkout (http://www.lukew.com/ff/entry.asp?1579)

KAPITEL 23

Daten absenden

Was passiert, wenn Sie ein Formular ausfüllen und auf den Absende-Button drücken? Meist werden die Daten an den Webserver zur weiteren Verarbeitung geschickt. Diese *action* wird direkt im <*form*>-Tag angegeben.

```
<form action="http://pmeerw.net/www-mm14/pizza/bestellung.cgi">
```

Für die Verarbeitung der Daten braucht man ein serverseitiges Programm, ein sogenanntes CGI Skript. Dieses kann in Python, PHP, Ruby oder einer anderen Programmiersprache erstellt werden.

23.1 Daten an E-Mail senden

Wenn Sie kein Möglichkeit haben, am Webserver zu programmieren, bleibt als Alternative, die Daten direkt per E-Mail zu senden. So sieht der entsprechende HTML-Code des Formulars aus:

```
<form action="mailto:ich@sbg.ac.at" method="POST" enctype="text/plain">
```

Achtung: diese Art die Daten zu senden ist sehr fehleranfällig: hierfür muß am Client-Computer nicht nur der Browser funktionieren, sondern auch das E-Mail Programm. Das E-Mail Programm muß richtig konfiguriert sein. Das ist auf vielen Computern am Internet nicht der Fall, oft werden die Daten also nicht erfolgreich versandt sondern gehen verloren.

23.2 Daten senden mit HTTP GET

Wird im *form*-Tag als *action* eine URL mit HTTP angegeben dann werden die Daten an den Webserver gesendet. Dabei gibt es zwei Methoden: GET und POST. GET ist die einfachere.

```
<h1>Bestellung</h1>
<form action="bestellung.cgi" method="GET">
    Anzahl <input name="anzahl">
    <input type="submit" value="Abschicken">
</form>
```

Wenn die UserIn auf „Abschicken“ drückt, baut der Browser aus der *action* und den Namen und Werten der einzelnen Eingabefelder eine URL zusammen, die dann aufgerufen wird.

URL der GET-Anfrage

Die URL der Bestellung oben könnte etwa lauten:

```
http://pmeerw.net/www-mm14/bestellung.cgi?anzahl=4
```

Die URL wird zusammengestellt aus:

- der *action* aus dem *form*-Tag, hier *bestellung.cgi*
- *?* (Fragezeichen)

- **für alle Eingabefelder, getrennt durch & (kaufmännisches Und):**
  - Name des Eingabefeldes
  - = (Gleich)
  - Eingegebener bzw. angeklickter Wert

Falls dabei Sonderzeichen vorkommen (z.B. Leerzeichen, Zeilenumbrüche, Umlaute, Fragezeichen, kaufmännisches Und) werden diese wie folgt encodiert: Statt Leerzeichen wird ein + oder *%20* gesetzt, bei allen anderen Zeichen wird ein % gefolgt von der Hexadezimaldarstellung des ASCII-Code gesetzt (siehe RFC 2396). Diese Codierung nennt man URL-Encoding, siehe HTML 4.01 Specification (http://www.w3.org/TR/html4/interact/forms.html#h-17.13.4.1).

KAPITEL 24

URL als Programm-Schnittstelle

Das Webformular ist nicht notwendig, um eine GET-Anfrage zu erzeugen. Wenn Sie z.B. das Eingabe-Formular von Google analysieren, werden Sie herausfinden, dass die Anfrage mit dem Suchwort „Schokolade“ so aussieht:

```
http://www.google.com/search?q=Schokolade
```

Sie können diese URL einfach direkt in den Browser eintippen, ohne das Eingabeformular von Google zu verwenden. Sie können diese URL in den Lesezeichen/Favoriten Ihres Browsers speichern oder in einem Link verwenden:

```
<a href="http://www.google.com/search?q=schokolade">Suche nach Schokolade</a>
```

Wenn Sie eine Web-Applikation erstellen müssen Sie auch darauf gefasst sein, dass die UserInnen nicht nur ihre Web-Formulare verwenden, sondern auch URLs konstruieren und aufrufen. Die **URL** (und nicht das Formular) ist also eine **öffentliche Schnittstelle** zu Ihrem Programm!

Nun könnte man meinen: “Wenn ich die URL geheim halte ist es doch keine öffentliche Schnittstelle”. Das ist aber ein Trugschluss: auch eine “geheime” URL, die ich nie bewusst öffentlich mache wird öffentlich werden. URLs sind im Browser-Cache gespeichert, werden im Browser vorgeschlagen wenn ich eine URL eintippe, sind in Logfile von Proxies gespeichert. Selbst URLs die ich blos im Skype-Chat weitergegeben habe, sind dritten bekannt, wie diese Untersuchung von Heise zeigt (http://www.heise.de/security/meldung/Vorsicht-beim-Skypen-Microsoft-liest-mit-1857620.html).

Das “Geheimhalten” einer URL ist also keine geeignete Sicherheitsmaßnahme! Wenn ich eine Webseite vor Zugriffen schützen will brauche ich dazu Passwörter, HTTPS, und Authentisieren nach RFC 2617.

Pricing Attack

Ein lehrreiches Beispiel aus der Frühzeit des Web: es gab einst Webshops, die den Preis der Waren als verstecktes Eingabefeld im Formular speicherten:

```
<form action="order.php" method="GET">
Anzahl: <input name="anzahl"> <br>
Adresse: <textarea name="adresse"></textarea><br>
Preis: 1000 €
<input type="hidden" name="preis" value="1000">
<input type="submit" value="Bestellung absenden">
</form>
```

Das gibt der KundIn die Chance ein „Gegenangebot“ zu senden (als „pricing attack“ bekannt):

```
http://happyshop.com/order.php?anzahl=4&adresse=Urstein+Süd+.1&preis=10
```

Wenn nun das Programm *order.php* einfach den Preis aus der URL übernimmt und auf die Rechnung schreibt, wird die Bestellung recht günstig (für die KundIn).

Merke: Alle Eingaben die eine Web-Applikation erhält sind mit extremer Skepsis zu betrachten!

KAPITEL 25

CSS Layout

Nachdem Sie nun einfache Formatierungen im CSS beherrschen lernen Sie wie Sie das Layout der ganzen Seite beeinflussen können.

Was Sie wissen sollten

- Ich weiss wie verschiedene Ausgabegeräte und verschiedene Bildschirmgrößen die Darstellung von Webseiten beeinflussen und die Gestaltung von Webseiten schwierig machen.
- Ich verstehe wie absolute Positionierung, Float, und *media queries* funktionieren.

Was Sie können sollten

- Ich kann An einem bestehen CSS-Layout Veränderungen vornehmen.
- Ich kann mittels CSS zwei oder drei Spalten nebeneinander positionieren.
- Ich kann mittels CSS eine Liste von Links als Navigations-Menü darstellen.
- Ich kann ein statisches Layout mit *media queries* so erweitern, dass die Webseite bei wesentlich kleineren oder größeren Ausgabegeräten noch sinnvoll dargestellt werden kann.

Vertiefung

- Marcotte (2011): Responsive Design (http://www.amazon.com/Responsive-Design-Brief-People-Websites/dp/098444257X/)
- Responsive News (2013): Tables (http://responsivenews.co.uk/post/52382349921/tables) - Wie macht man Tabellen responsive? eine harte Nuss!

KAPITEL 26

Rahmenbedingungen für Layout

Wie in Kapitel 1 beschrieben gibt es viele verschiedene Ausgabegeräte für Webseiten. Für die Gestaltung des Layouts von Webseiten spielt dabei die Bildschirmgröße bzw. die Auflösung eine wichtige Rolle.

Auflösung

Zuerst stellt sich die Frage: woher weiß ich, wie hoch die Auflösung am Computer meiner LeserIn ist? Woher weiß ich, wie viel Platz im Browserfenster zur Verfügung steht?

Die Antwort lautet: ich weiß es nicht, und es gibt keine zuverlässige Methode, mit der man diese Information in jedem Fall herausfinden kann. Es gibt eine Messmethode mit Hilfe der Programmiersprache Javascript, mit der man die Größe des Browserfensters messen kann – das Ergebnis der Messung ist natürlich dadurch verfälscht, dass Browser ohne Javascript ganz aus der Messung herausfallen. Diese Beschränkung sollten Sie bei den folgenden Überlegungen immer beachten.

Wir werden *media queries* kennen lernen - das ist eine Methode um in CSS auf die Größe des Browserfensters zu reagieren - aber auch diese Methode ist nicht absolut zuverlässig, auch wenn sie derzeit (2012) schon von allen aktuellen Browsern unterstützt wird. Microsoft Internet Explorer unterstütz *media queries* erst ab Version 9.0. Siehe can i use (http://caniuse.com/css-mediaqueries).

Alle Messmethoden die uns zur Verfügung stehen sind unvollständig.

Vergleichen Sie die höchsten hier dargestellte Auflösungen mit der geringsten Auflösungen. Da Breite und Höhe (mehr als) verdreifacht sind, steht bei der höchsten Auflösung also (mehr als) die neunfache Fläche zur Verfügung!

Die nächste Abbildung zeigt Statistiken über die Bildschirmauflösung von http://w3schools.org/browsers/browsers_display.asp von 2000 bis 2011. In diesem Zeitraum hat sich die Mehrheit langsam von 800x600 (bis 2003) auf 1024x768 (bis 2008) und schließlich auf höhre Auflösungen verschoben. Achtung: Auf Grund des Messverfahrens werden hier wahrscheinlich nur Desktops erfaßt, nicht mobile Geräte.

Pixeldichte

Die Angabe der Auflösung erfolgt in Pixel – die reale Größe des Ausgabegerätes (24“ Desktop, 13“ Laptop, mobiles Endgerät) ist bei gleicher Pixel-Auflösung sehr unterschiedlich! Mobile Geräte haben eine geringe Auflösung, aber eine höhere Pixeldichte:

Gerät	Erscheinungsjahr	Pixel	Diagonale in Inch	Pixel per Inch
19” LCD Display	2008	1280 × 1024	19 “	86 ppi
15” Macbook Pro	2007	1440 × 900	15.4 “	110 ppi
Sony PSP 7th gen	2005	480 × 272	4.3 “	128 ppi
Apple IPhone3	2009	480 × 320	3.5 ”	163 ppi
ASUS Zenbook	2012	1920 × 1080	13.3 ”	165 ppi
15” Macbook Pro ‘Retina’	2012	2880 × 1800	15.4 ”	220 ppi
Apple IPhone4	2010	960 × 640	3.5 “	326 ppi
Apple IPad 3rd gen	2011	2048 × 1536	12 “	264 ppi
Amazon Kindle Paperwhite	2012	768 × 1024	6 “	212 ppi
Google Nexus One	2010	800 × 480	3.7 ”	254 ppi
Android Galaxy Nexus	2011	1280 × 720	4.65 ”	316 ppi

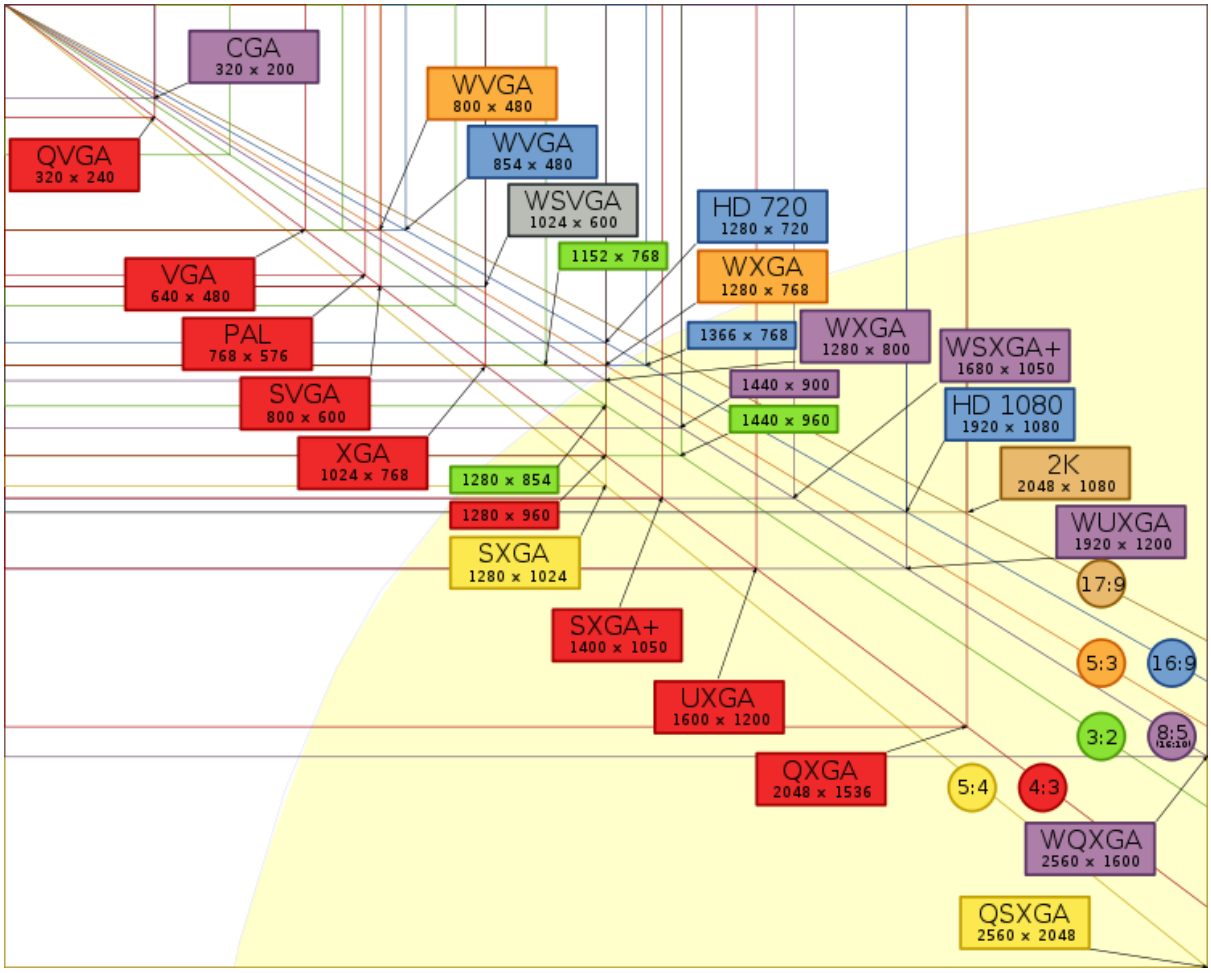


Abbildung 26.1: Abbildung: Einige mögliche Bildschirmauflösungen und Seitenverhältnisse (2011)

Date	Higher	1024x768	800x600	640x480	Other
January 2011	85%	14%	0%	0%	1%
January 2010	76%	20%	1%	0%	3%
January 2009	57%	36%	4%	0%	3%
January 2008	38%	48%	8%	0%	6%
January 2007	26%	54%	14%	0%	6%
January 2006	17%	57%	20%	0%	6%
January 2005	12%	53%	30%	0%	5%
January 2004	10%	47%	37%	1%	5%
January 2003	6%	40%	47%	2%	5%
January 2002	6%	34%	52%	3%	5%
January 2001	5%	29%	55%	6%	5%
January 2000	4%	25%	56%	11%	4%

Abbildung 26.2: Abbildung: Statistik über die Bildschirmauflösung

Brutto-Fläche vs. Netto-Fläche

Von diesen „Brutto-Angaben“ über die Größe der zur Verfügung stehenden Fläche sind nun noch der Platz für den Fensterrahmen des Browsers, für Scrollbalken, Symbolleisten, und eventuell eingeblendete Favoritenleisten abzuziehen, um den „netto“ verbleibenden Raum für die Gestaltung der Webseite zu erhalten.

Die Abbildung zeigt diese Problematik am Beispiel von Firefox.

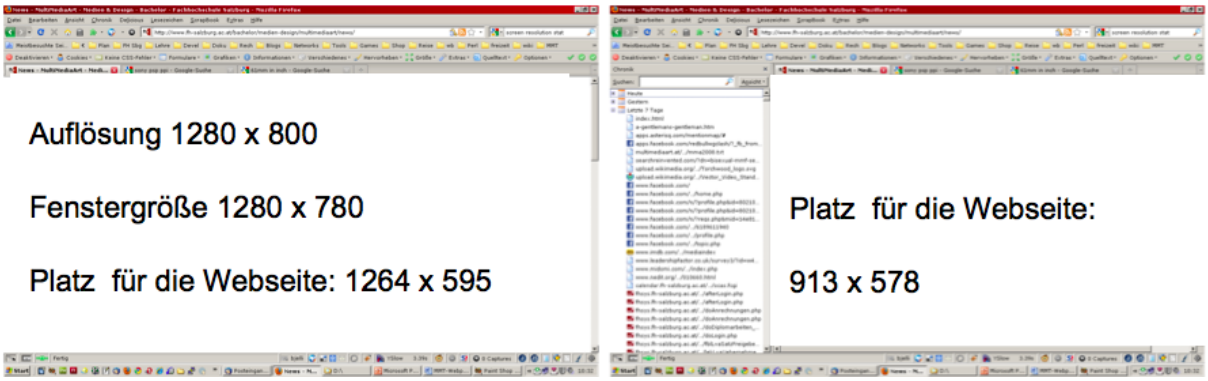


Abbildung 26.3: Abbildung: Platzbedarf von Browser-Elementen: Firefox ohne Sidebar, Internet Explorer mit Favoriten

Umgang mit der Problematik

Wie gehen WebdesignerInnen mit den verschiedenen Auflösungen um? Ein paar Varianten:

1. Ignorieren und für die eigene Bildschirmauflösung entwerfen. Manchmal in Kombination mit der Beschriftung „best viewed at 1600x1200“
2. Ignorieren dass es viele Bildschirmauflösungen gibt, und für das Minimum entwerfen.
3. Zwei oder drei Entwürfe, die den gleichen Inhalt für verschiedene Auflösungen unterschiedlich darstellen.

Dazu ein strenges Urteil:

1. Ist völlig inadäquat für das Medium Web. „best viewed“ ist eine Zumutung für alle LeserInnen auf “unpassenden” Ausgabegeräten. Stellen Sie sich vor, am Eingang eines Gebäudes wäre neben der Treppe ein Schild angebracht „nur benutzbar für Leute die Treppen steigen können“. Das Problem wurde erkannt, und absichtlich nicht gelöst?
2. Zeigt schon ein Minimum an Wissen über das Web, ignoriert aber die gestalterische Herausforderung des Mediums. Weil solch ein Entwurf auf einem Bildschirm mit hoher Auflösung sehr klein auf einer großen leeren Fläche erscheint wird es spöttisch „Briefmarkenlayout“ genannt.
3. Nur das verdient wirklich die Bezeichnung „Webdesign“.

Technische Umsetzung

Kleine Unterschiede im vorhandenen Platz kann man durch Zentrierung des Inhalts augleichen (siehe auch “Briefmarkenlayout”)

Fixen Breite und automatischer Seitenabstand ergeben eine Zentierung des Inhalts:

```
div#wrap {
  width: 76em;
  margin: 0 auto;
}
```

26.1 Responsive Design

Große Unterschiede im Platz kann man mit media queries in CSS behandeln. Die Verwendung von Mediaqueries wurde 2010 in einem Artikel von Ethan Marcotte in “A List Apart” unter dem Begriff Responsive Webdesign

(http://www.alistapart.com/articles/responsive-web-design/) popularisiert.

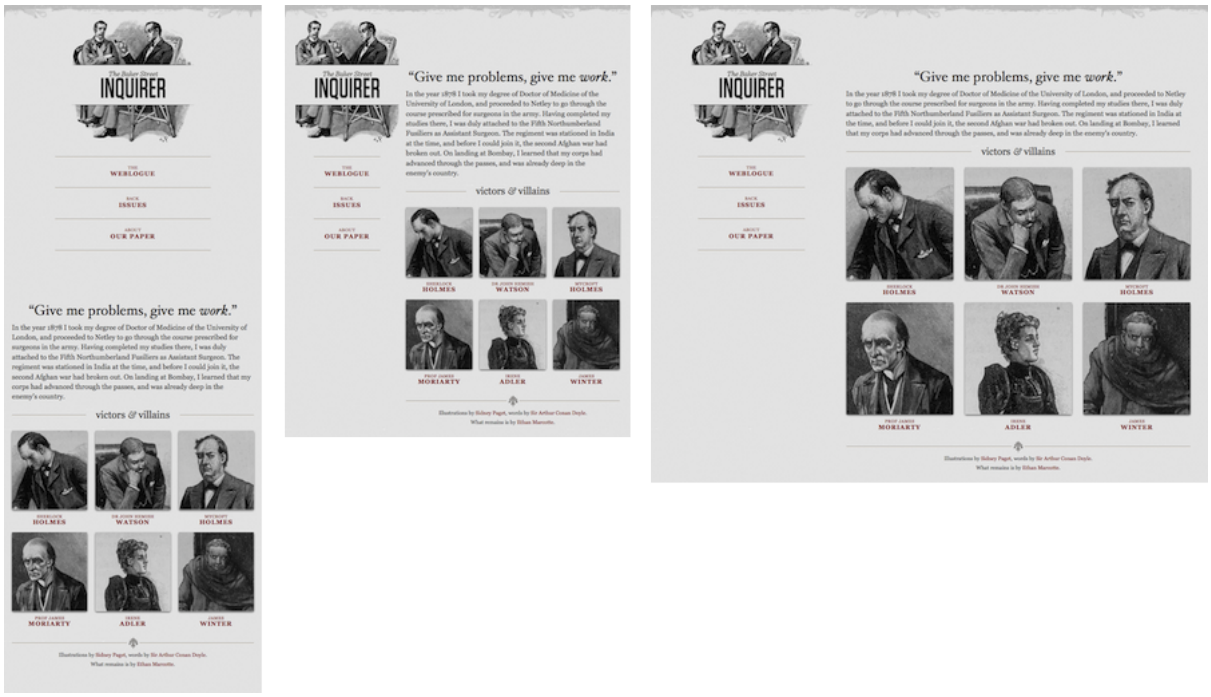


Abbildung 26.4: Abbildung: Beispiel für "Responsive Webdesign": Darstellung der Seite auf verschiedenen Breiten

26.2 Mobile First

Luke Wroblewski schlug schon 2009 unter dem Slogan Mobile First (http://www.lukew.com/ff/entry.asp?933) vor, zuerst die mobile Version der Website zu gestalten, und davon dann die "größeren" Versionen abzuleiten.

26.3 Media queries

Die technische Umsetzung des verschiedenen Layouts ist relativ einfach: Mediaqueries erlauben eine "Verzweigung" in CSS.

CSS mit Mediaqueries: Nur wenn Darstellung am Screen, und Breite kleiner gleich 480 Pixel

```
@media screen and (max-width: 480px) {
  .column {
  float: none;
}

/* weitere Regeln für kleine Screens */

/* allgemein gültige Regeln */
```

Bilder

Bilder waren lange Zeit ein Grund, warum das Layout von Webseiten nicht flexibel war: weil die Bilder nur für die Darstellung bei einer bestimmten Größe geeigenet waren. Das ist seit dem Jahr 2011 anders.

Pixel

Als Bildformate für <*img*> Tags in Webseiten werden nur Pixel-Formate von vielen Browsern unterstützt. Diese Formate (JPG, PNG, GIF) sind eigentlich für die Darstellung bei einer bestimmten Größe gedacht. Die vergrößerte Darstellung von Pixel-Bildern liefert keine guten Ergebnisse:

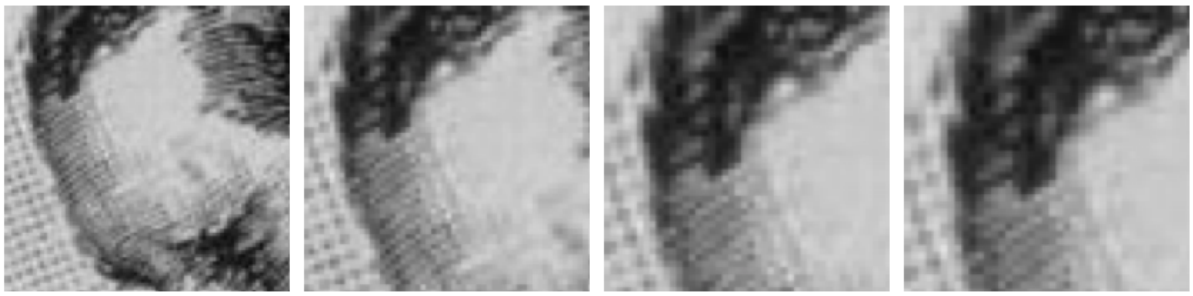


Abbildung 26.5: Abbildung: Ausschnitte aus einem Pixel Bild, vom Browser (Firefox) in 3 Stufen vergrößert dargestellt

Aktuelle Browser sind aber sehr gut bei der verkleinerten Darstellung von Pixel-Bildern.

Abbildung 26.6: Abbildung: Pixel-Bild wird vom Browser (Firefox) in 3 Stufen verkleinert dargestellt

Vektor

Mit dem Format SVG steht auch ein vektor-basiertes Bildformat für das Web zur Verfügung. SVG-Bilder können in beliebiger Größe verwendet werden. Die Einbindung erfolgt mit dem *img*-Tag:

```
<img src="circle.svg">
```

SVG erstellen

SVG-Dateien kann man im Code schreiben oder mit Inkscape, Adobe Illustrator oder anderen Vektor-Programmen erstellen.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
    "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
     version="1.1" baseProfile="full" width="100%" height="100%"
     preserveAspectRatio="xMinYMin meet" viewBox="0 0 300 300">
    <linearGradient id="orange_red"
        x1="0%" y1="0%" x2="100%" y2="0%">
        <stop offset="0%"
```

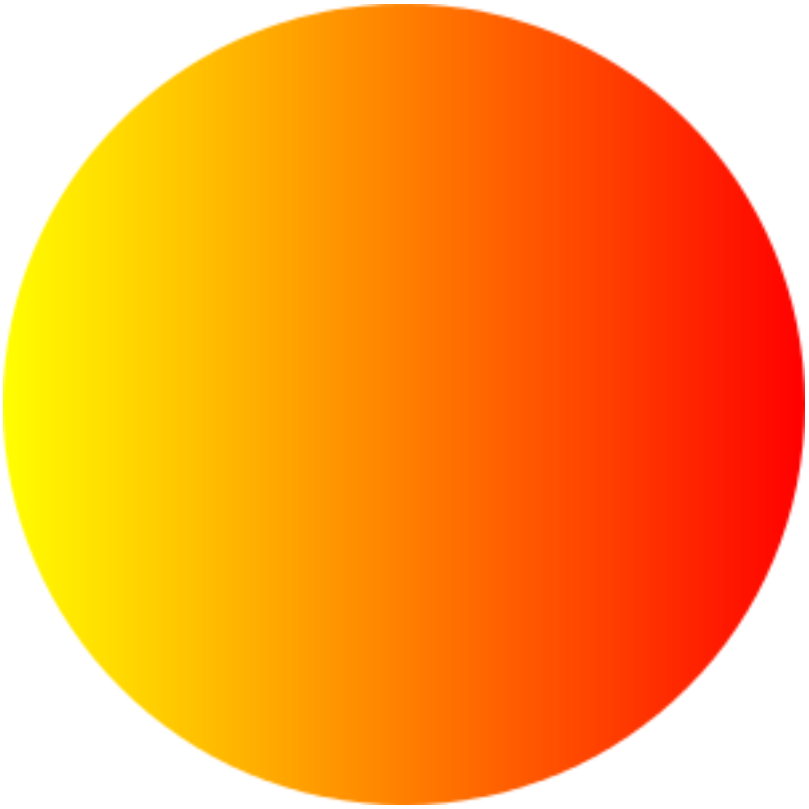


Abbildung 26.7: Abbildung: Kreis in SVG-Darstelllung

```
            style="stop-color:rgb(255,255,0); stop-opacity:1"/>
        <stop offset="100%"
            style="stop-color:rgb(255,0,0); stop-opacity:1"/>
    </linearGradient>
    <circle id="myCircle" cx="50%" cy="50%" r="100"
       fill="url(#orange_red)" />
</svg>
```

Das Attribut *preserveAspectRatio* im *svg*-Tag bestimmt wie das Bild auf verschiedenen Größen dargestellt werden soll.

Canvas

Der canvas-Tag bietet eine Leinwand, auf die mit Javascript in 2D oder 3D gezeichnet werden kann. Ohne Javascript ist er nur eine leere Leinwand, und wird deswegen hier noch nicht behandelt.

Schriftgröße

Die Schriftgröße im Browser unterliegt nur bedingte der Kontrolle durch HTML und CSS Code. Das „letzte Wort“ hat hier die LeserIn, die die Schrift größer oder kleiner stellen kann. z.B. im Internet Explorer unter Ansicht → Schriftgrad, in Firefox mit der Tastenkombination STRG + oder STRG –.

Je nach Schriftgröße und zur Verfügung stehendem Platz im Browser-Fenster wird der Browser die Absätze geeignet in Zeilen umbrechen, wie in der nächsten Abbildung gezeigt.

Skalieren

Beim Vergrößern und Verkleinern der Schriftgröße verwenden die Browser zwei verschiedene Methoden: entweder die Bilder werden mit der Schrift vergrößert und verkleinert (heute default), oder nur der Text wird verändern, die Bilder aber bleiben gleich. Hier das entsprechende Menü in Firefox.

Retina Displays

Mobile Geräte mit sehr hoher Pixeldichte stellen das Web Design vor eine neue Herausforderung: soll man –

Abbildung 26.8: Abbildung: Darstellung von Text bei verschiedenen Fensterbreiten und Schriftgrößen

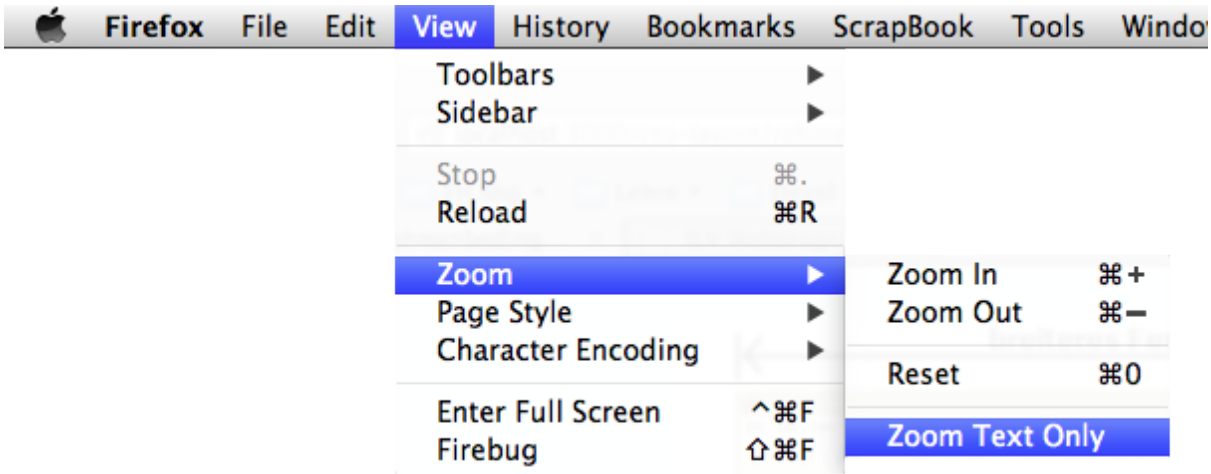


Abbildung 26.9: Abbildung: Zoom Menü in Firefox

wegen der Pixeldichte – riesige Bilder ausliefern? Oder – weil es doch ein mobiles Gerät ist, und die Datenübertragung über das Handy-Netz so langsam und teuer ist – doch kleine Bilder ausliefern?

Eine aktuelle Diskussion dazu findet man in

- Rupert (2012): Mo' Pixels Mo' Problems. In: A List Apart. (http://www.alistapart.com/articles/mo-pixels-mo-problems/)

Vertiefung

- Responsive Tables (http://blog.cloudfour.com/picking-responsive-tables-solution/)

CSS für Layout

Welche Gestaltungsmöglichkeiten bietet CSS nun?

Text

Die Darstellung einer HTML-Seite durch den Browser erfolgt von oben nach unten, je nach Sprache von links nach rechts oder von rechts nach links. Dabei wird zwischen blockbildenden Tags und nicht-blockbildenden Tags unterschieden. In nächsten Abbildung wurden <*p*> und <*em*>-Tags verwendet und mit folgendem Stylesheet formatiert:

```
p { background-color:#CCCCCC; }
em { background-color:#FFFF66; }
```

Bei der Darstellung im Browser kann der nicht-blockbildender Tag *em* dabei auf mehrere Zeilen verteilt werden und nimmt dann mehrere rechteckige Bereiche ein, der blockbildende Tag *p* wird immer als ein Rechteck dargestellt:

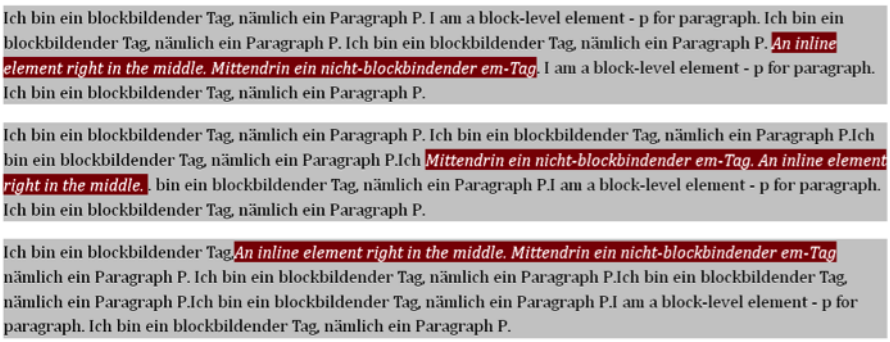


Abbildung 27.1: Abbildung: Darstellung von blockbildenden und nicht-blockbildenden Tags

Bild im Text

Ein Bild wird dabei wie ein Wort im Text behandelt, und nicht etwa frei auf der Webseite positioniert. Wenn Sie das Bild wie in der nächsten Abbildung mitten in einen Absatz hinein setzen, ergibt das meist ein sehr hässliches Layout.

Umbruch von Wörtern

Lange konnten Browser keine Wörter trennen und umbrechen, ein langes „Wort“ konnte dadurch die Breite eines Elements überschreiten.

CSS bietet ab Version 3 verschiedene Möglichkeiten diese Fälle zu behandeln, z.B mit der Eigenschaft text-overflow (http://css3clickchart.com/#text-overflow).

Derzeit ist es immer noch notwendig im Text die Soll-Bruchstellen als ** einzutragen, damit der Browser dann den Zeilenumbruch korrekt durchführen kann. Die Library hyphenator.js (https://code.google.com/p/hyphenator/) bietet den Umbruch u.a. für englische und deutsche Texte an.

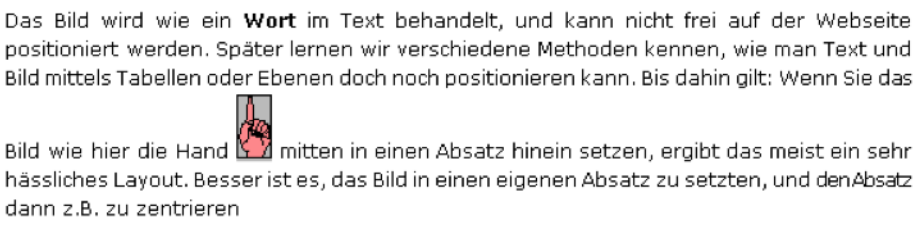


Abbildung 27.2: Abbildung: Bild mitten im Absatz

Width und Auto

Normalerweise nimmt ein Block die maximal zur Verfügung stehende Breite ein. Mit *width* kann eine andere Breite eines Blocks definiert werden:

```
div#main {  width:500px; }
```

Je nach *box-sizing* müssen Sie die Werte von *padding*, *border*, *margin* dazu addieren um den Gesamt-Platzbedarf zu errechnen, oder nicht: der default ist *box-sizing: content-box*.

```
box-sizing: content-box;
width: 200px;
padding: 10px;
border-width: 10px;
margin: 32px 0px
```

Gesamtbreite inklusive Rahmen = 0px + 10px + 10px + 200px + 10px + 10px + 0px = 240px

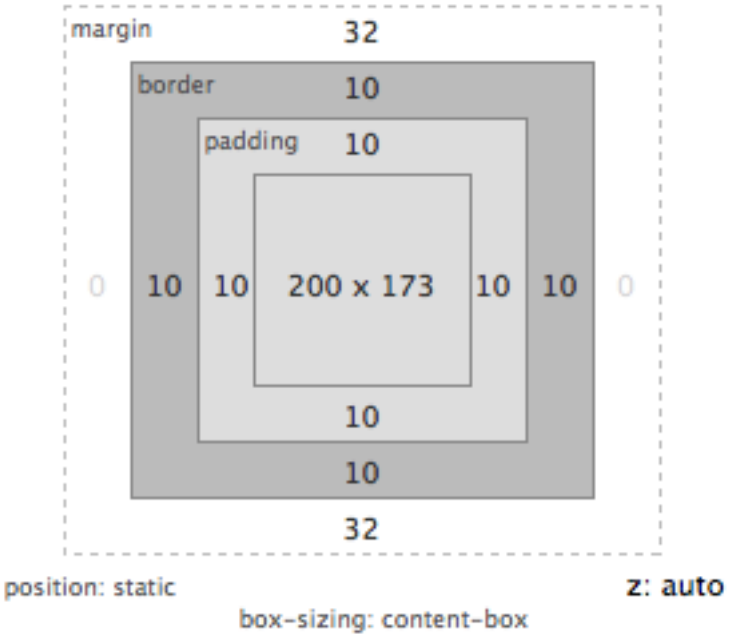


Abbildung 27.3: Abbildung: *width* im Box Model mit *box-sizing: content-box*

```
box-sizing: border-box;
width: 200px;
padding: 10px;
border-width: 10px;
margin: 32px 0px
```

Gesamtbreite inklusive Rahmen = 0px + 10px + 10px + 160px + 10px + 10px + 0px = 200px

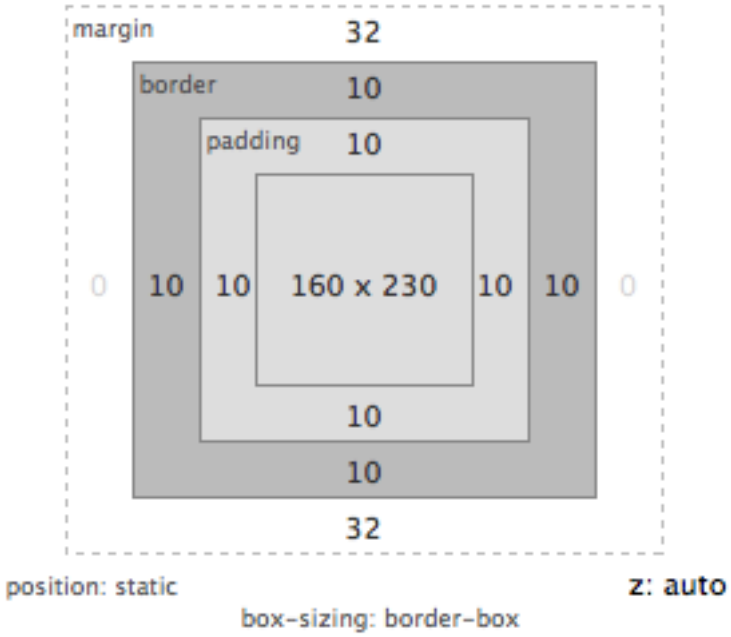


Abbildung 27.4: Abbildung: *width* im Box Model mit *box-sizing: border-box*

Zentrieren mit Auto

Um ein Element zu zentrieren kann *margin* mit Wert *auto* verwendet werden. Der zur Verfügung stehende Platz wird automatisch gleichmäßig verteilt.

```
div#main {
  width:500px;
  margin-left:  auto;
  margin-right: auto;
}
```

Float

Eine Möglichkeit aus der normalen Reihenfolge der Darstellung auszubrechen bietet die Deklaration *float* mit den Werten *left* und *right*. Damit wird ein Element nach links bzw. rechts gesetzt, der Rest des Inhalts „rutscht nach oben“ und wird neben das Element gesetzt („umfließt das Element“).

Hier sind drei Absätze zu sehen, die jeweils als erstes ein Bild enthalten. Im ersten Absatz ist die Darstellung ganz normal – das Bild wird wie ein Wort im Text behandelt. Im zweiten Absatz „floated“ das Bild nach rechts, der Text rutscht links davon nach oben. Im dritten Absatz „floated“ das Bild nach links, der Text rutscht rechts davon nach oben.

Bei Bildern funktioniert *float* besonders einfach, weil das Bild schon eine fixe Breite hat. Wird ein anderer HTML-Tag mit float versehen muß man auch die Breite des Tags setzen um einen sichtbaren Effekt zu erzielen. Soll ein Absatz nach rechts gehen, dann muss man dafür die Breite noch setzen.

Wenn viele Elemente mit *float* positioniert werden die die gleiche Höhe haben entsteht ein besonders flexibles Layout: im folgenden Beispiel werden je nach zur Verfügung stehendem Platz drei oder vier Bilder nebeneinander dargestellt:

Absolute Positionierung

Mit der Deklaration *position: absolute* wird ein Tag aus der normalen Darstellung herausgenommen und über dem restlichen Inhalt der Seite platziert. Mit *top* und *left* kann die linke obere Ecke des Elements positioniert werden.

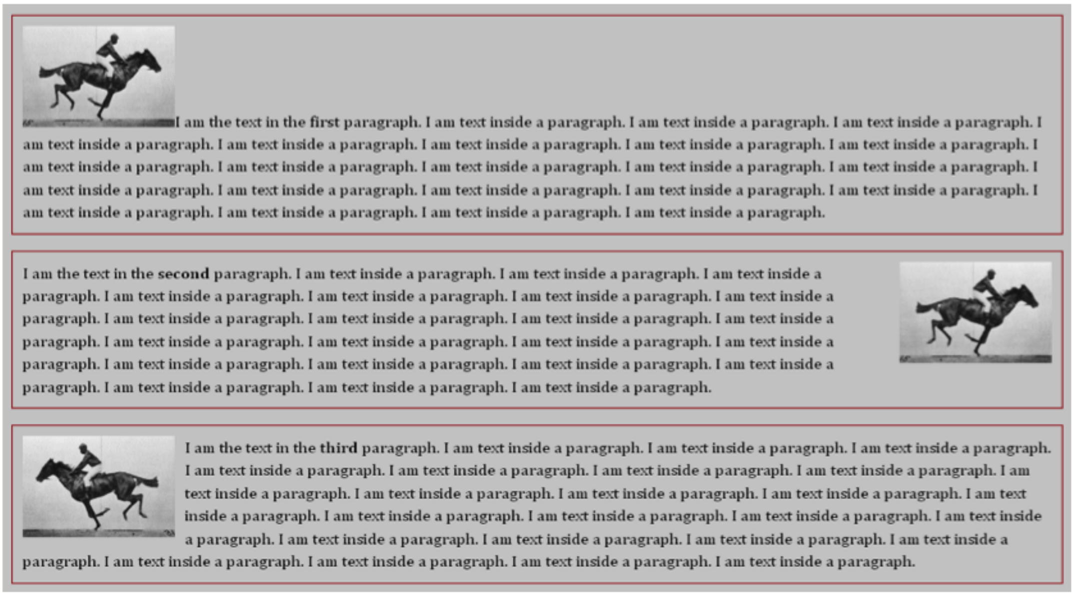


Abbildung 27.5: Abbildung: Bilder mit float

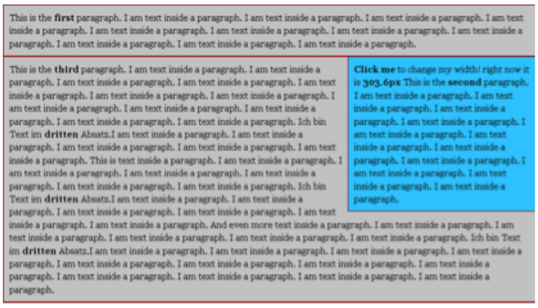


Abbildung 27.6: Abbildung: Ein Absatz mit *float*

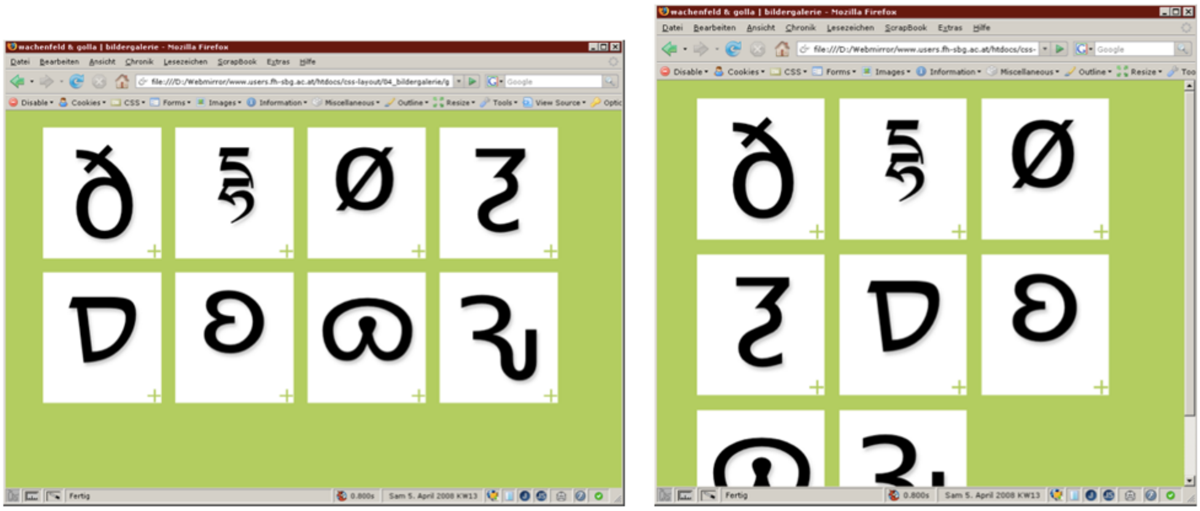


Abbildung 27.7: Abbildung: Zwei Darstellungen der Bildergalerie mit *float*

(oder mit *bottom* und *right* die rechte unter Ecke). In folgendem Beispiel wird ein absolut positiniertes Menü kombiniert mit einem Inhalt mit *margin-left* - sonst würde das Menü den Inhalt verdecken!

```
#navi {
  position: absolute;
  width: 80px;
  top:    0px;
  left:   0px;
}

#content {
  margin-left: 100px;
}
```

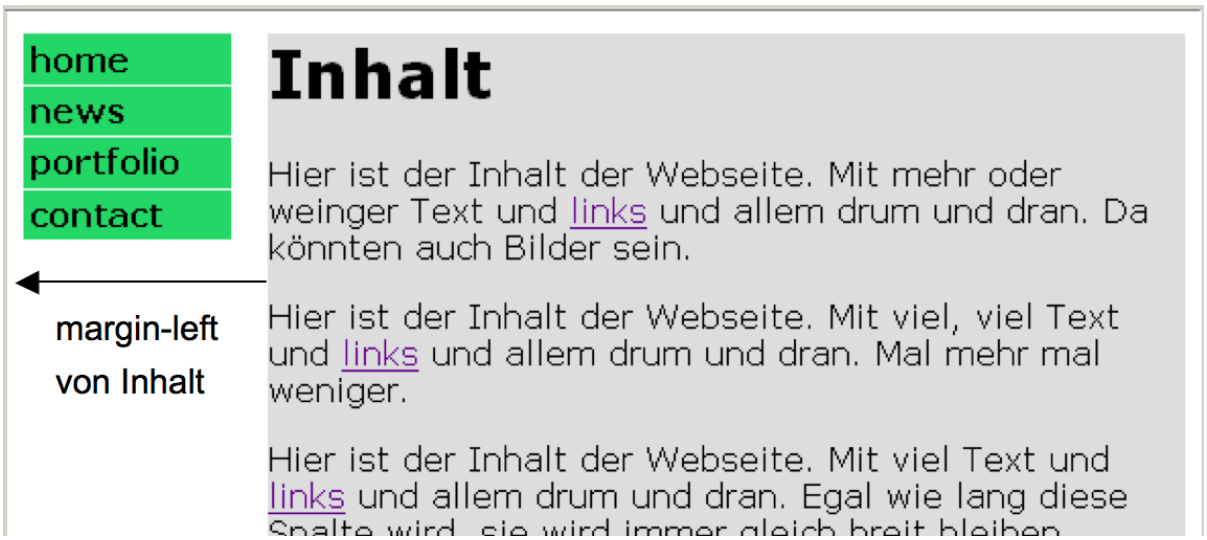


Abbildung 27.8: Abbildung: Kombination von absoluter Positionierung und *margin-left*

Koordinatensystem

Die Koordinaten (*top* und *left*) beziehen sich normalerweise auf die linke obere Ecke des Browserfenster.

Ein Element mit der Eigenschaft *position: absolute* oder *position: relative* bildet für seine untergeordneten Elemente ein neues Koordinationsystem. In folgendem Code definiert das *main div* ein neues Koordinatensystem, das *navi div* ist also in der linken oberen Ecke des *main div* und nicht in der linken oberen Ecke des Browser-Fensters positioniert.

```
<div id="main">
  <div id="content">
    <h1>Inhalt</h1>
    <p>Hier der Inhalt.</p>
    <p>Hier der Inhalt.</p>
    <p>Hier der Inhalt.</p>
    <p>Hier der Inhalt.</p>
  </div>
  <div id="navi">
    <p>a</p>
    <p>b</p>
    <p>c</p>
    <p>d</p>
    <p>e</p>
  </div>
</div>
```

```
#main {
  border: 1px red solid;
```

```
  position:relative;
  width: 700px;
  margin: 0px auto;
}
#content {
  margin-left: 120px;
  background-color: yellow;
}
#navi {
  position: absolute;
  width: 80px;
  top:0px;
  left:0px;
}
```

Viele Methoden

Wie Sie gesehen haben gibt es viele CSS-Properties die das Layout beeinflussen. Manche davon passen zusammen, andere widersprechen sich. So macht es keinen Sinn absolute Positionierung mit Float zu kombinieren.

Mit der Zeit werden Sie Erfahrungen sammeln und auch Ihren eigenen Stil entwickeln.

KAPITEL 28

Navigationsmenü

Wenn man mit CSS ein Navigationsmenü gestaltet, dann sollte man von einem HTML-Code ausgehen der auch ohne CSS gut benutzbar bleibt („graceful degradation“), und der nur mit CSS umgestaltet wird.

Der Ausgangspunkt für ein Navigationsmenü ist eine Liste mit Links:

```
<div class="navi">
  <h3 class="unsichtbar">Seitenauswahl</h3>
  <ul>
    <li><a href="index.html">home</a></li>
    <li><a href="ort.html">ort</a></li>
    <li><a href="dies.html">dies</a></li>
    <li><a href="das.html">das</a></li>
  </ul>
</div>
```

Ohne CSS wird die Liste ganz normal dargestellt.

Mit CSS kann man daraus ein vertikales Menü machen, dazu wird mit *list-style-type: none* der Listenpunkt *li* zu einem normalen Tag.

```
.unsichtbar { display: none; }

div.navi li {
   list-style-type: none;
   margin-bottom: 1px;
   background: #6C6;
   width: 6em;
   padding: 0.2em;
}

div.navi li a:link {
   text-decoration: none;
   font-weight: bold;
   color: black;
}
```

28.1 Horizontales Menü

Mit Hilfe von *float* kann man das Menü auch horizontal darstellen, dazu muß nur ein *float* eingefügt und der *margin* anders gesetzt werden:

```
div.navi li {
   float: left;
   list-style-type: none;
   margin-right: 1px;
   background: #6C6;
```

```
    width: 6em;
    padding: 0.2em;
}

div.navi li a:link {
    text-decoration: none;
    font-weight: bold;
    color: black;
}
```

28.2 Navigationsmenü überall

Das Navigationsmenü sollte natürlich in allen Seiten der Site gleich vorhanden sein. Dazu könnte man den Code in alle HTML-Dateien kopieren. Wenn man dann einen neuen Menüeintrag einfügen will, muss man wiederum alle Seiten editieren.

Deswegen bietet es sich an, bei dieser Gelegenheit von einfachem HTML auf PHP/Python umzusteigen oder Server Side Includes (SSI) zu verwenden

Das Navigationsmenü wird nun in einer separaten Datei gespeichert, z.B. *navi.php*. In den einzelnen Seiten wird das Menü dann mit *include* eingebunden:

```
<body>
<?php include "navi.php"; ?>
<h1>News</h1>
```

Server Side

Das Einfügen der inkludierten Datei erfolgt auf dem Server. Der Client erhält das vollständige HTML-Dokument inklusive Navigation, und merkt von der Inkludierung nichts.

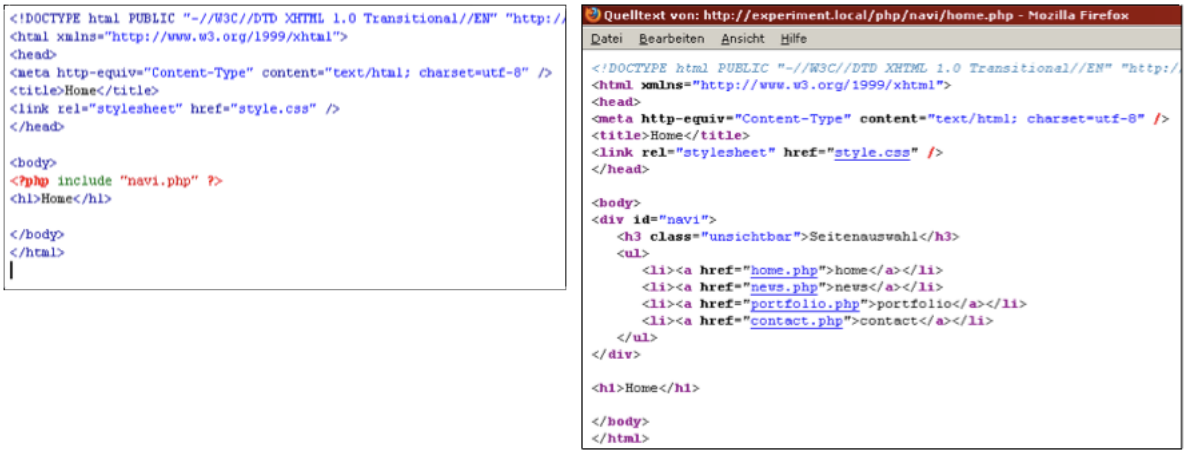


Abbildung 28.1: Abbildung: Original PHP-Datei am Server und Quelltext-Ansicht im Webbrowser

Ergebnis

Damit ist nun das Einfügen eines neuen Menüpunkts ins Navigationsmenü ganz einfach: nur die Datei *navi.php* muss editiert werden.

KAPITEL 29

CSS für Interaktion

Mit dem Selektor *:hover* wird eine CSS-Regel nur angewandt, wenn sich der Maus-Cursor über dem Element befindet. *:focus* hat eine ähnliche Bedeutung bei der Steuerung mit der Tastatur.

Achtung: auf Touch-Geräten wie Tables und Smartphones gibt es derzeit kein *:hover*! Deswegen sollte man *:hover* nur zur Dekoration, nicht aber für wichtige Funktionen verwenden!

```
div       { background-color: #ddd; }
div:hover { background-color: red; }
```

Hover mit Bild

Mit Hilfe von Hintergrundbildern kann man so zum Beispiel einen kitschigen 3d-Button darstellen, der beim Hovern „hineingedrückt“ wird.

```
div       { background-image: url(button-up.png);   }
div:hover { background-image: url(button-down.png); }
```

Beim Laden des zweiten Hintergrundbildes kann es dabei zu einer Verzögerung kommen. Um das zu verhindern kann man die beiden Hintergrundbilder in einer Bilddatei speichern und nur den Bildausschnitt wechseln (http://pmeerw.net/www-mm14/examples/logosplit.html):

Abbildung 29.1: Abbildung: Zweigeteiltes Bild für *hover*-Effekte ohne Verzögerung

```
#logo {
  background-image: url(logosplit.png);
  width: 135px;
  height: 34px;
  overflow: hidden;
}

#logo:hover {
  background-position: 0 -34px;
}
```

29.1 CSS Sprites

Eine extreme Anwendung dieses Prinzips nennt man CSS-Sprites: dabei werden möglichst viele Bilder in einer Bild-Datei zusammengefaßt. Falls viele kleine Icons verwendet werden, kann das die Darstellung der Webseite

erheblich beschleunigen.

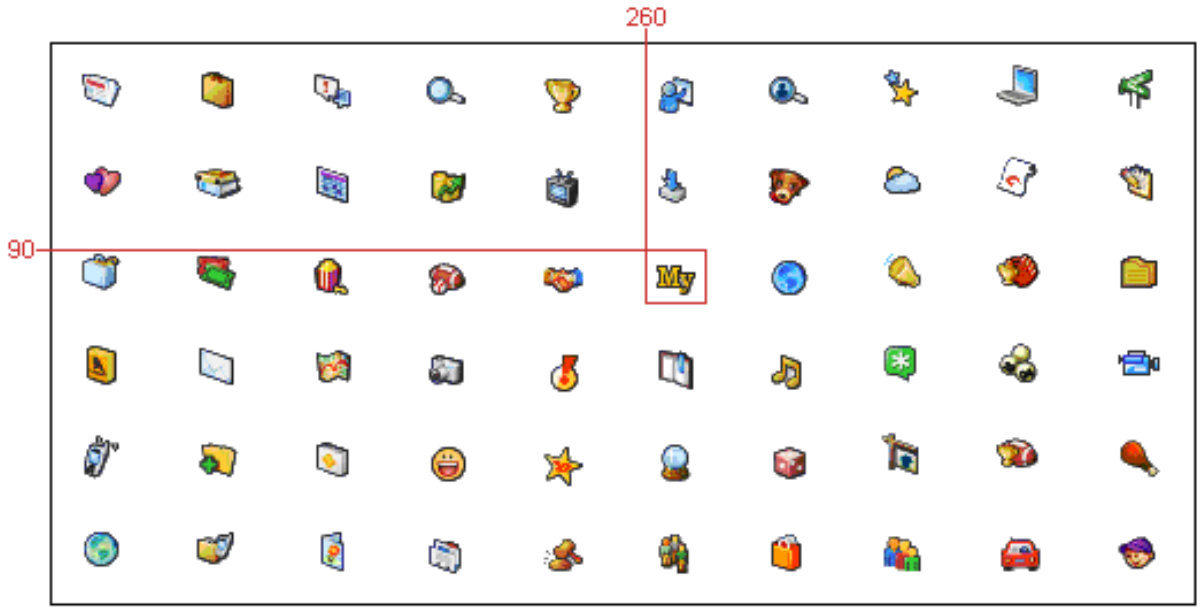


Abbildung 29.2: Abbildung: CSS Sprites von Yahoo

So könnte der CSS Code aussehen:

```
.icon {
  display:block;
  padding:8px 0 9px 40px;
  background:url(http://us.i1.yimg.com/pa-icons2.gif) 5px 3px no-repeat;
}

#messenger .icon {
  padding-left:31px;
  background-position:2px -497px;
}

#music .icon {
  background-position:5px -197px;
}
```

CSS Selektoren im Detail

Sie haben bereits einige CSS Selektoren kennen gelernt. In der CSS 2.1 Spezifikation werden u.a. folgende Schreibweisen für Selektoren beschrieben:

Name	**Muster**	**Beschreibung**
Universal selector	*	Stimmt mit jedem Element überein.
Type selectors	E	Stimmt mit jedem E-Element überein
Descendant selectors	E F	Stimmt mit jedem F-Element überein, das ein Nachfahre eines E-Elements ist.
Grouping	E,F,G	Stimmt mit jedem E-, sowie jedem F-, sowie jedem G-Element überein. E, F und G könnten auch komplexere Selektoren sein!
Child selectors	E > F	Stimmt mit allen F-Elementen überein, die Kindelemente eines Elements E sind.
Class selectors	E.c	Stimmt mit jedem E-Element überein, dessen *class* gleich „c“ ist.
Link pseudo-classes	:link	Stimmt mit Links überein, deren Ziel noch nicht besucht wurde (*:link*), oder deren Ziel bereits besucht wurde (*:visited*).
Dyn. pseudo-classes	:hover	Stimmt während bestimmter Interaktionen überein: Auswahl (*:focus*), Maus-Cursor (*:hover*), ...
id selectors	E#i	Stimmt mit jedem E-Element überein, dessen *id* gleich „i“ ist.
Adjacent selectors	E + F	Stimmt mit jedem F-Element überein, dem unmittelbar ein Element E vorausgeht.

Mit CSS3 kommen u.a. folgende Selektoren dazu:

Name	Muster	Beschreibung
Attribute selectors	E[attr^="str"]	Stimmt mit jedem Element überein, bei dem der Wert des Attributs *attr 'mit dem geforderten String 'str* beginnt.
UI pseudo-classes	:disabled	Diese Selektoren beziehen sich auf mögliche Zustände von Eingabefeldern wie *:disabled*, *:checked*.
Target pseudo-classes	:target	Stimmt mit dem Element überein, dessen *id* mit dem Anker in der URL übereinstimmt.
child pseudo-class	:first-child	Zählt die Geschwister des Elements, stimmt mit dem Element überein wenn die Zahl bestimmte Bedingungen erfüllt: erstes Geschwister, 13., ungerade Zahl, gerade Zahl, letztes.
Root pseudo-class	:root	Stimmt mit der Wurzel des DOM überein, immer <*html*>.
Empty pseudo-class	:empty	Element ist ganz leer – enthält weder Kinder-Knoten noch Text.
Pseudo-element	E::first-letter	Mit diesem Pseudo-Element kann man etwas aus HTML herausholen, was nicht explizit drinnen ist! Obwohl im Absatz der erste Buchstabe nicht extra in einen Tag eingeschlossen ist kann man ihn mit p::first-letter auswählen.

Child Selector

Wir kennen schon den Descendant Selector, der beliebige Nachfahren auswählt. Der Child Selector ist auf direkte Kinder beschränkt.

```
li > a { color: yellow; }
```

Pseudo Classes

Wir haben im Zusammenhang mit Links schon die Pseudo Classes *:link* und *:visited* kennen gelernt, und im Zusammenhang mit Interaktion *:hover* und *:focus*.

Es gibt noch eine Hand voll weitere Pseudo Classes:

- *:first-child* selektiert ein Element nur, wenn es das erste Kind seines Eltern-Nodes ist
- *:first-letter* selektiert nur den ersten Buchstaben! Das funktioniert auch, wenn der Buchstabe gar kein eigener Node in der DOM ist!
- *:first-word* selektiert das erste Wort. Das funktioniert auch, wenn das Wort gar kein eigener Node in der DOM ist!
- *::selection* der aktuell ausgewählte Bereich, siehe css-tricks (http://css-tricks.com/examples/DifferentSelectionColors/)
- *:nth-child(odd)* wählt jedes zweite, ungerade Element aus - 1,3,5,...
- *:nth-child(even)* wählt jedes zweite, gerade Element aus - 2,4,6,...

```
li:first-child { color: yellow; }
```

Adjacent selector

Dieser Selector wählt den direkten Geschwister-Node aus.

```
p + h1 { color: yellow; }
li + li { color: yellow; }
```

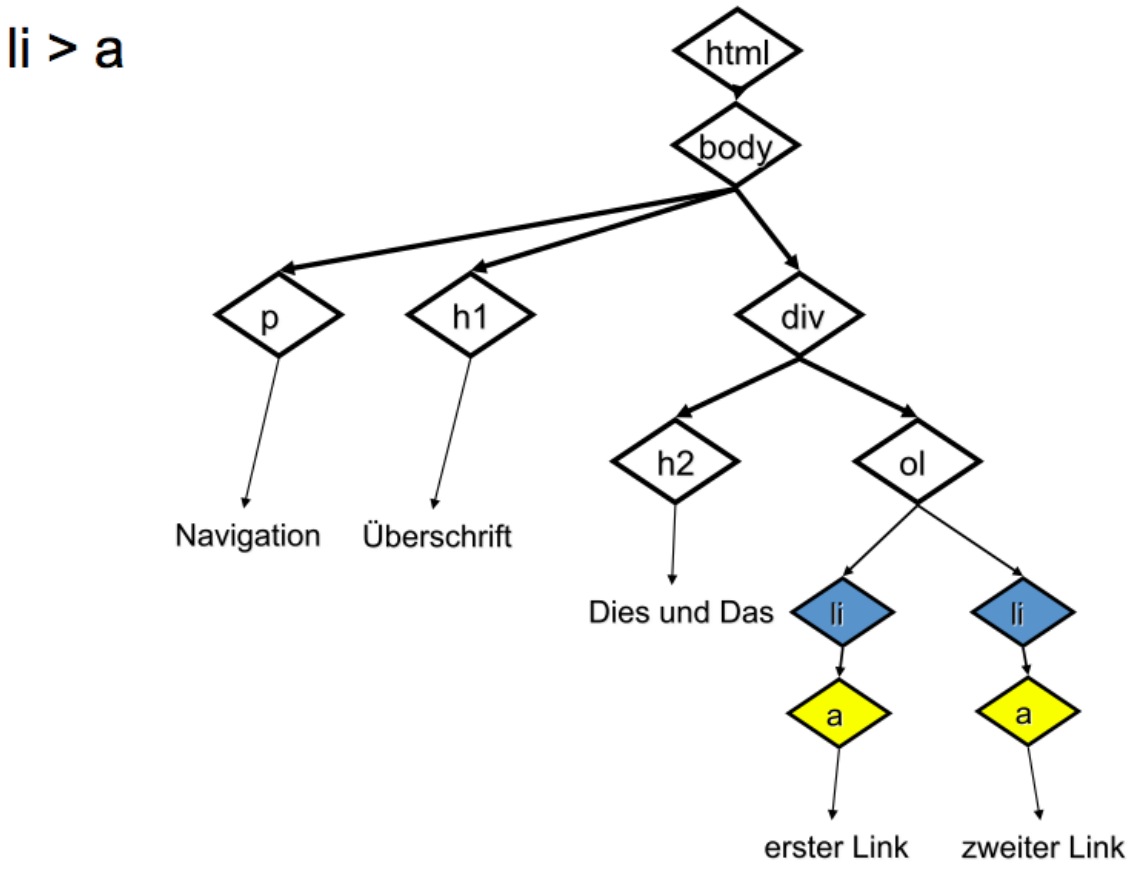


Abbildung 30.1: Abbildung: Document Object Model und Child Selektor

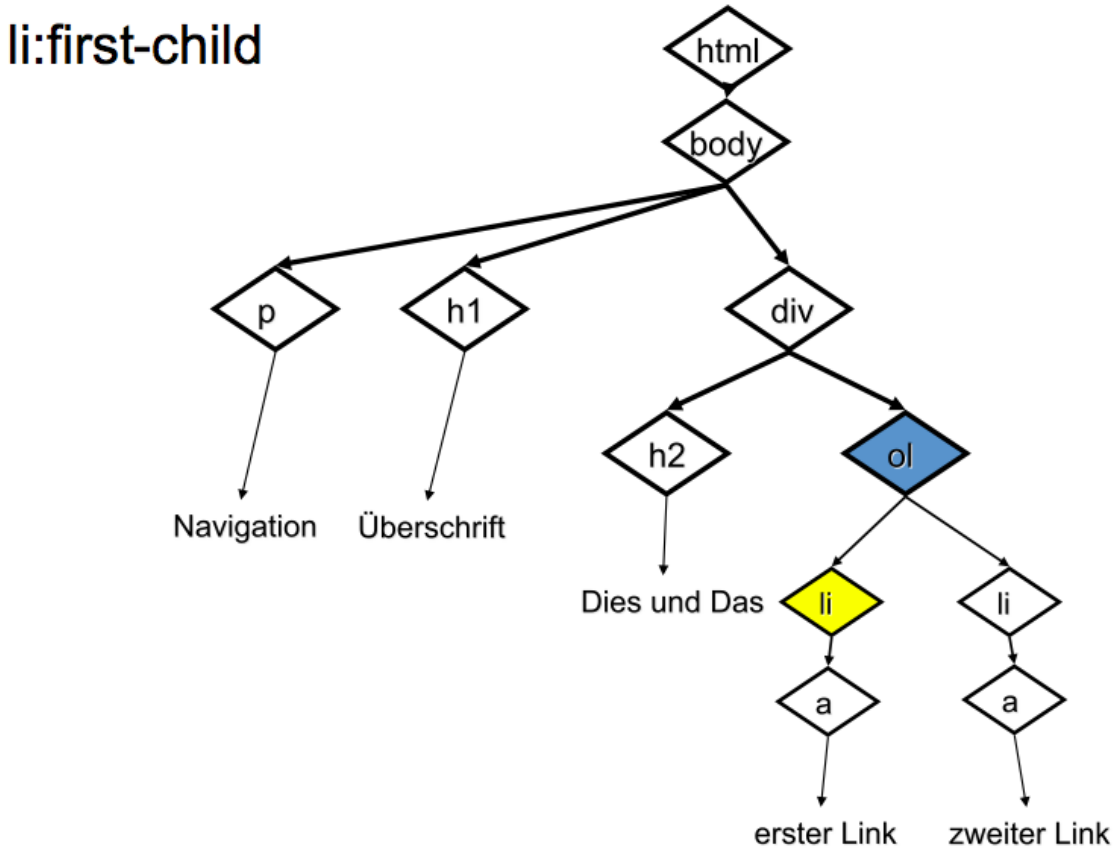


Abbildung 30.2: Abbildung: Document Object Model und First-child Selector

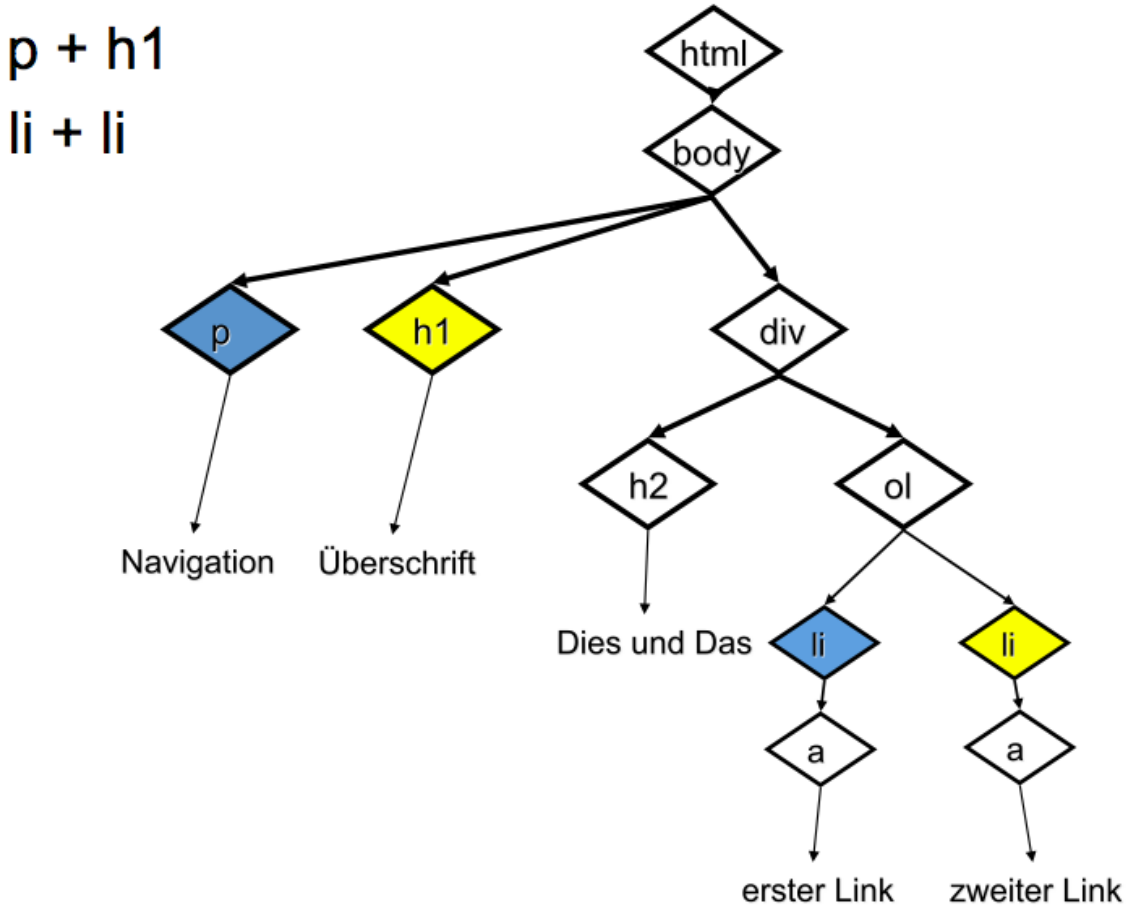


Abbildung 30.3: Abbildung: Document Object Model und Adjacent Selector

30.1 Kombination von Selektoren

Wenn man mehrere Selektoren kombiniert, hat das Komma die geringste Präzedenz:

```
p, div a { color: yellow; }
```

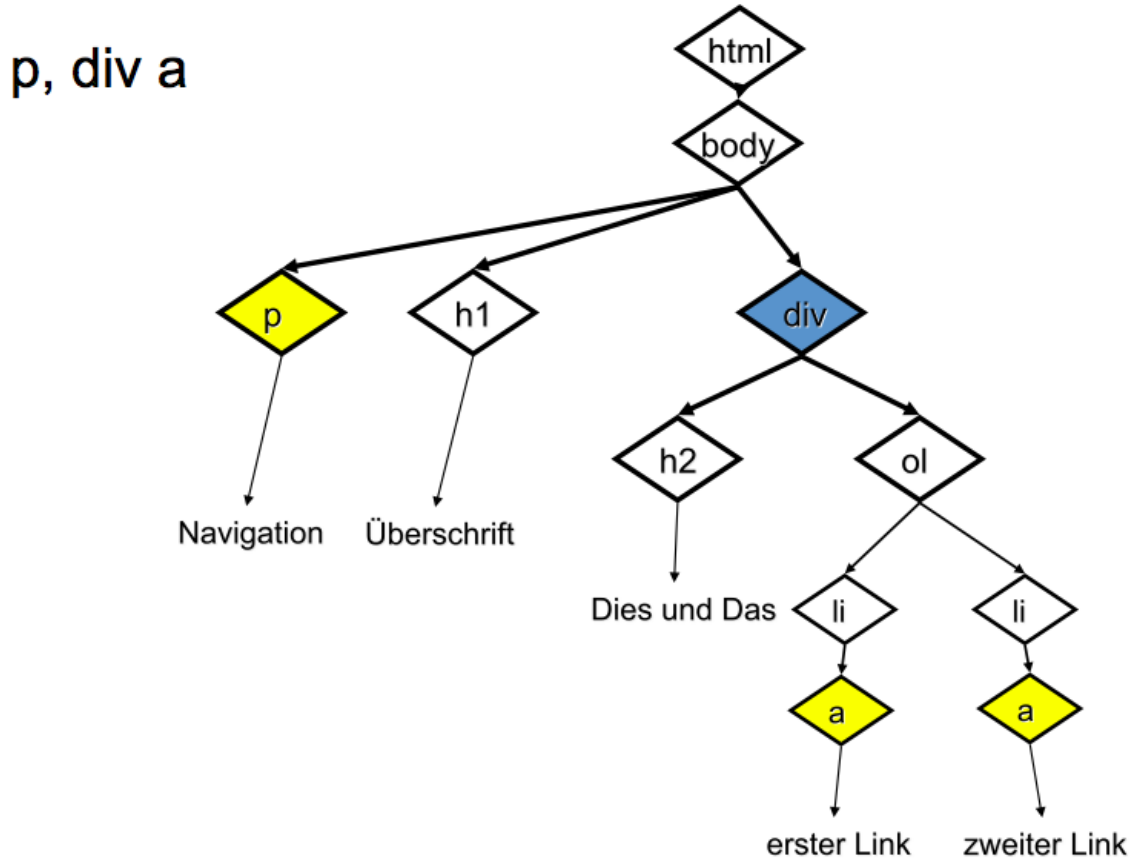


Abbildung 30.4: Abbildung: Selector Precedence im Document Object Model

KAPITEL 31

Login, Sessions

Sie lernen wie Sie ein Login für eine Web-Applikation programmieren können.

Mit Cookies wird HTTP stateful, damit werden Sessions und ein Login ermöglicht.

Was Sie alle wissen sollten

- Das HTTP ohne Cookies stateless ist. Wie Cookies funktionieren.
- Wie Login / Logout mit Hilfe einer Session realisiert werden kann

Was Sie können sollten

- Cookies setzen und auslesen
- Sessions benutzen

Cookies

Bis jetzt war jeder Zugriff auf die Webapplikation unabhängig von jedem anderen Zugriff: die Applikation weiß nicht, ob 10 verschiedene Leute die Homepage abrufen oder ob eine Person die Seite 10 mal lädt.

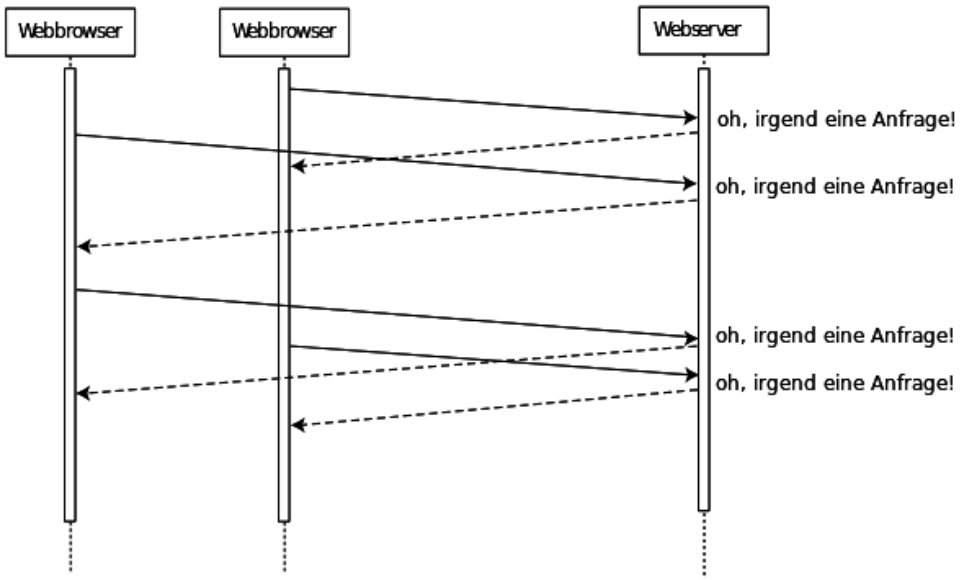


Abbildung 32.1: HTTP als “stateless protocol”

Das ist eine Grundeigenschaft von HTTP: es ist „stateless“ (zustandslos). Das Gegenteil davon wäre „statefull“ (zustandsbehaftet).

Also kann man mit HTTP alleine – wie wir es bisher kennen – kein „Login“ schaffen. Um zu wissen, dass der User Bob eingeloggt ist müsste er ja „wiedererkannt“ werden. Genau das macht den „state“ aus.

32.1 Cookies

Cookies werden im RFC 2965 beschrieben und bestehen ein oder mehreren key/value Paaren und optionalen Attributen (wie *Expires*, *Max-age*, *Path*, etc.).

Um Zustand zu ermöglichen wurde das HTTP-Protokoll um die sogenannten „Cookies“ erweitert: Ein Cookie besteht aus bis zu 4096 Bytes Daten, die der Webbrowser lokal speichert, und bei jedem Zugriff auf den Webserver wieder mitsendet.

Der Browser sendet nie ein Cookie an einen anderen Webserver als den von dem er es erhalten hat. Er kann aber viele verschiedene Cookies von verschiedenen Servern speichern (in einem Cookie-Jar).

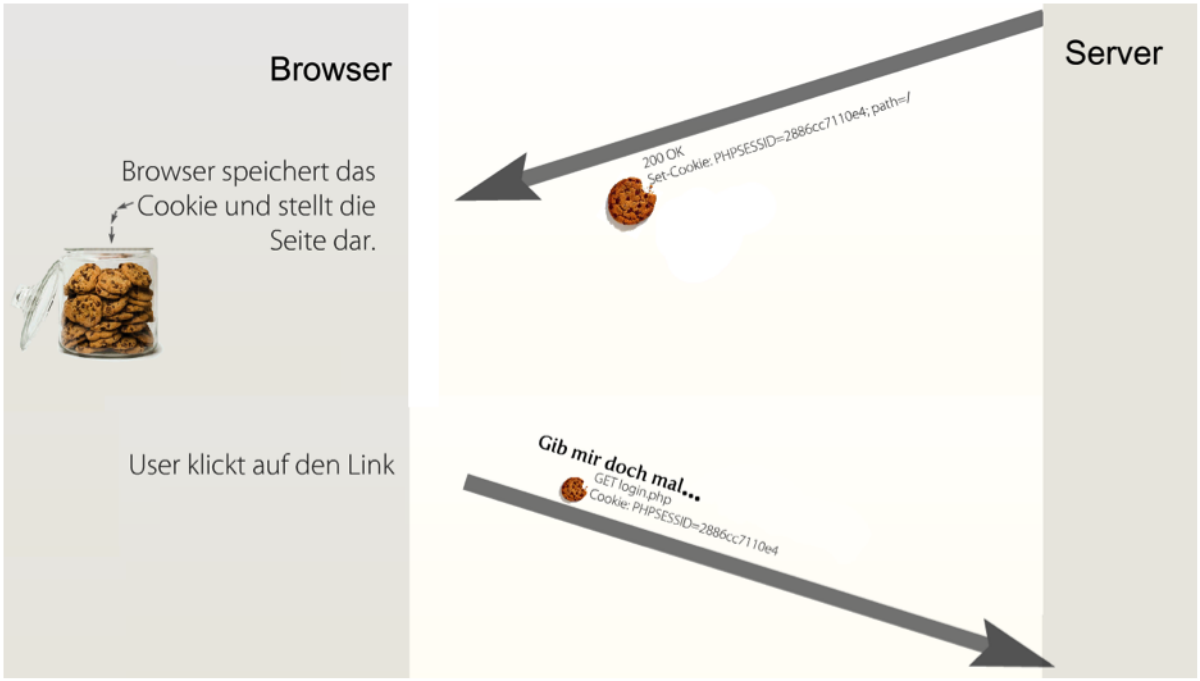


Abbildung 32.2: Cookie wird gesetzt und bei jedem weiteren Request gesendet

An Hand dieses Cookies kann eine Webapplikation einen bestimmten User wiedererkennen. Cookies können als Teil des HTTP-Protocols (nur) vom Server gesetzt werden. Dabei wird im HTTP Header der Name des Cookies angegeben, der Wert der gespeichert werden soll, und der Gültigkeitsbereich und Zeitraum:

```
Set-Cookie: style=gelb Set-Cookie: style=gelb; path=/admin/
Set-Cookie: style=gelb; expires=Tue, 29-Mar-2015 19:30:42 GMT; path=/admin/
```

Die einzige Art ein Cookie zu löschen ist ein Cookie mit gleichem Namen und Ablaufdatum in der Vergangenheit zu setzen:

```
Set-Cookie: style=wurscht; expires=Tue, 29-Mar-2005 19:30:42 GMT; path=/admin/
```

Cookies in Python

In Python bietet das Modul *Cookie* alles Notwendige an um mit Cookies zu arbeiten.

In Python finden Sie die bereits gesetzten Cookies, die vom Browser zurückgesendet werden in der Environment-Variable *HTTP_COOKIE*. Aus dieser kann ein Cookie-Objekt erzeugt werden und ab dann wie ein Dictionary verwendet werden, siehe unten:

```
import Cookie
cookie = Cookie.SmartCookie(os.environ['HTTP_COOKIE'])
style = cookie['style']
```

Cookies werden im HTTP-Header an den Client zurückgeschickt. Dazu wird das Cookie-Objekt einfach ausgegeben:

```
print 'Content-type: text/html'
print cookie
print # newline, separate HTTP header from HTTP body
print '<html>style: %s</html>' % (style)
```

Session und Login

HTTP ist “stateless” - jeder HTTP Request ist ein isoliertes Ereignis, der Server kann nicht erkennen ob Requests zusammen gehören.

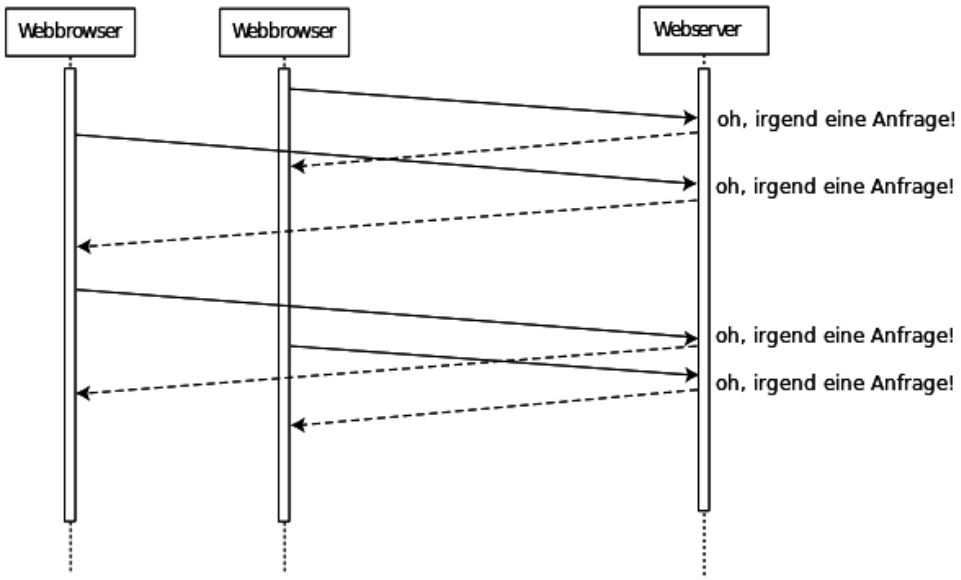


Abbildung 33.1: HTTP als “stateless protocol”

Mit der Einführung von Cookies, und damit von state, können wir nun erkennen, dass mehrere Requests zusammen gehören, vom selben Browser ausgelöst wurden.

Wir nennen diese Folge von Requests dann eine “Session”.

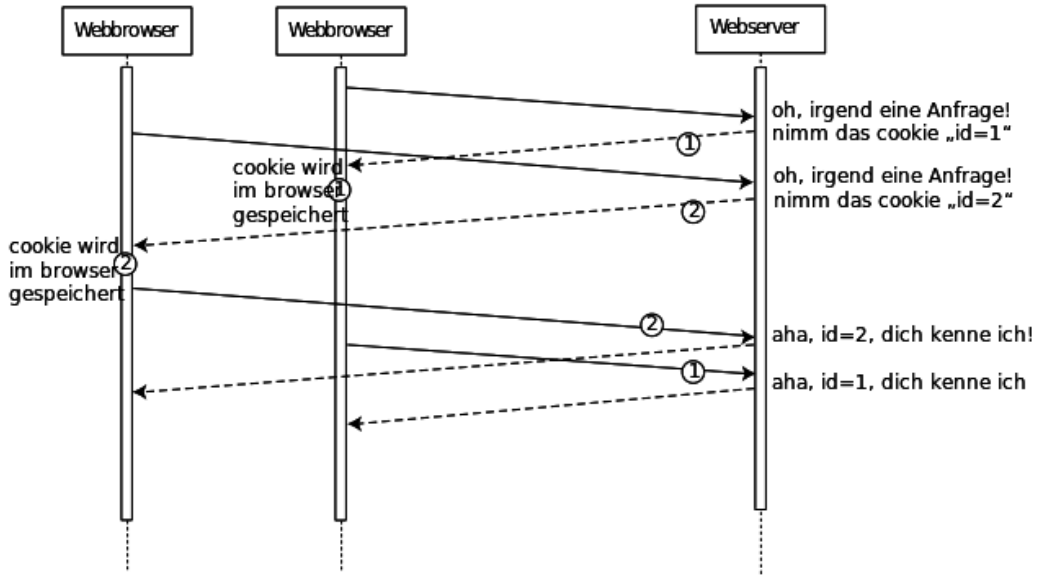


Abbildung 33.2: HTTP wird mit Cookies “stateful”

KAPITEL 34

Web Security

Sie erhalten einen Einblick in die Sicherheits-Probleme von Web-Applikationen.

Die Sicherheit von Web Applikationen ist ein komplexes Thema. Die OWASP gibt dazu regelmäßig Empfehlungen heraus. Die OWASP Top 10 von 2010 (https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project) sind die Grundlage für dieses Kapitel. Wir werden es speziell auf Python, MySQL, Apache anwenden.

Zehn Problemfelder die sie kennen sollten:

- Cross-Site Scripting (XSS)
- Fehler in Authentifizierung und Session Management
- Unsichere direkte Objekt-Referenzen
- Cross-Site Request Forgery (CSRF)
- Sicherheits-relevante Fehlkonfiguration
- Kryptografisch unsichere Speicherung
- Mangelhafter URL-Zugriffsschutz
- Unzureichende Absicherung der Transportschicht
- Ungeprüfte Um- und Weiterleitungen

Was Sie können sollten

- Sie können XSS Probleme erkennen und durch geeignetes Escapen oder Säubern des Outputs vermeiden
- Sie können eine Authentisierung mit Passwort und Cookie in Python implementieren, und dabei Session Fixation vermeiden
- Sie können CSRF Probleme erkennen und durch Verwendung eines CSRF Tokens vermeiden
- Sie können die Apache-Konfiguration überprüfen und Sicherheitsprobleme erkennen
- Sie können Sicherheitsprobleme beim Speichern von Passwörter und sensitiven Daten erkennen und vermeiden
- Sie können erkennen, wenn eine Applikation den Zugriffsschutz nicht konsequent für jede URL überprüft, und das Problem beheben
- Sie können erkennen wann SSL verwendet werden sollte
- Sie können Sicherheitsprobleme bei der Weiterleitung an externe Seiten erkennen und vermeiden

KAPITEL 35

Cross Site Scripting (XSS)

Die OWASP beschreibt dieses Problem allgemein so:

> **Achtung:** XSS-Schwachstellen treten dann auf, wenn die Anwendung von BenutzerIn X eingegebene Daten übernimmt und an den Browser anderer BenutzerInnen zurücksendet, ohne sie hinreichend zu validieren und zu escapen.

Wenn meine Seite eine XSS-Schwachstelle enthält könnte folgendes geschehen:

Cookies (und damit auch Sessions) von BenutzerInnen meiner Seite können gestohlen werden, die Darstellung der Webseite kann verfälscht werden, eine automatische Weiterleitung auf andere Seiten, z.B. Malware-Seiten kann eingebaut werden.

Der schlimmste Fall wäre die Installation eines sogenannten XSS-Proxies, der es der AngreiferIn ermöglicht die Browser von BesucherInnen fernzusteuern.

Was bedeutet Cross Site?

Diese Attacke wird "über die Bande gespielt":

- Hilda Harmlos stellt eine Webseite mit XSS-Schwachstelle aufs Netz, zum Beispiel mit einem Forum.
- Alyssa P. Hacker erstellt einen Forum-Eintrag der die XSS-Schwachstelle ausnutzt.
- Peter Publikum will das Forum lesen, und wird dabei attackiert.

Es ist also nicht die Site der Hackerin, die hier gefählich ist, sondern eine andere, scheinbar harmlose Seite.

35.1 Vermeidung von XSS

Cross Site Scripting kann komplett vermieden werden, wenn man niemals Input von BenutzerInnen auf der Webseite wiedergibt. Das ist eher unrealistisch in der Praxis...

XSS kann man mit zwei Verteidigungs-Linien vermieden werden:

- Schon bei der Eingabe (mit einer white-list) alles Entfernen was nicht harmlos ist.
- Bei der Ausgabe immer ein geeignetes Escaping verwenden

Mehr

Siehe XSS (Cross Site Scripting) Prevention Cheat Sheet (https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Pr

KAPITEL 36

Authentifizierung und Session-Management

Rund um Authentifizierung und Session-Management treten viele Probleme auf.

Keine Information in Fehlermeldungen preisgeben

Authentisierung kann aus vielen Gründen fehlschlagen:

- der Username existiert gar nicht
- dieser Account ist gesperrt
- das Passwort ist inkorrekt

Egal was die Gründe sind: die Rückmeldung an die UserIn muss immer genau gleich ausfallen:

> "Login fehlgeschlagen - Falscher Username oder Falsches Passwort."

Warum? Die Information, ob ein bestimmter Username im System exisitiert ist wertvoll! Es ist ja bereits die Hälfte der nötigen Information für ein Login. Deswegen soll man die Existenz von Usernamen genau so geheim halten wie die Passwörter.

Session ID wie Passwort schützen

Das Protokoll HTTP ist stateless. Wenn man trotzdem UserInnen authentifizieren will, dann muss bei jedem Request eine authentifizierende Information, zum Beispiel eine Session-ID, mitgeschickt werden. Dies geschieht zum Beispiel in einem Cookie.

Wenn es gelingt diese Information abzuhören und wieder zu verwenden, dann erhält man den gleichen Zugang wie die eigentliche UserIn. Diese Art der Angriffs nennt man "Replay Attacke" - ein Request wird abgehört und gleich wieder verwendet.

Um Passwörter ebenso wie Session IDs und Cookies vor dem Abhören zu schützen muss man den gesamten HTTP-Request verschlüssen - das ist über SSL/TLS möglich. Ein weiterer Punkt wo diese Informationen eventuell aufscheinen sind Logfiles. Auch beim Logging sollte man diese Informationen vorher ausfiltern oder verschlüsseln.

Nicht selbst implementieren

Es ist sehr schwierig, ein sicheres Authentifizierungs- und Session-Management zu implementieren. Man sollte nicht auf eigene Lösungen setzen - Diese haben dann oft Fehler bei Abmeldung und Passwortmanagement, bei der Wiedererkennung der BenutzerInnen, bei Timeouts, Sicherheitsabfragen usw. Das Auffinden dieser Fehler kann sehr schwierig sein, besonders wenn es sich um individuelle Implementierungen handelt.

Neue Session bei Login / Logout / neuen Rechten

Die "Session Fixation" Attacke funktioniert mit der Session-ID. Ein Beispiel:

Alyssa Hacker sendet einen Link an Peter Publikum. Dieser Link führt zur Bank von Peter Publikum, und gibt schon eine Session-ID vor. Wenn sich Peter Publikum nun bei seinem Online-Banking einloggt, die Session-ID aber gleich bleibt, dann kann Alyssa Hacker mit der gleichen Session die Online-Banking Seite aufrufen, und ist schon eingeloggt – als Peter Publikum.

Um diese Art der Attacke zu vermeiden, muss man beim Login und Logout jeweils eine neue Session starten und die ‘vorgegebene’ Session-ID verwerfen.

Mehr

Die OWAAS bietet noch mehr Informationen zu diesem Thema an im Authentication Cheat Sheet (https://www.owasp.org/index.php/Session_Management_Cheat_Sheet) und im Session Management Cheat Sheet (https://www.owasp.org/index.php/Session_Management_Cheat_Sheet).

Unsichere direkte Objektreferenzen

Die OWASP beschreibt dieses Problem ungefähr so:

> **Achtung:** Webanwendungen nutzen oft den internen Namen oder die Kennung eines Objektes, um auf dieses zu verweisen. BenutzerInnen können Parameter ändern, um diese Schwachstellen zu entdecken und auszunützen.

Beispiele für unsichere direkte Referenzen:

Ich benutze ein Online-Banking System. Die URL meines Kontos ist:

https://www.onlinebank.com/user?acct=6065

Ich verändere die URL, und probiere aus ob ich so Zugang zu weiteren Konten erhalte:

https://www.onlinebank.com/user?acct=6066

Ich benutze ein Galerie um Fotos zu betrachten. Die URL für ein bestimmtes Bild ist:

https://www.photos.com/show?img=100-0011_IMG.jpg&text=100-0011_IMG.txt

Ich verändere die URL, und versuche so die Anzeige von interessanten Dateien im System herbeizuführen:

https://www.photos.com/show?img=100-0011_IMG.jpg&text=/etc/passwd

Vermeidung von unsichere direkten Referenzen

- Nicht die Keys/IDs aus der Datenbank preisgeben, sondern durch "slugs" ersetzen.
- Nicht die Dateinamen in der URL preisgeben, sondern einen Code. Das serverseitige Programm interpretiert diese Codes und generiert selbst die echten Dateinamen.

Und in jedem Fall:

- Am Server prüfen ob genau diese UserIn Zugriff auf genau diese Ressource hat.

Slugs statt Keys

Slugs sind lesbare Texte, die einen Datensatz eindeutig identifizieren. Sie werden in der URL statt Datenbank-Keys/IDs verwendet. Sie sind auch unter dem Namen "friendly URLs" und (z.B. in Wordpress) unter "permalinks" bekannt.

Die Verwendung von Slugs hat mehrere Vorteile:

1. Das Erraten eines weiteren Keys ist nicht so leicht wie bei aufeinanderfolgende Zahlen
2. URLs werden dadurch kürzer, leichter lesbar und sind leichter zu merken
3. Auch Suchmaschinen lesen den Text des Slugs, die Seite kann auch unter den Stichwörtern des Slugs gefunden werden

Mit der folgenden Konfigurations-Datei *.htaccess* wir der Apache-Webserver angewiesen beim Aufruf der URL */item/text-der-slug* in Wirklichkeit das CGI-Skript *view_item.cgi* mit dem Parameter *slug=text-der-slug* aufzurufen:

```
RewriteEngine on
RewriteRule ^item/([-a-z]+) view_item.cgi?slug=$1
```

Kein Zugriff auf beliebige Dateien

Die Erwähnung von Dateinamen als Parameter in der URL ist immer eine schlechte Idee. Betrachten wir das schlechte Beispiel von oben noch einmal:

> https://www.photos.com/show?img=100-0011_IMG.jpg&text=100-0011_IMG.txt

Eine denkbar schlechte Implementiereung dieser Galerie würde die angegebene Datei über den Dateinamen, hier 100-0011_IMG.txt, einfach öffnen – aber was wenn ein völlig anderer Dateiname vom Browser geschickt wird? z.b. /etc/passwd?

Erfolgt keine Prüfung, kann man durch einfaches Ändern der URL beliebige Dateien am Server "erforschen".

Gegen diese Art von Attacke kann man auf mehreren Linien verteidigen:

- Im Betriebssystem: das CGI-Skript läuft unter dem User-Account des Webservers. Durch geeigenetes Setzen der Zugriffsrechte im Filesystem kann diesem User der Zugriff zu wichtigen Bereichen untersagt werden
- In der Applikation wird geprüft ob der Dateiname gefährliche Zeichen wie /, .., etc. enthält. Es ist aber je nach Verwendung des Dateinamens schierig alle problematischen Fälle zu erfassen.
- In der Applikation selbst: mit einer Whitelist wird nur der Zugriff auf ausgewählte Dateien erlaubt.

KAPITEL 38

Cross-Site Request Forgery (CSRF)

CSRF ist ein Angriff "über die Bande": Um Site C zu attackieren, wird auf Site B Code hinterlegt. Wenn Userin Alice mit Site B interagiert wird ohne ihr Wissen eine HTTP-Request an Site C geschickt. Unter der Annahme dass Alice bei C schon eingeloggt ist wird dieser Request "erfolgreich" durchgeführt.

Das Problem tritt also potentiell bei allen Sites auf, bei denen man auf Dauer eingeloggt bleibt.

Ein hypothetisches Beispiel: Die verwundbare Web-Applikation ("Site C") sei eine Spenden-Plattform. So könnte das Formular zum Spenden aussehen:

```
<form action="spende.php" method="...">
  <input name="betrag">
  <select name="kontonummer">
    <option value="23456789">Greenpeace</option>
    <option value="34567890">Amnesty International</option>
  </select>
  <input type="submit">
<form>
```

Je nachdem ob *GET* oder *POST* verwendet wird, sieht der Code der Attacke verschieden aus.

```
<img src="http://spendenportal.ch/spende.php?betrag=1000&kontonummer=6666666" alt="i can has chee
```

```
<form action="http://spendenportal.ch/spende.php" method="POST">
  <input name="betrag"      value="1000" type="hidden">
  <input name="kontonummer" value="6666" type="hidden">
  <input type="submit" value="Like this on Facebook">
</form>
```

Verteidigung gegen CSRF

Auf der eigenen Site sollen die UserInnen auf Dauer eingeloggt bleiben können, aber gleichzeitig sicher sein, dass nicht in ihrem Namen unerwartete Aktionen vorgenommen werden. Wie kann ich das sicher stellen?

Die OWASP schlägt als Verteidigungsstrategie einen "Synchronizer Token" vor: Im Source-Code aller Web-Formulare wird ein Token eingefügt, das zu einer konkreten Session gehört und nur eine beschränkte Zeit lang gültig ist.

Bei der Behandlung jeder Anfrage wird überprüft ob dieses Token vorhanden ist, zur Session passt und noch gültig ist. Ein Request der auf Grund einer CSRF-Attacke gesendet wurde, hat dieses Token nicht und wird daher nicht bearbeitet.

Wenn ein Request verarbeitet wird, wird überprüft ob das Token vorhanden ist und noch gültig ist.

Sicherheitsrelevante Konfiguration

Die OWASP beschreibt dieses Problem allgemein so:

> **Achtung:** Sicherheitsrelevante Fehlkonfiguration kann auf jeder Ebene der Anwendung, inkl. Plattform, Web- und Anwendungs-server, Frameworks oder Programmcode vorkommen. Die Zusammenarbeit zwischen Entwicklern und Administratoren ist wichtig, um eine sichere Konfiguration aller Ebenen zu gewährleisten.

In größeren Projekten / Firmen ist eine Arbeitsteilung üblich zwischen Entwicklung (Development) und Systemadmistration (Operations).

Sicherheit auf vielen Ebenen

Für eine Web-Applikation muss man dabei mindestens folgende Schichte beachten:

- Physikalische Sicherheit (Wer kann den Server ein- und ausschalten, zerlegen,...)
- Virtualisierungs-Schicht
- Betriebssystem, z.B. CentOS, Debian, Ubuntu
- Datenbank, z.B. MySQL, MongoDB, Postgres
- Interpreter, z.B. Python, PHP, Ruby, Shell/Bash
- Webserver, z.B. Apache
- Framework, z.B. Django, ZEND Framework, Rails
- Fremd-Applikation, z.B. Wordpress, Redmine
- Selbstgeschriebene Applikationen

Jeder dieser Schichten gilt es richtig zu konfigurieren und Sicherheits-Updates einzuspielen.

Wenn es eine Arbeitsteilung zwischen Development und Operations gibt ist zu klären wer für welche Schicht zuständig ist.

Konfiguration + Hardening

Zwei Szenarien:

- Entwicklungs-Rechner: möglichst viele Debug-Möglichkeiten, Bequemlichkeit wichtiger als Sicherheit
- Produktions-Server: Sicherheit wichtiger als Bequemlichkeit, Logging / Monitoring ja, aber nicht öffentlich zugänglich

Dafür gibt es oft schon fertige Konfigurationen, oder Tutorials

- Suhosin Hardened PHP (http://www.hardened-php.net/)
- MySQL Hardening (https://www.owasp.org/index.php/OWASP_Backend_Security_Project_MySQL_Hardening)

Sicherheitsupdates

Keine Software ist sicher, in jeder Software werden Sicherheitsprobleme entdeckt. Die relevante Fragen sind: werden Sicherheitsprobleme die bekannt werden möglichst schnell behoben? Und in Folge: Wenn ein Update zur Verfügung steht, wird es möglichst schnell installiert?

- Heise Security (http://www.heise.de/security/)

Meine Verantwortung

An dieser Stelle sollten Sie sich fragen: bei den Webprojekten, an denen Sie beteiligt sind ... wer ist für welchen Teil der Konfiguration / Updates zuständig? Wo liegt Ihre persönliche Verantwortung?

Kryptografisch unsichere Speicherung

Die OWASP beschreibt dieses Problem allgemein so:

> **Achtung:** Fehlende Verschlüsselung vertraulicher Daten ist die häufigste Schwachstelle, gefolgt von unsicherer Schlüsselerzeugung, der Speicherung statischer Schlüssel und die Nutzung schwacher Algorithmen. Schwache Hashwerte ohne Salt kommen zum Passwortschutz oft vor. Ein eingeschränkter Zugriff lässt externe Angreifer solche Probleme i.d.R. nicht leicht entdecken. Den nötigen Zugriff müssen sie vorher auf andere Weise erlangen.

Passwörter vermeiden

Es gibt mehrere Alternativen zum Speichern von Passwörtern.

Mit OAuth oder OpenID kann man die Authentisierung einem anderen Anbieter überlassen. Für die BenutzerInnen meiner Site heisst es dann nicht “erfinde ein Passwort”, sondern “Login with Facebook”.

Beispiele:

- Facebook Authentication (https://developers.facebook.com/docs/authentication/)
- Twitter Authentication (https://dev.twitter.com/docs/auth/oauth/single-user-with-examples)
- Google Authentication (https://developers.google.com/accounts/docs/OAuth2)

Alternativen die nicht mit einem Social Network verknüpft sind, sind OpenID und Persona (früher: BrowserId) von Mozilla. Dabei dient eine URL bzw. eine E-Mail Adresse zur Identifikation.

- OpenID (http://openid.net/get-an-openid/)
- Persona (https://login.persona.org/)

Passwörter speichern

Wenn man BenutzerInnen auffordert Usernamen + Passwort zu erfinden, muss man damit rechnen dass Passwörter wiederverwendet werden. Es geht also bei der Sicherheit von Passwörtern nicht nur um meine eigene Web-Applikation. Die Passwörter die ich sicher halte oder verliere gelten wahrscheinlich auch für andere, wichtigere Services!

Level 0 (unverschlüsseltes Speichern): Diese Variante hat den scheinbaren “Vorteil”, dass man den BenutzerInnen “vergessene Passwörter” wiedergeben kann. Diese Variante ist auf jeden Fall zu vermeiden!

Level 1 (Passwörter hashen und speichern): Diese Variante ist marginal besser. Hier ist ein Angriff mit “Rainbow Tables” möglich: Lange Listen von gängigen Passwörtern und den dazugehörigen Hash-Werten.

Level 2 (gesalzene Passwörter, stärkere Hashes): Das ist der aktuelle Stand der Technik.

Mehr dazu

Bachfeld (2011): Cracker-Bremse - Passwörter unknackbar speichern. In c’t 13/2011 http://www.heise.de/security/artikel/Passwoerter-unknackbar-speichern-1253931.html?view=print

Mangelhafter URL-Zugriffsschutz

Die OWASP beschreibt dieses Problem allgemein so:

> **Achtung:** In Anwendungen wird der Zugriff auf Seiten nicht immer verlässlich abgesichert. Manchmal wird Zugriffsschutz durch Konfiguration realisiert, die ggf. auch fehlerhaft sein kann. Manchmal vergessen die Entwickler auch nur, die notwendige Prüfung zu implementieren.

Nur was serverseitig geprüft wird ist sicher

Bei der Programmierung von Web-Applikationen muss man sich immer bewusst sein, dass alles was im Client passiert, bzw. vom Client geschickt wird, manipuliert werden kann. Meine serverseitigen Programme müssen jeden Input den sie bekommen selbst prüfen, und können sich nicht darauf verlassen dass so eine Prüfung bereits am Client passiert ist.

Gängige Fehleinschätzungen dieser Art sind:

- Wenn ich keinen Link zu dieser Seite hin setze, dann findet die Seite eh niemand.
- Wenn ich die Daten in ein hidden-field im Formular schreibe, können sie nicht verändert werden.
- Wenn die Daten des Formulars per POST übertragen werden, können sie nicht manipuliert werden.

Mehr

Die OWASP bietet noch vertiefende Informationen zu diesem Thema an:

- OWASP Development Guide: Chapter on Authorization (http://www.owasp.org/index.php/Guide_to_Authorization)
- OWASP Testing Guide: Testing for Path Traversal (http://www.owasp.org/index.php/Testing_for_Path_Traversal)

Unzureichende Absicherung der Transportschicht

Die OWASP beschreibt dieses Problem allgemein so:

> **Achtung:** Häufig wird der Netzwerkverkehr einer Anwendung nicht ausreichend geschützt, z.B. indem nur die Authentifizierung, nicht aber der Rest SSL/TLS nutzt. Hierdurch werden Daten und die Session ID gefährdet. Eventuell werden auch abgelaufene oder falsch konfigurierte Zertifikate verwendet. Durch die Beobachtung des Netzwerkverkehrs können einfache Schwachstellen gefunden werden.

KAPITEL 43

Webapplikationen mit MySQL

Sie lernen von Python aus auf MySQL zuzugreifen.

Was Sie wissen sollten

- Wie Sie von Python aus auf Ihre MySQL-Datenbank zugreifen können: Aufbau der Verbindung, Cursor, SQL-Statements ausführen und Rows lesen
- Dass das Filtern der Daten in der Datenbank passieren sollte und möglichst wenige Daten übertragen werden sollten.
- Wie ein Datensatz aus der Datenbank als Objekt in Python dargestellt wird.
- Wie ein Tabelle von Datensätzen aus der DB als Array von Objekten in Python dargestellt wird.

Was Sie können sollten

- In einer Webapplikation mit Datenbank kleine Veränderungen und Verbesserungen vornehmen.
- Daten aus der Datenbank lesen und mittels Python darstellen.

KAPITEL 44

Datenbank

MySQL (http://www.mysql.com/) ist die relationale Datenbank, die bei gemietetem Webspace am öftesten angeboten wird. MySQL ist Open Source.

MariaDB (https://mariadb.org/) ist ein “Fork” von MySQL, die Entwicklung hat sich im Jahr 2009 verzweigt.

Abbildung 44.1: Abbildung: MySQL und MariaDB Logos

Hier wird nicht die Funktionsweise einer relationalen Datenbank erklärt , sondern nur die Besonderheiten von MySQL/MariaDB und die für Web-Applikationen wichtigen Aspekte.

KAPITEL 45

MySQL Installation

Alle Code-Beispiel funktionieren für MySQL und MariaDB genau gleich.

MySQL-Shell

```
mysql -p -u USERNAME -D DATENBANKNAME
```

Die Option *-p* sorgt dafür, dass ich nach einem Passwort gefragt werde (die alternative wäre das Passwort als Argument anzugeben). Mit der Option *-u* gebe ich den Usernamen mit an. Zuletzt (und ohne Option) wird der Name der Datenbank als Argument übergeben.

SQL Importieren

Auf der Kommandozeile kann man auch ganze Dateien mit SQL-Befehlen auf einen Schlag einspielen, dabei verwenden wir die Eingabeumlenkung der Shell:

```
mysql -p -u USERNAME -D DATENBANKNAME < DATEINAME
```

Tabellen anzeigen

```
show tables
```

Tabellenstruktur beschreiben

```
explain tablename
```

zeigt den Aufbau einer bestimmten Tabelle:

```
mysql> explain users;
+-----------------+--------------+------+-----+---------+----------------+
| Field           | Type         | Null | Key | Default | Extra          |
+-----------------+--------------+------+-----+---------+----------------+
| id              | int(11)      | NO   | PRI | NULL    | auto_increment |
| firstname       | varchar(255) | YES  |     | NULL    |                |
| surname         | varchar(255) | YES  |     | NULL    |                |
| title           | varchar(255) | YES  |     | NULL    |                |
| email           | varchar(255) | YES  | UNI | NULL    |                |
| isfemale        | tinyint(1)   | YES  |     | NULL    |                |
| profile_visible | tinyint(1)   | YES  |     | NULL    |                |
| type            | varchar(255) | YES  | MUL | NULL    |                |
| is_admin        | int(11)      | YES  |     | 0       |                |
| description     | text         | YES  |     | NULL    |                |
| slug            | varchar(255) | YES  | UNI | NULL    |                |
| avatar          | varchar(255) | YES  |     | NULL    |                |
+-----------------+--------------+------+-----+---------+----------------+
12 rows in set (0.00 sec)
```

Abfrage

select und *join* funktionieren wie erwartet:

```
mysql> select id,firstname from users limit 8;
+----+-----------+
| id | firstname |
```

```
+----+-----------+
|  2 | Lea       |
|  3 | Stefan    |
|  4 | Karin     |
|  5 | Katharina |
|  6 | Julia     |
|  7 | Gianni    |
|  8 | Josef     |
|  9 | Michael   |
+----+-----------+
8 rows in set (0.01 sec)
```

Dokumentation

Die Details zu SQL in MySQL (Abweichungen vom SQL Standard, Erweiterungen) kann man der Dokumentation (http://dev.mysql.com/doc/refman/5.6/en/index.html) entnehmen.

13.7.5.38. SHOW TABLES Syntax

```
SHOW [FULL] TABLES [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

SHOW TABLES lists the non-TEMPORARY tables in a given database. You can also get this list using the mysqlshow db_name command. The LIKE clause, if present, indicates which table names to match. The WHERE clause can be given to select rows using more general conditions, as discussed in Section 19.32, "Extensions to SHOW Statements".

Matching performed by the LIKE clause is dependent on the setting of the lower_case_table_names system variable.

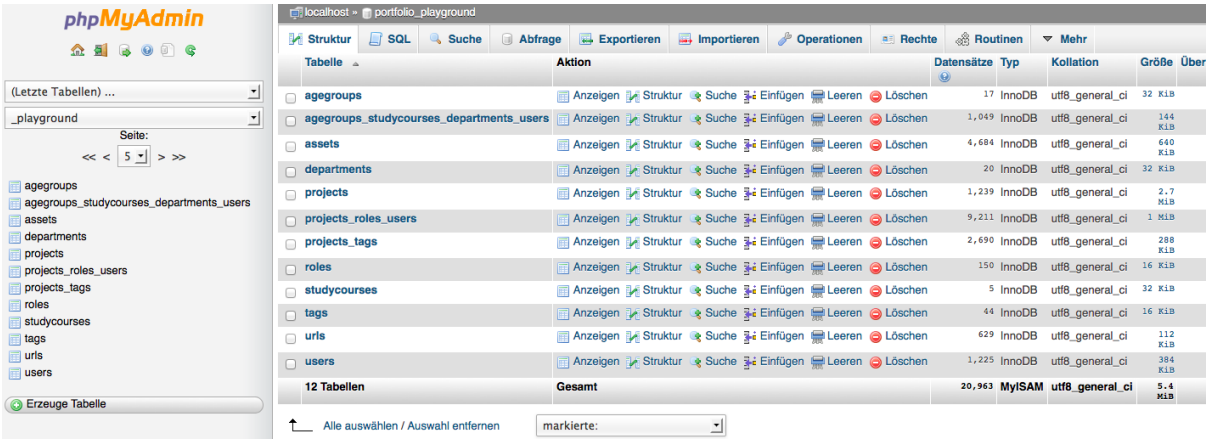
This statement also lists any views in the database. The FULL modifier is supported such that SHOW FULL TABLES displays a second output column. Values for the second column are BASE TABLE for a table and VIEW for a view.

If you have no privileges for a base table or view, it does not show up in the output from SHOW TABLES or mysqlshow db_name.

Abbildung 46.1: Abbildung: Dokumentation von MySQL auf http://dev.mysql.com

phpMyAdmin

PhpMyAdmin ist ein häufig verwendetes Tool zur Verwaltung von MySQL Datenbanken. Es ist in PHP geschrieben, kann also im Webspace installiert und über den Browser verwendet werden.



create table in phpMyAdmin

Über phpMyAdmin kann man viele SQL-Befehle durch Point & Klick formulieren, hier zum Beispiel ein *create table* Statement:

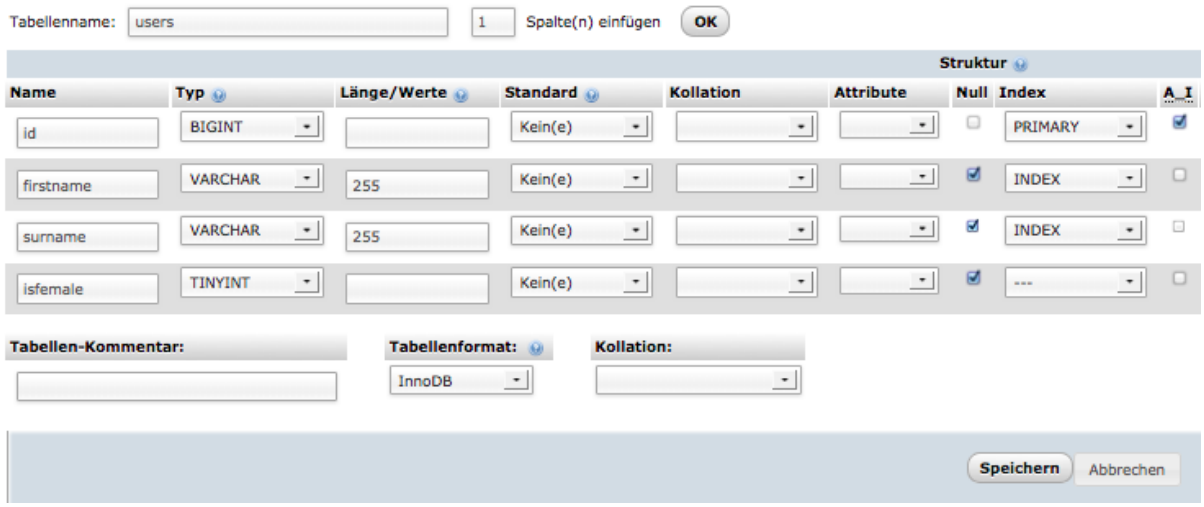


Abbildung 46.2: Abbildung: Tabelle anlegen mit phpMyAdmin

select in phpMyAdmin

Bei manchen Operationen zeigt phpMyAdmin das verwendete SQL-Statement an — das kann sehr lehrreich sein.

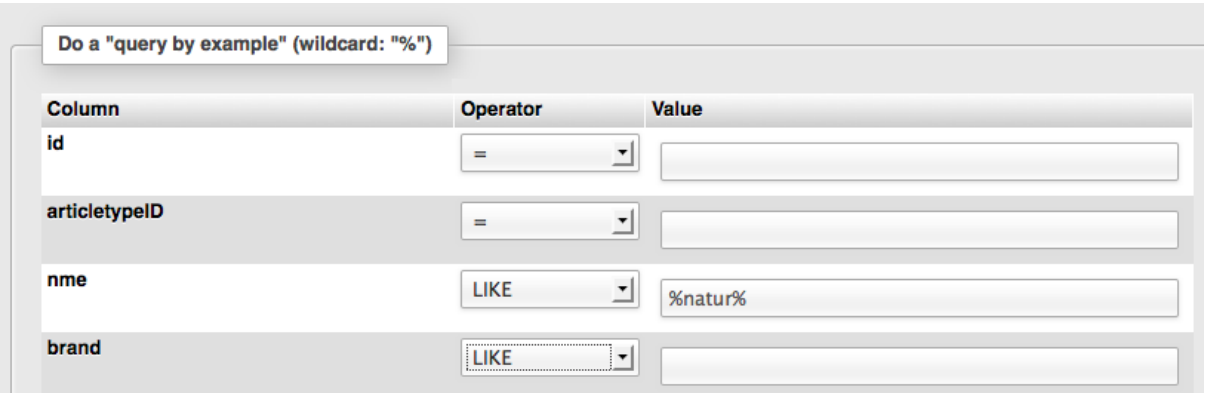
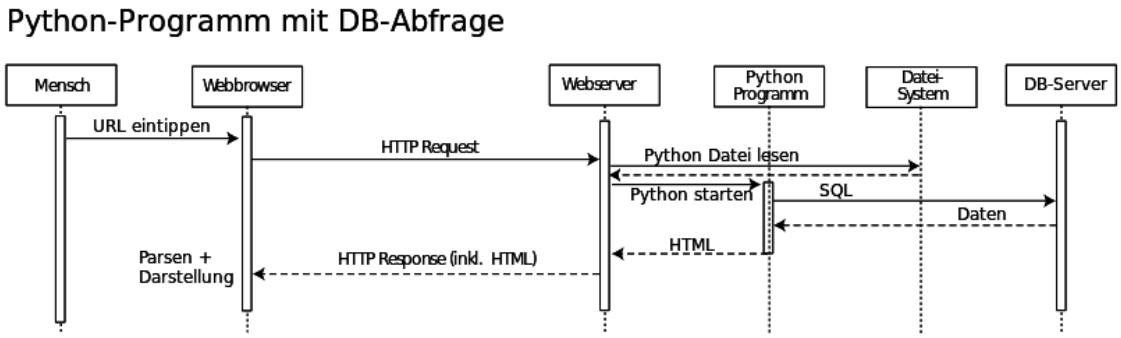


Abbildung 46.3: Abbildung: Suchen mit phpMyAdmin - Eingabemaske

Python und MySQL

Mit der Verwendung einer Datenbank wird der Aufruf einer Webseite noch einmal komplexer: der Webserver ruft das Python-Programm auf, das Python-Programm schickt eine Anfrage an die Datenbank.



Webseite wird von PHP erzeugt, mit DB-Abfrage

Ob der Datenbank-Server und der Webserver auf dem selben Computer laufen oder auf verschiedenen macht für die Programmierung kaum einen Unterschied.

Um von Python auf die Datenbank zuzugreifen gibt es verschiedene Schnittstellen. Dazu ist das Modul *python-mysqldb* zu installieren:

```
apt-get install python-mysqldb
```

Hier werden das „Python Database API“ vorgestellt, siehe auch PEP 249 Specification (http://legacy.python.org/dev/peps/pep-0249/).

Verbindungsaufbau

So funktioniert der Verbindung-Aufbau (und -Abbau) zur MySql-Datenbank:

```
import MySQLdb
con = MySQLdb.connect('localhost', 'testuser', 'testpwd', 'testdb')
```

Der Rückgabewert *con* ist ein sogenanntes “Datenbank-Handle” und stellt die Verbindung zum Datenbank-Server dar. Das erste Argument beschreibt, wie die Verbindung zur Datenbank vergestellt werden soll; in diesem Fall ist MySql am lokalen Rechner zu verwenden. Das zweite und dritte Argument sind einfach Strings mit dem Usernamen und Passwort für die Datenbank. Im letzten Argument wird der Datenbankname angegeben.

Mit dem Befehl *SET CHARACTER SET utf8* wird UTF-8 als Zeichensatz für die gelesenen Text-Daten festgelegt. Das macht natürlich nur Sinn, wenn die Daten in der Datenbank wirklich als UTF-8 gespeichert sind!

Anweisung an die Datenbank

Zuerst holt man sich einen *Cursor* von der Datenbank-Verbindung. Ein Cursor speichert den Zustand bei Lesen von Daten (z.B. viele Zeilen bereits gelesen wurden). Queries kann man einfach mit der Methode *execute* absetzen:

```
cur = con.cursor()
cur.execute("DELETE FROM users")
```

Hier werden alle Datensätze aus der Tabelle *users* gelöscht.

```
cur = con.cursor()
cur.execute("SELECT VERSION()")
version = cur.fetchone()
```

Hier wird die spezielle MySql-Funktion *version()* aufgerufen, die die Version der Datenbank- Software zurückliefert.

Abfrage der Datenbank

Eine Abfrage der Datenbank liefert normalerweise eine ganze Tabelle von Daten (mehrere Datensätze).

Ein Cursor bietet die Methoden *fetchone()*, *fetchmany()*, *fetchall()* mit denen einzelne Datensätze oder eine Reihe von Datensätzen abgerufen werden können.

KAPITEL 48

Javascript

Mit Javascript lernen Sie die wichtigste Programmiersprache des Web kennen.

Was Sie wissen sollten

- Ich weiss was der Unterschied zwischen Javascript und Java ist. Welche Sprache für eine bestimmte Problemstellung passend ist.
- Ich weiss, dass Javascript eine Programmiersprache im Web-Browser ist. Dass Javascript Kompatiblitätsprobleme hat, aber trotzdem die einzige Chance ist, Webseiten interaktiver zu machen.

Was Sie können sollten

- Ich kann mit den Javascript und dem DOM alle Elemente in eine Webseite manipulieren.

KAPITEL 49

Javascript Hintergrund

Anwendungsgebiet

Javascript ist eine Programmiersprache die in HTML eingebettet und vom Browser interpretiert wird. Im Browser ist der Wirkungsbereich von Javascript auf das Fenster und das aktuelle Dokument beschränkt – es kann nicht die Festplatte formatieren oder Excel starten (man sagt Javascript ist auf eine *sandbox* beschränkt).

Typische Verwendung

Neben der Verwendung für kleine Effekte im Browser hat Javascript noch weitere wichtige Anwendungsgebiete:

- Als „AJAX“ zum Nachladen von Daten – das macht das Web viel interaktiver und angenehmer in der Benutzung.
- Mit Javascript können Webseiten auch „offline“ – also ohne Verbindung zum Internet und dem Webserver – im Browser weiterarbieten.
- Der Server Node JS ist ein (Web-)Server, den man mit Javascript programmieren kann. Er wird besonders für realtime-Applikationen im Web eingesetzt, z.B. für Spiele.
- Javascript wird in einigen Programmen als Makro-Sprache verwendet, z.B. seit 2003 in Indesign und Illustrator.
- Die No-SQL Datenbanken CouchDB und MongoDB verwenden Javascript als Abfragesprache.
- Unter dem Namen Actionscript wird eine Variante von Javascript in Flash verwendet.

Javascript und Java

Es gibt noch eine zweite Programmiersprache mit sehr ähnlichem Namen: Java. Die beiden zu verwechseln ist recht peinlich. Die Verwechslung ist übrigens beabsichtigt: Im Jahr 1995 gab es einen großen Hype rund um Java, Netscape wollte davon profitieren und nannte die neue Skriptsprache im Browser „Javascript“.

	Javascript	Java
Wer hat's erfunden?	Netscape / Brendan Eich	Sun / James Gosling
Ausführung	Interpretiert Sprache	Kompilierte Sprache
Typen	Wenige Datentypen, Probleme werden erst zur Laufzeit erkannt.	Datentypen und Klassen werden streng unterschieden und zur Compilezeit geprüft.
Objektorientierung	Objekte und Prototypen	Objekte und Klassen
Verwendung	Im Webbrowser, am Webserver (node.js), in Flash, in Illustrator und Indesign.	Überall (Chipkarten, am Server, im Browser, im Handy).
Projekte	Früher nur winzige Projekte (z.B: Animations-Effekt), in den letzten Jahren: JS Frontend als wichtiger Teil einer Webappliaktion wie Facebook, Gmail, etc.	Alle, auch Großprojekte mit vielen Jahren Entwicklungszeit.
Für wen?	Auch Web-DesignerInnen	Nur InformatikerInnen

Geschichte von Javascript

Javascript wurde ursprünglich bei Netscape von Brendan Eich erfunden, und dann von verschiedenen Herstellern weiterentwickelt. Es war eines der Schlachtfelder im Browser-War: Microsoft und Netscape versuchten durch verschiedene Implementierungen die Position des eigenen Browsers zu verbessern und Webseiten im gegnerischen Browser unbrauchbar zu machen.

Erst nachträglich gab es (partielle) Einigung auf Standards. Mit dem ECMA-Standard 262 wurde die Syntax fixiert. Mit dem „Document Object Model" (DOM) wurde der Zugriff auf die Webseite vereinheitlicht. Libraries wie Prototype oder jQuery ebenen die letzten Unterschiede noch aus.

Lange Zeit waren kleine Javascript-Programme eine relative unabhängige Ergänzung für Web-Applikationen. Ein Beispiel dafür wäre ein Javascript-Kalender der die Eingabe eines Datums in ein Eingabefeld erleichtert.

Seit 2005 wird unter dem Schlagwort AJAX Javascript eng mit der serverseitigen Webapplikation verzahnt: AJAX steht für asynchrones Nachladen von Teilen der Webseite über Javascript.

Seit ca. 2008 sind offline-fähige Javascript-Applikationen möglich: Der Browser speichert alle notwendigen Teil der Website permanent ab, und kann die Applikation auch benutzen wenn keine Internetverbindung (mehr) besteht. Dies Applikationen haben z.B.: auf dem iPad eine große Bedeutung.

Unter den Stichwort „HTML5" sind auch viele Neuerungen in Javascript, viele neue APIs gemeint: File API, Websockets, Storage, WebGL, ...

49.1 Document Object Model

Das Document Object Model (DOM) ist ein allgemeines Modell wie ein Dokument (die Webseite oder auch ein XML-Dokument) von einer objektorientierten Skriptsprache aus manipuliert werden kann. Am 1. Oktober 1998 wurde das DOM eine offizielle W3-Empfehlung (recommendation) in der Version 1.0.

Das DOM einer Webseite kann man sich als Baum vorstellen, ähnlich der Ordnerstruktur im Dateisystem. Sie kennen diese Idee schon von CSS und den CSS-Selektoren.

In folgendem Beispiel ist der markierte *li*-Tag innerhalb des *ul*-Tags mit klasse *.sub*, der *li*-Tag enthält wiederum einen *a*-Tag.

Wenn man mit Javascript irgendeinen Teil der Webseite verändern will verändert man diesen Baum. Mögliche Veränderungen wären:

- Mache ein beliebiges Element des DOM unsichtbar (z.B. die ganze Liste)

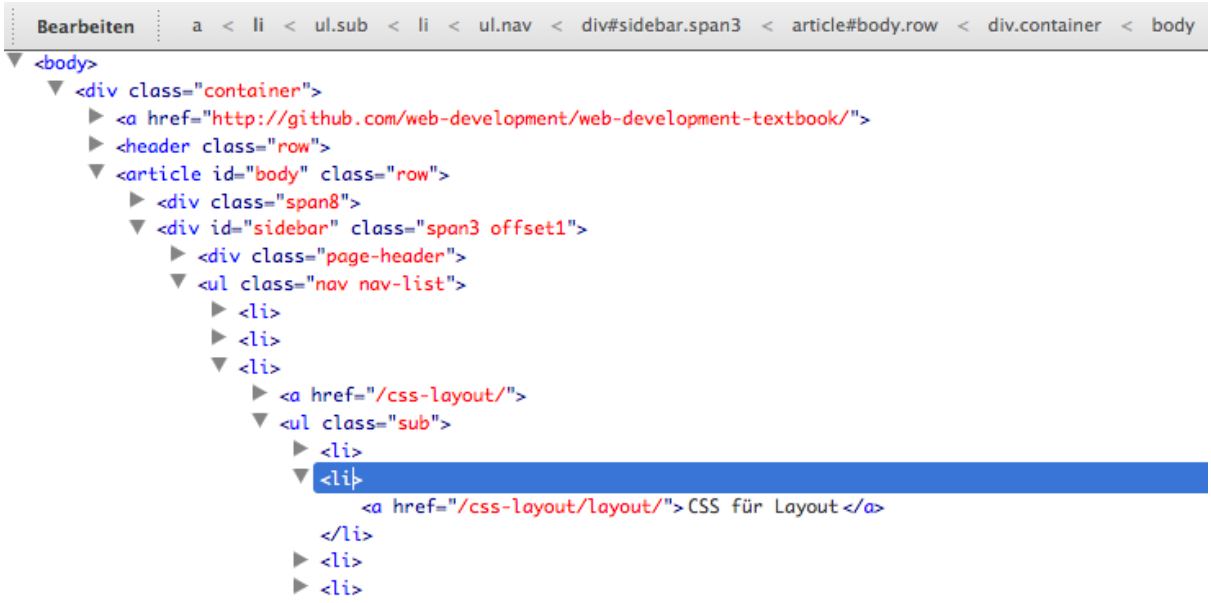


Abbildung 49.1: Abbildung: Bild des DOM, erzeugt mit dem DOM Inspektor von Mozilla

- Ändere den Style eines beliebigen Elements (z.B. die Schriftart der Überschrift)
- Füge neue Elemente ein (z.B. drei zusätzliche Listenpunkte)

KAPITEL 50

Javascript Basics

Ein vollständiges Beispiel für eine Webseite mit Javascript-Programm, auch live im browser (http://pmeerw.net/www-mm14/examples/farbwahl.html).

```
<body id="farbfeld">
<h1>Farbwahl</h1>
<form>
  <input type="button" value="Rot"  onclick="setcolor('red')">
  <input type="button" value="Grün" onclick="setcolor('#0F0')">
  <input type="button" value="Blau" onclick="setcolor('blue')">
</form>
<script>
function setcolor( c ) {
  b = document.getElementById('farbfeld');
  b.style.backgroundColor = c
}
</script>
</body>
```

In dieser Webseite ist an vier Stellen Javascript zu finden. Im *script* Tag am Ende des body, und dreimal in einem Attribut des *input*-Tags. Wie Sie sehen ist Javascript sehr eng mit HTML und CSS verzahnt.

Wer ein bestehende Website warten oder verändern will muss mindestens den bestehenden Javascript-Code erkennen können, um ihn nicht zu beschädigen. D.h. auch Leute die nur Design und keine Programmierung machen brauchen ein Grundverständnis von Javascript.

Es ist also erst einmal zu klären wie Javascript in HTML eingebunden wird.

Einbindung von Javascript

Möglichkeiten

- externe Javascript-Datei
- mit <script>-Tag
- Javascript in einer URL
- onEvent-Attribute

Externe Javascript-Datei

Man kann Javascript-Programme in eigenen Dateien speichern, diese haben traditionell die Endung .js. Wir werden später eine Javascript-Library namens jQuery verwendet. Mit dem *script*-Tag wird die externe Javascript-Datei eingebunden:

```
<script src="jquery.js" type="text/javascript"></script>
```

Wird der *script*-Tag auf diese Weise (mit dem Attribut *src*) verwendet, dann darf er keinen Inhalt zwischen <*script*> und </*script*> enthalten. Achtung: die Schreibweise ohne Ende-Tag: <*script src="jquery"*> ist nicht erlaubt!

Der <script>-Tag

Javascript-Programme können im HTML-Code mit dem *script*-Tag eingebettet werden. Das Programm wird dann beim Aufbau der Seite ausgeführt, siehe auch live im Browser (http://pmeerw.net/www-mm14/examples/countdown.html).

```
<h1>Selbstzerstörung</h1>
<script>
    i=10;
    while (i>0) {
        document.write("<br>in " + i + " Millisekunden");
        i--;
    }
</script>
<p><strong>Peng!</strong>
```

Dieses Programm ist übrigens ein gutes Beispiel für ein veraltetes Javascript-Programm. Die Methode *document.write()*, die hier für die Ausgabe verwendet wird, wurde durch das DOM größtenteils ersetzt. Nur sehr wenige Leute müssen bei sehr wenigen Gelegenheiten noch *document.write()* verwenden - z.B. die AutorInnen der Javascript-Libaries wie John Resig von jQuery.

Javascript in einer URL

Als URL kann man auch ein kleines Javascript-Programm angeben, z. B. bei einem Link:

```
<a href="javascript:location='http://www.google.at/'">zu Google nur mit Javascript</a>
```

Die "Javascript-in-einer-URL"- Schreibweise ist in HTML-Seiten nicht sehr sinnvoll, da sie für Browser ohne Javascript-Fähigkeit die Links unbrauchbar macht.

Hier eine Version die dem Prinzip der „graceful degradation" entspricht. Sie funktioniert mit und ohne Javascript sinnvoll:

```
<a href="http://www.google.at" onclick="ok=confirm('go?'); return ok;">google</a>
```

Ohne Javascript ist es ein ganz normaler Link zu Google.

Mit Javascript erscheint ein Dialog. Je nach Antwort wird der Link entweder aufgerufen oder nicht. Das funktioniert gleich wie beim onSubmit-Attribut des *form*-Tags: Wenn der Event-Handler *false* zurückgibt wird das Event unterbrochen.

Die onEvent-Attribute

Meist werden Javascript-Programme so geschrieben, dass sie nicht gleich beim Laden der Seite starten, sondern erst wenn gewisse Ereignisse (Events) eintreten.

Ein paar typische Events mit ihren typischen HTML-Tags:

```
<body onload="...">
```

Das Programm wird ausgeführt, nachdem die ganze Seite geladen und fertig dargestellt ist.

```
<a href="..." onmouseover="...">
```

Das Programm wird ausgeführt wenn die Maus über den Link bewegt wird (auch: *onmouseout*). (Achtung: funktioniert nicht auf Touch-Devices – so wie *:hover*).

```
<input type="button" onclick="...">
```

Das Programm wird ausgeführt wenn auf den Button geklickt wird. Das Programm muß *true* oder *false* zurückgeben um anzuzeigen ob die normale Funktion des Buttons wirklich ausgeführt werden soll.

```
<form onsubmit="...">
```

Das Programm wird ausgeführt wenn der Einsende-Knopf des Formulars betätigt wird, aber bevor die Daten wirklich gesendet werden. Falls der Javascript-Code *false* zurückgibt werden die Daten aber nicht versandt!

```
<a href="..." onclick="...">
```

Das Programm wird ausgeführt wenn der Link angeklickt wird. Falls der Javascript-Code *false* zurückgibt wird der Link aber nicht aktiviert!

```
<input onchange="...">
```

Das Programm wird ausgeführt wenn der Inhalt des Eingabefeldes verändert wurde.

```
<div ontouchstart="..." ontouchend="..." ontouchmove="....">
```

Nur auf Geräten mit Touchscreen.

```
<body onoffline="..." ononline="...">
```

Das Programm wird ausgeführt wenn das Gerät die Verbindung zum Internet verliert, bzw. wieder erhält.

Syntax von Javascript

Javascript hat eine ähnliche Schreibweise wie die Sprachen aus der C-Familie (C, C++, Java, Perl, PHP): Anweisungen werden mit einem Strichpunkt (Semikolon) getrennt, Blöcke werden mit geschwungenen Klammern gebildet.

Javascript ist eine objektorientierte Programmiersprache. Was bedeutet Objektorientierung? Die Grundidee ist, dass ein Objekt nicht nur eine Variable ist, die Daten speichert, sondern zusätzlich auch noch Funktionen existieren können, die zu diesem Objekt gehören.

Zum Beispiel ist das Objekt *document* bei Javascript im Browser immer vorhanden. Es repräsentiert die aktuell geladene Webseite. Dieses Dokument hat einige Eigenschaften (Variablen, Attribute) und einige Methoden (Funktionen):

```
// document.location
// diese Eigenschaft speichert die aktuelle URL
// wenn man einen neuen Wert in location speichert surft der Browser hin
document.location = "http://io9.com";
document.write("hi"); // die Methode write (über)schreibt die Webseite
```

Folgendes Beispiel verwendet die Methode *getElementById* des *document*-Objekts um ein bestimmtes *div* in der Webseite auszuwählen.

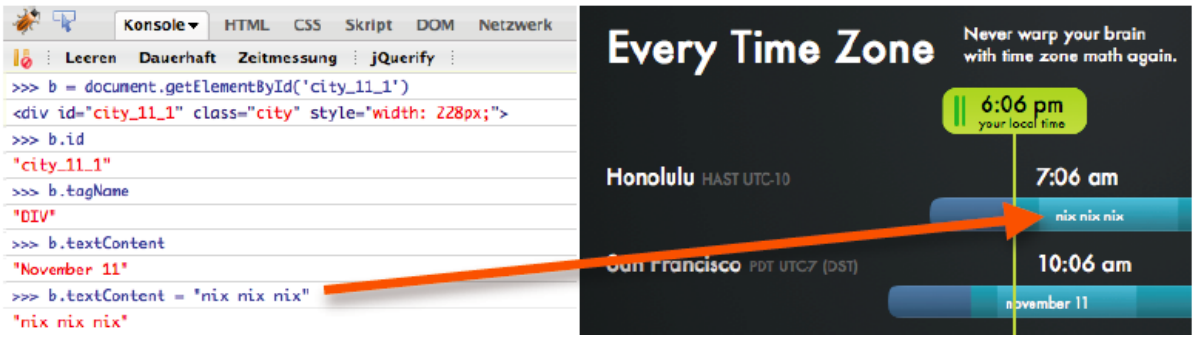


Abbildung 51.1: Abbildung: Javascript-Beispiel in FireBug auf der Website http://everytimezone.com/

Variablen

In anderen Programmiersprachen müssen Variablen deklariert (ein Datentyp für die Variable wird festgelegt) und initialisiert werden (ein erster Wert wird in die Variable gespeichert). In Javascript ist die Deklaration nicht nötig. Wenn eine Variable zum ersten Mal im Programm erwähnt wird, wird sie vom Interpreter angelegt.

Auch die folgenden Beispiele können Sie direkt in der Console ausprobieren wie in der Abbildung gezeigt: Wenn Sie einen Ausdruck eintippen wird er ausgewertet. Mit dem Befehl *console.log()* können Sie direkt auf die Konsole schreiben.

Datentypen

Variablen in Javascript können Zahlen, Strings, Arrays, Objekte enthalten – der Interpreter trennt Variablen nicht nach verschiedenen Datentypen:

```
a = 10;           // typeof(a) == "number"
a = 3.141;        // typeof(a) == "number"
a = "ein text";   // typeof(a) == "string"
a = true;         // typeof(a) == "boolean"
a = undefined;    // typeof(a) == "undefined"
```

Bei Zahlen in Javascript wird nicht zwischen integer und float unterschieden: bis 2⁵³ (9.007.199.254.740.992) können Ganzzahlen gespeichert werden, darüber nur noch floats. Die Details können Sie in [How numbers are encoded in JavaScript](http://www.2ality.com/2012/04/number-encoding.html) nachlesen.

Arrays

Arrays in Javascript können wie in C mit eckigen Klammern und Integer-Index ausgelesen werden: *b[0]*, *b[1]*, ... Aber eigentlich sind Arrays schon Objekte.

Für das Erzeugen des Arrays gibt es zwei Schreibweisen:

```
var b;
b = ["eins", 2, 3.141, true];           // JSON-Schreibweise
b = new Array("eins", 2, 3.141, true);  // Objekt-Schreibweise
// typeof(b) == "object"
```

Die Werte im Array können verschiedene Datentypen haben (String, Number, Boolean,...). Die Größe des Arrays ist nicht beschränkt, die aktuelle Länge des Arrays kann aus der Eigenschaft *.length* ausgelesen werden.

```
var i,t;
i = 0;
t = "Das Array:\n";
while( i < b.length ) {
    t += "Index " + i    + "\n";
    t += "Wert  " + b[i] + "\n";
    i++;
}
alert(t);
```

Für das Erzeugen von Objekten gibt es zwei Schreibweisen: die JSON-Schreibweise mit geschwungenen Klammern eignet sich gut für einmalige Objekte. Will man mehrere Objekte mit denselben Eigenschaften erzeugen, dann ist eine Construktor-Funktion besser geeignet.

```
var c;
c = {"farbe" : "rot", "beschriftung": "int pi == 3", "verkauft": true};

// Construktor-Funktion
function Shirt( f, b, v ) {
    this.farbe        = f;
    this.beschriftung = b;
    this.verkauft     = v;
}
d = new Shirt("rot", "int pi == 3", true);

alert("Das Shirt ist " + c.farbe ); // gibt "Das Shirt ist rot"
alert("Das Shirt ist " + d.farbe ); // gibt "Das Shirt ist rot"

// typeof c === "object"
// c.constructor === Shirt
// d.constructor === Object
```

Zugriff auf Eigenschaften

Eine Besonderheit von Javascript (die Sie nicht in anderen Programmiersprachen finden werden) ist, dass Eigenschaften eines Objekts nicht nur über die Punkt-Schreibweise, sondern auch über eckige Klammern – also wie ein Array – angesprochen werden können:

```
alert("Das Shirt ist " + c.farbe );
alert("Das Shirt ist " + c["farbe"] );
```

Schleife über Eigenschaften

Mit dieser Schreibweise und der *for*-Schleife kann man über alle Eigenschaften eines Objektes iterieren:

```
var i, t;
t = "Das Objekt:\n";
for( i in c ) {
    t += "Eigenschaft = " + i    + ", ";
    t += "Wert        = " + c[i] + "\n";
}
alert(t);
```

KAPITEL 52

Document Object Model

52.1 Lesen aus dem DOM

Die wichtigsten Befehle zur Manipulation des DOM finden Sie im Objekten „document“ und in Objecten vom Typ „node“. Ein Node ist ein Knoten des DOM-Baums, entspricht also einem HTML-Tag in einem HTML-Dokument. Die Attribute des HTML-Tags sind über *getAttribute* bzw. *setAttribute* zugänglich und manipulierbar.

Hier eine Liste der wichtigen Objekte, Methoden, Eigenschaften für die DOM:

```
document.getElementById()
document.getElementsByTagName()
document.querySelectorAll()    /* liefert Array   */
document.querySelector()       /* liefert 1 Node  */

document.createElement()

node.parentNode    /* liefert 1 Node           */
node.childNodes    /* liefert Array von Nodes */
node.firstChild
node.lastChild
node.previousSibling
node.nextSibling
node.data
node.attributes
node.innerHTML

node.getElementById()
node.getElementsByTagName()
node.getElementsByClassName()

node.appendChild()
node.cloneNode()
node.getAttribute()
node.setAttribute()
node.hasChildNodes()
node.insertBefore()
node.removeAttribute()
node.removeChild()
node.replaceChild()
```

Simples Beispiel

Ein bestimmter Tag wird über die *id* ausgewählt und sein *style*-Attribut gesetzt:

```
d = document.getElementById("person_25");
d.setAttribute("style", "display:none");
```

Diese beiden Zeilen könnten auch zu einer kombiniert werden:

```
document.getElementById("person_25").setAttribute("style", "display:none");
```

Achtung: Falls der Tag schon ein *style*-Attribute hatte wird dieses überschrieben. Der Wert des Attributes ist ein einfacher String.

Selektieren

Man kann CSS-Selektoren verwenden um Element auszuwählen, und zwar mit der Methode *document.querySelectorAll()*:

```
inputs = document.querySelectorAll("input");
i = 0;
while(i < inputs.length) {
  console.log("input mit name " + inputs[i].name );
  i++;
}
```

Text

Den eigentlichen Text der HTML-Seite kann man als *data* eines Text-Nodes auslesen.

```
<span id="v_25" class="vorname">Benjamin</span>
```

Das erste und einzige Kind des Spans ist ein Text-Node:

```
vn = document.getElementById("v_25").firstChild.data;
```

oder – etwas kürzer – über die Eigenschaft *textContent*:

```
vn = document.getElementById("v_25").textContent;
```

textContent funktioniert auch bei Nodes die noch weitere verschachtelte Tags enthalten und extrahiert immer den gesamten Text aus allen "Blättern" des DOM-Baums.

52.2 Manipulation des DOM

Noch einmal eine Liste der wichtigen Objekte, Methoden, Eigenschaften die für das Erzeugen, Zerstören oder Verändern des DOM notwendig sind:

```
node = document.createElement("h1");
              // erzeugt eine Node, Tag-Name angeben
node.innerHTML = "bla <strong>bla</strong bla";
              // Zugriff auf den "Inhalt" der node als String,
              // ist of schneller als DOM manipulation!
node.appendChild(newchild);
              // fügt die newchild als Kind an
newnode = node.cloneNode()
              // gibt eine Kopie der Node zurück,
              // die Kopie ist noch nicht im DOM Baum eingefügt!
node.insertBefore(newchild, oldchild)
              // fügt newchild als Kind von node ein, vor dem oldchild
oldchild = node.removeChild(child)
              // löscht child als Kind von node,
              // gibt es als "frei schwebende" node zurück
node.replaceChild(newchild, oldchild)
              // ersetzt kind oldchild durch kind newchild

node.setAttribute('value', 42)  // setzt ein attribut
node.removeAttribute('value')   // löscht ein attribute
```

Einfügen

Das Einfügen eines ganz neuen Elements in die Webseite ist am einfachsten mit der Eigenschaft *innerHTML*:

```
document.querySelector("body").innerHTML = "<p>Alles <b>neu</b></p>";
```

Duplizieren

Mit *cloneNode* kann man einen ganzen Teilbaum duplizieren, und wo anders wieder einfügen. So kann man z.B. in einem Pizza-Bestell-Formular die Eingabeelement für eine Pizza in einer *div* zusammenfassen:

```
<div class="pizza">
  <p>
    <select name="pizzatype[]">
      <option>Magaritha</option>
      <option>Vegetarian</option>
      <option>Quattro Staggione</option>
    </select>
    ...
  </p>
</div>
```

Dieses *div* kann man dann clonen, falls man mehrere Pizzen braucht:

```
var first_pizza = document.querySelector("div.pizza");
var new_pizza = first_pizza.cloneNode(true);
first_pizza.insertBefore(new_pizza, null);
```

Manipulieren der Klassen

```
document.querySelector('#foo').classList.add('bar');
document.querySelector('#foo').classList.remove('bar');
document.querySelector('#foo').classList.toggle('bar');
if( document.querySelector('#foo').classList.contains('bar') ) {
  //
}
```

Einfügen von Event Handlern

Wir haben im schon gesehen, wie Event-Handler direkt im HTML definiert werden können:

```
<form>
  <input type="button" value="Rot"  onclick="setcolor('red')">
  <input type="button" value="Grün" onclick="setcolor('#0F0')">
  <input type="button" value="Blau" onclick="setcolor('blue')">
</form>
<script>
  function setcolor( c ) {
    b = document.getElementById('farbfeld');
    b.style.backgroundColor = c
  }
</script>
```

Mit der Methode *addEventListener* kann das auch von Javascript aus erfolgen.

Hier ein erster Entwurf: Um die Buttons einzeln anzusprechen, müssen wir eine id hinzufügen:

```
<form>
  <input type="button" value="Rot"  id="r">
  <input type="button" value="Grün" id="g">
  <input type="button" value="Blau" id="b">
</form>
<script>
  function setcolor( ev ) {
    b = document.getElementById('farbfeld');
    b.style.backgroundColor = 'red';
  }
  document.getElementById('r').addEventListener('click', setcolor);
  document.getElementById('g').addEventListener('click', setcolor);
  document.getElementById('b').addEventListener('click', setcolor);
</script>
```

Wir haben aber ein Problem: der Methode *addEventListern* wird als zweites Argument die Methode *setcolor* übergeben. Das ist nicht dasselbe wie ein Aufruf der Methode, dann würde man schreiben: *setcolor()*.

Hier gibt es keine einfache Möglichkeit ein Argument für die Farbe mit zu geben!

Eine Lösung für dieses Problem: was in der Methode *setcolor* zur Verfügung steht ist *this*: der Node der angeklickt wurde, in unserem Fall der jeweilige Button.

Wir müssen also einen Weg finden die Farbe direkt aus dem Button angeklickten auzulesen. So können wir zum Beispiel die Hintergrundfarbe des Buttons verwenden:

```
<form>
  <input type="button" value="Rot"  style="background-color:red"  id="r">
  <input type="button" value="Grün" style="background-color:#0F0" id="g">
  <input type="button" value="Blau" style="background-color:blue" id="b">
</form>
```

```
<script>
  function setcolor( ev ) {
    b = document.getElementById('farbfeld');
    b.style.backgroundColor = this.style.backgroundColor;
  }
  document.getElementById('r').addEventListener('click', setcolor);
  document.getElementById('g').addEventListener('click', setcolor);
  document.getElementById('b').addEventListener('click', setcolor);
</script>
```

Programm live im browser (http://pmeerw.net/www-mm14/examples/farbfeld-dom.html).

KAPITEL 54

jQuery - Einführung

Mit der einfachen Schreibweise von jQuery lernen Sie schneller und einfacher Javascript-Programme zu erstellen.

Was Sie wissen sollten

- Ich kann die Fachbegriffe „unobstrusive Javascript“ / „graceful degradation“ / „progressive enhancement“ erklären
- Ich weiss, dass Javascript-Funktionen in Variablen gespeichert werden können, was der Unterschied zwischen *f = c();* und *f = c;* ist·
- Ich weiss, dass Javascript-Funktionen ohne Namen definiert werden können, wie man so eine anonyme Funktion definiert und verwendet.

Was Sie können sollten

- Ich kann mit jQuery Teile einer Webseite aus- und einblenden
- Ich kann Mit jQuery dynamische Formulare gestalten

Vertiefung

- Responsive News (2012) Cutting The Mustard (http://responsivenews.co.uk/post/18948466399/cutting-the-mustard) - wie die BBC progressive enhancement verwendet
- Archibald (2013) Progressive Enhancement is Still Important (http://jakearchibald.com/2013/progressive-enhancement-still-important/)

Unobstrusive Javascript

In Zusammenhang mit jQuery werden die Fachbegriffe „graceful degradation“, „progressive enhancement“ und „unobstrusive“ verwendet. Dahinter verbergen sich zwei verwandte, aber verschiedene Konzepte:

graceful und progressive

Die Library jQuery unterstützt das Prinzip der „graceful degradation“ – auch ohne Javascript sind Webseiten mit jQuery immer noch gut verwendbar. Dieses Prinzip wird auch „progressive enhancement“ genannt, und bezieht sich nicht nur auf Javascript, sondern auch auf andere „Zusatz-Technologien“ wie z.B. Flash.

Die Idee ist dabei immer die Gleich: man baut die Webseite zuerst ohne Javascript, und fügt dann Javascript hinzu (ohne die Verwendbarkeit ohne Javscript zu zerstören). Der Inhalt (Content) der Webseite bleibt auch ohne Javascript zugänglich.

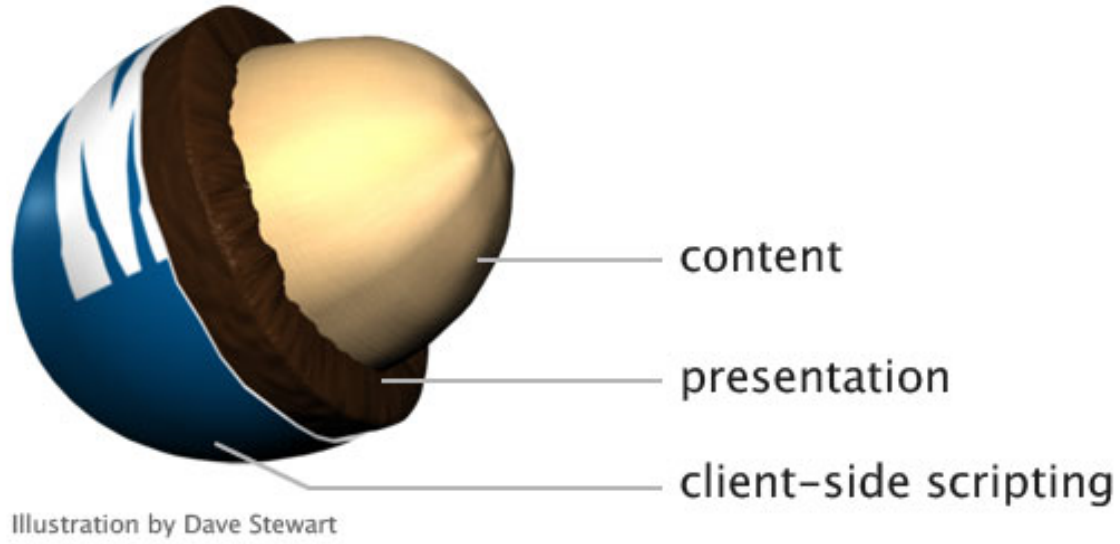


Abbildung 55.1: Abbildung: Die Rolle von Content, Darstellung und Programmierung (Unobstrusive Javascript)

Von dieser Herangehensweise profitieren nicht nur Blinde, Menschen mit veralteten Browsern oder exotischen Ausgabegeräte. Auch für Suchmaschinen wie Google oder andere Programme die die Information aus den Webseiten weiter verarbeiten ist dieses Prinzip hilfreich.

Grenzen von Graceful Degradation

Es gibt Websites, bei denen dieser Ansatz nicht funktionieren kann. z.B. für einen Shooter als Browsergame kann man nicht nicht eine Javascript-freie Alternative anbieten.

Für viele Apps funktioniert das aber. Probieren Sie z.B. gmail ohne javascript aus. Die Interaktion ist etwas umständlicher, aber man kann alle Features benutzen.

unobstrusive

Bei der Verwendung von jQuery bleibt der HTML-Code „javascript-frei“: jQuery wird nur an einer Stelle, im Head des Dokuments eingebaut. Das nennt man „unobstrusive Javascript“.

jQuery vom CDN

Die URLs https://codeorigin.jquery.com/jquery-latest.min.js und http://codeorigin.jquery.com/jquery-latest.min.js kann man für alle Webseiten die online sind verwenden: hinter codeorigin.jquery.com steht nicht nur ein Server, sondern der das MaxCDN (http://www.maxcdn.com/) CDN (Content Delivery Network). Nur wenn man offline entwickeln will muss man die Library wirklich herunterladen.

KAPITEL 56

Graceful Degradation

Die Aufgabenstellung: auf einer langen Webseite sind mehrere Anker-Punkte mit *<a name=...>* gesetzt, über ein Navigationsmenü soll man diese erreichen können. Diese Seite (http://brigitte-jellinek.at) zeigt ein funktionierendes Beispiel.

Version 1

Wir beginnen mit völlig problemlosem HTML:

```
<div id="navigation">
  <a href="#s0">Thema 0</a>
  <a href="#s1">Thema 1</a>
  ...
</div>
<section id="s0">
  <h2>Thema 0</h1>
  <p>bla bla ...</p>
</section>
<section id="s1">
  <h2>Thema 1</h1>
  <p>bla bla ...</p>
</section>
```

```
#navigation {
  position: fixed;
  z-index: 10;
  bottom: -1px;
}
```

Version 2

In einem Versuch die Seite zu verbessern ersetzen wir nun die Links durch den Aufruf einer Javascript-Funktion:

```
<div id="navigation">
  <a onClick="scrollToMe('#s0')">Thema 0</a>
  <a onClick="scrollToMe('#s1')">Thema 1</a>
  ...
</div>
```

Die Javascript-Funktion verwendet jQuery und die übergebene ID um den Ziel des Links ausfindig zu machen, und dann die jQuery Methode *offset* um die Position des Ziels im Dokument zu berechnen.

Mit der jQuery-Methode *animate* wird dann eine Animation erzeugt: binnen 800 Millisekunden wird die Seite durch Setzen von *scrollTop* von der aktuellen Scrollposition in die Scrollposition gebracht, die das Ziel ganz oben im Fenster anzeigt.

Mit *return false* wird die "normale" Funktion des Links deaktiviert.

```
function scrollToMe(id) {
  var top  = $(id).offset().top;
```

```
  $('body').animate({ scrollTop: top }, 800);
  return false;
}
```

Diese Version ist kein Beispiel für gutes Javascript: in manchen Browsern funktioniert das Scrollen der Seite mit *scrollTop* nicht.

Mit dieser Version haben wir

- die klassischen Links zerstört
- für manche Javascript-fähige Browser eine Animation eingefügt
- für manche Javascript-fähige Browser keine Animation eingefügt

Wir haben dabei beide Prinzipien gebrochen

- kein progressive enhancement: Links funktionieren nicht mehr
- kein unobstrusive javascript: Javascript-Code direkt in HTML-Attributen

Version 3

Im nächsten Versuch werden wir jQuery verwenden, um unobstrusive zu programmieren:

Die Navigation wird wieder zurückgestellt auf normale HTML-Links:

```
<div id="navigation">
  <a href="#s0">Thema 0</a>
  <a href="#s1">Thema 1</a>
  ...
</div>
```

Die Funktion *scrollToMe* wird als Eventhandler implementiert: sie erwartet ein Event als Argument und den angeklickten Noden in der Variable *this*. Ausserdem verwendet die Funktion die jQuery-Methode *preventDefault* um das "normale" Verhalten des Links zu unterbinden.

In der letzten Zeile wird an alle Links in der Navigation die Funktion *scrollToMe* als Eventhandler für *click* angefügt.

```
$(document).ready( function () {

  function scrollToMe(event) {
    var link = $(this).attr('href'),
        top  = $(link).offset().top;
    $('body').animate({
      scrollTop: top
    }, 800);
    event.preventDefault();
  }

  $('#navigation a').on('click', scrollToMe);
});
```

Diese Variante funktioniert schon besser:

- die klassischen Links funktionieren für Browser ohne Javascript
- für manche Javascript-fähige Browser eine Animation eingefügt
- für manche Javascript-fähige Browser ist die Animation immer noch kaputt

Wir haben damit schon ein Prinzip eingehalten, und sind beim anderen Prinzipien auf halben Weg

- teilweise progressive enhancement: Links funktionieren für Browser ohne Javascript, in der Javascript-Variante bleibt die URL immer gleich
- unobstrusive Javascript: erfüllt

Version 4

Im nächsten Schritt werden wir sicher stellen, dass die Animation nur in solchen Browsern verwendet wird, wo sie auch funktioniert.

Achtung: hier gibt es einen falschen und einen richtigen Ansatz:

1. Browser Detection: Unterscheidung nach Namen, Versionsnummer, Betriebssystem des Browsers
2. Feature Detection: Unterscheidung nach genau der Fähigkeit, die ich verwenden will

Die erste Variante funktioniert nicht: die Selbstoffenbarung der Browser kann falsch sein, ich kenne nicht alle Browser und ihre Fähigkeiten. Siehe auch Zakas (2009) Feature detection is not browser detection (http://www.nczonline.net/blog/2009/12/29/feature-detection-is-not-browser-detection/).

Die Funktion *scrollToMe* bleibt unverändert. Wir testen ob die Funktion scrollTop wirklich den Wert von scrollTop verändern kann. Wenn das funktioniert wird die globale Variable *can_scroll* auf *true* gesetzt, andernfalls auf *false*.

```
// try out scrollTop,
// set global Flag can_scroll
var old_scrolltop = $('body').scrollTop();
$('body').scrollTop(10);
window.can_scroll = ( $('body').scrollTop() > 0 );
$('body').scrollTop(old_scrolltop);

if ( window.can_scroll ) $('#navigation a').on('click', scrollToMe);
```

Diese Herangehensweise (Feature Detection, dann Flags setzen, das im weiteren Code verwendet werden kann) wird von der Library modernizr (http://modernizr.com/) für eine lange Liste von Features angeboten.

Nebenbemerkung: In ganz seltenen Fällen muss man doch Browser Detection machen. Eine gute Library dafür ist HTML5 please (http://api.html5please.com/). Damit kann man eine Liste von Features angeben die erfüllt sein müssen damit die Seite funktioniert. Ist das nicht der Fall, dann wird eine entsprechende Meldung angezeigt

Diese Variante behebt das Problem mit nicht-funktionierenden Javascript-Browsern:

- die klassischen Links funktionieren für Browser ohne Javascript
- die klassische Version wird auch für "Animations-unfähige" Javascript Browser verwendet
- für manche Javascript-fähige Browser funktioniert die Animation

Es bleibt aber noch ein Problem:

- teilweise progressive enhancement: in der Javascript-Variante bleibt die URL immer gleich
- unobstrusive javascript: erfüllt

Version 5

In der klassischen Version ändert sich beim Navigieren zwischen den Ankern jeweils die URL im Browser. Wenn ich ein Ziel annavigiere, und dann die URL kopiere um einen Link zu setzen bzw. mir ein Bookmark setze, dann verweist die URL die ich verwende wirklich wieder genaue auf das Ziel.

Dieses Verhalten ist erstrebenswert, wird aber von der "animierten" Version derzeit nicht geliefert.

Diese "Navigierbarkeit" ist auch ein klassisches Problem von AJAX-Applikationen, die Lösung die wir hier entwickeln funktioniert auch dort:

In die Funktion *scrollToMe* wird eine Zeile eingefügt. Mit dem History-Objekt kann man den Browser von Javascript aus "navigieren": mit *history.back()* zum Beispiel einen Schritt zurück gehen.

Mit *history.pushState()* kann man zu einer neuen Seite navigieren, sie wird dabei an die History angefügt - das ist das "normale" Verhalten des Browsers.

Eine Alternative ist *history.replaceState()* - dabei wird die aktuelle Seite ersetzt, die Browser-History wird nicht länger.

Beide Methoden erwarten drei Argumente - ein Objekt und zwei Strings - aber nur das letzte Argument wird derzeit benutzt. Es ist ein String mit der URL die geladen werden soll.

```
function scrollToMe(event) {
  var link = $(this).attr('href');
  ...
  window.history.pushState( {}, "Thema " + link, link);
}
```

Mit dieser Variante haben wir für die Javascript-Browser alle Funktionalität der einfachen HTML-Version wiederhergestellt. Und zusätzlich gibt es eventuell noch eine schöne Animation.

Damit sind beide Prinzipien voll erfüllt:

- progressive enhancement - alle Browser erhalten die maximal mögliche Funktionalität
- unobstrusive javascript - kein Javascript-Code im HTML

KAPITEL 57

Javascript im Browser

In diesem Kapitel werden wir die Verwendung von Javascript im Browser näher betrachten.

Was Sie wissen sollten

- Was die Unterschiede von jQuery und plain Javascript sind
- Was ein Event ist
- Was AJAX bedeutet

Was Sie können sollten

- Eine Website mit Events interaktiver machen
- Sowohl mit als auch ohne jQuery Javascript programmieren
- Mit jQuery einen Teil der Webseite mit AJAX (nach-)laden
- Mit AJAX Daten laden

jQuery und reines Javascript

Das jQuery Objekt

Alle Funktionen die wir hier disskutieren werden geben immer ein jQuery Objekt zurück, welches eine Liste von Nodes anhalten kann. Manchmal ist diese Liste leer, manchmal enthält sie nur ein Element, manchmal mehrere.

Um die jQuery Objekte besser von anderen Javascript Variablen unterscheiden zu können werden wir ihre Variablennamen immer mit einem Dollar beginnen.

Plain Javascript:

```
// von Javascript zu jQuery

// Eine Node nach ID auswählen
node = document.getElementById("id");

// jQuery Objekt konstruieren
$q = $(node);
```

jQuery:

```
// Eine Node mit ID auswählen
$q = $("#idname");

// DOM-Node extrahieren:
node = $q.get(0);
node = $q[0];
```

Wenn mehrere Nodes gefunden werden muss das in jQuery nicht besonders behandelt werden, in reinem Javascript schon:

```
// plain Javascript
// Nodes mit CSS-Selektor auswählen
arr = document.querySelectorAll(".class");

// wie viele sind es?
if ( arr.length > 1 ) {
  console.log("mehrere");
}
```

Wie wählt man Nodes aus der DOM aus?

```
// Eine Node nach ID auswählen
node = document.getElementById("id");

// Erste Node auswählen
node = document.querySelector("h2");

// Alle Nodes auswählen
arr= document.querySelectorAll("h2");
```

... und mit jQuery:

```
// Eine Node nach ID auswählen
$q = $("#id");

// Erste Node auswählen
$q = $("h2:first");
$q = $("h2").first();

// Alle Nodes auswählen
$q = $("h2");
```

Attribute

Wie liest man Attribute aus, wie setzt man sie? Das ist in reinem Javascript sogar etwas einfacher als mit jQuery: die Attribute sind einfach Eigenschaften des jeweiligen Nodes und könnnen direkt manipuiert werden.

jQuery erleichtert einerseits das Verarbeiten einer ganzen Liste von Nodes, und bietet andererseites ein paar besser verständliche Werte für Attribute:

```
// plain Javascript - nur erstes h2!
document.querySelector('h2').align;
document.querySelector('h2').align = 'center';

// Form-Element deaktivieren
document.querySelector('input').disabled = "disabled";

// jQuery - alle h2!
$('h2').attr('align');
$('h2').attr('align', 'center');

// Form-Element deaktivieren
$('input').attr('disabled', true);
```

Bei der Manipulation von CSS ist zu beachten: die Schreibweise von CSS-Eigenschaften mit einem Minus-Zeichen lässt sich nicht nach Javascript übertragen. In Javscript wird deswegen aus *background-color* die Eigenschaft *backgroundColor*:

```
document.querySelector('h2').style.backgroundColor = "yellow";
```

bzw.

```
$('h2').css('background-color', 'yellow');

// mehrere Eigenschaften setzen
$('h2').css({backgroundColor: 'yellow', color: 'red'});
```

HTML einfügen

Wie fügt man eine Node in die DOM ein?

versus

```
// jQuery

// Text einfügen
$n.append("Hallo");

// Tag einfügen
$l = $(
  "<a href='page.html'>Hallo</a>"
);
$n.append($l);


// oder
```

```
$l = $("<a>")
     .attr('href', 'page.html')
     .text('Hallo');
$n.append($l);
```

Event-Listener

Wie fügt man einen Event-Handler ein?

```
// plain Javascript
node.addEventListener("click", f);
```

bzw.

```
// jQuery
$n.on("click", f);
```

Wann brauche ich jQuery? Wann brauche ich ein Framework?

Als jQuery im Jahre 2006 erschien brachte es große Fortschritte gegenüber "reinem Javascript". In den Jahren seither hat jQuery die Weiterentwicklung von Javascript beeinflusst: so wurde *querySelector* und *querySelectorAll* erst nach jQuery in den Javascript Standard aufgenommen, und landet z.B. in Firefox 3.5 im Juni 2009.

Die "reine" Javascript Lösung bringt Performance-Vorteile, besonders auf mobilen Endgeräten, wo das Laden der Library und der Speicherverbrauch durch die Library größere Auswirkungen haben als am Desktop.

Siehe auch

- Vortrag von Estelle Weyl: You don't need a Framework for that! (http://www.youtube.com/watch?v=FbpUt3XLGlE), Slides (http://estelle.github.io/fluentconf/#slide1).

AJAX

Wir kennen schon die Funktionsweise von HTTP. Bisher wurde ein HTTP Request durch eine Handlung der UserIn ausgelöst (URL eintippen, Link anklicken), oder um Ressourcen nachzuladen die zu einem HTML-Dokument gehören.

Mit AJAX lernen wir nun eine neue Art kennen, wie HTTP-Request verwendet werden: Asynchrone Requests.

Was ist AJAX?

AJAX ist die englische Abkürzung für „Asynchrones Javascript und XML“. In diesem Kapitel lernen Sie was das genau bedeutet, und dass sich hinter dem X zum Schluss auch andere Format verbergen können

Ein Beispiel für die Verwendung von AJAX ist das in der Abbildung gezeigte Eingabefeld: schon während des Eintippens eines Suchwortes wird eine Anfrage an den Webserver geschickt. Dieser antwortet mit einer Liste von vorgeschlagenen Namen. Diese Liste wird mit Javascript in einem *div* unterhalb des Eingabefelds angezeigt:

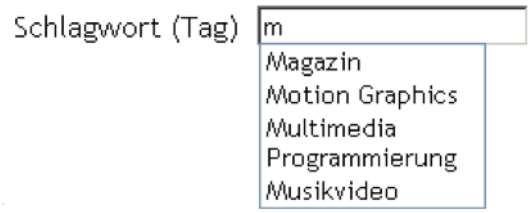


Abbildung: Vorschläge für die Eingabe werden über AJAX geladen

Mit AJAX wird hier eine HTTP-Anfrage gesendet. Der Unterschied zu einer „normalen“ HTTP-Anfrage: Bei einer „normalen“ HTTP-Anfrage schaltet der Browser auf „Warten“, eine neue vollständige Webseite wird geladen und angezeigt. Asynchron heisst: der Request wird abgesetzt, das Javascript-Programm läuft sofort weiter, die UserIn kann weiterhin mit der Webseite interagieren. Erst wenn die Antwort des Servers vorliegt wird die normale Darstellung der Seite kurz unterbrochen und ein Javascript-Programm fügt die Daten in die Seite ein.

AJAX im Javascript Programm

Auf der Ebene des Javascript Programm-Codes sieht der Unterschied zwischen synchron und asynchron so aus:

Bevor *Befehl3* ausgeführt werden kann muss erst die Antwort des Servers vorliegen – hier kann also eine Wartezeit von mehreren Sekunden entstehen.

Befehl3 kann sofort ausgeführt werden, egal ob und wie schnell der Server antwortet. Wenn die Daten vom Server schließlich einlangen wird die Funktion *handle_data* aufgerufen um die Daten zu verarbeiten. Das kann z.B. gleichzeitig mit *Befehl4* erfolgen.

HTTP

Betrachten wir nun den Ablauf für ein Textfeld mit *Autocomplete*-Funktion, wie in der obigen Abbildung gezeigt. Folgende Abbildung ist ein Sequenz Diagramm (http://de.wikipedia.org/wiki/Sequenzdiagramm), die Zeit läuft von oben nach unten.

Zuerst wird die Webseite mit dem Formular geladen: der Browser schickt die Anfrage an den Server und erhält eine Antwort. Was immer zuvor im Browser angezeigt wurde wird - nach Ankunft der HTTP Response - gelöscht, die neue Seite wird im Browser dargestellt. Diese Verhalten des Browsers ist uns schon bekannt.

Nun kommt der neue Teil: das Eintippten des ersten Buchstabens ins Eingabefeld löst ein Javascript-Programm aus, das einen AJAX-Request absetzt. Im Netzwerk ist das ein ganz normaler HTTP Request, für den Server gibt es keinen Unterschied zu jedem anderen Request.

Was anders ist, ist das Verhalten des Browsers: Wenn die Daten der Response einlangen wird nicht die Seite gelöscht, sondern es wird eine Javascript-Funktion in der Seite aufgerufen, die die Daten entgegen nimmt. Für das Autocomple-Verhalten bestehen die Daten aus einer Liste von Vorschlägen, die Javascript-Funktion zeigt diese Vorschläge unterhalb des Eingabefeldes an.

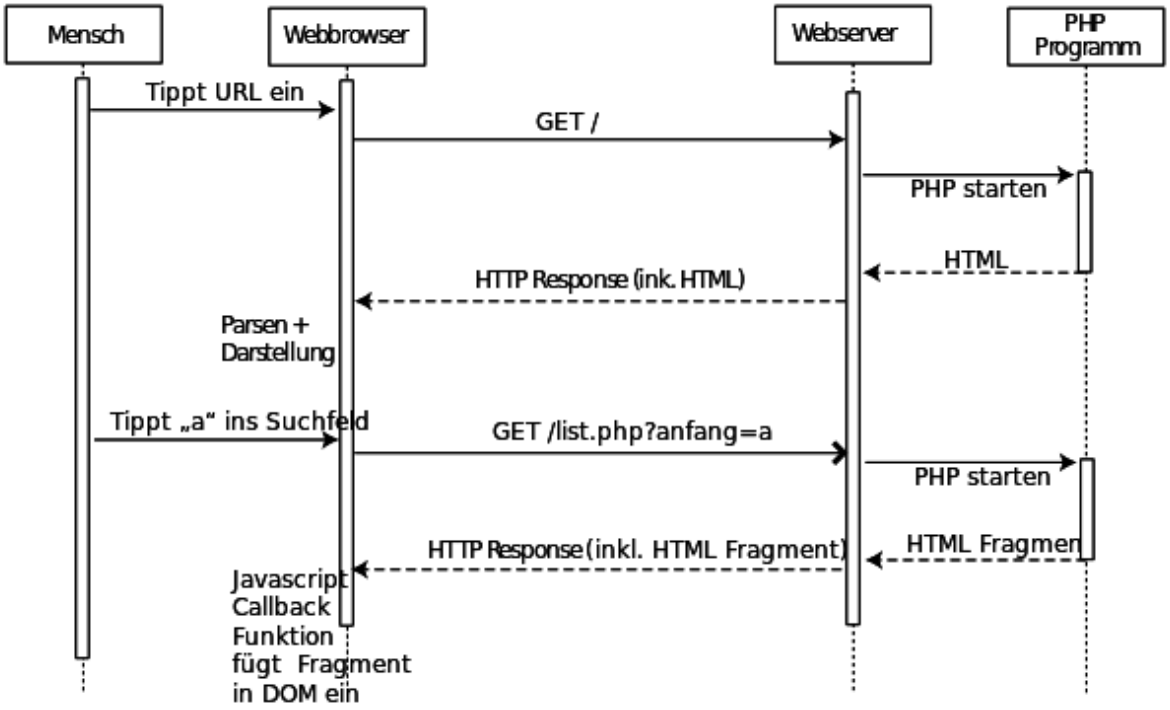


Abbildung 59.1: Abbildung: AJAX Ablauf

Datenformate - mehr als nur XML

Das X am Ende von AJAX steht für XML – das stimmt aber nicht: die Daten vom Server können im XML-Format gesendet werden, aber genauso auch als HTML oder reiner Text oder JSON. Man könnte das X in AJAX auch als „X-beliebiges Format" deuten. Das wichtigste Javascript-Konstrukt für AJAX ist das *XMLHTTPRequestObject*, das der Javascript-Interpreter des Browsers zur Verfügungs stellt. Leider gibt es bei diesem Objekt Unterschiede zwischen den Browsern. Um diese Unannehmlichkeiten zu vermeiden, sollte man fertige Libraries verwenden, die die Browser-Unterschiede verbergen.

Simples Javascript Beispiel

Im ersten AJAX Beispiel wird der Output eines Counters in eine HTML-Seite eingebunden.

```
<html>
<head>
  <title>AJAX counter</title>
  <style>
     p#counter_zeile { display: none; }
  </style>
</head>
<body>
  <h1>Webseite</h1>
  <p>mit total viel Inhalt</p>
```

```
  <p id="counter_zeile">Counter: <span id="counter_zahl">?</span></p>

  <script>
    window.addEventListener('load', loadCounterWithAjax);

    function loadCounterWithAjax() {
      document.getElementById('counter_zeile').style.display = "block";
      var ajax_request = new XMLHttpRequest();
      ajax_request.addEventListener('load', handleCounterData);
      ajax_request.open("GET", "counter_ajax.cgi");
      ajax_request.send();
    }

    function handleCounterData() {
      document.getElementById('counter_zahl').innerHTML =
        this.responseText;
    }
  </script>
</body>
</html>
```

Für den Fall das Javascript nicht funktioniert wird die ganze Counter-Zeile nicht angezeigt (*display:none* als Style). Falls Javascript funktioniert wird die Zeile eingeblendet.

Das *XMLHttpRequest* Objekt liefert verschiedene Events, hier wird nur für das *load* Event eine Funktion als Listener angebracht. Mit der *open* methode wird der HTTP-Request konfiguriert, aber erst mit *send* wirklich abgeschickt. Da er Request asynchron erfolgt geht der Javascript-Interpreter weiter, und wartet nicht auf den HTTP-Response.

Erst sehr viel später, wenn der HTTP-Response vorliegt, wird die Funktion *handleCounterData* aufgerufen. Die Funktion erhält das *XMLHttpRequest* Objekt in der Variable *this*.

Simples jQuery Beispiel

jQuery bietet einige Vereinfachungen gegenüber Javascript: die Funktion *load* erledigt nicht nur den AJAX Request, sondern auch das Einfügen des Rückgabewerts in einen DOM Node:

```
<html>
<head>
  <title>AJAX counter</title>
  <style>
      p#counter_zeile { display: none; }
  </style>
</head>
<body>
  <h1>Webseite</h1>
  <p>total viel Inhalt</p>
  <p id="counter_zeile">Counter: <span id="counter_zahl">?</span></p>

  <script src="jquery.js"></script>
  <script>
  $(document).ready(function(){
      $("p#counter_zeile").show();
      $("#counter_zahl").load("counter_ajax.cgi");
  });
  </script>
</body>
</html>
```

Die ganze Arbeit macht hier jQuery in der Zeile:

$("#counter_zahl").load("counter_ajax.cgi");

Das Element mit der ID counter_zahl wird ausgewählt. Mit dem Load-Befehl wird eine AJAX-Anfrage an die angegebene URL abgesetzt. Wenn der HTTP-Response beim Browser ankommt, werden die gelieferten Daten

in das ausgewählte Element eingefügt. (Die gelieferten Daten sollten also reiner Text oder ein HTML-Fragment sein.)

KAPITEL 60

Index

- *genindex*
- *modindex*
- *search*