Front-End Development

21.02.2014 @ next level GmbH

Einleitung

Dieses Handout fasst die Inhalte der Präsentation zum Thema „Front-End Development“ vom 21.02.2014 zusammen. Die Inhalte basieren zu einem großen Teil auf Fragen von Mitarbeitern der next level GmbH.

Es wird sowohl das grundlegende Zusammenspiel von HTML, CSS und JavaScript umrissen, als auch neuere Entwicklungen in den Bereichen HTML5/CSS3, Server-seitige Technologien und Aspekte im Recruiting-Bereich.

Die Basics

HTML in Blitzgeschwindigkeit

Die Hauptfunktion von HTML ist seit jeher die Strukturierung von Textdokumenten. Das Internet besteht auch heute noch zum größten Teil ganz unspektakulär aus endlosen Seiten von Text. Und damit dieser Text nicht einfach so aussieht

```
Unsere Mission Die Idee, die uns antreibt. IT-Experten sind strukturiert und chaotisch, nüchtern und versponnen, kreativ und rational. Sie denken logisch um die Ecke und begeistern sich für Dinge, die Außenstehende oft schwer verstehen. ITler sind anders. Auch bei der Jobsuche. Deshalb sind wir ein anderer Personalvermittler. Einer, der ITlern wirklich etwas bringt. Alle bei next level sind selbst IT-Menschen und wissen bei jedem Projekt genau, worum es geht. Weil wir Experten sind für die Technologien und Anforderungen, finden IT-Spezialisten bei uns die spannendsten Kunden und die coolsten Jobs. IT-Experten und Unternehmen arbeiten über einen langen Zeitraum eng mit uns zusammen. Wir kennen ihre Stärken, Vorlieben und Traumjobs. So bekommt jeder genau die Infos und Angebote, die ihn wirklich interessieren. Bei uns zählt der Mensch, nicht der Job. Wir glauben, dass wir die beste Personal- und Projektvermittlung sind, die ein ITler sich wünschen kann. Das ist unser Ziel und daran arbeiten wir jeden Tag mit großer Begeisterung.
```

braucht es eine maschinenlesbare Möglichkeit, die Bestandteile dieses Texts, also Überschriften, Absätze, Links, Media-Elemente, als solche zu kennzeichnen.

Idealer Weise steht am Beginn eines HTML-Dokuments der Doctype, eine Zeile Text in der dem verarbeiten Programm, also in der Regel dem

Browser mitgeteilt wird, an welche der diversen HTML-Syntaxen sich das folgende Dokument orientiert. Das sieht dann zum Beispiel so aus:

HTML 4.01 Strict (HTML-Syntax, invalide wenn nicht mehr zulässige HTML-Elemente verwendet werden)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

XHTML 1.0 Transitional (XML-Syntax, valide auch wenn nicht mehr zulässige XHTML-Elemente verwendet werden)

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

HTML5

```
<!DOCTYPE html>
```

Jedes HTML-Dokument sollte in seinen Grundzügen folgenden Aufbau haben:

```
<!DOCTYPE html>
<html> <!-- Das Wurzel-Element der Seite -->
    <head> <!-- Erstes Kind-Element von <html>.
Enthält Meta-Informationen über die aktuelle Seite,
zum Beispiel den Titel, Angaben zur Sprache, Elemente
für die Inhaltszusammenfassung, Schlüsselwörter, CSS-
Style-Informationen -->
    </head>
    <body> <!-- Zweites Kind-Element von <html>. Hier
gehört der Seiteninhalt hinein -->
    </body>
</html>
```

Innerhalb des `<body>`-Elements finden wir den eigentlichen Inhalt der Seite, strukturiert mit Elementen wie diesen:

- `<h1>`, „Heading Level 1“, die Hauptüberschrift; geht hierarchisch absteigend weiter bis zu `<h6>`
- `<p>`, „Paragraph“, repräsentiert einen Textabsatz
- `<a>`, „Anchor“, ein Link zu einer Stelle im aktuellen oder einem anderen Dokument
- `<img>`, „Image”, ein Bild
- `<span>`, ein Abschnitt von Text innerhalb eines Absatzes
- `<div>`, „Division“, ein universell einsetzbarer Container zum Gruppieren anderer Elemente

Mit der Hilfe solcher Elemente wird aus dem eingangs gezeigten Textklumpen so etwas:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Unsere Mission</title>
    </head>
    <body>
    <h1>Unsere Mission</h1>
    <h2>Die Idee, die uns antreibt.</h2>
    <p>IT-Experten sind strukturiert und chaotisch,
nüchtern und versponnen, kreativ und rational. Sie
denken logisch um die Ecke und begeistern sich für
Dinge, die Außenstehende oft schwer verstehen. ITler
sind anders. Auch bei der Jobsuche. Deshalb sind wir
ein anderer Personalvermittler. Einer, der ITlern
wirklich etwas bringt.</p>
    <p>Alle bei next level sind selbst IT-Menschen
und wissen bei jedem Projekt genau, worum es geht.
Weil wir Experten sind für die Technologien und
Anforderungen, finden IT-Spezialisten bei uns die
spannendsten Kunden und die coolsten Jobs.</p>
    <p>IT-Experten und Unternehmen arbeiten über
einen langen Zeitraum eng mit uns zusammen. Wir
kennen ihre Stärken, Vorlieben und Traumjobs. So
bekommt jeder genau die Infos und Angebote, die ihn
wirklich interessieren. Bei uns zählt der Mensch,
nicht der Job.</p>
    <p>Wir glauben, dass wir die beste Personal- und
Projektvermittlung sind, die ein ITler sich wünschen
kann. Das ist unser Ziel und daran arbeiten wir jeden
Tag mit großer Begeisterung.</p>
    </body>
</html>
```

Und das sieht im Browser dann so aus:

CSS in Blitzgeschwindigkeit

Das ist natürlich schön und gut, aber besonders ansehnlich ist die gerade entstandene Seite ja nicht unbedingt. Wie kriegen wir diese Seite also so richtig sexy?

In früheren Versionen von HTML gab es dafür eine Kombination aus HTML-Elementen und -Attributen, die Informationen über die darzustellende Optik enthielten.

Das `<font>`-Element zum Beispiel diente dazu, für einen bestimmten Textabschnitt die Schriftart, -farbe und -größe zu definieren.

Elemente wie `<table>` unterstützten eine ganze Reihe von Attributen, um ihre Optik zu beeinflussen:

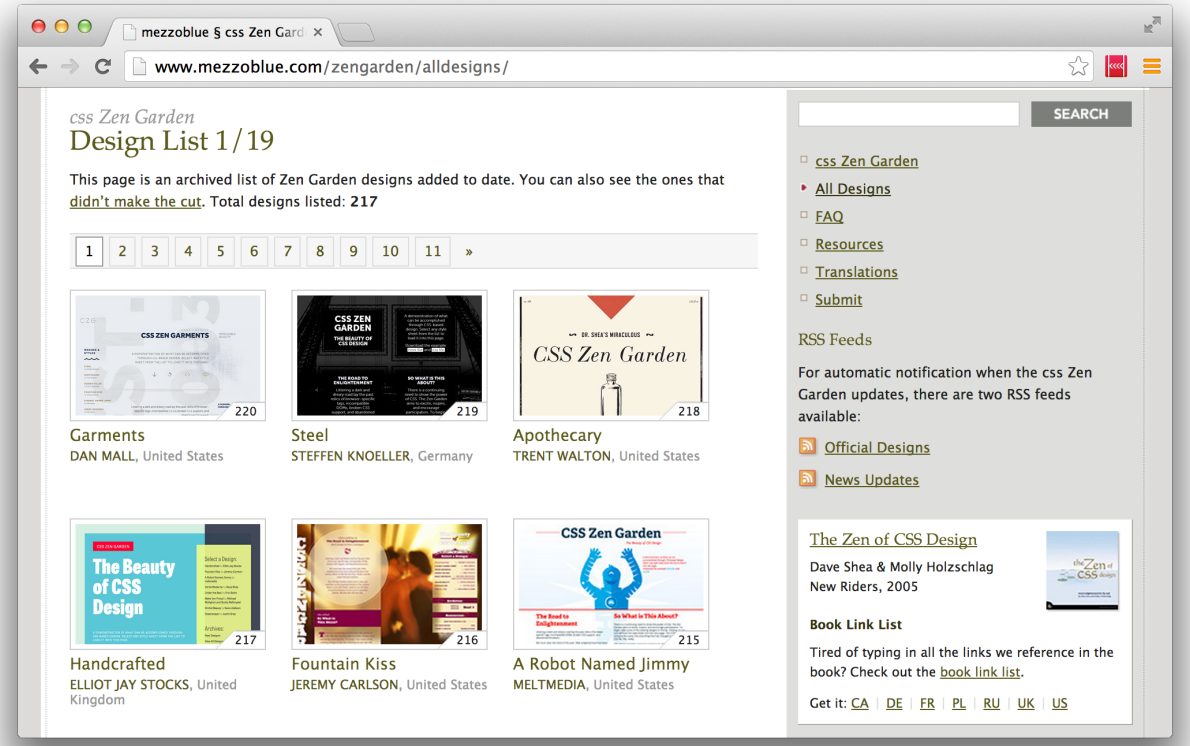
```
<table align="left" bgcolor="#ff0000" border="0"
cellpadding="7" cellspacing="3"></table>
```

Die mit dieser Art des Stylings verbundenen Probleme werden hoffentlich schnell klar. Angenommen unser Chef hat auf seinem Flug von Berlin nach Köln gelesen, dass Verdana eine viel geilere Schrift als Arial ist, so sitzen wir plötzlich den Rest des Wochenendes in unserem IT-Keller und ändern auf allen 700 Unterseiten unserer Webseite die jeweils 30 `<font>`-Elemente, nur um am Montag zu erfahren, dass unsere Tabellen in 2014 natürlich nicht mehr rot unterlegt sind, sondern gefälligst blau. Wir haben ja sonst nix zu tun.

Da nicht alle ITler Masochisten sind, gab es schon früh verschiedenste Ansätze, das Styling von strukturierten Dokumenten zentral zu steuern. Einer dieser Ansätze entwickelte sich im Laufe der Zeit zu CSS Level 1, dessen finale Spezifikation 1996 vom W3C veröffentlicht wurde.

Mit CSS hat ein Entwickler die Möglichkeit, seine Seiteninhalte in separaten Dateien (oder über das `<style>`-Element direkt ins HTML eingebettet) optisch zu gestalten.

Vermutlich immer noch das beste Beispiel für die Macht von CSS ist die Seite csszengarden.com. Eine simple HTML-Seite, für jeden Besucher frei zum Download verfügbar, die durch von Usern erstellte Stylesheets die unterschiedlichsten, teilweise sehr ausgefallene Looks erhält.



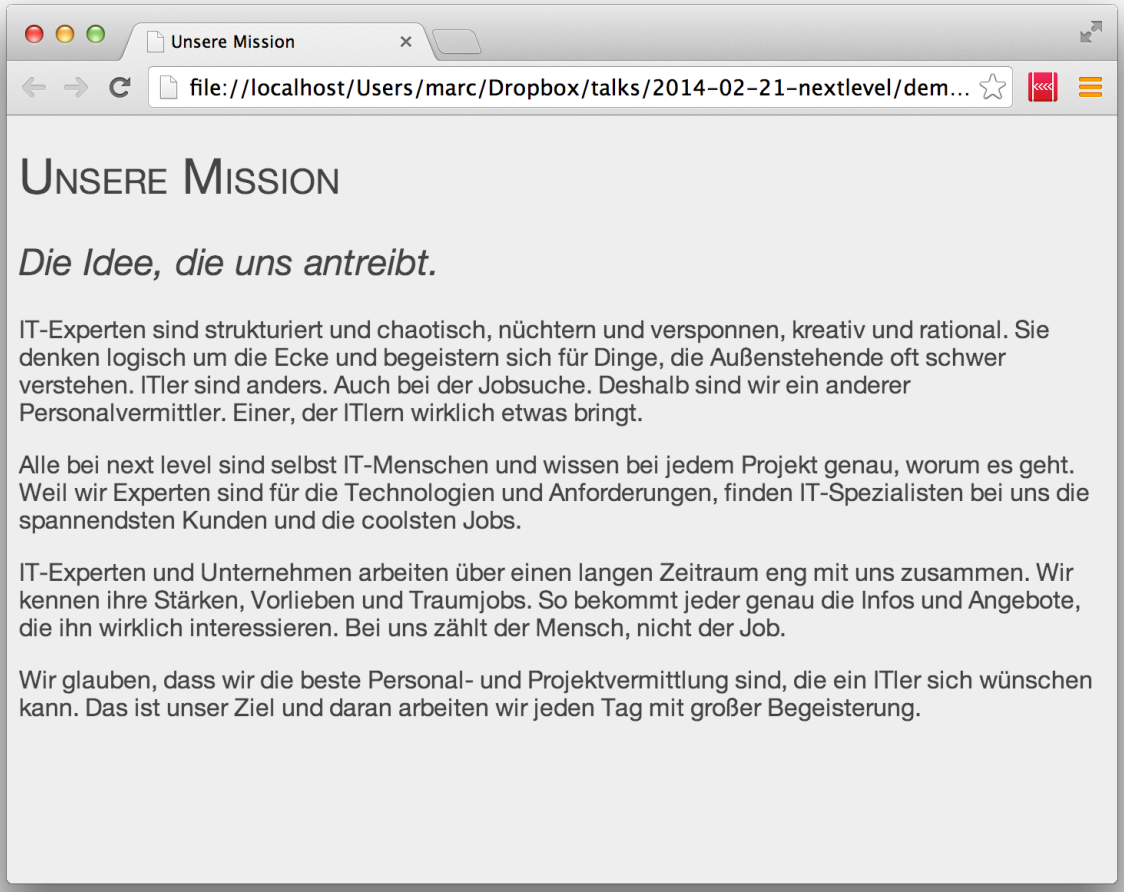
Wir ergänzen unsere Beispielseite aus dem HTML-Kapitel um ein paar Zeilen, um die Optik der Seite ein wenig zu verändern.

```
<head> <link rel="stylesheet" type="text/css"
href="nextlevel.css" /> <!-- Hier binden wir über ein
<link>-Element das Stylesheet ein --> </head>
```

In der Datei nextlevel.css legen wir ein paar Style-Anweisungen fest:

```
html, body {
    background-color: #eeeeee;
    color: #444444;
    font-family: 'Helvetica Neue', 'Arial', sans-
serif;
}

h1, h2 {
    font-weight: normal;
}

h1 {
    font-variant: small-caps;
}

h2 {
    font-style: italic;
}
```

Im Browser sieht das so aus:



JavaScript in Blitzgeschwindigkeit

JavaScript hat seine Anfänge Mitte der 90er Jahre bei Netscape. Sind HTML und CSS noch statische Angelegenheiten („so ist das Dokument aufgebaut", und „so wird das Dokument dargestellt"), ist JavaScript eine echte Programmiersprache, mit der sich bei Bedarf auch komplexe Applikationen schreiben lassen.

Ein zentrales Element ist hierbei das „Document Object Model" (DOM). Hierin findet sich die Struktur des aktuellen HTML-Dokuments als in JavaScript verarbeitbares Objekt wieder. Einzelne Elemente des Dokuments können abgefragt und in ihren Attributen verändert werden:

```
document.getElementById('content').className =
'meinContent';
```

Zusätzlich können über Event-Handler Ereignisse abgefragt werden. So kann das Skript z.B. unterschiedlich reagieren, wenn der Benutzer den Mauszeiger über ein Element bewegt, klickt, oder ein Formular abschickt.

```
// alle Elemente vom Typ <p> durchlaufen
for(
     var allePTags = 0;
     allePTags < document.getElementsByTagName('p').length;

     allePTags++) {
     document.getElementsByTagName('p')[allePTags].className =
'closed'; // <p>-Elementen "geschlossen"-Klasse zuweisen
     document.getElementsByTagName('p')[allePTags].onclick =
function(){ // Beim Klick auf ein <p>-Element, auf/zu per Klasse
regeln
          if(this.className == 'open') { this.className =
'closed'; }
          else { this.className = 'open'; }
     };
}
```

Lassen sich viele Arbeiten gut mit reinem JavaScript erledigen, sind im Laufe der Jahre auch immer mehr Helfer-Bibliotheken wie jQuery, Prototype oder MooTools entstanden, die zum Beispiel mächtigere und flexiblere Möglichkeiten zum Finden von Elementen und die schmerzlose Umsetzung komplexer Manipulationen bis hin zu Animationen bieten.

Abseits der Ergänzung von Funktionalität auf ansonsten statischen Seiten, wird JavaScript für die unterschiedlichsten Zwecke eingesetzt. Der Less-CSS-Präprozessor bietet einen JavaScript-Helfer, der übergebenes Less auf Browserseite verarbeitet und in CSS umwandelt. Mit Modernizr können Entwickler den Browser auf das Vorhandensein einer Vielzahl von Features prüfen. Selectivizr durchsucht Stylesheets nach im aktuellen Browser nicht unterstützten Selektoren und empfindet diese per JavaScript nach.

Und zu guter Letzt ist JavaScript in letzter Zeit mehr und mehr die Sprache der Wahl in serverseitigen Anwendungen geworden, z.B. für den Server node.js oder als Scripting-Sprache für die Datenbank Apache CouchDB.

Semantisches HTML

Da der grundlegende Sinn von HTML wie bereits erwähnt die Strukturierung von Textdaten ist, ergibt sich daraus auch die Bedeutung des Begriffs „semantisches HTML“. Schließlich hat jedes HTML-Element seine eigene Bedeutung. Im Prinzip wäre es auch möglich, ein Dokument fast nur mit `<p>`-Elementen aufzubauen, diese mit Klassen zu versehen und per CSS in jede Gestalt zu zwingen, aber das ist natürlich nicht Sinn der Sache.

Wenn ich ein Dokument habe, dann packe ich seinen Titel in ein `<h1>`, den Untertitel in eine `<h2>`. Tiefer gehende Abschnitte erhalten entsprechend ihrer Verschachtelungstiefe `<h3>` bis `<h6>`-Überschriften. Absätze kommen in `<p>`-Elemente. Wenn ich eine Textpassage hervorheben will, tue ich dies mit dem `<em>`-Element (Emphasis), geht es um eine besonders starke Hervorhebung nutze ich `<strong>` (*wichtig* versus **wirklich** wichtig!).

Mit HTML5 stehen Entwicklern zusätzliche Elemente zur Verfügung, wie etwa `<article>`. Damit wird zum Beispiel angezeigt „hier drin befindet sich ein in sich abgeschlossener irgendwie gearteter Textbeitrag“.

Andere Anwendungsfälle von semantischer Textauszeichnung finden sich im Bereich der „Microformats“. Diese definieren Standard-Klassen, über die in normalem HTML nicht klar abbildbare Strukturen beschrieben werden können. Ein Beispiel hierfür ist das hCard-Format. In HTML gibt es keine eindeutige Methode, eine digitale Visitenkarte abzubilden. Unter Verwendung der hCard-Klassen wird dies möglich:

```
<div class="vcard">
    <a class="fn org url" href="http://
www.commerce.net/">CommerceNet</a>
    <div class="adr">
        <span class="type">Work</span>:
        <div class="street-address">169 University
Avenue</div>
        <span class="locality">Palo Alto</span>,
<abbr class="region" title="California">CA</abbr>
        <span class="postal-code">94301</span>
        <div class="country-name">USA</div>
    </div>
    <div class="tel">
    <span class="type">Work</span> +1-650-289-4040
    </div>
    <div class="tel">
        <span class="type">Fax</span> +1-650-289-4041
    </div>
    <div>Email: <span
class="email">info@commerce.net</span></div>
</div>
```

Die so formatierten Daten wären nun z.B. für ein entsprechendes Browser-Plugin problemlos identifizierbar und entsprechend darstellbar.

HTML5

HTML5 ist grob seit 2007 in Arbeit (das politische Hick-Hack von W3C/WHATWG, HTML5 und XHTML 2.0 soll hier mal außen vor bleiben) mit dem Ziel, 2014 endgültig als Standard verabschiedet zu werden.

HTML5 lässt sich grob aufteilen in Änderungen an der Auszeichnungssprache HTML selbst, sowie in die zahlreichen neuen APIs, die Entwicklern zur Verfügung stehen sollen.

Orientierten sich die ursprünglichen HTML-Elemente noch stark an der Idee, der Strukturierung von vernetzten Textdokumenten zu dienen, führt HTML5 viele Elemente ein, die wirklich im Hinblick auf ihren Einsatz in Webseiten konzipiert wurden, z.B.:

- `<section>`: Repräsentiert einen beliebigen Inhaltsabschnitt in einem Dokument oder einer Anwendung
- `<article>`: Ein selbständiges Stück Inhalt in einem Dokument, z.B. ein Blogeintrag
- `<header>`: Gruppierung von einleitenden oder der Navigation dienenden Informationen
- `<nav>`: Ein Bereich, der Navigationselemente enthält
- `<video>, <audio>`: Multimedia-Elemente, mit Browser-internem Player und einer API für Player von Drittanbietern
- `<canvas>`: „Leinwand” zur dynamischen Erzeugung von Grafiken, z.B. für Graphen oder Animationen
- `<main>`: Der Haupt-Seiteninhalt

Ebenfalls neu sind zusätzliche Arten von Eingabefeldern, z.B. zur Auswahl von Datumswerten oder Farben, oder die simple Unterscheidung zwischen einem Textfeld und einem Textfeld für E-Mail-Adressen. Das macht sich z.B. auf Smartphones durch die Anzeige spezialisierter Tastaturen bemerkbar.

Andere Elemente und Attribute, von deren Verwendung bereits seit HTML 4.01 abgeraten wurde, werden in HTML5 nicht mehr unterstützt.

Dies betrifft insbesondere darstellende Elemente wie `<font>`, `<big>` oder `<center>`.

Darüber hinaus beinhaltet HTML5 Spezifikationen für zahlreiche Entwicklerschnittstellen:

- canvas API, um `<canvas>`-Elemente per JavaScript zu steuern
- Drag-and-drop
- Cross-document messaging
- Browser history management
- Web Storage: Ein Key-Value Speicher ähnlich Cookies, mit größerem Speichervermögen und verbesserter API
- Web Workers: Hintergrundprozesse für HTML-Anwendungen.
- Geolocation
- File API: Asynchrone Drag & Drop Dateitransfers vom Desktop ohne vorgeschalteten Upload.

CSS 3 und mehr

Seit der Veröffentlichung der CSS Level 1 Empfehlungen 1996 wurde die Sprache beständig weiterentwickelt. Die aktuell verabschiedeten Standards sind Level 1, Level 2 und eine Revision von Level 2.

Seit dem Jahr 2000 läuft die Arbeit an CSS Level 3 (CSS3). Dabei wird CSS3 nicht als in sich geschlossenes System entwickelt. Vielmehr wurde CSS3 in mehrere Untermodule aufgesplittet, die voneinander unabhängig entwickelt werden. Diese Entscheidung führte auch dazu, dass das W3C 2012 bekanntgab, dass es kein CSS4 als solches geben wird, sondern dass vielmehr jedes einzelne Modul irgendwann seine ganz eigene nächste 4. Entwicklungsstufe erreichen wird.

Unter der Überschrift „CSS3“ existiert eine Vielzahl von Modulen, von denen lediglich vier den Status „Recommendation“ haben, mit einigen weiteren Modulen in der Vorstufe zur „Recommendation“.

Recommendations, also im Entwurf abgeschlossen und so von Browserherstellern vorbehaltlos umsetzbar sind diese Module:

- Media queries
- Selectors Level 3
- Color Level 3
- Namespaces

Weitere, noch nicht finalisierte Level 3 Features sind:

- Übergänge & Animationen
- Transformationen in 2D und 3D
- Verwendung von Schriften per @font-face
- CSS-Layouts

Media queries

Media queries ermöglichen es, bestimmte Styles nur bei Erfüllen spezifizierter Umstände gelten zu lassen.

```
@media screen and (max-width: 960px) {
    /* alle Anweisungen in diesem Block werden nur
auf Bildschirmen in einem Browserfenster von bis zu
960 Pixeln Breite angewendet */
}
```

Media queries können gezielt verschiedene Medien-Typen wie Bildschirme, Ausdrucke oder Blindenschrift-Displays ansteuern und zusätzlich auf Parameter wie die verfügbare Anzeigefläche im Browser oder auf dem Display an sich, die Pixeldichte oder die Farb-Unterstützung des Displays beschränkt werden.

Selectors Level 3

- `E[att^="val"]`: Findet E-Elemente, deren att-Attribut mit „val" anfängt.
- `E[att$="val"]`: Findet E-Elemente, deren att-Attribut auf „val" endet.
- `E[att*="val"]`: Findet E-Elemente, deren att-Attribut „val" enthält.
- `E:nth-child(n)`: Findet E-Elemente, die das nte Kind ihres Elternelements sind.
- `E:first-of-type`: Findet E-Elemente, die in ihrer Umgebung das Element ihres Typs sind.

Color Level 3

Das Color Level 3 Modul bietet insbesondere neue Möglichkeiten Farbwerte zu definieren.

`rgba` ermöglicht zum Beispiel die Definition eines Rot/Grün/Blau-Farbwertes mit einem zusätzlichen Wert für Alpha-Transparenz:

`rgba(255, 0, 0, .5)` entspräche einem halb-transparenten Rot.

Außerdem wurde die Liste der Schlüsselwörter für Farbwerte extrem erweitert, so dass wir statt

```
color: #8A2BE2;
```

ebenso gut

```
color: blueviolet;
```

schreiben können.

Namespaces

In XML-verwandten Auszeichnungssprachen besteht die Möglichkeit, ein Dokument oder auch Abschnitte eines Dokuments einem Namensraum zuzuweisen. D.h. alle Elemente unterhalb der Zuweisung haben die Bedeutung, die ihnen in diesem Namensraum zugewiesen ist. HTML5 bietet zum Beispiel Unterstützung für MathML, eine Sprache zur Formatierung mathematischer Formeln. Innerhalb eines HTML-Dokuments kann uns also plötzlich solch ein Block begegnen:

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
 <mrow>
  <mi>a</mi>
  <mo>&InvisibleTimes;</mo>
  <msup>
   <mi>x</mi>
   <mn>2</mn>
  </msup>
  <mo>+</mo>
  <mi>b</mi>
  <mo>&InvisibleTimes; </mo>
  <mi>x</mi>
  <mo>+</mo>
  <mi>c</mi>
 </mrow>
</math>
```

Sind die Elemente in MathML noch alle eindeuting anders als in HTML benannt, gibt es leider nichts was verhindert, dass wir irgendwann mit einer anderen Syntax zu tun haben, nennen wir sie mal UnfugML, in der z.B. dem `<a>`-Element ein ganz anderer Zweck gegeben wird. Bis zur Einführung von CSS-Namespaces wäre dieses UnfugML-`<a>` wie HTML-`<a>`s dargestellt worden. Dank CSS-Namespaces können wir am Anfang eines separaten Stylesheet aber folgendes sagen:

```
@namespace 'UnfugML';
```

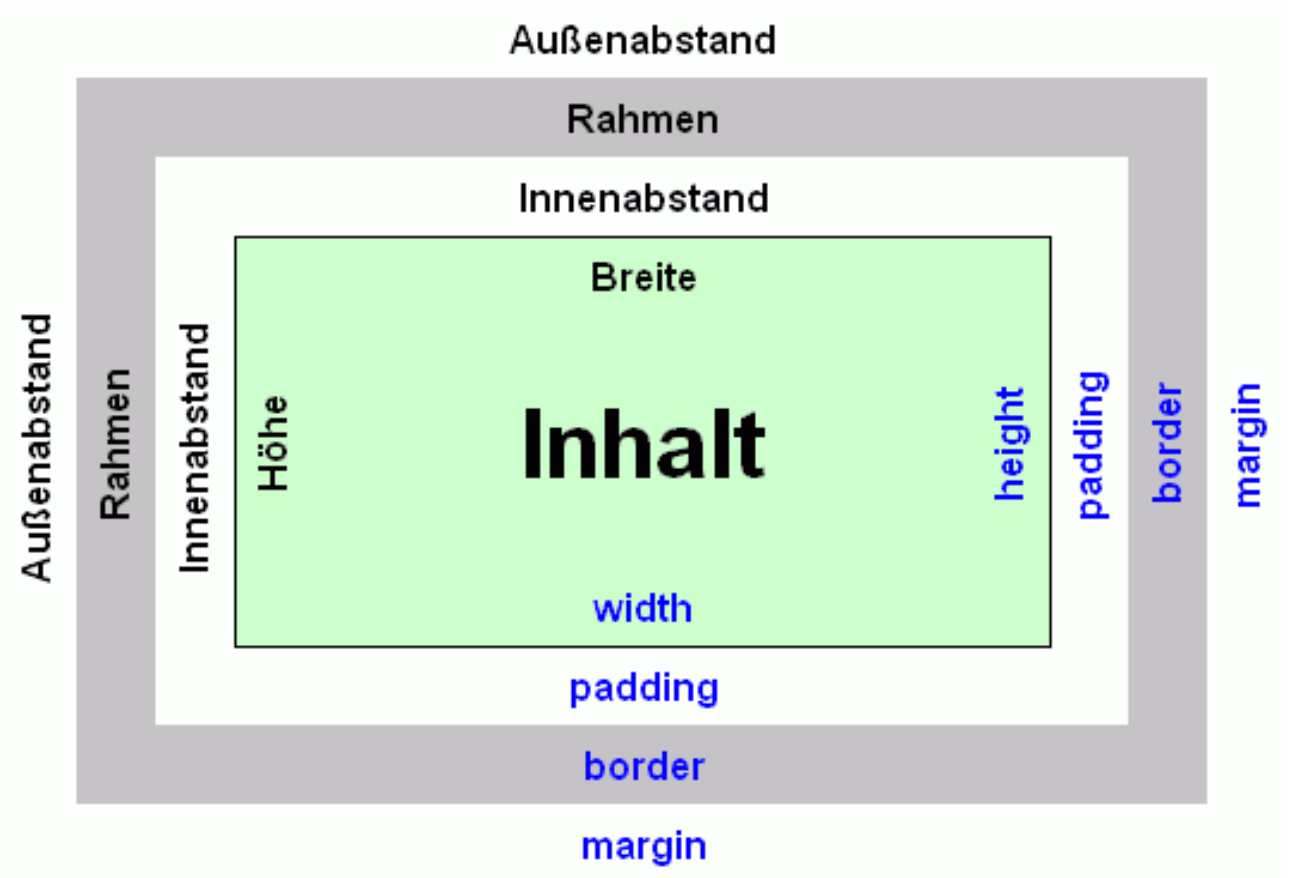
Alle folgenden Styles gelten nun nur für Elemente, die in einem Eltern-Element mit `xmlns="UnfugML"` erscheinen.

Für Fortgeschrittene

Im Browser

CSS-Floats

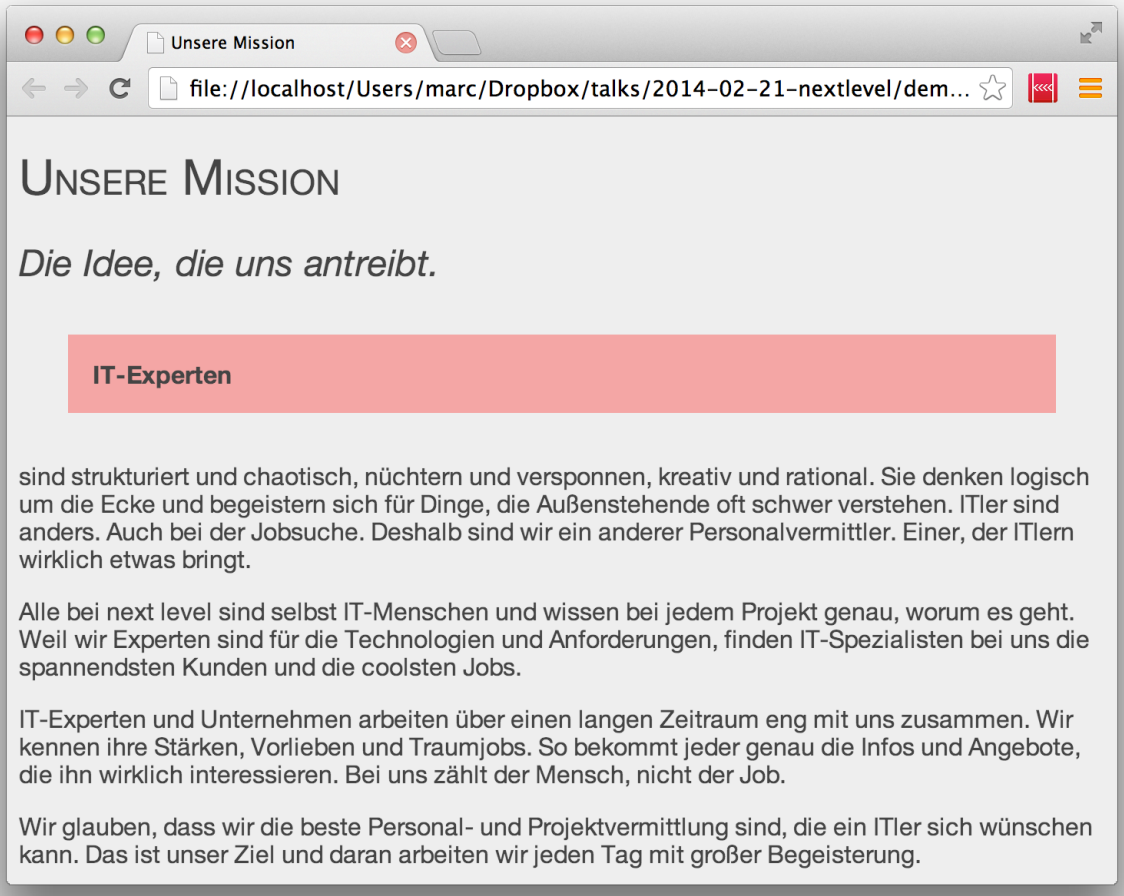
Jedes HTML-Element ist von einem Rechteck umgeben. Die einzelnen Bestandteile dieses Rechtecks (eigentlicher Inhalt, Innenabstand, Rahmen, Außenabstand) sind im sogenannten Box-Model definiert.



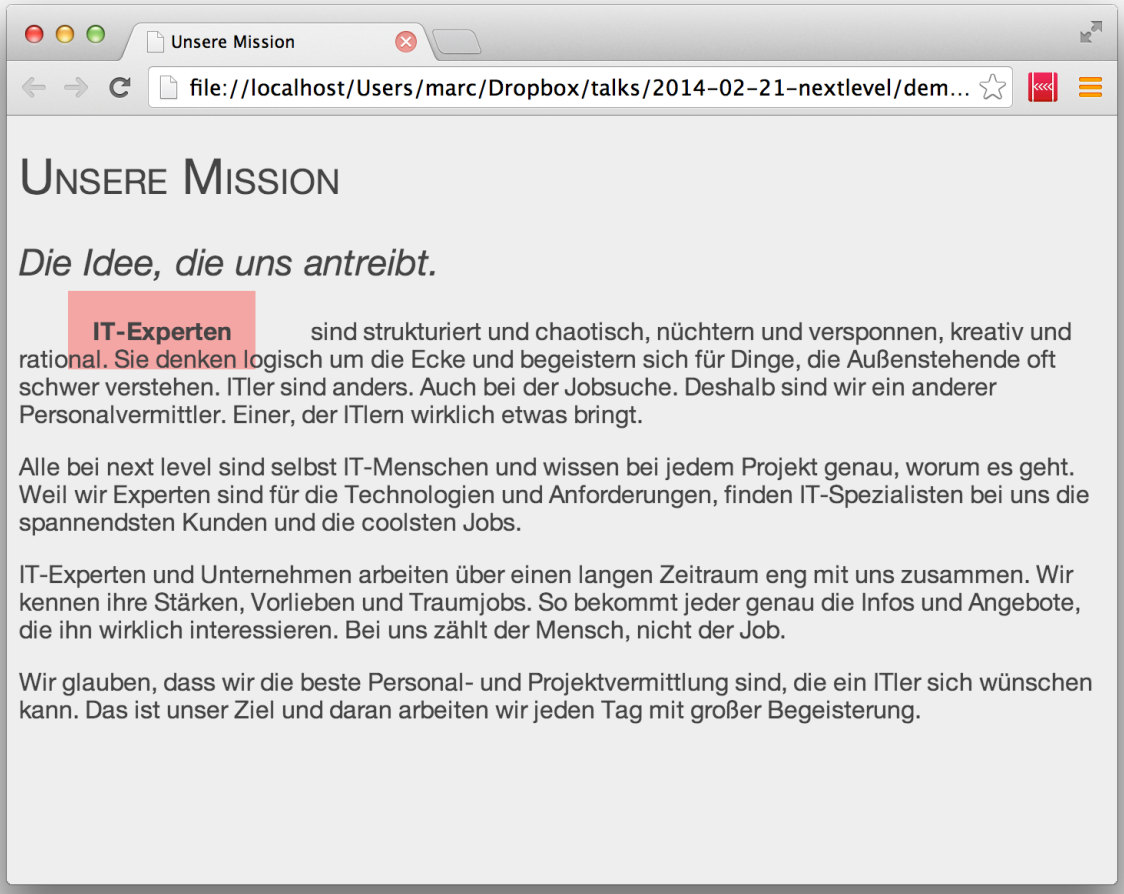
Quelle: SelfHTML

Über die CSS-Eigenschaft `display` wird definiert, wie die Box eines Elements sich in Bezug auf ihre Umgebung verhalten soll.

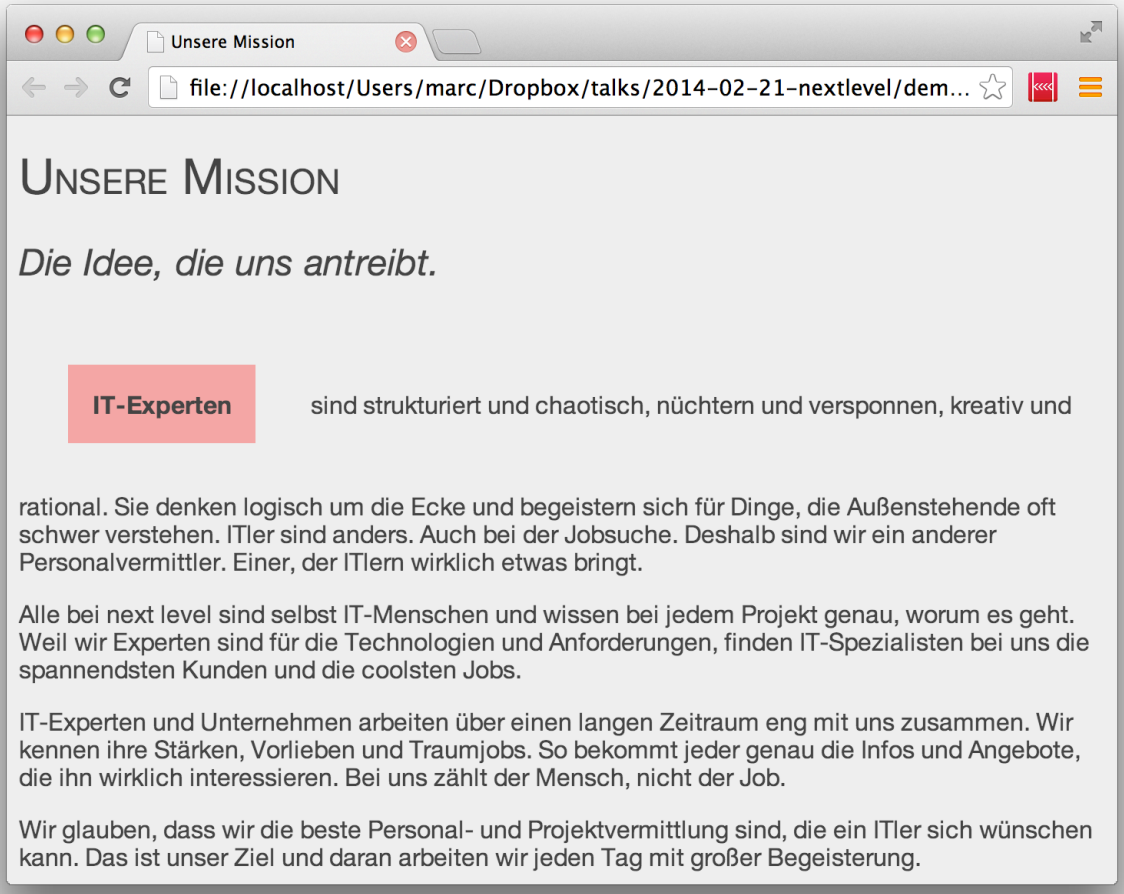
`display: block` sorgt dafür, dass das Element einen Absatz erzeugt. Das heißt zwischen dem aktuellen und dem folgenden Element wird ein Zeilenumbruch erzeugt. Innen- und Außenabstände werden horizontal wie vertikal berücksichtigt.



`display: inline` lässt das Element dem umgebenden Textfluss folgen. Dies hat auch zur Folge, dass nur horizontale Abstände berücksichtigt werden.



Die etwas neuere (CSS 2.1) Deklaration `display: inline-block` bildet eine Kombination aus den beiden vorherigen Werten, indem das Element zwar dem umgebenden Textfluss folgt, gleichzeitig aber alle Abstände berücksichtigt werden, als handelte es sich um ein Block-Element.



Häufig kommen wir in Szenarien, in denen ein Element wie ein Block formatiert werden soll, gleichzeitig aber innerhalb des Textflusses nach einer Seite hin ausgerichtet sein soll. Das klassische Beispiel ist hier ein Bild, das Abstände nach außen haben und in einem Textabsatz rechts in der Ecke sitzen soll:

Hier kommt die CSS-Eigenschaft `float` zum Einsatz. Sie ermöglicht es, Elemente aus dem normalen Textfluss zu lösen und sie links oder rechts „schweben“ zu lassen. Der umgebende Text umfließt dieses schwebende Element daraufhin.

Der Einsatz von `float` ist auch beim Layout von Seiten gebräuchlich. Früher wurden Seiten häufig mit Hilfe von Tabellen strukturiert. Gegen den Einsatz von Tabellen für Layouts spricht natürlich, dass wir hier ein Element für einen ganz anderen als den gedachten Zweck missbrauchen. Tabellen sind für die Formatierung tabellarischer Daten

da, sonst nichts. Zudem sind Tabellen zwar komfortabel aufzubauen, gleichzeitig aber sehr starr in ihrer Aufteilung.

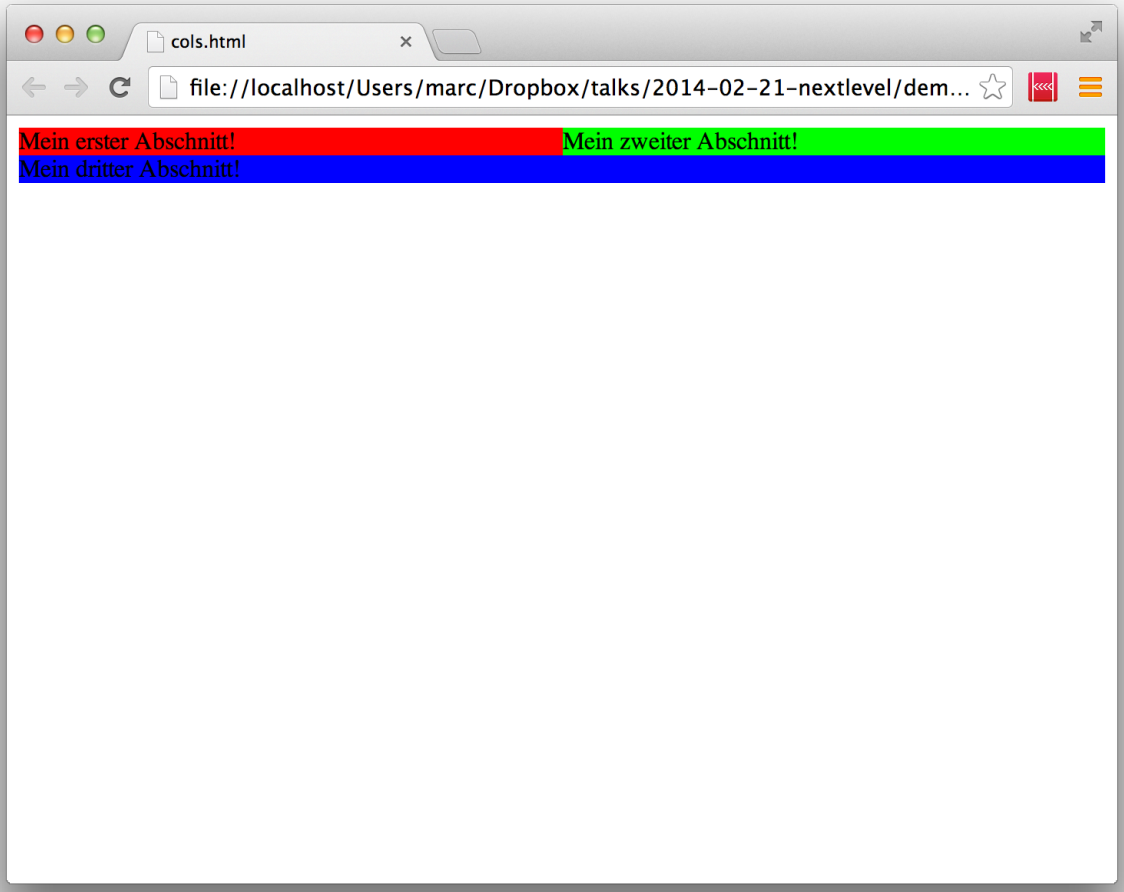
Mit `float`enden Elementen sind wir hier flexibler. Angenommen wir haben diese Elemente:

```
<div id="eins">Mein erster Abschnitt!</div>
<div id="zwo">Mein zweiter Abschnitt!</div>
<div id="drei">Mein dritter Abschnitt!</div>
```

Mit Hilfe von `float` können wir diese Elemente bequem layouten.

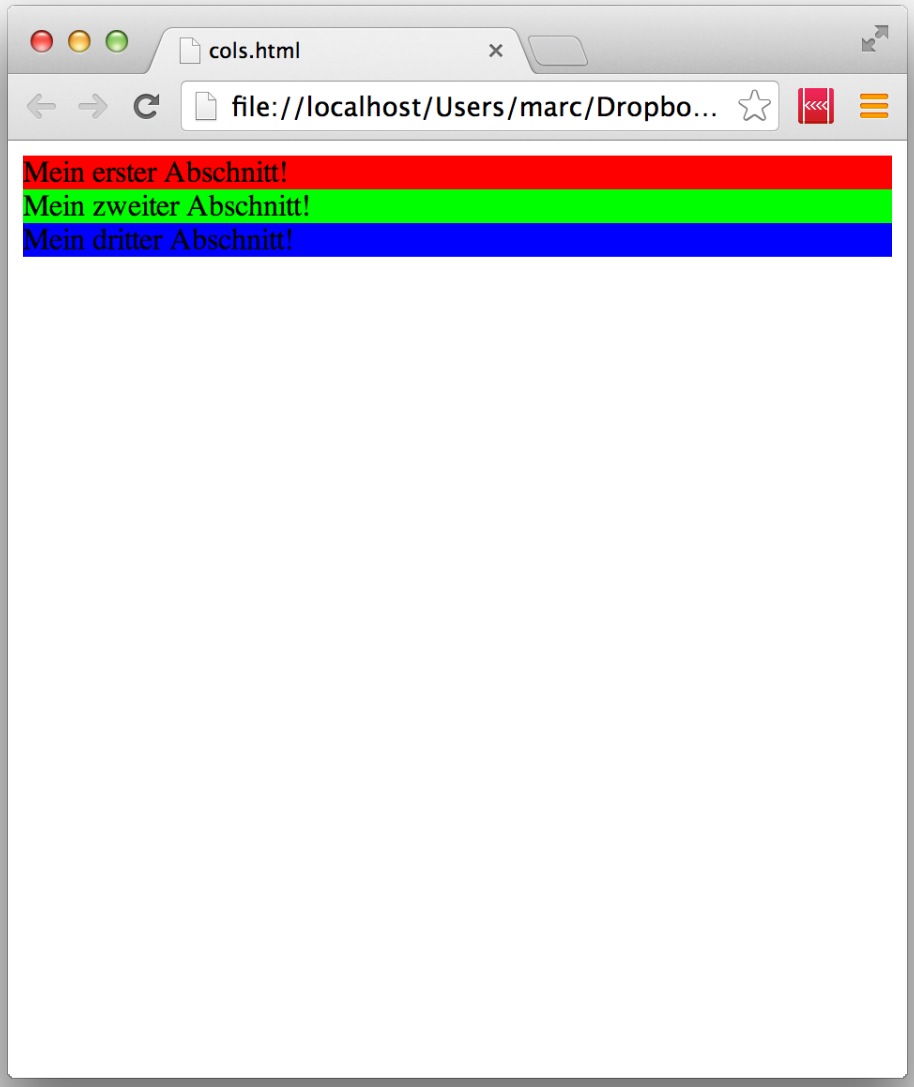
Bauen wir uns ein einfaches Zwei-Spalten-Layout:

```
<style>
    #eins {
        background-color: #ff0000;
    }

    #zwo {
        background-color: #00ff00;
    }

    #drei {
        background-color: #0000ff;
        clear: both;
    }

    div#eins, div#zwo {
        float: left;
        width: 50%;
    }
</style>
```



Ergänzend eine Media query, um ab einer bestimmten Breite jede Spalte in eine eigene Zeile in voller Breite umspringen zu lassen, weil wir alle kleine Bildschirme in der Tasche haben:

```
@media screen and (max-width: 480px) {
    div#eins, div#zwo {
        float: none;
        width: auto;
    }
}
```



Diese Lösung ist natürlich schon und gut und funktioniert auch seit Jahren in den meisten Fällen. Aber letztendlich wird auch hier natürlich lediglich eine Eigenschaft zum Ausrichten von Elementen im Textfluss zweckentfremdet, um damit ganze Seiten zu layouten.

Dieses Problem will das u.a. Flexbox-Modul lösen, das seit Kurzem in allen aktuellen Browserversionen unterstützt wird (und somit in 5 bis 10 Jahren so richtig echt in Kundenprojekten einsetzbar sein wird).

Unsere Elemente strukturieren wir nun wie folgt:

```
<div class="flex-container">
    <div id="eins">Mein erster Abschnitt!</div>
    <div id="zwo">Mein zweiter Abschnitt!</div>
</div>
<div id="drei">Mein dritter Abschnitt!</div>
```

Nachdem wir dem Container gesagt haben, dass er Flexbox anwenden soll, können wir seinen Kindern relative Breiten mitgeben:

```
.flex-container {
    display: -webkit-flex;
}

div#eins {
    -webkit-flex: 2;
    flex: 2;
}

div#zwo {
    -webkit-flex: 1;
    flex: 1;
}
```

Die Breite des Containers wird basierend auf den `flex`-Werten entsprechend aufgeteilt.

CSS-Präprozessoren

Seit ein paar Jahren groß im Kommen sind CSS-Präprozessoren. Erwähnenswert sind hier Less, Sass und Stylus. All diesen Tools gemein ist, dass nicht mehr reines CSS geschrieben wird. Vielmehr schreibt der Entwickler eine Datei in einer dem jeweiligen Präprozessor eigenen Syntax. Diese wird dann vor der Auslieferung an den Browser von einem Tool auf der Kommandozeile, einem Script im Browser oder auf dem Server in CSS umgewandelt.

Was ist an Präprozessoren jetzt so toll? Auf großen Seiten kann CSS schnell ausufern. Selektoren können ewig lang werden. Viel zu viele Elemente teilen sich gemeinsame Styles, die sich nun immer wieder wiederholen. Werte, z.B. Farben, die aufeinander abgestimmt sind, müssen bei Änderung des Grundwertes neu berechnet und eingetragen werden, gerne auch wieder an 50 Stellen im Code.

Präprozessoren können hier sehr bei der Übersichtlichkeit des zu bearbeitenden Codes helfen. Sie bieten Features wie **verschachtelte Regeln**:

```
header {
    color: black;
    .navigation {
        font-size: 12px;
    }
    .logo {
        width: 300px;
    }
}
```

wird zu

```
header {
    color: black;
}
header .navigation {
    font-size: 12px;
}
header .logo {
    width: 300px;
}
```

Variablen:

```
$text-color: #444444;

body {
    color: $text-color;
}

a {
    color: $text-color;
}
```

wird zu

```
body {
    color: #444444;
}

a {
    color: #444444;
}
```

Mixins:

```
.demo-box() {
    border: 1px solid #000000;
    // noch mehr Zeug
}
sidebar {
    .box {
        .demo-box();
    }
}
footer {
    .node-otherthing {
        .demo-box();
    }
}
```

wird zu

```
sidebar .box {
    border: 1px solid #000000;
    // noch mehr Zeug
}
footer .node-otherthing {
    border: 1px solid #000000;
    // noch mehr Zeug
}
```

Berechnungen und Helfer aller Art:

```
a:hover {
    color: lighten(10%, $text-color);
    padding: 13px + 0.5em;
}
```

wird zu

```
a:hover {
    color: #5E5E5E;
    padding: 19px;
}
```

Diese und andere Features von CSS-Präprozessoren (im Fall dieser Beispiele Less) bieten dem Developer ein sehr mächtiges Werkzeug, um den an sich starren CSS-Code dynamischer und effizienter zu entwickeln.

AJAX

Klassische Webseiten werden seit jeher synchron abgerufen und angezeigt. Wir geben eine Adresse an, der Browser fragt die Seite beim Server an, erhält eine Antwort und stellt die erhaltenen Informationen dar.

Mal ehrlich, das ist doch langweilig, oder?

Klar, aber deswegen gibt es ja AJAX (Asynchronous JavaScript and XML). AJAX bietet Entwicklern die Möglichkeit, Informationen vom Server abzurufen, ohne dafür die Seite komplett neu laden zu müssen. So werden zum Beispiel solche Szenarien möglich:

- wir haben einen Newsticker auf unserer Seite, der live mit aktuellen Meldungen gefüllt wird
- wir bauen uns einen Chat
- Facebook (wer es nicht kennt, kann auf Bing danach suchen) lädt den Großteil seiner Seitenbestandteile erst nach und nach, z.B. den Newsfeed und die Ticker- / Chat-Kontakte-Spalte

Und wie geht das nun?

Bereits seit Version 5 des geschätzten Internet Explorers existiert das XMLHttpRequest-Objekt in JavaScript. Andere Browserhersteller haben dieses Feature seinerzeit wegen extremer Nützlichkeit schnell nachgebaut. Dieses Objekt kann im Code einer JavaScript-Anwendung konfiguriert werden, und schickt in der Folge eine Anfrage an den Server. Je nach erhaltener Antwort kann das Skript dann entsprechend reagieren:

```
var xmlHttp = null;
try {
    // Mozilla, Opera, Safari sowie Internet Explorer
(ab v7)
    xmlHttp = new XMLHttpRequest();
} catch(e) {
    try {
        // MS Internet Explorer (ab v6)
        xmlHttp = new
ActiveXObject("Microsoft.XMLHTTP");
    } catch(e) {
        try {
            // MS Internet Explorer (ab v5)
            xmlHttp = new
ActiveXObject("Msxml2.XMLHTTP");
        } catch(e) {
            xmlHttp = null;
        }
    }
}

if (xmlHttp) {
    xmlHttp.open('GET', 'beispiel.xml', true);
    xmlHttp.onreadystatechange = function () {
        if (xmlHttp.readyState == 4) {
            alert(xmlHttp.responseText);
        }
    };
    xmlHttp.send(null);
}
```

Das ist natürlich eine ganze Menge Code für eine simple und auch recht häufig gebrauchte Anfrage. Außerdem müssen wir natürlich auch entsprechend reagieren, wenn der `readyState` andere Werte annimmt. Deswegen haben JavaScript-Libraries wie jQuery fertige AJAX-Helfer-Funktionen an Bord, die das inzwischen alltäglich gewordene Absenden asynchroner Abfragen bedeutend erleichtern:

```
$.get('beispiel.xml', function(data){
    alert(data);
});
```

Auf dem Server

Front-End / Back-End

Da im Laufe der Jahre die Fähigkeiten der geläufigen Browser immer weiter angewachsen sind, hat sich auch die Struktur vieler moderner Web-Anwendungen verändert.

Die klassische Web-basierte Anwendung oder auch eine normale Webseite ist meist nach einem ähnlichen Schema konzipiert:

- Daten und Programmlogik liegen auf dem Server
- Anfragen werden auf Server z.B. in PHP verarbeitet
- HTML-Seiten werden gebaut
- HTML wird an Browser geschickt
- Browser fragt verlinktes CSS, JS, Medien ab
- fertig ist die Seite

Moderne Anwendungen können inzwischen in großen Teilen oder sogar komplett Client-seitig existieren, der Server wird zum reinen Daten-Lieferanten:

- Server liefert eine fast leere HTML-Seite aus
- CSS und JavaScript werden abgerufen
- JavaScript enthält den Großteil der Anwendungslogik
- JavaScript fragt vom Server Daten ab
- JavaScript injiziert Templates für Seitenelemente

Dank HTML5-Features wie Datenbanken im Browser können eines Tages theoretisch sogar die Anfragen in Richtung des Servers minimiert werden, indem wir die Datenbank einmal beim initialen Aufruf laden und danach auf die im Browser gelagerten Daten zurückgreifen. Damit wäre unsere Anwendung, solange die Daten aktuell sind, unabhängig von der Qualität des verfügbaren Netzwerks.

Was hier letztendlich aber gesagt sein soll ist, dass es ganz auf die jeweilige Anwendung ankommt, wo das Front-End und das Back-End ihre Grenzen haben.

Welche Technologien wo?

Grundsätzlich lässt sich sagen, dass in der Theorie so ziemlich jede Programmiersprache im Front- wie im Back-End laufen *könnte*, solange die entsprechende Laufzeitumgebung vorhanden ist. Wenn irgendwer bei Microsoft einen Hitzschlag erleidet und Internet Explorer 12 mit einer Umgebung für COBOL ausliefert, ist COBOL schlagartig eine Front-End-Sprache.

Client- wie Server-seitig haben sich im Laufe der Zeit einige Sprachen als Standards etabliert. Im Browser ist dies eigentlich nur JavaScript mit nativer Unterstützung in allen relevanten Browsern. Flash / Actionscript und Java sind zwar auch vertreten, benötigen aber Plugins und sind gerade im mobilen Web letztendlich schon irrelevant.

Auf dem Server ist die Vielfalt deutlich größer, hier sind unter anderem verbreitet:

- PHP
- Java
- C# in ASP.NET
- Ruby
- Perl
- JavaScript

Moment, JavaScript? Auf dem Server?

Ja, im Jahr 2009 wurde die erste Version der Node.js-Plattform veröffentlicht. Node.js nutzt die von Google ursprünglich für den Chrome-Browser entwickelte V8 JavaScript-Engine, um seinen Code auszuführen.

Die Architektur von Node.js ist darauf ausgelegt, Anfragen und Antworten möglichst schnell vom zentralen Server-Prozess fort zu delegieren. So ist die Server-Anwendung schneller wieder ansprechbar als in anderen Architekturen, in denen eine Anfrage einen kompletten Prozess belegt, bis sie komplett beantwortet ist.

Zudem existieren im Umfeld von Node.js zahlreiche Implementierungen von WebSockets. WebSockets ist grob gesagt eine Standleitung vom Browser zum Server. War es bisher so, dass eine Anwendung, die aktuelle Daten vom Server wollte, regelmäßig AJAX-Anfragen stellen musste, so bieten WebSockets die Möglichkeit, dass der Server bei Änderungen entsprechende Informationen direkt an verbundene Clients pusht.

CMS für HTML-Schubser

Wie wichtig sind Contentmanagement-Systeme in der Frontend Entwicklung […]?

Content Management Systeme (CMS) sind natürlich sehr wichtig. Viele Webseiten basieren auf CMS wie Wordpress, TYPO3, Drupal etc.

Front-End-Entwickler sollten sich in den in ihrem Umfeld gebräuchlichen CMS gut auskennen. Schließlich sind sie in gewisser Hinsicht dem jeweiligen System ausgeliefert, bzw. dem Output, den das CMS produziert.

Das heißt auch, es liegt oft in der Verantwortung des Entwicklers zu erkennen, ob ein vorgeschlagenes Design so umsetzbar ist, oder ob irgendeine Eigenart von Drupal verhindert, dass wir bestimmte Elemente unbedingt an genau die vorgeschlagene Stelle setzen.

Und wenn es um die eigentliche Umsetzung geht, muss der Entwickler natürlich das Templating-System des jeweiligen CMS kennen. Was nützt schließlich das tollste HTML, wenn ich nicht weiß, wie ich daraus ein funktionierendes Wordpress-Theme baue?

Server-Musts für CSS-Friseure

[…] und wie viel Backend sollte ein Frontend Dev können?

Das kommt meiner Meinung nach sehr auf die Arbeitsteilung im konkreten Unternehmen an. Ich persönlich würde auch in einer reinen Front-End-Position so viel wie möglich von Back-End des Systems, mit dem ich arbeite verstehen wollen einfach weil ich ein Gefühl dafür haben will, woher die Inhalte kommen, die ich strukturieren und stylen soll. Ich denke, dass es vielen Entwicklern so geht. Aber verallgemeinern lässt sich das sicher nicht.

Toolchain

Was muss ein Front-End Entwickler beherrschen?

Die Frage lässt sich leider beim besten Willen nicht allgemeingültig beantworten. Wäre ja auch viel zu einfach. Es folgen stattdessen ein paar Hinweise und Nice-to-haves, die helfen dürften, einen Entwickler besser einzuschätzen.

Es gibt keine spezifischen Tools, die *jeder* Developer beherrschen sollte, weil der Workflow jeder Firma, jedes Teams, jedes Entwicklers grundverschieden sein kann. Jenseits von der Erwartung, dass der Entwickler mit absoluter Standardsoftware wie Office und Photoshop umgehen können sollte, wird es schnell schwammig. Arbeitet der Entwickler in TextMate? BBEdit? Textpad++? Eclipse? NetBeans? Letztlich ist es egal, solange am Ende guter Code dabei herumkommt.

Der dabei entstehende Output sollte valide sein (HTML ist ein Muss, CSS wäre schön). Der Code sollte so gut es geht selbsterklärend sein (Was macht die Variable $i?) und wenn's geht auch ein wenig dokumentiert, zumindest an potentiellen Knackpunkten.

Der Entwickler sollte auch eine Strategie für Cross-Browser-Output haben. Sei es, dass er gezielt per Conditional Comments Internet Explorer Versionen anspricht. Oder dass er über Progressive Enhancement / Graceful Degradation mit Tools wie Modernizr dafür sorgt, dass Browser immer nur versuchen, eine Seite so darzustellen, wie ihre Fähigkeiten es auch zulassen.

Kaum vermeiden lässt sich heutzutage auch, dass der Entwickler Ideen hat, mit dem mobile Web umzugehen. Soll heißen, er kann Seiten für Tablets und Smartphones umsetzen, z.B. unter Verwendung von Media Queries. Er weiß, welche Bedienkonzepte für Touch-Displays geeignet sind. Er hat Lösungen parat für Problemfälle wie die Darstellung von großen Tabellen auf kleinen Bildschirmen.

Zu guter Letzt ist es nur hilfreich, wenn der Entwickler mit mindestens *einem* Versionierungssystem wie CVS, SVN / Subversion oder Git gearbeitet hat. Welches genau? Egal, solange er sich überhaupt angewöhnt hat, seine Arbeit brav und wie ein echter Teamplayer einzuchecken.

Mein ganz persönliches Toolkit

Mein persönlicher Arbeitsablauf unterscheidet sich von Projekt zu Projekt, je nachdem ob der Schwerpunkt der Arbeit mehr im Front-End oder im Back-End liegt. Generell kann ich sagen, dass ich Webseiten mit einer Handvoll Tools entwickele:

- TextMate 2 für Mac OS X ist der Code meiner Wahl
- als Textumgebung verwende ich einen MAMP-Server (Mac, Apache, MySQL, PHP)
- jQuery kommt in fast jedem Projekt zum Einsatz, ist bei CMS wie Drupal sowieso mit dabei
- Less ist der CSS-Präprozessor in unserem Haus
- LESS Elements ist eine bezaubernde Sammlung an Less-Mixins für Standard-Aufgaben wie Schatten, Animationen etc.
- mit Modernizr befrage ich den Browser nach seinen Fähigkeiten und kann diese gezielt per CSS und JavaScript ansprechen
- als CSS Reset verwende ich Normalize, da es das Default-Styling von Elementen erhält und lediglich für alle Browser angleicht
- Selectivizr ermöglicht den Einsatz moderner CSS-Selektoren auch auf älteren Browsern

Diese Bibliotheken und Skripte finden sich eigentlich in jedem Projekt wieder, dass ich bearbeite, egal ob sie in einem Drupal-Theme liegen oder einer PHP-Anwendung.

Recruiting

Wie erkennt man einen guten Front-End Developer?

Ein wasserdichtes System, um einen wirklich guten Front-End Developer zu finden, kann ich hier leider nicht offerieren. Was folgt sind jedoch einige Beobachtungen aus den letzten Jahren, die vielleicht in die richtige Richtung weisen mögen.

Es empfiehlt sich selbstverständlich immer, ein wenig das relevante Grundwissen abzuklopfen. Es ist klar, dass der technische Sachverstand des Kandidaten den des Recruiters übersteigt (hoffen wir es mal!), aber mit Grundkenntnissen in den relevanten Themengebieten sollte es zumindest möglich sein, absoluten Unfug zu erkennen.

Überhaupt sollte ein Kandidat in der Lage sein, aus dem Stegreif über aktuelle Entwicklungen in seiner Branche zu sprechen, die ihn interessieren. Was zählt ist das Interesse, der Wille zur Weiterentwicklung.

Oft fällt es Bewerbern schwer, zu ihren Schwächen zu stehen. Einfach mal einräumen, dass man sich mit HTML5 vielleicht noch nicht so sehr auseinandergesetzt hat? *Undenkbar!* Dann lieber erzählen, dass man, ja klar, sich mit HTML5 voll gut auskennt, es aber noch voller gefährlicher, unerprobter Technologien stecke und somit höchste Vorsicht geboten sei. Bei solchen Spinnereien, und generell bei sehr absolut formulierten Aussagen, wirkt eine einfache Bitte, das doch mal näher zu erklären, wahre Wunder.

Überhaupt ist es das Sahnehäubchen auf dem Entwickler-Cupcake, wenn er nicht nur 50 Fachbegriffe pro Minute absondern kann, sondern auch in der Lage ist, die gleichen Themen auch mal verständlich zu erklären. Fast, als wäre er ein ganz normaler Mensch.

Darüber hinaus enden meiner Meinung nach aber auch schnell die Recruiter-seitigen Möglichkeiten, die Befähigung eines Entwicklers im Gespräch abzuklopfen. Hier beginnt schon bald der

Zuständigkeitsbereich des suchenden Unternehmens, den Kandidaten in Tests oder Workshops genauer unter die Lupe zu nehmen. Selbst erfahrenen IT-Leuten kann ich im Gespräch sonst etwas erzählen, was zählt ist auf dem Keyboard.

Beispiele für gute / schlechte Front-Ends

Auch dieses Thema lässt sich (leider, leider) nicht gut allgemein abhandeln. Gute Beispiele wie schlechte Beispiele können oft subjektiv sein. Zudem kann eine schlechte Seite über Nacht durch ein gutes Redesign ersetzt werden. Genauso gut kann eine gute Seite schlagartig zu absolutem Schrott werden. Deswegen folgen auch hier nur Dinge, auf die man achten kann.

Ist die Referenzseite zweckmäßig und navigierbar? Der Punkt sollte sich von selbst verstehen. Vermittelt der Kandidat mit seiner Referenz den Eindruck, er könnte eine Seite herstellen, die gut zu bedienen ist? Bei der schnell klar wird, was die Funktion der Seite ist und wie ich diese Funktion nutzen kann?

Ist der Code valide? Niemand freut sich über invalides HTML. Google bestraft diesen Unfug. Browser zeigen Inhalte falsch an oder rechnen sich einen Wolf, um die Fehler im Code zu erkennen und zu kompensieren. Evtl. nicht semantisches Markup erschwert es z.B. Screenreadern, die Inhalte vernünftig zu verarbeiten. *Niemand* freut sich über invalides HTML.

Und ist der Code zeitgemäß? Niemand verlangt, dass die Referenzseite voll von CSS-Transforms ist und per Flexbox inklusive Float-Fallback aufgebaut wurde, aber zumindest die Semantik der Elemente sollte stimmen, Tabellen sollten nur für Ergebnisse in Hallenfußballturnieren genutzt werden, und die Optik sollte aus CSS kommen, nicht aus dreifach verschachtelten `<big>`-Tags.

Hat die Seite einen akzeptablen PageSpeed Score? Was nützt die schönste Seite, wenn sie aus welchen Gründen auch immer völlig unperformant ist? PageSpeed, so wie z.B. auch YSlow, ist ein

praktisches Hilfsmittel, Optimierungsbedarf an Webseiten festzustellen. Seien es zu große Bilder, langsame Skripte oder verschenkte Bandbreite durch unkomprimierte Inhalte. Wenn ein Kandidat Referenzen mit PageSpeed Resultaten im roten Bereich vorlegt, sollte er besser eine verdammt gute Ausrede parat haben.

Wie viel Selbstgemachtes steckt in der Seite? Will der Kandidat uns gerade weismachen, er könnte CMS *und* Responsive, weil er mal bei Joomla ein fertiges responsives Theme installiert und eine Farbe geändert hat? Einfach mal Firebug oder den WebInspector von Chrome / Safari aufmachen. Einfach mal die Stylesheet-Links anschauen, den Theme-Namen ermitteln, durch Google jagen und das ursprüngliche Theme mit der Referenz vergleichen.

Der Referent

Marc-Oliver Teschke (* 1984) verließ 2004 die romantische Fachwerkstadt Celle, um am *b.i.b. International College* in der romantischen Weltmetropole Paderborn seine Ausbildung zum **staatlich geprüften Informatiker Multimedia** zu absolvieren.

Die Gelegenheit, sich die zwei Ausbildungsjahre anrechnen zu lassen und in nur einem Jahr an der *Southampton Solent University* in der romantischen Engländerstadt Southampton den **BSc Internet Application Development** draufzusetzen, konnte er sich nicht entgehen lassen.

Seit 2007 ist er bei der *PLANWERK 6 websolutions GmbH* in der romantischen Nicht-Köln-Stadt Düsseldorf als **Entwickler für Internet-Anwendungen**, professioneller HTML-Schubser, CSS-Friseur, JavaScript-Bändiger, PHP-Töpfer und Datenbank-Datenbanker tätig und in alle Schritte der Entstehung von Webseiten eingebunden, von Akquise und Planung, Design und Entwicklung bis zu Übergabe und Support.

Links

Basics

- Mozilla Developer Network Web Technologies Dokumentation
- SitePoint Doctypes
- CSS Zen Garden
- jQuery
- MooTools
- Microformats
- hCard
- HTML5: Differences from HTML4
- CSS current work & how to participate

Für Fortgeschrittene

- SELFHTML CSS/Box-Modell
- MDN display
- MDN float
- MDN Using CSS flexible boxes
- Solved by Flexbox
- Less CSS-Präprozessor
- Sass CSS-Präprozessor
- Stylus CSS-Präprozessor

Toolchain

- Conditional Comments
- Conditional HTML
- Graceful degradation versus progressive enhancement
- Modernizr
- TextMate
- MAMP
- LESS Elements

- Normalize
- Selectivizr

Recruiting

- „How to hire a programmer when you're not a programmer” - Basecamp (ehem. 37signals), 2010
- W3C Markup Validation Service
- PageSpeed Insights
- YSlow
- Firebug