Kai Günster

Mit Syntax-Highlighting!

Inkl. Downloads

Schrödinger lernt HTML5, CSS & JavaScript

Das etwas andere Fachbuch

- Lerne die Sprachen des Webs von Grund auf
- Mobile Layouts, Geolocation, Touchevents, Audio & Video ... alles drin
- Durchblicken, mitmachen und genießen!

VIERTE AUFLAGE

Rheinwerk Computing

Widmung

Mit Dank an meine Freunde, meine Familie und ganz besonders an Sizinha, die alle noch mit mir sprechen, obwohl ich nicht die Zeit für sie hatte, die sie verdient haben. Und mit Dank an das Web als Ganzes, denn ohne die Arbeit, die viele Andere dort geleistet haben, wäre auch dieses Buch nicht möglich gewesen.

Liebe Leserin, lieber Leser

SIE HABEN SICH EINE MENGE VORGENOMMEN:

HTML, CSS und JavaScript.

Drei Sprachen.

Wissen Sie was? Wir können das **gut verstehen**. Das ist schließlich nicht irgendein IT-Zeugs, das sind **Weltsprachen**. Da sind wir dabei.

Und zum Glück sind wir **bestens vernetzt**, so wie es sich im **WWW** gehört.

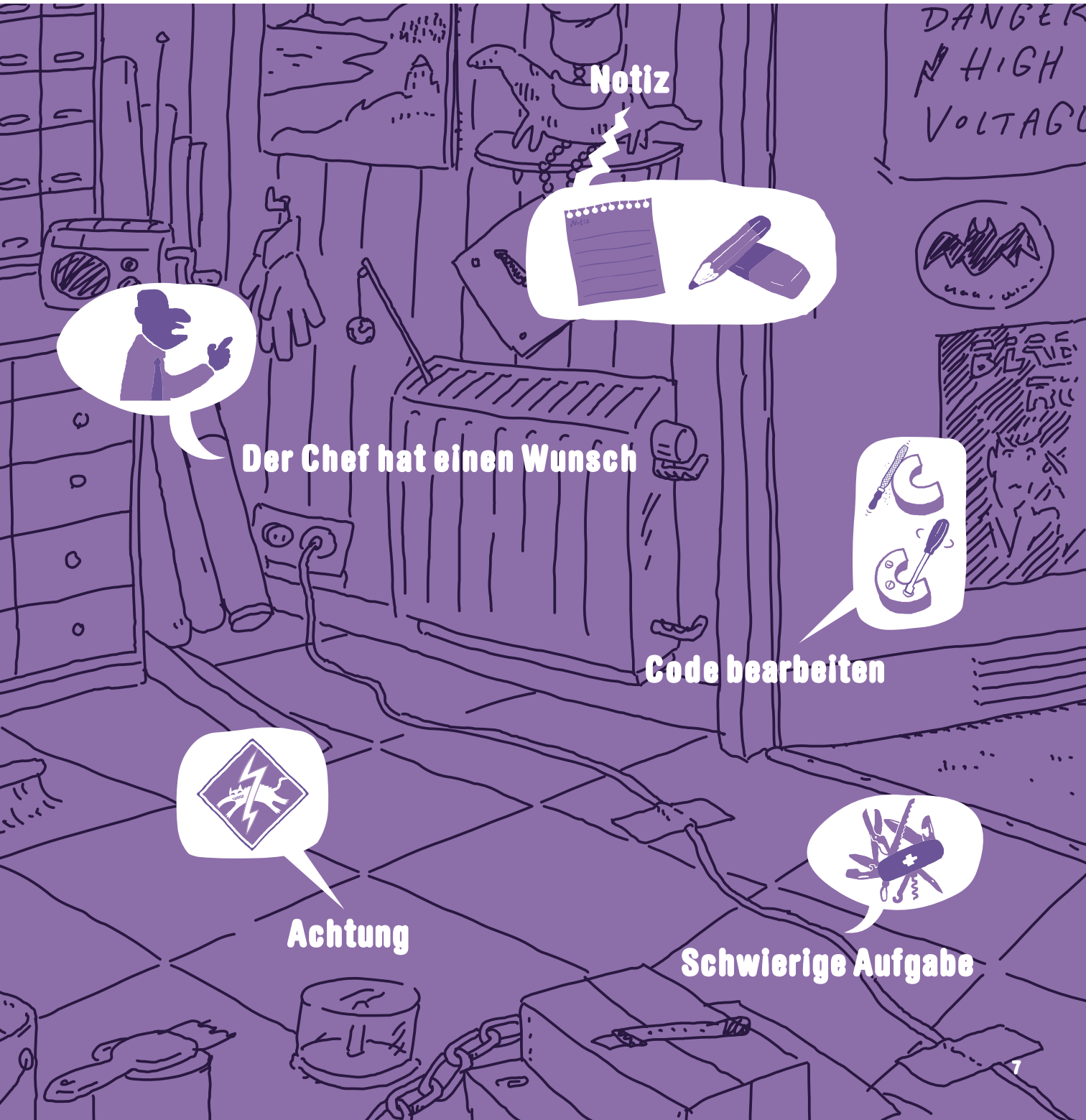
Na dann: Viel Erfolg!

Schrödingers Büro

Die nötige Theorie, viele Hinweise und Tipps

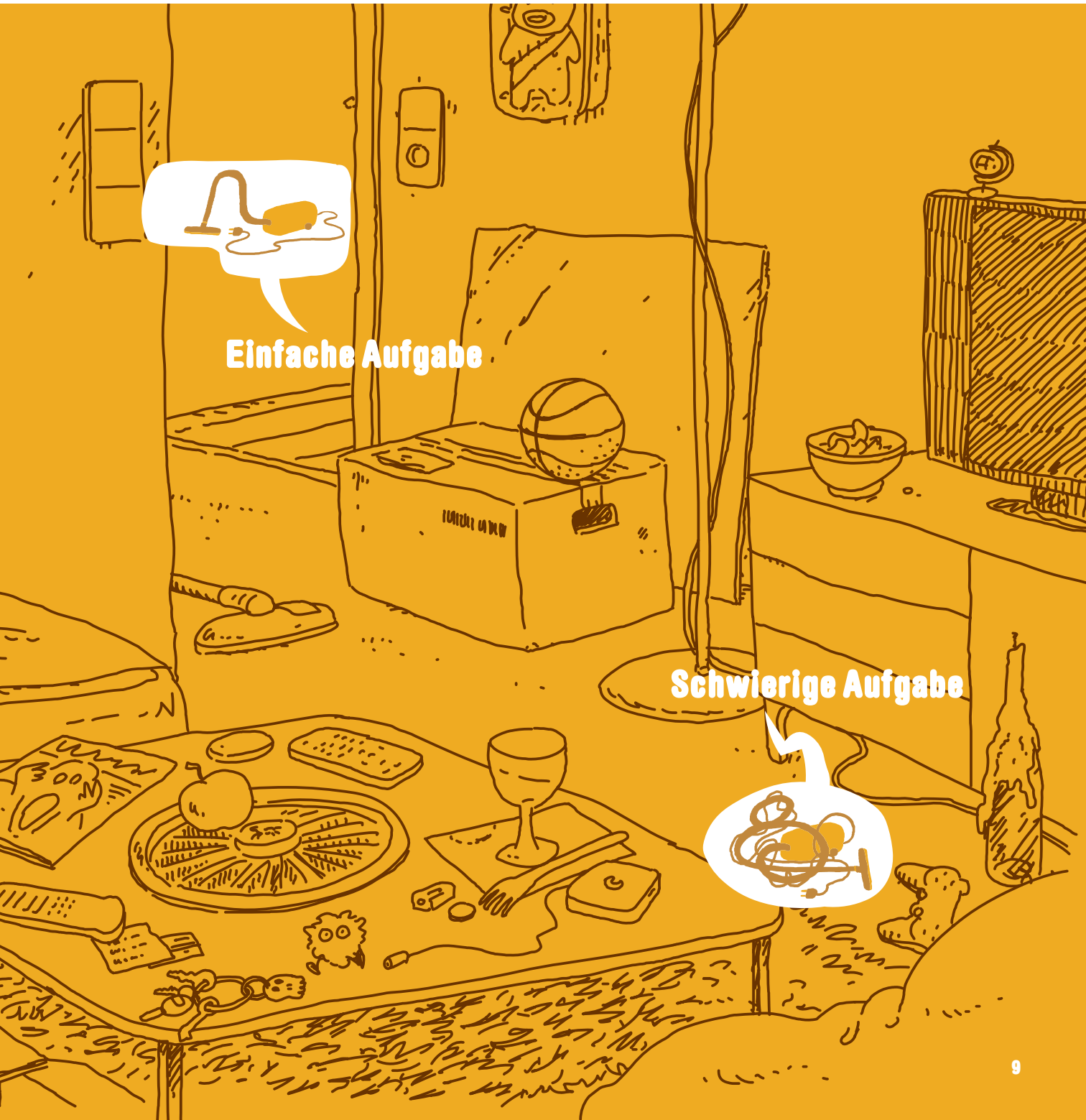
Schrödingers Werkstatt

Unmengen von Code, der ergänzt, verbessert und repariert werden will



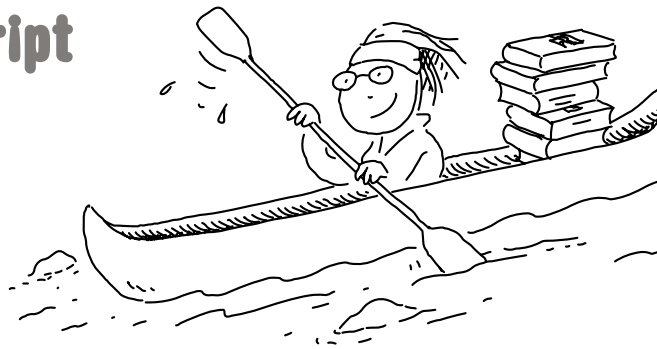
Schrödingers Wohnzimmer

Viel Kaffee, Übungen und die verdienten Pausen



IMPRESSUM

Schrödinger lernt HTML5, CSS und JavaScript

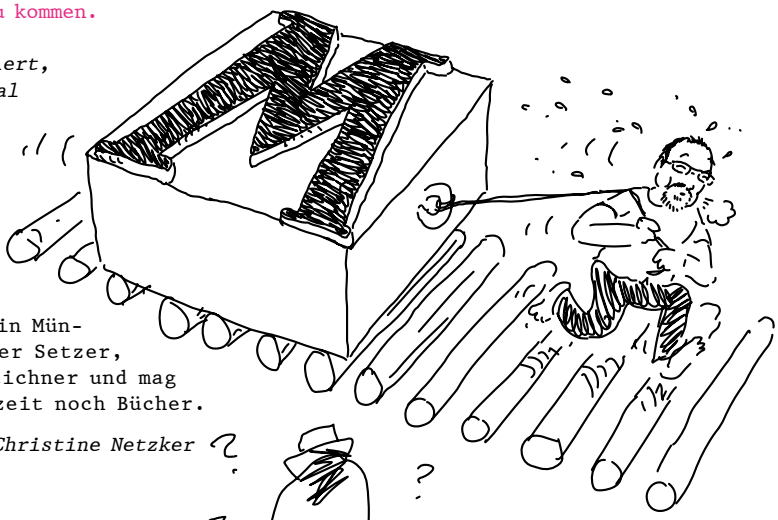


Als Kanutin umschifft Almut Stromschnellen mit notfalls nur einem Paddel. Eine Fähigkeit, die sie zur Fachbuchlektorin prädestiniert.

Lektorat: Almut Poll

Gutes Timing ist in der Buchproduktion unerlässlich. Die Hoffnung auf Bestzeiten hat Norbert allerdings aufgegeben: Schrödinger legt offenbar mehr Wert darauf, als Lustigster ins Ziel zu kommen.

Herstellung: Norbert Englert, Denis Schaal



Okay, die Baumstämme hatten wir schon hingelegt, bevor Markus den Satz des Buches übernahm. Und der Rest war, wie man sieht, ein Kinderspiel, gell?

Markus lebt und arbeitet in München als selbstständiger Setzer, Bildbearbeiter und Reinzeichner und mag auch in seiner Freizeit noch Bücher.

Satz: Markus Miller, Janne Brönner, Christine Netzker

Schon zu Schulzeiten zeichnete Leo am liebsten die Bücher voll, von denen er am wenigsten kapierte.

Seit er weiß, daß man das auch gegen Bezahlung machen kann, kapiert er gar nichts mehr.

Leo Leowald lebt und arbeitet in Köln als freiberuflicher Illustrator. Er veröffentlicht unter anderem in *titanic*, *jungle world* und bei *reprodukt* und zeichnet seit 2004 den Webcomic *www.zwarwald.de*.

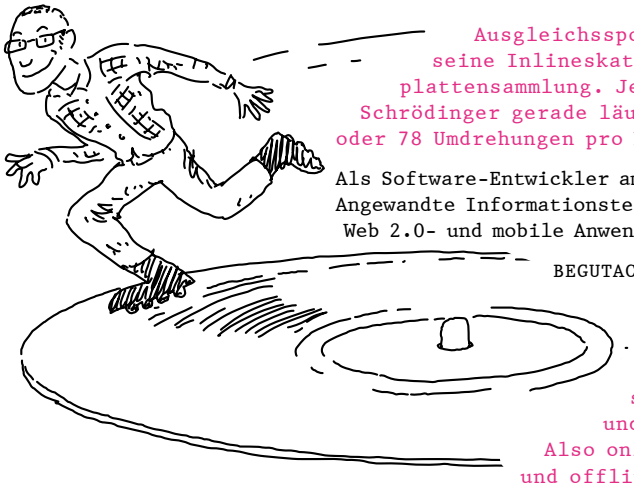
Andreas' zweite Leidenschaft neben der Buchgestaltung ist kochen. Wie auch immer: Hauptsache rare – VERY RARE!

Andreas Tetzlaff ist selbstständiger Buchgestalter in Köln. Er arbeitet normalerweise für Kunstbuchverlage – dass ausgerechnet ein IT-Fachbuch ihn vor künstlerische Herausforderungen stellt, hätte er sich vorher nicht träumen lassen …

Annette ist von Haus aus Archäologin, da ist es nur ein kleiner Schritt bis zum Lektorat, und der Vorteil ist: Bei Schrödinger und Co. findet sie immer was.

Annette Lennartz ist freiberufliche Lektorin in Bonn. Für Schrödinger hat sie immer eine offene Tür. Privat schätzt sie augenzwinkernde und gruselige Geschichten oder bastelt an filigranen Schiffsmodellen.

KORRIGIERT VON: *Annette Lennartz, Anne Scheibe*



Ausgleichssport für Philip sind seine Inlineskates und seine Vinylplattensammlung. Je nachdem, wie gut es bei Schrödinger gerade läuft, rotiert er mit 33, 45 oder 78 Umdrehungen pro Minute.

Als Software-Entwickler am Fraunhofer-Institut für Angewandte Informationstechnologie FIT stellt *Philip Ackermann* Web 2.0- und mobile Anwendungen auf den Prüfstand.

BEGUTACHTET VON: *Philip Ackermann*

Im Gegensatz zu Schrödinger liebt Kai seine Katze Pixie. Außerdem liebt er Brettspiele und schreibt darüber im *Meople's Magazine*. Also online über Offline-Spiele und offline über Webentwicklung, während sich Pixie ganz aufs Spielen konzentrieren kann.

Kai Günster, AUTOR

Einbandgestaltung: Andreas Tetzlaff und Leo Leowald
Initiales Design: Andreas Tetzlaff

FÜR DIE, DIE ES GENAU WISSEN WOLLEN

Dieses Buch wurde gesetzt aus unzähligen Schriften (u.a. aus der WIMBY von Evert Ypma: Danke, Evert!), Tonnen an Illustrationen und anderen komischen Zeichen, die alle Beteiligten in den Wahnsinn trieben.

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de abrufbar.

ISBN 978-3-8362-9598-7

4., aktualisierte Auflage 2023

INHALTSVERZEICHNIS

Kapitel 3: Jetzt kommt Farbe ins Spiel

Einführung in CSS

Kapitel 4: Kaskaden für Bossingen

CSS-Selektoren und Typografie

Kapitel 5: Ordnung in die Plattensammlung

Listen und Tabellen

Kapitel 6: Von der Wiege bis zur Bahre – Formulare

Formulare

Kapitel 7: Von Rändern und Schuhkartons

Seitenlayout in HTML und CSS

Kapitel 8: ENTlich, eine Website! Schrödinger setzt das Gelernte zusammen

Eine Website von Anfang an

Kapitel 9: Schöner wohnen mit CSS3

CSS3

Kapitel 10: Jetzt muss es sich aber endlich bewegen

JavaScript

Kapitel 11: Programmieren mit Bausteinen

Funktionen

Kapitel 12: Augen auf, du hast User!

Eventhandler

Kapitel 13: Gerade stand das da noch nicht

DOM-Manipulation

Kapitel 14: Schrödingers Welt der Programmierung

Objekte und JSON

Seite 563

Kapitel 15: Halt, hiergeblieben! Cookies, WebStorage und File-API

Cookies, WebStorage und File-API

Seite 609

Kapitel 16: Alles kann ein Radio sein, oder ein Fernseher, oder sogar eine Leinwand

Multimedia

Kapitel 17: Schrödinger will's wissen

Ajax

Kapitel 18: Verwandlungskunst

Responsive Webdesign und Mobile Devices

Seite 759

Kapitel 19: Der Blick nach vorn – was geht noch?

Was geht noch?

Seite 821

Anhang: Reguläre Ausdrücke und Zeichencodes

Das Vorwort: seriös und ohne Schnickschnack

Ich habe keine Angst zu übertreiben, wenn ich sage, dass das World Wide Web eine der großen, transformativen Erfindungen der Menschheit ist. So wie die Dampfmaschine oder die Druckerpresse. Es hat sogar sehr viel gemeinsam mit der Druckerpresse, denn genau wie diese hat das World Wide Web unseren Umgang mit Informationen grundlegend verändert. Durch die Druckerpresse wurden Bücher zur Massenware: Jeder, der Lesen konnte, konnte sich günstig Bücher beschaffen, sie lesen und sich weiterbilden. Dass die Welt ohne diese Erfindung heute anders aussähe, siehst du bestimmt genauso.

Das World Wide Web geht noch weiter. Es macht mehr Informationen schneller und einfacher zugänglich als jemals zuvor in der Geschichte. Noch wichtiger ist aber: Jeder Benutzer kann jederzeit seine Meinung zu etwas sagen, kann eigene Informationen verbreiten und mit Menschen in der ganzen Welt kommunizieren, als säßen sie im nächsten Raum. Menschen aus aller Welt bilden Gemeinschaften, diskutieren, tauschen sich aus und arbeiten zusammen, um Neues zu erschaffen.

Und das hört nicht bei technischer Information auf; der freie Meinungsaustausch durch das Web hat längst auch die Politik erreicht. Das WWW hat jetzt schon, mit seinen gerade 20 Jahren, Revolutionen ausgelöst. Auch bei uns in Deutschland wurden schon politische Vorhaben gestoppt und in Gang gesetzt, weil sich Menschen über das Internet gefunden und zusammengearbeitet haben. Und all das geht unabhängig von Geschlecht, Hautfarbe, Nationalität und Reichtum der Eltern. Das ist die Änderung, die das Web bringt. Und ich bin mir sicher, es geht noch weiter.

Aber was hat das alles mit dir zu tun? Du hast dieses Buch in der Hand, weil du die drei Sprachen des Webs lernen möchtest: HTML, CSS und JavaScript. Das ist dein Werkzeugkasten, um selbst etwas im Web zu erschaffen. Ob du „nur“ deine eigene, kleine Homepage haben möchtest, wichtige Informationen darstellen und verbreiten oder ganz neue Wege entwickeln, zu kommunizieren – diese Werkzeuge geben dir die Möglichkeiten dazu. Oder auch, wenn du das nächste große Browserspiel entwickeln willst, das Millionen Menschen zusammenbringt, um Spaß zu haben.

Und vielleicht möchtest du auch dabei mitmachen, wenn die nächste große Idee das Web – und damit die Welt – verändert. Das aktuelle Modell Social Media ist noch nicht das Ende der Fahnenstange; wir werden noch andere Wege finden, uns durch das Web auszutauschen, zusammenzukommen, Neues zu erschaffen. Und mit dem Inhalt dieses Buches hast du alles, was du brauchst, um dabei mitzumachen. Das Web ist die erste Erfindung, an der wirklich jeder mitarbeiten kann, und du wirst sehen, dass die Einstiegshürde dafür gar nicht so hoch ist, wie du jetzt noch glaubst. Also los, und ich hoffe, du hast Spaß dabei!

Kai

—EINS—

Aufbau einer Seite und die wichtigsten Elemente

Fangen wir mit einem Gerüst an

Schrödinger fühlt sich ziemlich verloren mit seiner neuen Aufgabe. Klar, er hat schon Tausende von Webseiten besucht, aber wie man sie selbst erstellt, da hat er nie drüber nachgedacht.
Er weiß ja nicht einmal, mit welchem Werkzeug er eine Seite erstellen soll. Und so tut Schrödinger, was jeder in seiner Situation tun würde: Er geht nach Hause. Natürlich nicht alleine, davon würde seine Unwissenheit auch nicht besser. Aber Schrödinger kennt zum Glück jemanden, der ihm weiterhelfen kann.

Die drei ??? – HTML, CSS und JavaScript

... ich hab ihm immer mit seinem Office und seinem E-Mail-Programm geholfen, und heute kommt der alte Bossingen zu mir und sagt: „Unsere Webseite muss dringend überarbeitet werden, Schrödinger, Sie sind doch Computerspezialist." Ich hab doch keine Ahnung von Webseiten, ich surfe doch nur.

Mach dir mal keine Sorgen, Schrödinger. Webseiten zu bauen, ist viel einfacher, als du denkst. Fast, wie Texte zu schreiben in Word, nur dass du nicht im Menü auswählst, wie dein Text aussehen soll, sondern mit Tags Bereiche markierst und dann im Stylesheet beschreibst, wie sie aussehen sollen.

Und da geht's schon wieder los mit dem Fach-Chinesisch. Täcks? Steilschrift? Ich versteh ja nicht mal, wovon du redest.

Okay, okay, fangen wir doch einfach am Anfang an. Es gibt drei wichtige Elemente für Webseiten, drei Sprachen, um genau zu sein. Zuerst gibt es **HTML.** In HTML schreibst du die Struktur und den Inhalt deiner Seite. Du schreibst deinen Text und markierst Überschriften, Absätze und so weiter, und der Webbrowser macht dann daraus den formatierten Text, der angezeigt wird.

[Zettel]
Der Browser ist das Programm, mit dem man Webseiten anschaut und im World Wide Web surft. Die bekanntesten sind Internet Explorer, Firefox, Safari und Chrome.

Eine Sprache, und der Webbrowser macht was draus? So wie beim Programmieren? Man schreibt seitenweise unverständlichen Kram, dann lässt man dieses Programm laufen ...

Du meinst den **Compiler**?

Genau, den Compiler. Man schreibt kryptischen Code, lässt den Compiler laufen, und Hokus Pokus hat man ein Computerprogramm, zum Beispiel World of Warcraft.

Nein, so schlimm ist es wirklich nicht. **HTML ist nicht kryptisch, das meiste ist einfach Text**. Und zwischendrin stehen **Markierungen**, die erklären, was der Text bedeutet, zum Beispiel „hier ist der Seitentitel".
Die zweite wichtige Sprache ist **CSS**, für Cascading Style Sheets. HTML für sich funktioniert, aber es sieht aus wie **dein Referat aus der fünften Klasse**. CSS sorgt dafür, dass alles gut aussieht, oder zumindest so aussieht, wie du es haben möchtest. Ob es dann auch wirklich gut aussieht, ist deine Sache.

Sehr, sehr witzig.

Und dann gibt es noch JavaScript. **JavaScript** ist für alles da, was nicht nur einfach auf dem Bildschirm stehen soll, sondern auch etwas tun soll. **Rechnen, sich bewegen, die Lottozahlen vorhersagen**, das ist alles JavaScript.

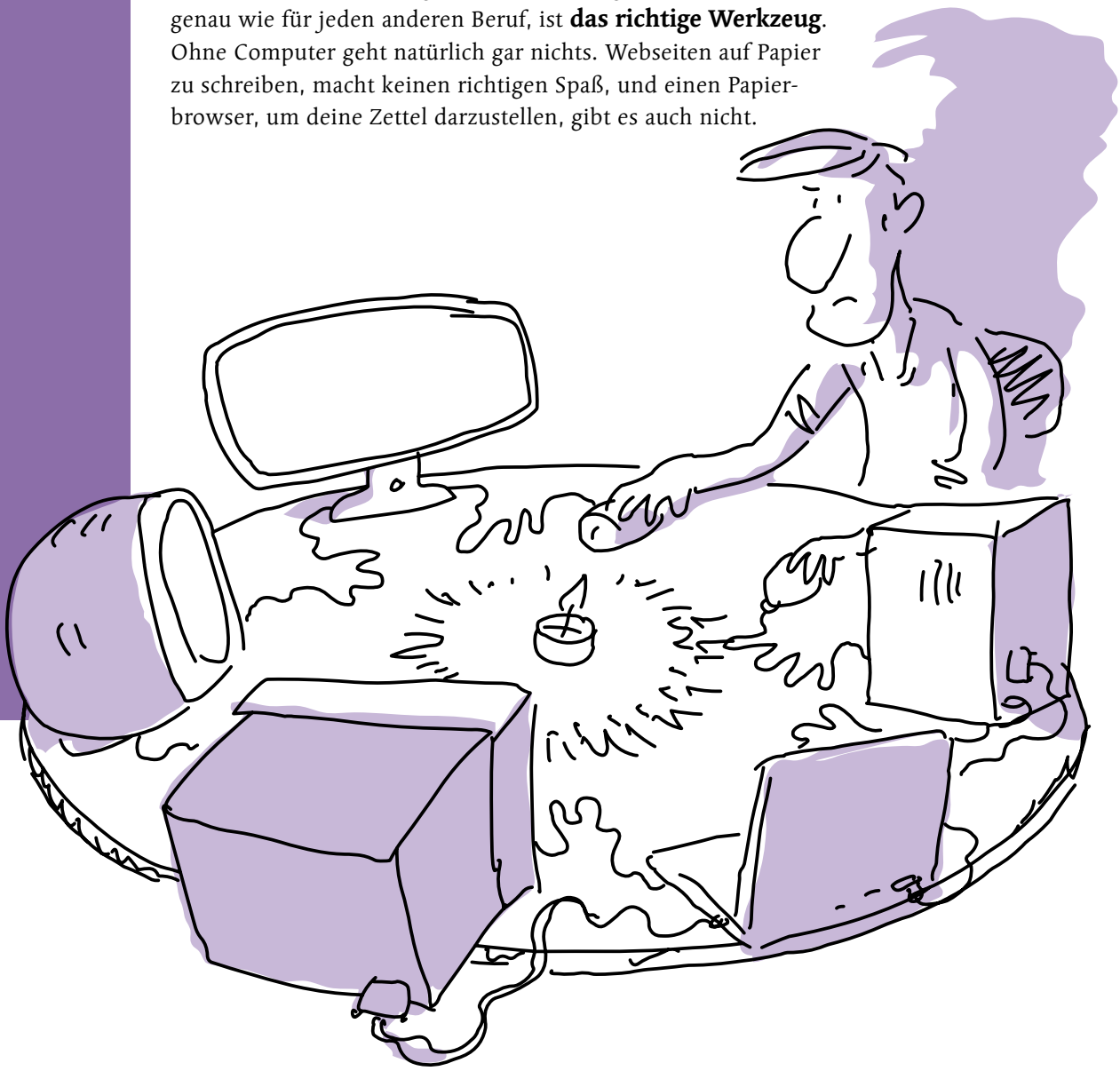
Du kannst dich ja ziemlich gut damit aus. Kannst du mir nicht vielleicht ein paar Sachen beibringen? Zumindest genug, dass Bossingen mit der neuen Webseite zufrieden ist?

Na klar, dafür hat man ja Freunde. Wenn ich mal was über deine Arbeit wissen möchte, würdest du mir ja auch weiterhelfen, oder? Warum machst du nicht schnell einen Kaffee, und dann erklär ich dir HTML? Zu CSS und JavaScript kommen wir dann später, die haben ohne HTML sowieso keinen Sinn.

[Zettel]
Wenn man Webentwickler um Hilfe bittet, ist eine Tasse Kaffee fast immer das geeignete Mittel zur Bestechung.

Der Werkzeugkasten

So, dann los. Das Wichtigste für den Anfang als Webentwickler, genau wie für jeden anderen Beruf, ist **das richtige Werkzeug**. Ohne Computer geht natürlich gar nichts. Webseiten auf Papier zu schreiben, macht keinen richtigen Spaß, und einen Papierbrowser, um deine Zettel darzustellen, gibt es auch nicht.



Welches System du benutzt, ist egal; das Tolle an Webseiten ist ja gerade, dass sie überall funktionieren. Windows, Mac OS oder Linux sind alle okay. Wenn du ein anderes Betriebssystem benutzt, bist du an manchen Stellen auf dich allein gestellt, weil ich dann keine Ahnung habe. Aber wenn du wirklich ein anderes System benutzt, kennst du dich wahrscheinlich gut genug aus, dass das kein Problem ist.

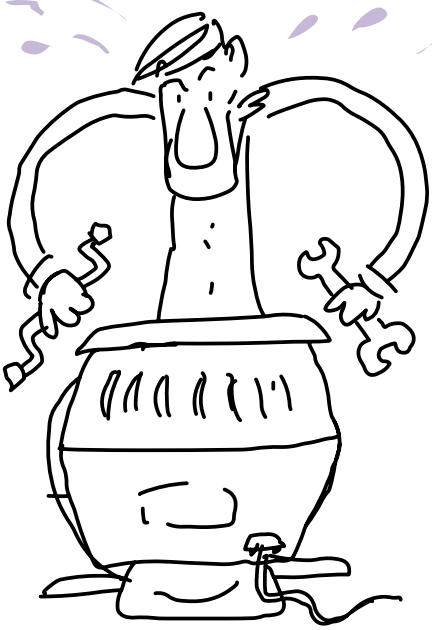
Webbrowser

Als Nächstes brauchst du **Webbrowser** – Plural, also mehr als einen. Ein Browser ist heute bei jedem Betriebssystem dabei, aber um Webseiten zu schreiben, sollte es mindestens noch ein weiterer sein.




Hast du nicht gerade noch gesagt, dass Webseiten überall funktionieren?

Du hast Talent dafür, Salz in offene Wunden zu streuen, weißt du das, Schrödi? Theoretisch funktionieren Webseiten überall, aber leider sehen sie nicht immer überall so aus, wie du es möchtest. Das ist der ewige Fluch der Webentwickler. Na gut, das ist vielleicht etwas dramatisch. In den letzten Jahren hat sich vieles verbessert, und Webseiten sehen in verschiedenen Browsern meist sehr ähnlich aus. Trotzdem ist es immer besser, mehr als einen Browser zum Testen zu haben. Firefox und Chrome kannst du auf jedem System installieren (und herunterladen unter **www.firefox.com** bzw. **www.google.com/chrome**). Safari ist auf jedem Mac schon installiert, für alle anderen Systeme gibt es ihn hier: **www.apple.com/safari**. Selbst Microsoft Edge gibt es inzwischen für die verbreiteten Betriebsysteme. Für Linux und MacOS kannst du ihn unter **https://www.microsoft.com/en-us/edge/download** herunterladen, bei Windows ist er sowieso dabei... es sei denn, dein Windows hat noch einen Internet Explorer. Den unterstützt Microsoft selbst inzwischen nicht mehr, deswegen wollen wir hier auch nicht weiter darüber sprechen. Installiere einfach einen der anderen Browser oben und sei glücklich: Der IE ist nämlich der Hauptgrund für den ewigen Fluch von vorhin.



Editor

Und damit haben wir auch schon fast alles, was wir brauchen, es fehlt nur noch ein **Texteditor**. Man kann HTML – und auch CSS und JavaScript – in jedem Editor schreiben, der einfache Textdateien ohne Formatierung erzeugt. Alles, was **TXT-Dateien** erzeugt, funktioniert. Aber trotzdem sind manche Editoren besser als andere. Ein guter Editor für HTML zeigt an, wenn das Dokument korrekt ist, hebt die Struktur hervor und erleichtert die Fehlersuche. Für Windows ist Notepad++ (**http://notepad-plus-plus.org**) sehr geeignet, unter Linux funktioniert Geany gut (**www.geany.org** oder mit dem Package Manager deiner Linux-Distribution), Bluefish (**http://bluefish.openoffice.nl** oder wieder mit dem Package Manager) ist auch nicht schlecht. Bluefish gibt es auch für den Mac, TextWrangler ist eine Alternative.

Genug Vorbereitung. Es ist Zeit, ins kalte Wasser zu springen und die erste Seite zu schreiben.

Warte mal! Das sind ja alles Programme, um Text zu bearbeiten. Es gibt doch für HTML bestimmt was Einfacheres, so mit Maus hin- und herschieben und so.

Klar, da gibt es vieles. Aber zum Lernen sind WYSIWYG-Tools keine gute Idee, du sollst ja erst mal wissen, was du tust. Später kannst du dann zum Beispiel Dreamweaver benutzen oder einen der vielen kostenlosen Editoren. Wenn du das dann noch möchtest – Webentwickler streiten sich gerne, ob es sich mit einem WYSIWYG-Editor besser arbeiten lässt als mit einem guten Texteditor. Das musst du nach ein paar hundert Seiten selbst entscheiden.

[Zettel]
WYSIWYG steht kurz für „What you see is what you get“, also „Was du siehst ist auch das, was rauskommt“, und beschreibt alle Programme, bei denen das Dokument beim Bearbeiten genauso aussieht wie später für den Leser.

Das erste Dokument

Fertig, Schrödinger? Genieße deine **letzten Sekunden** als jemand, der noch nie eine Webseite geschrieben hat. Es geht jetzt schneller, als du denkst.

[Einfache Aufgabe]

Starte den Editor deiner Wahl. Erstelle eine neue Datei, und speichere sie mit der Endung **`.htm`** oder **`.html`**.

[Zettel]

Die meisten Editoren erkennen an der Dateiendung, dass es sich zum Beispiel um eine HTML-Datei handelt und sie damit beim Bearbeiten unterstützen sollen. Deshalb ist es besser, die Datei mit der richtigen Endung zu speichern, bevor man anfängt, damit zu arbeiten.

[Einfache Aufgabe]

Jetzt tippe diesen HTML-Code ein, und speichere die Datei.

```
<!DOCTYPE html>
<html>
<!--Das ist wirklich meine erste Webseite -->
  <head>
    <title>Meine erste Webseite</title>
  </head>
  <body>
    <h1>Dies ist meine erste Überschrift</h1>
    <p>Hallo Webwelt!</p>
  </body>
</html>
```

[Zettel]

Den HTML-Code einzurücken, wie gezeigt, ist für die Funktion der Seite nicht wichtig. Du könntest alles in eine Zeile schreiben, und es sähe im Browser gleich aus. Aber die Einrückung macht es viel angenehmer, die Datei zu bearbeiten.

Wenn alles korrekt abgetippt ist, dann sollte die Datei im Editor jetzt zum Beispiel so aussehen:

```
1  <!DOCTYPE html>
2  <html>
3  <!--Das ist wirklich meine erste Webseite -->
4    <head>
5      <title>Meine erste Webseite</title>
6    </head>
7    <body>
8      <h1>Dies ist meine erste Überschrift</h1>
9      <p>Hallo Webwelt!</p>
10   </body>
11 </html>
```

HTML-Datei im Editor: Quelltext mit Farbcodierung

[Achtung]
Achte besonders darauf, alle **<**, **>** und **/** zu setzen, wie im Beispiel.

Und jetzt kommt der spannende Moment: ein Doppelklick auf das Datei-Icon im Verzeichnis, und die Datei sollte sich im Browser öffnen und etwa so aussehen:

HTML-Datei im Browser: Webseite mit Titel

Ich gratuliere, deine erste Webseite! Sieht doch schon toll aus, oder?
Damit sind wir auch fertig und …

Was? Fertig? Das sieht aber noch nicht so aus wie die Webseiten, die ich jeden Tag sehe!

… fertig und können ins Büro gehen, wo ich dir erkläre, was genau du gerade getan hast. Dachtest du, ich lasse dich mit so einer oberflächlichen Einführung sitzen?

Markup und Tags

Was haben wir jetzt gerade getan? Unsere erste HTML-Seite erstellt, klar, aber warum funktioniert sie und sieht so aus, wie sie aussieht? Und was bedeuten diese vielen **Größer- und Kleiner-als-Zeichen**?
Diese Zeichen sind das A und O von HTML, ohne sie wäre es einfach nur Text. Alles, was zwischen einem **<** und einem **>** steht, ist ein **Tag**. **<html>**, **<body>**, ** ** sind Tags, öffnende Tags, um genau zu sein. Kommt noch ein Slash **/** dazu, wird aus dem öffnenden ein **schließendes Tag**: **</html>**, **</body>**, ** **.

[Zettel]
Tag ist ein englisches Wort und hat nichts mit dem deutschen Zeitraum zwischen Sonnenauf- und Sonnenuntergang zu tun. Gesprochen wird es deshalb Täg, und bedeutet unter anderem „Markierung". Ein Tag markiert Text.

[Hintergrundinfo]
HTML steht für Hypertext Markup Language. Es gibt viele Markupsprachen; ihnen allen ist gemein, dass sie Bereiche mit Tags markieren. Die bekannteste Markupsprache neben HTML ist XML, aber das wird erst viel später interessant für uns.

Alles zwischen einem öffnenden und dem dazu passenden schließenden Tag ist der **Tag-Body**, der Inhalt des Tags.

```
<title>*1Dies ist der Seitentitel*2</title>*3
```

*1 öffnendes Tag

*2 Tag-Body

*3 schließendes Tag

Tags können und müssen ineinander **verschachtelt** werden: In der Beispielseite sind **<head>** und **<body>** Teil des Tag-Bodys von **<html>**, **<title>** steht im Tag-Body von **<head>**.

[Achtung]
Tags dürfen zwar beliebig tief ineinander verschachtelt werden, aber nicht jedes Tag darf an jeder Stelle stehen. **<head>** innerhalb von **<body>** geht zum Beispiel nicht, deswegen hätte Dalí kein Webentwickler werden können.

Eins ist beim Verschachteln von Tags sehr wichtig: Wenn das öffnende Tag innerhalb eines anderen Tags verschachtelt ist, dann muss das passende schließende Tag in dem gleichen Tag verschachtelt sein.

```
<p>
  Hier steht Text
  <span>Mehr Text</span>
</p>

<p>
  Hier steht Text
  <span>Mehr Text</p>
</span>
```

X

[Achtung]
Falsch verschachtelte Tags verursachen im Browser keine Fehlermeldung, aber die Seite kann falsch dargestellt werden. Wenn später CSS und JavaScript dazu kommen, führen falsche Verschachtelungen zu Fehlern, die nur schwer zu finden sind.

[Begriffsdefinition]
Ein HTML-Dokument, in dem alle Tags korrekt geschlossen und verschachtelt sind, heißt wohlgeformt (well-formed).

Also wie Schachteln ineinanderpacken. Entweder die Schachtel ist in der anderen oder nicht, halb geht nicht.
Das ist aber genug Tag-Theorie für einen Tag, was bedeuten diese Dinger nun?

Das ist am einfachsten an einem Beispiel zu erklären.

Struktur einer HTML-Seite

Nummer 1 ist doch gar kein ordentliches Tag! Was soll dieses Ausrufezeichen?

```
<!DOCTYPE html> *1
<html> *2
<!--Das ist wirklich meine erste Webseite --> *3
  <head> *4
    <title>Meine erste Webseite</title> *5
  </head>
  <body> *6
    <p>Hallo Webwelt!</p> *7
  </body>
</html> *8
```

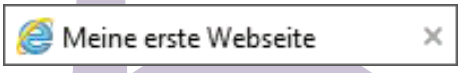
*1 Schön, dass du noch aufpasst, Schrödinger. Und du hast vollkommen Recht, das ist kein echtes Tag, sondern die **Doctype-Deklaration**. Damit teilst du dem Browser mit, dass jetzt wirklich ein HTML-Dokument beginnt – falls er der Dateiendung nicht vertraut. Außer DOCTYPE gibt es aber keine Deklarationen, diese Zeile bleibt immer allein.

*2 Das Root-Tag **<html>**. Das muss bei jedem HTML-Dokument vorhanden sein, vor dem öffnenden **<html>** darf nur der Doctype stehen.

*3 Du brauchst nichts zu sagen, Schrödinger, ich gebe es zu. Auch das ist kein Tag. Alles zwischen **<!--** und **-->** ist ein Kommentar und wird vom Browser komplett ignoriert. Aber wenn sich jemand den Quelltext der Seite anschaut, sind die Kommentare zu sehen, also kein guter Ort, um dein Passwort fürs Online-Banking zu speichern.

*4 In den Head-Bereich der Seite gehört alles, was zwar mit der Seite zu tun hat, was aber nicht zum Inhalt der Seite gehört, zum Beispiel, aber nicht nur, …

*5 … der Seitentitel. Der Titel wird nicht in der Seite angezeigt, sondern zum Beispiel in der Titelzeile des Browserfensters.



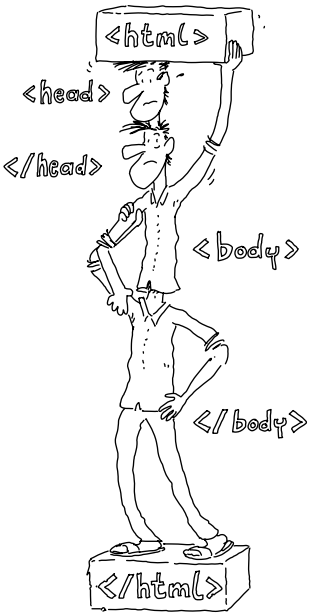
So zeigt der Internet Explorer den Titel an.

*6 Im **<body>** steht dann wirklich der Inhalt der Seite, alles, was hier steht, wird im Browserfenster angezeigt.

*7 **<p>** steht für Paragraph, also Absatz – Textabsatz, nicht Schuhabsatz. Man muss seinen Text nicht in **<p>**-Tags einpacken. Aber wenn man Fließtext schreibt, bietet es sich an, denn dann funktionieren Dinge wie das Einrücken der ersten Zeile per CSS.

*8 Das Ende des HTML-Dokuments. Nach **</html>** darf nichts mehr kommen, hier ist Schluss. **Finito.**

```
<html>
  <head>
  </head>
  <body>
  </body>
</html>
```



Was noch fehlt, um einen gut strukturierten Text zu schreiben, sind **Überschriften**. HTML unterstützt **sechs Ebenen von Überschriften**, die Tags dafür heißen **<h1>** bis **<h6>**. Der Text der Überschrift steht natürlich im Tag-Body.

[Einfache Aufgabe]

Füge über dem Textabsatz eine Überschrift der Ebene 1 (**<h1>**) ein, und gib ihr den Text „Dies ist meine erste Überschrift".

Schau dir die Seite im Browser an, die Überschrift sollte groß und fett oben auf der Seite stehen. Falls sie nicht zu sehen ist oder falsch aussieht, lautet das korrekte HTML so:

```
…
<body>
  <h1>Dies ist meine erste Überschrift</h1>*1
  <p>Hallo
…
```

*1 Überschrift Ebene 1

Wow, das ist so ... so ...

Cool? Einfach? Brillant erklärt?

Hässlich. Es ist hässlich.

Ja, hässlich ist es wohl auch. Das Standardaussehen von **<h1>** wird in den meisten Browsern von der **Hausnummern-industrie** gesponsert. Man kann es problemlos von der Straße aus lesen. Aber die gute Nachricht ist: Wenn wir später über CSS reden, kannst du alles Hässliche abstellen.

[Zettel]

Bitte mach die Überschrift nicht kleiner, indem du **<h4>**, **<h5>** usw. anstelle von **<h1>** benutzt. Es funktioniert zwar, dem Browser ist es egal; aber zum Beispiel Leseprogramme für Blinde müssen sich auf die richtige Abfolge der Überschriften verlassen können, um ihre Leser durch die Webseite zu navigieren.

Du kennst jetzt schon fast alles, was für die Sprache HTML wichtig ist. Klar, es gibt noch viel, viel mehr Tags – von denen ich dir die meisten später noch zeige – aber die funktionieren genauso wie die aus diesem Abschnitt. Aber über die Syntax von HTML gibt es nur noch zwei Dinge zu lernen, und zu denen kommen wir jetzt.

Attribute, leere Tags und Links

Das **World Wide Web** wäre nie so groß und populär geworden, wie es ist, wenn man immer nur einzelne Seiten betrachten könnte. Die große, tolle Neuheit waren die **Links**: Textmarken, die man anklicken kann, um zu einer anderen Webseite oder einer anderen Stelle auf derselben Seite zu gelangen.

[Zettel]
Links werden manchmal auch Hyperlinks genannt, daher stammt das „Hypertext" in HTML.

[Zettel]
Bisher war HTML viel zu einfach, deshalb hier ein bisschen Extraverwirrung: Es gibt auch ein `<link>`-Tag, das hat aber nichts mit diesen Links zu tun. Wir werden es später noch häufig brauchen, aber falls du schon jetzt neugierig bist, gibt es im World Wide Web einige sehr gute HTML-Referenzen, zum Beispiel **https://developer.mozilla.org/en-US/docs/Web/HTML/Reference** (auf Englisch).

Aber auf der Seite herumzuspringen, macht keinen wirklichen Spaß, wenn die Seite zu kurz ist. Zuerst pumpen wir unsere Beispielseite mal ein wenig auf. Das Mittel der Wahl, wenn man schnell und ohne nachzudenken viel Text braucht, heißt schon seit dem späten Mittelalter **Lorem Ipsum**, ein langer, sinnloser Text, nur um den Platz auf der Seite zu füllen. Wir haben es heute natürlich einfacher als ein spätmittelalterlicher Drucker, angefangen damit, dass wir nicht mehrere Seiten Kauderwelsch aus Bleilettern setzen müssen. Heute gibt es im Internet Lorem-Ipsum-Generatoren (zum Beispiel bei **www.lipsum.com**), aus denen wir nur noch ca. 30 Absätze Text kopieren und um jeden Absatz **<p>**-Tags setzen müssen. Oder du startest einfach mit der Datei **`Beispiel_1_3.html`** aus den Beispielen.

[Hintergrundinfo]
Lorem Ipsum wird heute noch von Webdesignern gerne genutzt. Da der Text vollkommen sinnlos ist, versucht das Gehirn nicht, den Inhalt zu verstehen. So kann man sich besser auf das Layout konzentrieren.

Jetzt, wo wir unseren langen Text haben, können wir Links setzen, um innerhalb der Seite hin und her zu springen. Das Tag, um einen Link zu setzen, heißt **<a>**. Das Tag, um ein Sprungziel zu setzen, heißt … auch **<a>**.

Weil das Alphabet nicht genug Buchstaben hatte, um den beiden Funktionen jeweils ihr eigenes Tag zu geben? Zwei Funktionen, ein Tag – das kann nicht gutgehen. Der Browser muss doch unterscheiden können, was ich will.

Und genau dazu brauchen wir eines der zwei Dinge, die es in der HTML-Syntax noch gibt: **Attribute**. Attribute werden in ein **öffnendes Tag** geschrieben und haben einen **Namen** und meistens einen **Wert**. Ein Link-Tag mit Attribut sieht so aus:

```
<a href="#unten" id="meinLink">Link nach unten</a>
```

*1 Name des Tags

*2 Attribute: Das Attribut **href** legt fest, wo der Klick auf einen Link hinspringt.

*3 der Attributwert, in Anführungszeichen

*4 Ein Tag kann beliebig viele Attribute haben. **id** ist ein Attribut, das in fast jedem HTML-Tag erlaubt ist. Es weist dem Tag einen eindeutigen Bezeichner zu, und wir werden gleich viel Spaß damit haben.

[Zettel]
Es kann natürlich nicht jedes Attribut in jedem Tag stehen. Für jedes Tag gibt es eine Liste von gültigen Attributen. Die Referenzseiten helfen auch da weiter, wenn du's genau wissen willst.

Und damit haben wir schon eine Hälfte unseres Problems gelöst. **<a>**-Tags, die das Attribut **href** haben, sind Links. Wenn man darauf klickt, gelangt man zu dem Ziel, das im Attribut **href** angegeben ist.

Das will ich ausprobieren.

Du kannst es gerne versuchen: Wenn du das Beispiel in die lange Beispieldatei vor dem ersten **<p>** einfügst und die Datei im Browser öffnest, dann wird der Text „Link nach unten" von deinem Browser schon als Link dargestellt. Nur wird noch nichts passieren, wenn du darauf klickst.

[Zettel]
In den meisten Browsern werden Links blau und unterstrichen angezeigt.

Warum wird noch ...
oh, klar, es gibt noch kein Sprungziel für den Link.

Genau so ist es. Aber auch das haben wir schnell behoben. Das Attribut dafür heißt **`id`**, und weil ein **Sprungziel** nur eine Marke im HTML ist und keinen Text enthält, nutzen wir hier das letzte bisschen Syntax von HTML aus, das wir noch nicht gesehen haben: **leere Tags**.

```
<a id*1="unten"/*2>
```

*1 **`id`** ist das Attribut, das einen Link zum Sprungziel macht. Es kann später noch viel mehr …

*2 Es muss immer einen **/** geben, um ein Tag zu schließen. Bei leeren Tags wandert der Slash nur an eine andere Stelle.

[Hintergrundinfo]
Früher wurde zur Definition von Sprungmarken **`name`** benutzt, nicht **`id`**. Das funktioniert auch heute noch, aber das **`name`**-Attribut ist in der neuesten Version von HTML eigentlich nicht mehr gültig.

Wenn es sowieso keinen Tag-Body gibt, dann brauchen wir auch kein schließendes Tag, wir packen den Slash einfach ans Ende des öffnenden Tags, damit weiß der Browser, dass es keinen Tag-Body gibt. Es spart zwar gegenüber

```
<a id="unten"></a>
```

nur drei Zeichen ein, aber wie alle Programmierer und Entwickler neigen auch Webentwickler zu **Schreibfaulheit**. Und bei einem **`<a>`** mögen es nur drei Zeichen sein, die man spart, aber bei einem **`<div>`** sind es schon fünf. Leere Tags können einem Webentwickler Jahre seines Lebens sparen!

[Achtung]
Achte genau auf den Unterschied zwischen den Werten für **`href`** im Link und **`id`** im Sprungziel. Das **`href`** beginnt mit einem Hash (#), die **`id`** nicht.

Das Beispiel oben hat schon gezeigt, wie man das Sprungziel definiert.
Jetzt kannst du es ausprobieren.
Wir haben ja unsere lange Datei schon vorbereitet, es wird Zeit, dass wir damit auch etwas tun.

[Einfache Aufgabe]

Öffne die Datei im Editor, und füge oberhalb des ersten **<p>** einen Link ein:

```
...
<body>
  <a href="#unten">Nach unten springen</a>
  <p>Hallo Webwelt!...
```

Jetzt füge unterhalb des letzten **<p>** noch ein Sprungziel ein:

```
  ...</p>
  <a id="unten"/>
</body>
```

Wenn du die Seite jetzt im Browser öffnest, siehst du vor allem anderen den Link „Nach unten springen“. Klickst du darauf, landest du am Ende der Seite. Scroll ruhig noch mal ganz nach oben, und klick noch mal drauf, nur um sicher zu sein. Und um ganz sicher zu sein, mach es gleich noch mal.

[Zettel]

Achte nebenbei mal auf die Adresszeile des Browsers: Wenn du zum ersten Mal auf den Link klickst, wird **`#unten`** an die Adresse angehängt.

Das Scrollen ist ganz schön nervig, du hättest die Beispielseite ruhig was kürzer machen können.

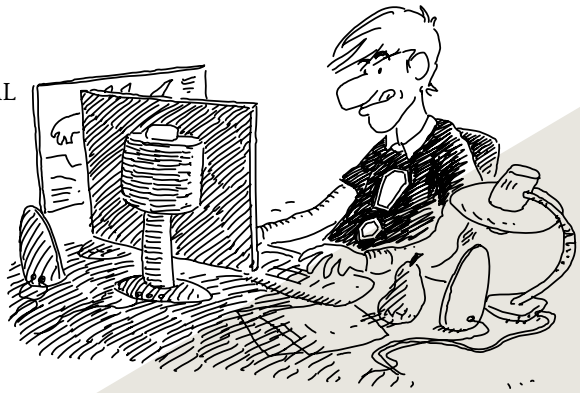
Ja, vielleicht. Aber, Schrödinger, du willst Webentwickler werden. Und ein Webentwickler, der zu faul zum Scrollen ist, der tut was dagegen.

[Einfache Aufgabe]

Wenn du zu faul zum Scrollen bist, dann bau dir einen Link, um von ganz unten wieder nach oben zu springen. Du weißt doch jetzt, wie es geht.

Wenn du unten einen Link und oben eine Sprungmarke eingefügt hast, dann kannst du jetzt von oben nach unten und wieder zurück springen, ganz ohne zu scrollen. **Faulheit ist eine Tugend**.

Falls es nicht auf Anhieb funktioniert, vergleich mal, ob dein HTML diesem hier ähnlich sieht:



```
...
  <body>
    <a href="#unten">Nach unten springen</a>*1
    <a id="oben"/>*2
    <p>
      Lorem ipsum ...*3
    </p>
    <a href="#oben">Nach oben springen</a>*4
    <a id="unten"/>*5
  </body>
...
```

*1 Link nach unten

*2 Sprungziel oben

*3 seitenweise Text, an dem du wirklich nicht entlangscrollen möchtest

*4 Link nach oben

*5 Sprungziel unten

[Achtung]

Du hast es dir bestimmt schon gedacht, aber ich sag es trotzdem noch mal: Mehrere Sprungziele mit derselben **`id`** sind keine gute Idee, das kann nur schiefgehen. Mehrere Links mit demselben **`href`** sind dagegen kein Problem.

Jetzt musst du nicht mehr nach oben und nach unten scrollen, aber es stehen zwei **<a>**-Tags mit verschiedenen Attributen direkt übereinander. Ich weiß nicht, wie es dir geht, aber mich als schreibfaulen Webentwickler stört das schon wieder ein bisschen. Das geht bestimmt kürzer.

[Einfache Aufgabe]

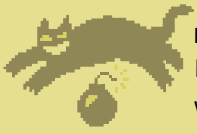
Fasse jeweils die beiden Tags oben und die beiden Tags unten auf der Seite zusammen, so dass sie so aussehen:

```
<a href="#unten"*1 id="oben"*2>Nach unten springen</a>
```

*1 Attribut für **<a>** als Link

*2 Attribut für **<a>** als Sprungziel

Es wird dich jetzt wahrscheinlich nicht überraschen, dass auch das funktioniert. Und es ist wieder weniger zu tippen. Du machst dir ja gar keine Vorstellung, wie viele Einzelheiten in der Webentwicklung durch reine Faulheit motiviert sind.



[Achtung]

In der Theorie können auch andere Tags als Sprungziele benutzt werden, solange sie eine **`id`** haben. In der Praxis gibt es damit aber Probleme im Internet Explorer. So ist es zumindest für den Moment sicherer, nur zu **<a>**-Tags zu springen.

Links zwischen zwei Seiten – über den Gartenzaun

Auf einer Seite von oben nach unten zu scrollen, ist ein guter Anfang, aber auch damit wäre das World Wide Web nie zu einem Erfolg geworden, der Nutzen ist doch sehr eingeschränkt. Aber **<a>** kann noch mehr, viel mehr.
Stell bitte sicher, dass **die beiden Testseiten, die wir anlegen werden, im gleichen Verzeichnis liegen**; sonst funktioniert das, was wir tun wollen, nicht so, wie es soll, und wir sind beide unglücklich.
Für unseren nächsten Trick werden wir jetzt eine Webseite aus dem Browser verschwinden lassen und durch eine andere ersetzen. Ist vielleicht nicht Copperfields Niveau, aber sein HTML ist auch ziemlich mies.

[Einfache Aufgabe]
Erzeuge zwei HTML-Seiten mit den Namen **`links_eins.html`** und **`links_zwei.html`** im gleichen Verzeichnis.
Füge in beiden Seiten einen Link zur jeweils anderen ein, und probier es aus.

Es ist dir vielleicht in der Eile entgangen, aber du hast noch nicht erklärt, wie ich das mache ...

Kleinigkeit, wirklich. Stell dir den einfachsten Weg vor, wie der Link von einer Seite zur anderen aussehen könnte. Vergiss die ganzen komplizierten Adressen, die du im Browser eintippen musst, dazu kommen wir noch früh genug. Was ist die einfachste Methode?

Ich schreibe den Namen der Datei, auf die ich verlinken möchte, ins **href** des Links.

So ist es. Probier es aus. Und **vergiss die Dateiendung nicht**, ohne `.html` am Ende geht's nicht.

[Achtung]
Bei einem Link auf eine andere Seite wird kein Hash an den Anfang des `href` gestellt, die ist nur für Links innerhalb der Seite da.
Immer, wenn du einen Dateinamen angibst, ist es wichtig, auf Groß- und Kleinschreibung zu achten. Ansonsten kann es sein, dass ein Link zwar bei dir funktioniert (wenn du Windows benutzt), aber später auf dem Server nicht mehr.

So sollte ein Link aussehen:

```
<a href="links_zwei.html"*1>Zu Seite zwei</a>
```

*1 einfach nur der Dateiname mit Endung

Damit kannst du jetzt von einer Seite zur anderen springen, das ist endlich das, womit das World Wide Web populär geworden ist.

[Zettel]
Diese Art von Link nennt man einen relativen Link, weil er Dateien relativ zu der Datei sucht, in der sich der Link befindet. Die Zieldatei muss nicht im selben Verzeichnis liegen.
In ein Unterverzeichnis kannst du mit `href="verzeichnis/datei.html"` verlinken, in ein Verzeichnis weiter oben mit `href="../datei.html"`. Warnung für Windows-Benutzer: Benutzt `/`, nicht `\`!

Zumindest fast. Nur auf die Seite neben unserer zu verlinken, ist immer noch nicht alles. Es hilft dir zum Beispiel nicht, wenn du neue Bücher kaufen willst. Es sei denn, du würdest Buchhändler, dann vielleicht, aber sonst nicht. Die meisten von uns brauchen dafür einen Link, der zu einer weiter entfernten Webseite führt. Versuch mal den hier:

```
<a href="http://www.rheinwerk-verlag.de/">Rheinwerk</a>
```

Füg ihn in eine der Beispieldateien ein, und probier ihn aus. Zurück musst du allerdings mit dem Zurück-Knopf des Browsers; Rheinwerk hat zwar viele nette Leute, aber ich konnte sie nicht dazu überreden, einen Link zu unserer Beispielseite auf ihre Webseite zu setzen.

[Achtung]
Damit dieser Link funktioniert, musst du mit dem Internet verbunden sein.

Dieser Wert von **`href`** heißt im Alltag **Webadresse**, technisch nennt man es eine **absolute URL**, und wir werden in Kapitel 2 sehen, was drin steckt. Für den Moment reicht es, wenn du weißt, dass du jede Adresse aus dem Browser kopieren und als **`href`** einfügen kannst, und der Link wird funktionieren.

[Schwierige Aufgabe]

Hier ist eine Kleinigkeit für dich zum Ausprobieren: Füge in einer der beiden Beispieldateien einen Link ein, der in die lange Datei aus dem vorigen Abschnitt führt, ans untere Ende der Datei.

Die beiden Arten von Linkzielen zu kombinieren, war nicht schwierig, oder? Du packst das Sprungziel innerhalb der Seite hinten an den Dateinamen, und schon funktioniert es.

```
<a href="langeseite_mit_links.html*1#*2unten*3">Andere Seite unten</a>
```

*1 Name der Zieldatei

*2 das wichtige und mächtige Hash-Zeichen

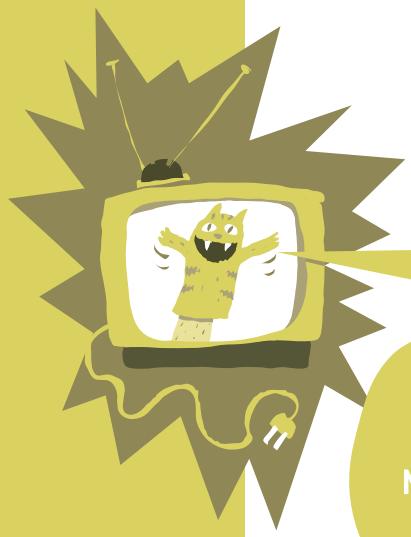
*3 Sprungmarke innerhalb der Datei

[Notiz]

Man will nicht immer, dass sich das Linkziel im Browser öffnet. Manchmal ist ein Link nur dazu da, dass der Benutzer sein Ziel auf seinem Rechner speichert. Das kann er mit Rechtsklick und "Speichern unter" immer selbst bestimmen, aber mit dem Attribut **`download`** kannst du es auch zur Standardaktion des Links machen. Der Wert des Attributs wird als Dateiname vorgeschlagen, den der Benutzer aber selbstverständlich ändern kann.

Was ist eigentlich aus dem Kaffee geworden, von dem wir vorhin gesprochen haben? Inzwischen ist der bestimmt durch?

Das Ziel im Auge – das Attribut target



Jetzt habe ich noch einen letzten Trick mit dem **<a>**-Tag auf Lager. Ich komme mir inzwischen schon vor wie einer von diesen Fernsehwerbeansagern.

ABER DAS IST NOCH NICHT ALLES.

Das <a>-Tag kann nicht nur innerhalb einer Seite von oben nach unten springen, es kann nicht nur zu einer anderen Seite springen, und es kann auch nicht nur beides gleichzeitig.

Nein, Das <a>-Tag kann noch mehr: Es kann auch neue Fenster öffnen. ES IST UNGLAUBLICH!

Danach habe ich aber auch wirklich nichts mehr mit dem **<a>**-Tag. Es gibt zwar noch mehr Attribute, die tun aber nichts Interessantes. Wenn du die Liste mit allen Attributen sehen willst oder wenn du sonst etwas nachschlagen willst, was nicht ins Buch gepasst hat, dann helfen auch da wieder die Referenzseiten. Was uns jetzt noch interessiert, ist das Attribut **`target`**. Bei **`target`** geht es nicht darum, zu welcher Seite der Link führt, sondern darum, in welchem Fenster es aufgeht.

[Zettel]

Benutze **`target`** sparsam. In jedem modernen Browser kann der Leser einen Link im neuen Fenster öffnen, wenn er möchte. Ihm ein neues Fenster aufzuzwingen ist aber meistens eher nervig.

Es gibt zwei Arten, **`target`** zu verwenden, die an dieser Stelle wichtig sind. Die erste ist, den Wert mit **`_blank`** anzugeben:

*1 Der Unterstrich ist wichtig, sonst versteht der Browser es als Namen eines neuen Fensters, wie unten beschrieben.

```
<a href="..." target="_blank*1">Ein Link</a>
```

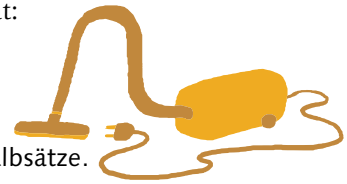
Mit **`_blank`** wird der Link **in einem neuen Fenster** geöffnet. Wird der Link ein zweites Mal geklickt, öffnet sich ein weiteres Fenster, wird ein anderer Link mit **`target="_blank"`** geklickt ebenso.

[Zettel]
Neues Fenster heißt heute immer neues Fenster oder neuer Tab. Als Webentwickler hast du keinen Einfluss darauf, ob ein neues Fenster oder ein neuer Tab aufgeht, das liegt allein am Browser.

In der zweiten Variante gibst du den Namen eines Fensters an – eigentlich eines Frames, aber über Frames, die keine Fenster sind, reden wir ein anderes Mal. Ist noch kein Fenster mit dem angegebenen Namen geöffnet, öffnet sich ein neues Fenster, genau wie mit **`target="_blank"`**. Ein zweiter Klick auf denselben Link oder einen weiteren Link mit demselben **`target`** öffnet die Zielseite aber im selben Fenster wie vorher.

Tinks und Largels

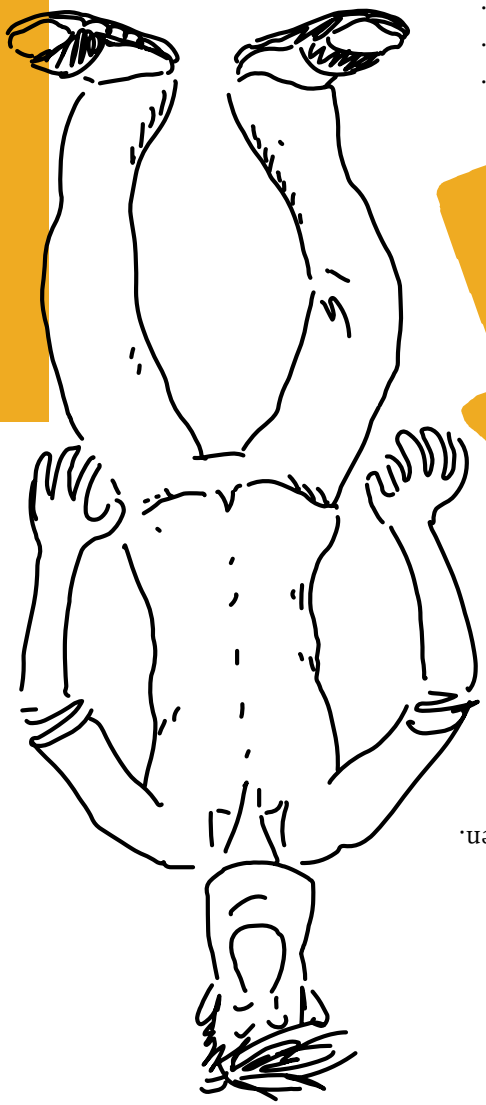
Manche Leute behaupten, Tinks und Largels könne man nicht verwechseln. Zur Sicherheil üben wir das tieber nochmat:



[Einfache Aufgabe]
Nimm einen Bleistift, und verbinde die Halbsätze.

Das **`target`**-Attribut ...
Das <**`link`**>-Tag ...
Das **`<a>`**-Tag ...
Das **`href`**-Attribut ...

... gibt an, zu welcher Seite oder Sprungmarke ein Link führen soll.
... öffnet einen Link in einem anderen Fenster.
... definiert sowohl Links als auch Sprungmarken.
... kann eine absolute oder relative URL enthalten.
... sollte sparsam genutzt werden.
... wird durch ein gesetztes **`id`**-Attribut zur Sprungmarke.
... ist zwingend notwendig, damit ein **`<a>`**-Tag korrekt ist.



Das **`href`**-Attribut gibt an, zu welcher Seite oder Sprungmarke ein Link führen soll.
Das **`target`**-Attribut öffnet einen Link in einem anderen Fenster.
Das **`<a>`**-Tag definiert sowohl Links als auch Sprungmarken.
Das **`href`**-Attribut kann eine absolute oder relative URL enthalten.
Das **`target`**-Attribut sollte sparsam genutzt werden.
Das **`<a>`**-Tag wird durch ein gesetztes **`id`**-Attribut zur Sprungmarke.
Nichts ist zwingend notwendig, damit ein **`<a>`**-Tag korrekt ist.

[Einfache Aufgabe]

Jemand fängt auf der Seite http://www.stapelzeug.de (**index.html**) an, zu lesen.
Er klickt auf jeden Link, der ihm unterkommt. Folge ihm beim Lesen. Leg den Bleistift nicht weg, und verbinde die Links mit ihrem Ziel.

index.html

```
…
<body>
    <p>Willkommen auf der Homepage
von Stapelzeug, dem weltgrößten
Hersteller von Stapelwaren aller Art.
Kisten, Fässer, Mikadostäbchen, wenn
man mehreres davon stapeln kann, dann
stellen wir es her!</p>
<p><a href="produkte.html">Kommen
Sie rein und sehen Sie selbst, dass
wir nicht hochstapeln, sondern hoch
stapeln.</a></p>
</body>
…
```

produkte.html:

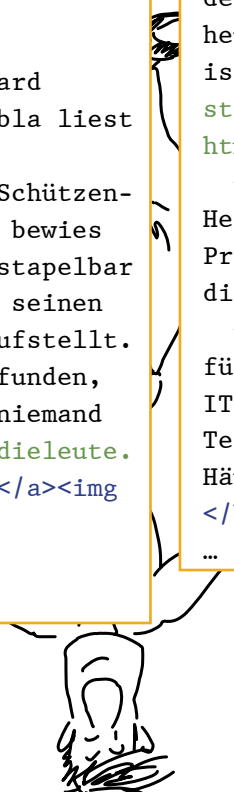
```
<body>
  <p><a href="dieleute.
html">Unser Team</a> <a
href="geschichteundgeschichten.html">Die
Firma</a> <a href="kisten.html">Kisten
und Schachteln</a> <a href="anderes.
html">Sonstige Stapelware</a></p>
</body>
…
```

geschichteundgeschichten.html:

```
…
<body>
  <p><a id="firmengeschichte"/>
Stapelzeug, gegründet von Burkhard
Bossingen im Jahr 1981 bla bla bla liest
das eigentlich jemand?</p>
  <p><a id="stapelrekord">Beim Schützen-
fest in Hintermöhrenhausen 1986 bewies
Burkhard Bossingen, dass alles stapelbar
ist. Hier sieht man ihn, wie er seinen
Rekord im Bierflaschenstapeln aufstellt.
Durch Zufall haben wir herausgefunden,
dass der Knirps im Hintergrund niemand
anderes ist als unser <a href="dieleute.
html#schroedinger">Schroedinger</a><img
src="…" alt="Stapelrekord"></p>
</body>
…
```

dieleute.html:

```
…
<body>
  <p><a id="bossingen"/>Der
Firmengründer und Geschäftsführer,
Burkhard Bossingen. Geboren in grauer
Vorzeit, hält er bis heute das Ruder
seine Firma fest in der Hand. Und
deshalb geht es Stapelzeug auch
heute noch gut. Aber auch Bossingen
ist nicht immer <a href="http://www.
stapelzeug.de/geschichteundgeschichten.
html#stapelrekord">bierernst</a></p>
  <p><a id="heisenberg"/>Helena
Heisenberg, zweite Geschäftsführerin und
Prokuristin, lässt keine Unschärfe in
die Firmenbücher kommen</p>
  <p><a id="schroedinger"/>Das Mädchen
für alles im Büro und außerdem unser
IT-Experte. Wenn sich Schrödinger am
Telefon meldet, dann sind Sie in guten
Händen.</p>
</body>
…
```



index.html ⇨ produkte.html ⇨ dieleute.html ⇨
geschichteundgeschichten.html#stapelrekord ⇨
dieleute.html#schroedinger ⇨ Ende

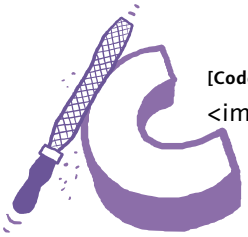
Text war gestern – Bilder

Bilder in ein HTML-Dokument einzufügen, birgt jetzt keine großen Geheimnisse mehr. Ein neues Tag, ein paar Attribute, fertig. Und ein Bild brauchst du natürlich auch. Falls du gerade keins zur Hand hast, nimm einfach das Beispielbild aus dem Download zum Buch.

Wenn man Entwickler malen lässt ...

Das Tag, das du dazu brauchst, heißt **`<img>`**, und die URL des Bildes, relativ oder absolut, steht im Attribut **`src`**.

Red nicht weiter, damit weiß ich alles, was ich brauche. So muss es aussehen, richtig?



[Code bearbeiten]

```
<img src="beispielbild.jpg"/>
```

[Zettel]

<img> ist ein sogenanntes Void-Element, ein HTML-Tag, das niemals einen Tag-Body haben kann. Es ist deshalb falsch, **<img ...></img>** zu schreiben. Außerdem darfst du bei Void-Elementen den schließenden Schrägstrich weglassen. Wieder ein Zeichen gespart =)

Im Prinzip ja, aber es gibt eine Kleinigkeit, die fehlt. **<img> muss** immer die Attribute **src** und **alt** haben, um valide zu sein. In **alt** steht **eine kurze Beschreibung des Bildes**, die zum Beispiel von Lesegeräten für Blinde verwendet wird.

[Begriffsdefinition]

Ein HTML-Dokument ist valide, wenn es allen Regeln für HTML entspricht: Es verwendet nur offizielle Tags und Attribute an Stellen, an denen sie erlaubt sind, es schließt alle Tags korrekt und enthält alle verpflichtenden Tags und Attribute. Valide ist für den Webentwickler koscher, halal und zuckerfrei in einem.

```
<img src="beispielbild.jpg"*1 alt="Krakelige Katze"*2/>
```

*1 Das **src**-Attribut enthält eine URL zu einer Bilddatei. Es gelten dieselben Regeln wie für **href** im **<a>**-Tag, nur dass wirklich ein Bild am anderen Ende der Adresse stehen muss.

*2 **alt** enthält eine kurze, aber aussagekräftige Beschreibung des Bildes.

[Zettel]

Manche Bilder stellen nicht wirklich Inhalt dar, sondern dienen nur als Dekoration. Auch diese Bilder müssen ein **alt**-Attribut haben, aber der Wert darf leer sein: **alt=""**.

[Einfache Aufgabe]

Spätestens jetzt solltest du das **<img>**-Tag in einem der Beispiele ausprobieren.

Es gibt noch zwei weitere Attribute, die für **<img>** gerne und häufig verwendet werden: **height** und **width**. Die beiden Attribute geben **Höhe und Breite des Bildes** an; und auch wenn sie nicht verpflichtend sind, ist es eine gute Idee, sie anzugeben. Sind die Attribute gesetzt, reserviert der Browser an der Stelle des Bildes ausreichend Platz, um das Bild darzustellen, sogar wenn es noch nicht geladen wurde. Der Rest der Seite sieht so aus, als sei das Bild schon geladen. Sind **height** und **width** nicht

gesetzt, wird auch kein Platz für das Bild reserviert; es wird erst eingefügt, wenn es vollständig geladen wurde. Der Rest der Seite wird dann entsprechend angepasst, und das Layout „springt“.

Bevor das Bild geladen wurde ...

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur vel erat at felis venenatis tempor pulvinar vulputate nisl. Nam eget mollis metus. Integer malesuada auctor feugiat. Donec vitae scelerisque arcu. Ut varius, risus venenatis rutrum tempor, tellus lacus blandit orci, ut dapibus nisl ipsum sit amet sapien. Vestibulum quis metus at metus tempor rutrum. Cras cursus interdum sem, ut viverra velit convallis ac. Donec convallis ornare vehicula. Aliquam accumsan convallis tellus in laoreet. Fusce massa sem, ullamcorper eu auctor in, viverra in ipsum. Nulla rhoncus semper varius. Nunc eget felis ut justo viverra fringilla vel sed sapien. Sed dictum, purus sit amet convallis fringilla, sem nibh egestas justo, eu porta leo risus eget lectus. Fusce lacinia mauris sed urna dignissim congue.

... und hinterher

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur vel erat at felis venenatis tempor pulvinar vulputate nisl. Nam eget mollis metus. Integer malesuada auctor feugiat. Donec vitae scelerisque arcu. Ut varius, risus venenatis rutrum tempor, tellus lacus blandit orci, ut dapibus nisl

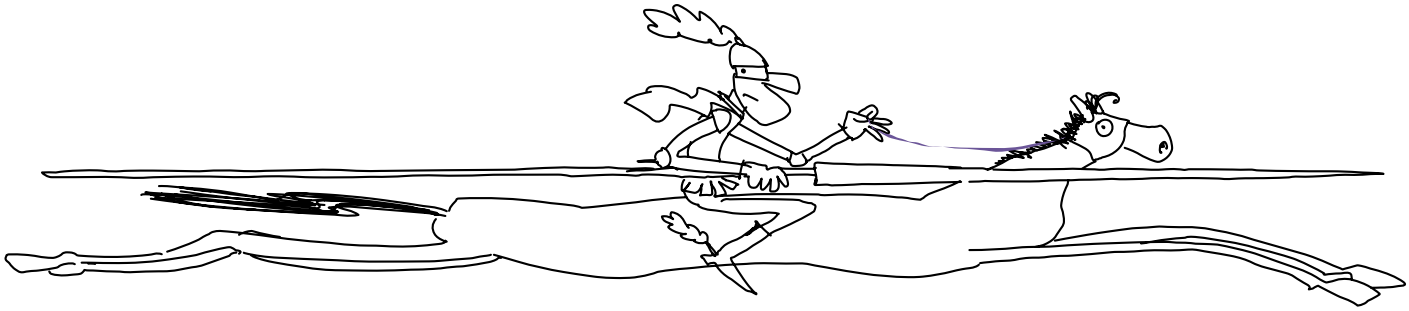
Siehst Du, wie sich der ganze Text verschiebt, wenn das Bild dazukommt? Ziemlich nervig, wenn man diesen Absatz gerade liest.

[Zettel]
Deine Bilder werden noch nicht so schön im Absatz stehen, sondern nur fies in der ersten Zeile. Dagegen können wir bald mit CSS etwas tun.

Und was, wenn ich eine falsche Breite und Höhe angebe?

Dann **vertraut der Browser dir**, dass du schon weißt, was du tust.
Und wenn du es mal nicht weißt, dann vertraut dir der Browser trotzdem: Er staucht und streckt das Bild so, dass es die angegebene Größe hat.

Entweder dieses Bild wurde schlecht gestreckt, oder der Vogel hat mindestens Warp 5 drauf. Vielleicht ist es auch die „Millenium-Drossel"?



[Einfache Aufgabe]
Spiel ein bisschen mit den `height`- und `width`-Werten im `<img>`-Tag, um zu sehen, was ich meine. Wenn du das Beispiel aus dem Download benutzt, versuch zum Beispiel mal `height="50"` und `width="200".`

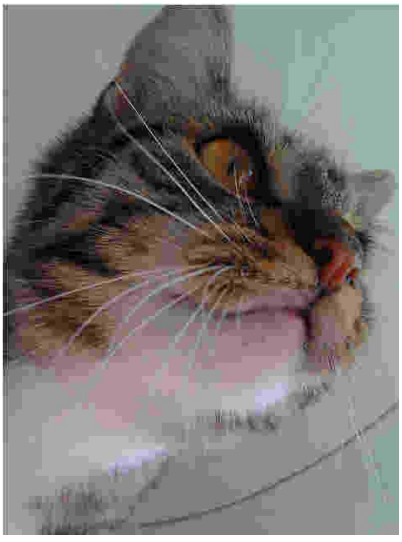
[Achtung]
Es ist keine gute Idee, ein Bild mit `height` und `width` auf eine andere Größe zu zwingen. Vergrößerst du das Bild zu sehr, wird es pixelig, verkleinerst du es, werden unnötig viele Daten übertragen.

Ein paar Worte zu Bildformaten im World Wide Web: Du solltest immer nur **komprimierte Formate** verwenden, um keine unnötigen Daten zu übertragen. Auch im Zeitalter von Glasfaser und LTE sind riesige Bilddateien ein großes DON'T. Gerade viele mobile Benutzer bezahlen nach Datenvolumen, und die werden dir nicht dankbar sein für das 120 MB große Foto auf deiner Seite. Außerdem könnte auch dein

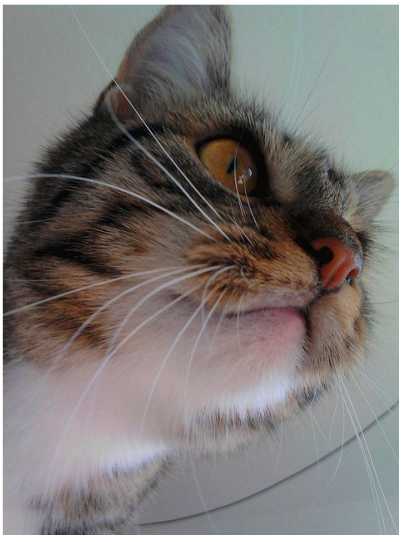
Webhoster nach Datenvolumen kassieren, und dann möchtest **du** dieses Foto nicht auf deiner Seite haben.
Die aktuellen Geschmacksrichtungen für komprimierte Bilder:

- JPEG erzeugt **die kleinsten Dateien**, benutzt aber eine **verlustbehaftete Kompression**; das heißt, JPEG-Bilddateien können Fehler enthalten, sogenannte Artefakte, die in anderen Formaten nicht auftreten. Je kleiner die Bilddatei sein soll, desto schlimmer die Artefakte.
- PNG hat für Fotos eine weniger gute Kompression als JPEG, ist aber dafür **verlustfrei**. Bei anderen Arten von Bildern, vor allem wenn sie wenige verschiedene Farben enthalten, kann die Kompression von PNG sogar besser sein als die von JPEG. Außerdem unterstützt PNG **Transparenz**, JPEG nicht.
- GIF war als Vorgänger von PNG populär, konnte aber nicht mehr als **256 Farben** benutzen. Heute fristet es eher ein Nischendasein mit kurzen **Animationen**: Animierte GIFs sind wesentlich einfacher herzustellen als „echte" Videos.

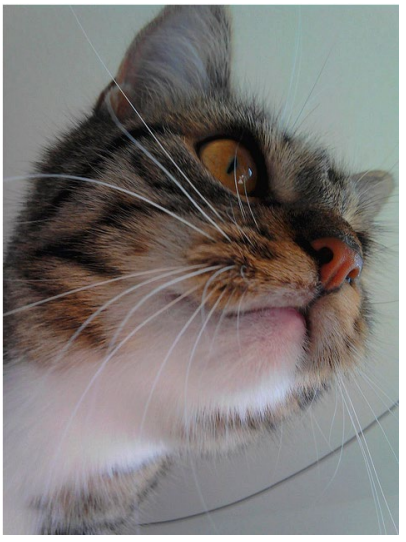
Pixie, die Artefaktkatze



JPEG mit starken Artefakten, dafür nur 8 kB groß



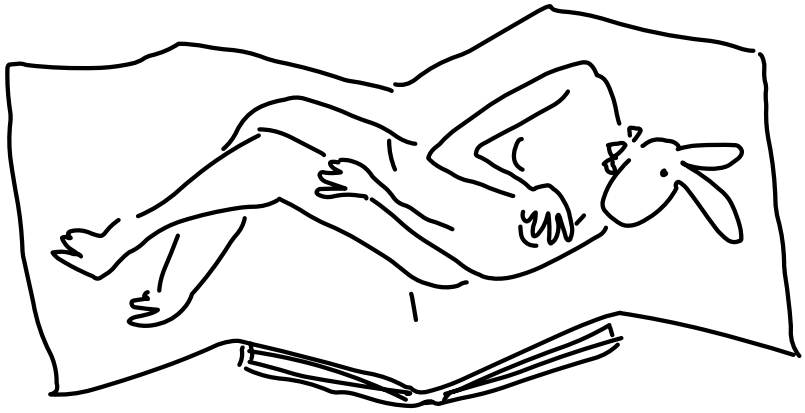
JPEG mit sehr wenigen Artefakten, aber schon 142 kB groß



PNG – keine Artefakte, aber stolze 570 kB groß, zu groß fürs Web!

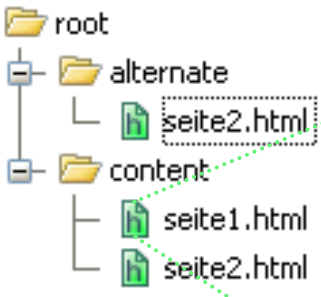
Das sollte man im Kopf haben – mehr vom <head>

Wir haben die letzten ca. 20 Seiten über Dinge gesprochen, die in den **<body>** gehören. Dabei haben wir den anderen Teil des HTML-Dokuments komplett ignoriert – aber Parallelen zu anderen Arten von Lektüre sind rein zufällig.



Alles, was bunt und eindrucksvoll aussieht, steht zwar im **<body>**, aber der **<head>** ist deshalb noch lange nicht unwichtig oder langweilig. Okay, doch. Das meiste, was im **<head>** steht, ist eher langweilig, aber es ist deswegen nicht weniger wichtig. **<title>** kennen wir ja nun schon: Es ist nichts Besonderes, aber eine Seite ohne Titel ist auch doof. Aber was können wir sonst noch im Kopf haben?

Zunächst haben wir da **<base>**. Das **<base>**-Tag bestimmt eine URL, zu der alle relativen Links relativ aufgelöst werden. Klingt sehr abstrakt, aber hier ist ein Beispiel: Nimm an, du hast die abgebildeten Ordner und Dokumente.




```
<html>
  <head>
    <title>Beispiel</title>
  </head>
  <body>
    <a href="seite2.html*1">Wo soll es denn hingehen?</a>
  </body>
</html>
```

*1 Eine relative URL: Welche **seite2.html** ist gemeint?

*1 das **<base>**-Tag ..

*2 ... mit dem Attribut **href** ...

So wie es da steht, führt der Link zu **root/content/seite2.html**, zu dem Dokument, das neben **seite1** steht.
Wenn du jetzt im **<head>** folgende Zeile einfügst

```
<base*1 href*2="../alternate/*3">
```

*3 ... und einem Wert, der auf das Verzeichnis **alternate**, neben **content**, verweist

werden alle relativen URLs, egal, ob von **<img>**, **<a>** oder einem anderen Tag, so behandelt, als läge **seite1** im Verzeichnis **alternate**. Der Link öffnet jetzt **root/alternate/seite2.html**.

Und wozu ist das Ganze jetzt gut?
Ich hätte auch einfach den Link auf die **alternate**-Seite setzen können, wenn ich dahin verlinken will.

In dem kleinen Beispiel hier hast du vollkommen Recht, aber stell dir vor, du hast **Hunderte von HTML-Seiten** in verschiedenen Verzeichnissen, und alle verlinken aufeinander. Und jetzt verschiebst du ein Verzeichnis in ein anderes Verzeichnis. Wenn du auf allen Seiten **<base>** auf das oberste Verzeichnis oder auf deine Domäne gesetzt hast, musst du zwar alle Links ändern, die in das verschobene Verzeichnis führen, aber alle Links aus dem Verzeichnis bleiben gleich.

So weltbewegend klingt das jetzt nicht ...

Ich gebe zu, **<base>** braucht man nicht so oft. Aber es schadet auch nicht, es zu kennen.
Das nächste Tag im **<head>** ist **<meta>**. **<meta>** gibt **Informationen über das Dokument** an, eben sogenannte **Metadaten**. Dazu gehören zum Beispiel der Name des Autors – das wird im Moment meistens deiner sein –, die Kurzbeschreibung des Inhalts und einige mehr.

```
<meta name="author*1" content="Schroedinger*2"/>
```

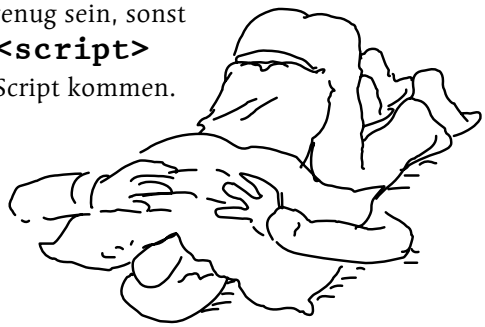
*1 Der Name des Meta-Wertes: Hier geben wir an, wer Autor des Dokuments ist.

*2 der dazugehörige Wert

Bevor du dich beschwerst: Sogar ich finde **<meta>** ein unglaublich trockenes Thema, aber der alte Bossingen wird dir jedes **<meta>**-Tag danken.

Warum? Soll ich ihm erklären, wofür es gut ist, damit es mir dann danken kann?

Er wird's dir danken, weil **<meta>**-Tags **von Suchmaschinen wie Google ausgewertet werden**. Wenn du nach etwas suchst, und beim Suchergebnis steht eine Zusammenfassung der Seite, dann ist das der Inhalt von **<meta name="description".../>**. Mit diesen beiden Tags im **<head>** soll es aber hier erst einmal genug sein, sonst brauchen wir gleich beide noch mehr Kaffee. Über **<link>** und **<script>** sprechen wir sowieso noch ausführlich, wenn wir zu CSS und JavaScript kommen. Für das **<meta>**-Tag gibt es aber noch eine wichtige Funktion.



Andere Länder, andere Zeichen: Character Encoding

Computer können bekanntlich nur mit **Zahlen** umgehen. Trotzdem sehen wir täglich mehr Buchstaben als Zahlen auf dem Bildschirm. Wie kommt also der Computer von der Zahl zum **Buchstaben**? Dafür ist das **Character Encoding** da, eine Tabelle, die zum Beispiel sagt, dass 41 = A ist oder 4D = M. Wäre schön, wenn es wirklich so einfach wäre, oder? Ist es aber leider nicht, es gibt nämlich nicht ein Character Encoding, es gibt Hunderte von den Dingern, wenn nicht sogar noch mehr. Für jedes Schriftsystem, das auf der Welt verwendet wird, gibt es mindestens eins, aber eher fünf oder sechs. Ein altes, ein neues, eins von Microsoft, eins von Apple, eins, um Unterhosen zu beschriften ... Du verstehst schon, was ich meine. Zum Glück gibt es seit einigen Jahren Unicode, eine Sammlung von Character Encodings, deren Ziel es ist, alle Zeichen, die irgendwo auf der Welt verwendet werden, in eine Tabelle zu packen.

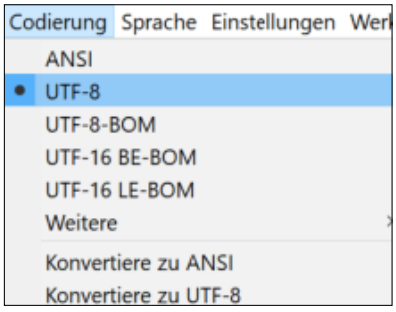
[Hintergrundinfo]
Zeichencodes werden meistens hexadezimal angegeben, also in einem Zahlensystem, das auf der 16 basiert statt auf der 10. Zum Beispiel ist 4D hexadezimal = 77 dezimal, aber du musst die Umrechnung nicht im Kopf beherrschen.

[Hintergrundinfo]
Jedes der Unicode-Encodings enthält dieselbe Sammlung von Zeichen, sie werden nur anders in Zahlen übersetzt. Wirklich kennen muss man nur UTF-8 und UTF-16, die sich darin unterscheiden, wie viele Zahlen für einen Buchstaben verwendet werden.

EIN Character Encoding, sie zu knechten, alle Zeichen zu finden, zusammenzutreiben und an Zahlen zu binden.

Unicode wird von allen Systemen und Browsern unterstützt, du musst nur für zwei Dinge sorgen: Dein Dokument muss **in einem Unicode Encoding gespeichert sein**, und der Browser muss das auch **wissen**.
Teil eins erledigst du in deinem Texteditor. In Notepad++, Geany und Bluefish ist es jeweils sehr einfach, das Encoding einzustellen:

- Notepad++ : Menü „Encoding“, hier wählst du „Encode in UTF-8“ aus.
- Geany: Menü „Document“ • „Set Encoding“ • „Unicode“ • „UTF-8“
- Bluefish: Menü „Document“ • „Character Encoding“ • „UTF-8“



Das Encoding kann man im Menü einstellen.

[Zettel]
Was passiert, wenn der Browser das falsche Encoding zur Anzeige verwendet, hängt davon ab, wie falsch das Encoding ist. Im schlimmsten Fall werden nur wilde Sonderzeichen angezeigt. Oft ist das Encoding aber nur ein wenig daneben, dann ist der Text noch lesbar aber mit Rechtecken oder Fragezeichen gespickt.

Damit ist im Dokument alles klar, jetzt muss der Browser es nur noch richtig lesen. Dafür muss wieder das **<meta>**-Tag ran:

```
<meta charset*1="utf-8*2">
```

*1 Extra zu diesem Zweck hat **<meta>** das Attribut **charset**.

*2 UTF-8 ist das verbreitetste der Unicode-Encodings. **"utf-8"** klein zu schreiben ist hier richtig. Das ist auch eigentlich immer die richtige Wahl für Encodings im Web.

[Zettel]
Webbrowser sind gut im Raten des verwendeten Encodings, falls das **<meta>**-Tag fehlt. Aber warum sollte man sich darauf verlassen, wenn man es auch angeben kann?
Und damit funktioniert das Dokument so korrekt, wie wir es machen können.

Was meinst du damit? Es funktioniert trotzdem nicht überall?

Zumindest nicht sicher. Es kann zum Beispiel sein, dass der Browser die Zeichen zwar richtig dekodiert, sie aber nicht darstellen kann.

Wie kann das passieren? Wenn der Computer weiß, welches Zeichen das richtige ist, dann muss es es doch auch anzeigen können.

Leider nicht, es gibt noch einen Spieler, der beteiligt ist: der verwendete **Font**. Wenn der Computer eine 41 findet, weiß er durch das Character Encoding, dass du ein A meinst. Aber er weiß noch nicht, wie ein A aussieht. Der Font ist eine weitere Tabelle, und mit der Information „dies ist ein A" findet der Computer dort so etwas wie „zeichne einen Schrägstrich nach oben, einen Schrägstrich nach unten und einen waagerechten Strich zwischen den beiden".

[Hintergrundinfo]
Font ist Englisch für Schriftart. Times New Roman, Arial, Comic Sans sind Beispiele für Fonts.

Wenn der Computer die Zahl 16A0 findet (mit Encoding UTF-8), weiß er zwar, dass der Buchstabe Fehu aus dem Germanischen Runenalphabet gemeint ist, aber wenn er keinen Font für Runen kennt, hat er keinen Plan, dass damit dieses Symbol gemeint ist:

ᚠ

Du kannst in HTML auch Zeichen verwenden, die du auf deiner Tastatur nicht findest, indem du **Character Entities** verwendest. Character Entities fangen mit der Zeichenfolge **&#x** an und hören mit einem **;** auf.

&#x*1 03A3 *2 ; *3

*1 Anfang der Character Entity

*2 der Zeichencode (hexadezimal)

*3 Ende der Character Entity

[Zettel]
Du kannst Character Entities auch im Dezimalsystem angeben, dann schreibst du statt **&#x** nur ein **&#** an den Anfang.

Versuch es ruhig mit dem Beispiel, dieses Zeichen sollte auf jedem Computer zu finden sein.
Jetzt hast du mit **<meta>**-Tags und Character Encodings den trockensten Teil von allem hinter dir, versprochen.
Neben den Character Entities mit Zeichencode gibt es auch noch einige **benannte Character Entities**, für diese bleibt **&** am Anfang und **;** am Ende gleich, aber dazwischen kommt der Name. Zum Beispiel codiert **©** das Copyrightsymbol. Es gibt recht viele von diesen benannten Entities, aber in Zeiten des Unicode sind sie immer seltener notwendig. Heute kann man eigentlich alle Zeichen direkt ins Dokument schreiben.

[Achtung]
Es gibt allerdings drei Zeichen, die immer als Character Entity eingegeben werden müssen, sonst kann der Browser die Seite nicht darstellen: **<**, **>** und **&** müssen als **<**, **>** und **&** codiert werden, sonst interpretiert sie der Browser als Teil eines Tags oder einer Entity – mit katastrophalen Folgen.

Denk noch mal drüber nach: Übungen

Und ich hab jetzt die ganze Zeit geredet und geredet, aber Kaffee hab ich immer noch keinen. Lass dir noch mal in Ruhe alles durch den Kopf gehen, worüber wir gesprochen haben, ich verwüste inzwischen deine Küche auf der Suche nach Espresso.
Lass uns mit ein paar einfachen Dingen anfangen, die du immer und immer und immer wieder brauchen wirst. Versuch zuerst, die Aufgaben zu lösen, ohne nachzuschlagen, wie es geht. Aber mach dir keine Sorgen, falls du doch mal zurückblättern musst.
Mit etwas Übung geht das alles, ohne zu spicken.

[Einfache Aufgabe]
Erstelle zwei neue HTML-Dokumente, die jeweils zueinander verlinken.

Soweit kein Problem, oder? Wo ist denn dein Kaffeepulver?

Ich hab nur ganze Bohnen, du wirst mahlen müssen. Aber schau bitte trotzdem mal auf mein HTML: Es funktioniert zwar, aber ich will sicher sein, dass ich nichts vergessen habe.

```
<!DOCTYPE html>*1
<html>*2
  <head>*3
    <meta charset="utf-8"/>*4
    <title>Beispiel</title>*5
  </head>
  <body>*3
    <p>*6
      <a href="andereseite.html">Zur anderen Seite</a>*6
    </p>
  </body>
</html>
```

*1 **DOCTYPE** sollte immer gesetzt sein.

*2 Der gesamte Seiteninhalt wird von **<html>** umschlossen.

*3 **<head>** und **<body>**, die beiden großen Blöcke einer Seite

*4 Das Character Encoding zu setzen, ist nicht zwingend nötig, aber empfohlen.

*5 Den Seitentitel zu vergessen, macht auch keinen guten Eindruck.

*6 und endlich der Link zur anderen Seite

Wenn du diese Elemente alle hast, dann ist deine Seite okay.
Ist die Kaffeemühle im Regal nur Show oder funktioniert die?
Sieht alt aus.

Das ist die richtige. Was mach ich weiter, während du Kaffee kurbelst?

[Einfache Aufgabe]

Nimm eine der beiden Seiten, und füge über dem Link drei Überschriften ein, eine **<h1>**, eine **<h2>** und eine **<h3>**.

Dann füge unter dem Link noch ein Bild ein, nimm ruhig wieder das Beispielbild von vorhin, wenn du kein anderes hast.

Das ist ja einfach. So muss es aussehen.

```
...
  <body>
    <h1>Oberüberschrift</h1>*1
    <h2>Mittelüberschrift</h2>*1
    <h3>Unterüberschrift</h3>*1
    <p>
      <a href="andereseite.html">Zur anderen Seite</a>
      <img src="beispielbild.png" alt="Beispielbild"*2>
    </p>
  </body>
</html>
```

*1 Drei Überschriften. Überschriften gehören nicht zu einem Absatz, deshalb stehen sie außerhalb des **<p>**.

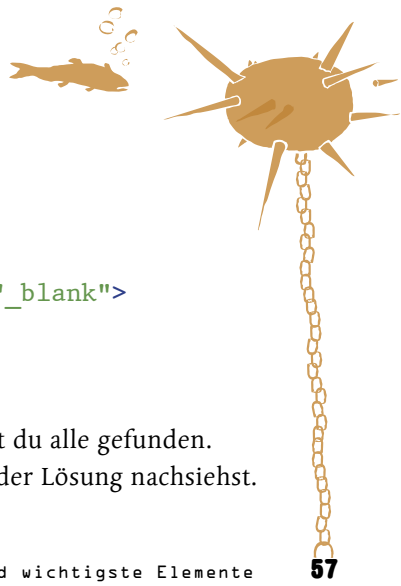
*2 Vergiss bei **<img>** nicht das **alt**-Attribut.

So, der Kaffee brüht. Aber bis er fertig ist, hab ich jetzt noch eine interessantere Aufgabe für dich.

[Schwierige Aufgabe]

Im folgenden HTML sind einige Fehler versteckt. Viel Spaß beim Suchen.

```
<html>
  <head>
    <meta charset="utf-23"/>
    <title>Beispiel<title>
  </head>
  <body>
    <h1>Überschrift
    <p>
      <a href="andereseite.html">Zur anderen Seite</a target="_blank">
    </body>
  </p>
</html>
```



Wie viele Fehler findest du? Wenn es sechs sind, dann hast du alle gefunden. Wenn nicht, dann schau noch mal genau hin, bevor du in der Lösung nachsiehst.

[Belohnung]

Und jetzt ist endlich der Kaffee fertig.
Den haben wir uns ehrlich verdient!


```
*1
<html>
  <head>
    <meta charset="utf-23"/>*2
    <title>Beispiel<title>*3
  </head>
  <body>
    <h1>Überschrift*4
    <p>
      <a href="andereseite.html">Zur anderen Seite</a target="_blank"*5>
    </body>*6
  </p>*6
</html>
```

*1 Der **DOCTYPE** fehlt.

*2 Es gibt zwar mehrere Unicode-Encodings, aber UTF-23 gehört nicht dazu.

*3 Dem schließenden **<title>** fehlt der Schrägstrich.

*4 Das schließende **<h1>**-Tag fehlt komplett.

*5 Attribute können nur im öffnenden Tag stehen.

*6 **<body>** und **<p>** werden in der falschen Reihenfolge geschlossen.

—ZWEI—

Server-kommunikation, Adressen, Standards

Das World Wide Web, unendliche Weiten

Schrödinger erfährt, dass Webseiten auf Servern liegen, die Anfragen aus der ganzen Welt entgegennehmen und als Antwort HTML-Seiten verschicken, und staunt über das weltweite Adressensystem. Am liebsten würde er gleich seine eigene Website ins Netz stellen, aber das übernimmt besser ein Provider. Schrödinger bekommt trotzdem einen eigenen Server, um seine Seiten besser testen zu können. Das Frage-Antwort-Spiel gehört einfach dazu. Danach hört er sich die Geschichte von den Browserkriegen an.

Wo finde ich denn nun meine Seite? Von Webservern und DNS

Okay, ich verstehe jetzt zumindest in groben Zügen, wie ich eine Webseite erstelle. Nicht weiter schwierig. Aber die kann doch bisher nur ich sehen, oder?

So ist es, was du auf deinem Computer schreibst, kannst nur du sehen.

Aber das bringt mir doch gar nichts. Ich mache die Seiten doch, damit die ganze Welt sie sehen kann. Es heißt doch World Wide Web, nicht My Computer Web.

Deswegen sollen die Seiten auch **nicht auf deinem Computer bleiben**. Also, die Beispielseiten vielleicht schon, aber die echten Webseiten, die du schreibst, eher nicht. Damit der Rest der Welt sie sehen kann, musst du sie auf einem **Webserver** zugänglich machen.
Die meisten Leute, sogar Webentwickler, haben den Server nicht zu Hause im Wohnzimmer stehen. Es gibt da natürlich Ausnahmen, aber in dem Fall sind die schon sehr selten. Man geht eher zu einem **Webhoster** und bezahlt dafür, dass der den Server betreibt.

Das klingt teuer ...

Nicht so sehr, um eine normale Website hosten zu lassen, **bezahlst du im Monat weniger als fürs Rauchen oder Kaffeetrinken**. Wenn du mehrere Millionen Besucher im Monat hast, dann wird es natürlich etwas teurer, aber dann kannst du mit deiner Website auch Geld verdienen, und es passt wieder.

Und wenn ich jemanden dafür bezahle, **meine Site zu hosten? Dann kann jeder meine Seiten sehen, indem er `beispiell.html` in seinen Browser eintippt? So einfach kann es doch nicht sein, ich bin ja nicht der einzige mit derart kreativen Namen, wie findet jemand mein `beispiell.html`?**

Du hast Recht, ganz so einfach ist es nicht. Wenn deine Website gehostet wird, brauchst du noch eine **Domäne**, damit sie gefunden werden kann. So was wie „schroedingersseite.de“. Wenn du bei einem Hoster eine Site anmietest, gibt es eigentlich immer ein Paket, in dem eine Domäne mit drin ist. Die wird dann für dich registriert und so eingerichtet, dass sie auf deine Website zeigt. Und dann kann jeder in seinem Browser die URL http://schroedingersseite.de/beispiel1.html eingeben, um genau deine Beispielseite zu sehen. Der Domänennamen (schroedingersseite.de) gibt an, auf welchem Server deine Seite liegt. Die gesamte URL oder **Webadresse** besteht aus dem Domänennamen sowie weiteren Angaben darüber, was du von diesem Server haben willst, in diesem Fall die Seite **`beispiel1.html`**.

Okay ... wie? Er gibt diese URL ein, und sein Browser weiß, wo er diese Seite findet? Ist das im Browser mit drin? Was passiert dann, wenn der Browser älter ist und meine Seite nicht kennt?

[Zettel]
Die Domäne schroedingersseite.de ist nur ein Beispiel. Im Moment ist sie nicht registriert, aber selbst wenn sie es eines Tages sein sollte, hat sie nichts mit uns zu tun.

Der Browser muss deine Domäne nicht kennen, keine Sorge. Er muss nur einen **DNS-Server kennen**, und bei dem fragt er dann nach, wie er deine Website finden kann.

[Begriffsdefinition]
DNS steht für Domain Name System und ist ein System von Servern, deren Aufgabe es ist, Domänennamen in sogenannte IP-Adressen zu übersetzen. Mit dieser IP-Adresse kann dann der richtige Server erreicht werden.

Und der DNS-Server? Woher kennt der meine Website?

Bei dem hat dein Hoster sie eingestellt. Das genau bedeutet „**eine Domäne registrieren**“: einem DNS-Server bekannt machen, dass es zum Beispiel www.schroedingersseite.de überhaupt gibt und wo der Server dazu zu finden ist. Und es ist auch wirklich nur *ein* DNS-Server, dem der Hoster das mitteilt. Alle anderen DNS-Server müssen erst bei diesem einen anfragen.

Aber damit ist doch niemandem geholfen. Jetzt müssen alle DNS-Server wissen, welcher DNS-Server meine Domäne kennt. Dann könnten sie auch gleich meine Domäne selbst kennen.

Zum Glück nicht, DNS ist da nämlich ziemlich **clever**. Es ist **hierarchisch** aufgebaut, also in mehreren Ebenen, und jeder DNS-Server weiß, bei welchem Server der nächsten Ebene mehr Infos zu bekommen sind.

Root DNS Server

2 *„Hier sucht einer www.schroedingersseite.de“*

3 *„Kenn ich nicht, aber der da unten kennt sich mit .de aus“*

1 *„Ich möchte www.schroedingersseite.de lesen“*

4 *„Kennst du www.schroedingersseite.de?“*

Schrödingers Computer

8 *„Der Bayer sagt, du suchst 200.100.xxx.xxx“*

DNS Server bei Schrödingers Internet-Provider

5 *„Nicht echt, aber ich kenn da jemanden“*

DNS Server für .de

9 *„Hey du, gib mir was zu lesen“*

6 *„Man munkelt, du kennst www.schroedingersseite.de“*

7 *„Joa freili, dös lieagt doch oaf 200.100.xxx.xxx“*

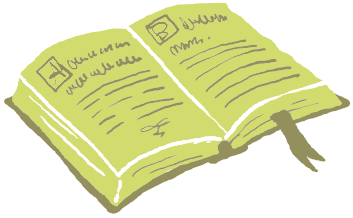
Webserver mit Schrödingers Seite. IP-Adresse: 200.100.xxx.xxx

DNS Server für schrödingersseite.de

... ich habe keine Ahnung, warum der DNS-Server für www.schroedingersseite.de Bayer ist ...

[Begriffsdefinition]

Der .de-Teil eines Domänennamens heißt TLD, das ist kurz für Top Level Domain.



Am Ende kommst du dann endlich bei deinem Webserver an, und der gibt dir die Seite, die du lesen möchtest.

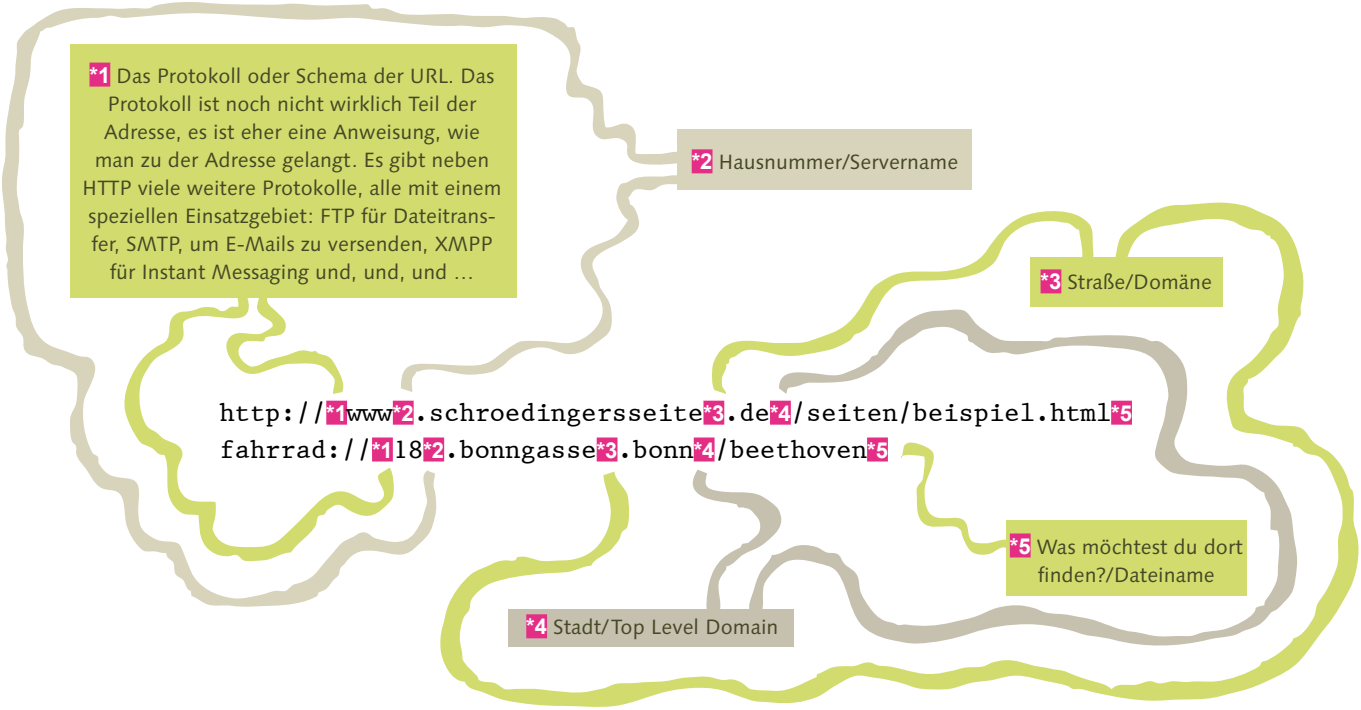
Das sind aber 'ne Menge Schritte, nur um meine Beispielseite zu finden.

Meistens nicht, der DNS-Server mit dem du sprichst, merkt sich alle Ergebnisse, die er schon mal bekommen hat, damit er nicht beim nächsten Mal wieder suchen muss. Ich sag doch: DNS ist clever.

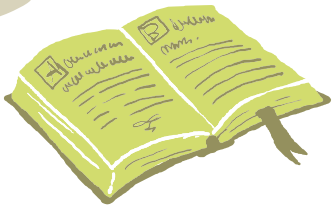
Das ist also der mittlere Teil von diesen „absoluten URLs", von denen du vorhin gesprochen hast. Und was bedeutet der ganze andere Kram drumherum? Ich hoffe, das ist nicht auch so kompliziert wie DNS?

URLs – alles an der richtigen Adresse

URL steht für **Uniform Resource Locator**, also ein einheitliches Format, das angibt, wo eine Ressource gefunden werden kann. In etwa so wie eine Straßenadresse, die angibt, wo ein Gebäude steht.



Fangen wir mal vorne an, beim Protokoll. **HTTP** ist mit Sicherheit das Protokoll, mit dem du am meisten zu tun haben wirst. Es ist das Standardprotokoll für alles, was im World Wide Web getan wird.



[Begriffsdefinition]
HTTP steht für Hypertext Transfer Protocol. Im Namen steckt schon drin, dass damit Dokumente in der Hypertext Markup Language übertragen werden. Aber auch alles andere im Web – Bilder, Videos, Stylesheets, Skripte – wird mit HTTP übertragen.

Ein verwandtes Protokoll ist **HTTPS. Das „S" steht für Secure**. Mit HTTPS werden die übertragenen Daten verschlüsselt, aber sonst bleibt gegenüber HTTP alles beim Alten.

Bisher haben wir in allen Beispielen ein anderes Protokoll verwendet, ohne es anzugeben. Es ist aber in der Adresszeile des Browsers zu sehen: file://. Das File-Protokoll greift auf das lokale Dateisystem zu.

Dann der Domänenname. Den kennst du schon. Und schließlich Pfad und Dateiname. Dieser Teil, `/seiten/beispiel.html` aus dem Beispiel, wird an den Webserver übermittelt, damit der weiß, was du überhaupt von ihm willst.

[Zettel]
Es muss auf dem Server nicht unbedingt eine Datei mit dem angegebenen Namen existieren. Es könnte dort auch ein Programm geben, das aufgerufen wird und die Seite, die du lesen möchtest, erzeugt.

[Hintergrundinfo]
Egal, was du eingibst, der Computer arbeitet nur mit absoluten URLs. Aus einer relativen URL macht der Browser zunächst eine absolute. Fängt die relative URL mit einem Slash an, ersetzt sie Pfad und Dateinamen der aktuellen URL. Ohne den Slash wird die relative URL an den Pfad angehängt.

Aber meistens sag ich ihm das gar nicht. Wenn ich im Browser eine Adresse eingebe, dann hör ich doch nach der Domäne auf.

Dann bekommst du die Eingangsseite des Servers. Die heißt meistens **`index.html`**, aber ansonsten ist nichts Besonderes daran, eben eine HTML-Seite wie jede andere auch.

Und wenn ich eine Datei anfrage, die es gar nicht gibt?

Dann bekommst du den wahrscheinlich häufigsten Fehler im World Wide Web:

404 File Not Found

[Hintergrundinfo]
404 ist ein Fehlercode des HTTP-Protokolls.
Mehr dazu im nächsten Abschnitt.

Es gibt noch ein paar wichtige Elemente in der URL. Die bekommst du häufig nicht zu sehen, sie sind aber trotzdem wichtig. Eins davon kennst du schon vom **`<a>`**-Tag: das Fragment. Mit einem **`#`**, am Ende einer URL angehängt, gibt das Fragment an, zu welcher Sprungmarke des Dokuments gesprungen werden soll.

[Zettel]
Falls es die genannte Sprungmarke nicht gibt, wird das Fragment ignoriert. Es gibt in diesem Fall keinen Fehler.

Und dann gibt es noch ein Element: den **Query String**. Vor dem Fragment, aber nach dem Pfad kann es den **Query String** geben, mit dem du Parameter an den Server übergibst. Für HTML-Dokumente, wie wir sie im Moment erstellen, ist der Query String uninteressant. Aber wenn die URL auf einen dynamischen Inhalt zeigt – sagen wir zum Beispiel die Google-Suche –, dann gibt er an, mit welchen Werten der Server seine Arbeit machen soll.

[Achtung]
Nur Protokoll und Domäne müssen in einer absoluten URL zwingend vorhanden sein, alle anderen Teile sind optional. http://www.schoedingersseite.de?suche=katze#suchergebnis hat zum Beispiel Query String und Fragment, aber keinen Pfad. Trotzdem ist die URL gültig.

*1 der alte, langweilige Kram

*2 Der Query String beginnt immer mit einem **?** ...

```
http://www.google.com/search*1?*2q=schroedinger*3
```

*3 ... und besteht aus Paaren von Parameternamen und Werten, hier wird der Parameter **q** mit dem Wert **schroedinger** übergeben. Mehrere Parameter werden mit einem **&** getrennt.

Diese URL findet übrigens deinen berühmten Namensvetter bei Google. Du weißt schon, den mit der Katze.

Bleib mir bloß weg mit dem! Dieses Schrödinger hat mir meine gesamte Schul- und Unizeit zur Hölle gemacht, der und seine Katze.

Sorry, ich wusste nicht, dass er dir so auf die Nerven geht.

Ach, und noch ein URL-Bestandteil: der **Port**. Der wird gleich wichtig, wenn du deinen ersten Server installierst. Bisher haben wir so getan, als würde jeder Server genau einen Dienst ausführen und genau eine Aufgabe erfüllen. Ein Webserver ist ein Webserver und nichts anderes. Aber das ist so nicht richtig. Ein Server kann fast beliebig viele Aufgaben erfüllen. Der Port ist eine Zahl zwischen 0 und 65535, durch die der Server weiß, mit welchem Dienst du dich verbinden möchtest.

[Zettel]
Um bei unserer Adressanalogie zu bleiben, ist der Port die Türklingel an einem Haus mit 65535 Wohnungen.

Man muss den Port fast nie in einer URL angeben, weil es für die meisten Protokolle einen **Standardport** gibt. HTTP liegt zum Beispiel fast immer auf Port 80. Wenn HTTP auf einem anderen Port liegt, gibt es eigentlich nur zwei Möglichkeiten, warum: Entweder der Server wird nur zur Entwicklung benutzt und soll vom Internet aus nicht erreichbar sein, oder der Besitzer wollte den HTTP-Dienst verstecken, weil er zweifelhafte Dinge tut.
Um einen vom Standard abweichenden Port in der URL anzugeben, schreibst du ihn **mit einem Doppelpunkt hinter die Domäne**.

```
http://www.schroedingersseite.de:80*1/beispiel.html
```

*1 Die Portangabe. In diesem Fall kann sie auch wegfallen, 80 ist der Standardport für HTTP.

[Hintergrundinfo]
Der andere wichtige Standardport für Webentwickler ist 443 für HTTPS. Das File-Protokoll hat keinen Standardport, weil der Zugriff nicht über ein Netzwerk erfolgt.

Ferngespräch für Herrn Web Server – HTTP

Jetzt weißt du, wie die URL zu der Webseite führt, die wir sehen wollen, und wie der Server zu finden ist, der diese Seite hat. Ein letztes Stück im Puzzle fehlt noch: **Wie spricht der Browser mit dem Server**? Ein „Hey, wie geht's?" hilft bei Computern ja meistens nicht weiter.

Zum Glück ist das HTTP-Protokoll dennoch freundlich: Es ist ein Textprotokoll, das heißt, wir können mitlesen, was passiert. Und der Grundaufbau ist sehr einfach. Es gibt keine dauerhafte Kommunikation zwischen Client und Server. Der **Client**, für uns fast immer der Webbrowser, schickt eine **Anfrage** (Request) an den Server und erhält eine **Antwort** (Response) zurück. Transaktion abgeschlossen. Der Request enthält einige Informationen über den Client, auf welche Ressource er zugreifen möchte und was er damit tun will.

[Zettel]
Eine Ausnahme ist das SMTP-Protokoll zum Versenden von E-Mails. Da wird ein freundliches HELO zum Login verwendet.

Dabei ist HTTP **zustandslos (stateless)**. Mit jedem Request müssen wieder alle deine Informationen mitgegeben werden. Ungefähr so, als würdest du im Call Center deiner Versicherung anrufen: Bei jedem Weitervermitteln des Anrufs darfst du wieder deine gesamte Lebensgeschichte erzählen, deine Versicherungsnummer, den Mädchennamen deiner Mutter, die Vorgangsnummer, die dir der andere freundliche Mitarbeiter gegeben hat, … Und dann bekommst du eine einzige Antwort und ein „dafür muss ich Sie weitervermitteln, einen Moment bitte", und alles geht von vorne los. Das ist zustandslos.

Ein einfacher HTTP-Request

*1 Das HTTP-Verb. Was wollen wir mit der Ressource tun? **GET**, „holen“, lädt die Ressource einfach. Fast alles, was wir tun, funktioniert mit **GET**, die anderen Request-Arten brauchen wir erst, wenn wir über Formulare und später über AJAX sprechen.

*2 Der relative Anteil der zu ladenden URL – der Server weiß ja schon, wer er ist, also brauchen wir den Hostname nicht.

*3 Die Protokollversion – hier kann 1.0, 1.1 oder 2.0 stehen, aber als Entwickler von Webseiten ist uns das ganz ehrlich egal. Browser und Server machen das unter sich aus.

```
GET*1 /index.html*2 HTTP/1.1*3
User-Agent: Mozilla/5.0 (Windows NT 5.2; WOW64; rv:15.0)↩
Gecko/20100101 Firefox/15.0.1*4
```

*4 Ein HTTP-Header. Header enthalten zusätzliche Informationen über den Client, darüber, welche Antwort wir gerne hätten, und vieles mehr. Der **User-Agent**-Header teilt dem Server mit, welcher Webbrowser und welches Betriebssystem die Anfrage stellen. Ein Request kann beliebig viele Header enthalten.

Und die Antwort des Servers dazu:

*1 wieder die Protokollversion, falls wir sie inzwischen vergessen haben

*2 Der Statuscode. Hieran erkennt der Browser, ob alles funktioniert hat und, falls nicht, was der Fehler war.

*3 Statusmeldung: dieselbe Information wie der Statuscode, aber für uns Nicht-Computer nett aufbereitet

```
HTTP/1.1*1 200*2 OK*3
Content-Length: 17438*4
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>*5
<html>
…
```

*4 Und auch die HTTP-Response kann Header enthalten. **Content-Length** sagt uns, wie viele Byte die Antwort enthält (ausgenommen Header und Statuszeile). **Content-Type** gibt an, dass in diesem Fall ein HTML-Dokument mit Encoding UTF-8 geliefert wird.

*5 das Dokument, das wir sehen wollen

Statuscode 200, wie im Beispiel, ist für uns immer positiv. **200 bedeutet, dass alles funktioniert hat** und die Ressource, die wir sehen wollen, geliefert wird.

Statuscode	Statusmeldung	Bedeutet
200	OK	Alles super, Inhalt kommt.
302	Found	Es gibt den Inhalt zwar, aber nicht hier. Der Header **Location** enthält die Information, wo das Dokument jetzt zu finden ist – aber normalerweise erledigt der Browser das schon für uns. Status 302 ist der richtigste Weg, den Leser auf eine andere Seite weiterzuleiten (Redirect).
304	Not Modified	Der Inhalt wurde gefunden, aber wir haben die aktuelle Version des Inhalts schon im Cache (welche Version des Inhalts wir kennen, weiß der Server aus anderen Headern, die wir senden).
400	Bad Request	An unserem Request war etwas faul. Das kann ein Header oder Parameter sein, den wir nicht oder falsch gesetzt haben, oder ziemlich alles andere, das dem Server nicht passt.
401	Unauthorized	Nur angemeldete User dürfen diesen Inhalt sehen, und wir sind nicht angemeldet.
403	Forbidden	Jetzt sind wir zwar angemeldet, aber dieser Inhalt ist für uns nicht freigegeben.
404	Not Found	Diesen Inhalt gibt es nicht.
500	Internal Server Error	Der Server ist beim Versuch, den Request zu verarbeiten, vor eine Wand gelaufen. Status **500** wird dir häufiger begegnen, wenn du eigene Server-Anwendungen schreibst, zum Beispiel in PHP oder Java.

[Achtung]

Status **304** solltest du immer im Hinterkopf behalten, wenn du Änderungen an deiner Webseite vornimmst, aber die Seite im Browser sich nicht verändert. Sie könnte einfach noch im Browsercache liegen. In dem Fall hilft es fast immer, mit Strg + F5 neu zu laden.

[Zettel]

Mit HTTP/2 wird nicht mehr zwingend jede Ressource mit einem eigenen Request geladen. Ressourcen direkt mit der Seite auszuliefern ist eine der Beschleunigungen der neuen Version. Wir müssen uns darum aber nicht kümmern, für uns ändert sich nichts.

Aber mit dem Ende der Response ist die Unterhaltung mit dem Server noch nicht abgeschlossen. Es gibt heute kaum noch eine Webseite, die nur aus dem HTML-Dokument besteht. **Dazu kommen Bilder, Stylesheets, JavaScripts und einiges andere**. Das alles sind eigenständige Ressourcen, die mit einem eigenen Request geladen werden müssen. 20 oder mehr Requests für eine Seite sind keine Seltenheit:

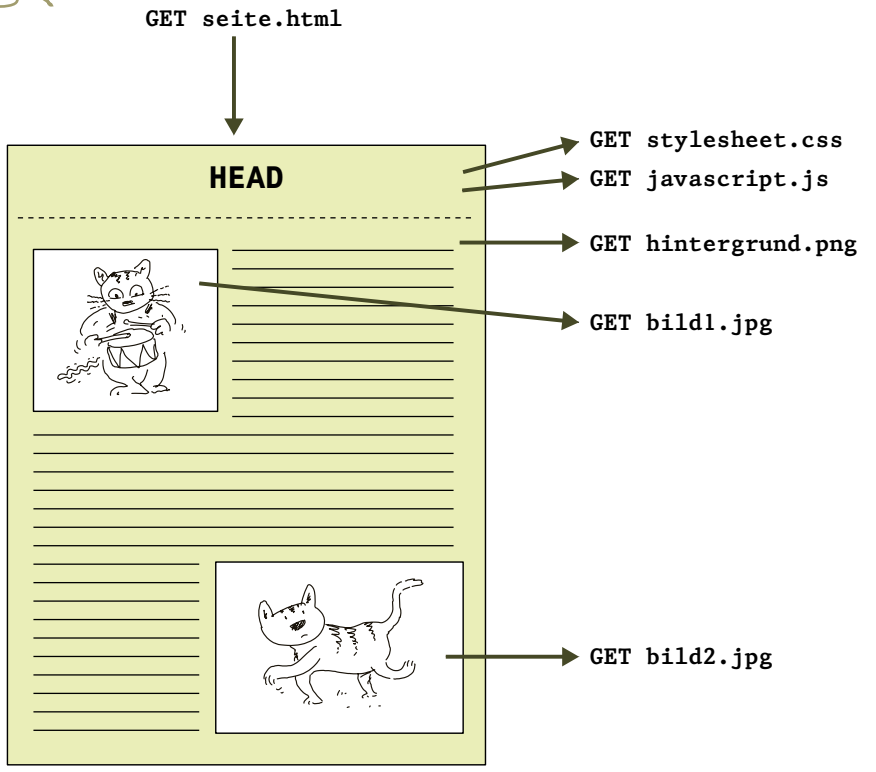
„Hallo, ich hatte gerade schon mal angerufen wegen **`seite.html`**. Gerade fällt mir auf, dass ich dazu **`stylesheet.css`** brauche.“

„Kein Problem, hier ist es, auf Wiederhören.“ *click*

„Hi, ich schon wieder, mir fehlt noch **`hintergrund.png`**.“

„Alles klar, hier ist es, Wiederhören.“ *click*

„Ja, ich schon wieder, jetzt brauche ich noch …“



Konversation mit dem Server

[Hintergrundinfo]

Aktuelle Browser können nur begrenzt viele Requests parallel ausführen, sehr begrenzt viele, zum Beispiel drei. Müssen viele Ressourcen geladen werden, wirkt die Seite dadurch insgesamt sehr träge. Es gibt einige Tricks, um das zu vermeiden, die wir später sehen werden.

Und damit jetzt erst mal genug Theorie.

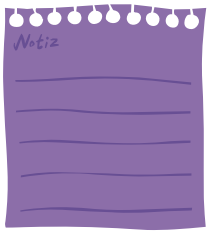
Jetzt wird es ernst – unser eigener Webserver

Bisher hat mit `file://` alles blendend funktioniert. Aber um wirklich Webentwicklung zu betreiben, ist es **einfach nicht dasselbe** wie `http://` mit einem echten Webserver. Und nicht nur ein emotionales „es ist einfach nicht dasselbe". Es funktioniert zwar fast alles auch mit dem File-Protokoll, aber eben nicht alles: Formulare und AJAX brauchen unbedingt einen Server, bei beiden Themen geht es nämlich genau um Kommunikation mit dem Server.

Und auch bei anderen Themen fühlt es sich einfach echter an.

Ich will aber nicht eine eigene Domäne registrieren und für Hosting bezahlen, nur um HTML zu lernen.

Musst du auch nicht, wir können einen Server auf deinem Computer installieren, der zum Lernen alles kann, was wir brauchen. Es ist auch gar nicht schwierig, andere haben sich schon die Arbeit gemacht, das für uns Webentwickler vorzubereiten.



[Notiz]

Wenn der Server bei dir nicht funktioniert oder du ihn nicht installieren willst, kannst du den Rest des Buches auch problemlos ohne benutzen. Nur in den Kapiteln zu Formularen und AJAX kannst du die Beispiele dann nicht ausprobieren.

Na, dann mal los.

Wir benutzen für unsere Beispiele einen **Apache-Server**, sehr einfach zu installieren mit einem Paket, genannt XAMPP, kurz für Cross(**X**)-Platform **A**pache, **M**ySQL, **P**HP and **P**erl.

[Notiz]

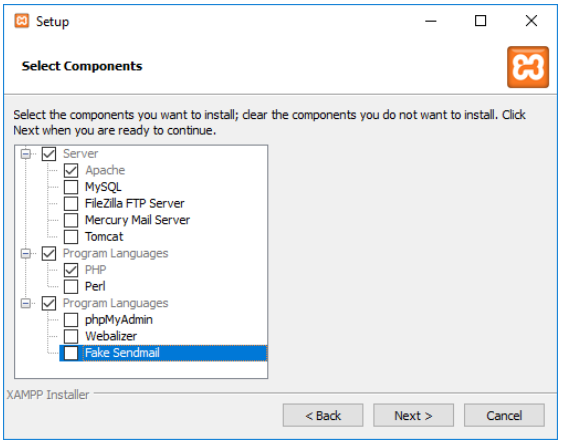
XAMPP ist ein Paket, dass die verbreitetste Softwaresammlung für Webseiten enthält: den Apache-Webserver, PHP für serverseitige Programmierung und die MySQL-Datenbank. Genau diese drei Komponenten kannst du bei jedem Webhoster bekommen.

Für den Moment interessiert uns nur der Apache-Server, aber das Gesamtpaket ist schön einfach zu installieren und zu bedienen. Nur wie es installiert wird, hängt von deinem Betriebssystem ab.

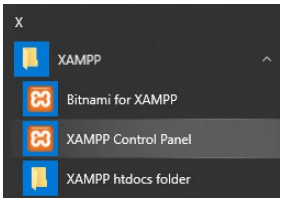
[Notiz]
XAMPP für Windows, Linux und macOS bzw. OS X findest du beim Download zum Buch oder unter **www.apachefriends.org**.

Windows

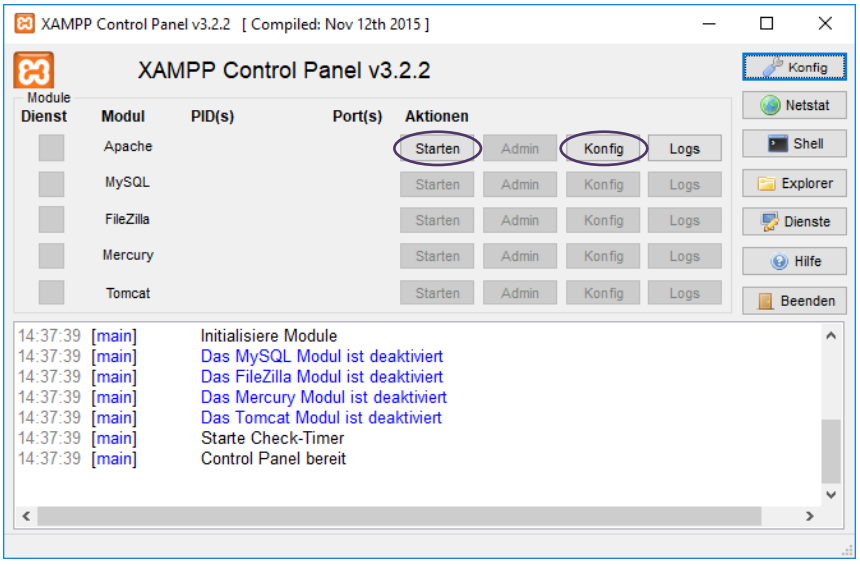
Unter Windows ist die Installation einfach und komfortabel, wie man es von guten Windows-Programmen gewohnt ist: Der **Installations-Wizard** macht die gesamte Arbeit. Als Zielort kannst du auswählen, was du möchtest. Wichtig ist, dass du anschließend keine der Komponenten als Dienst installierst, denn dann werden sie bei jedem Start deines Systems mitgestartet, auch wenn du sie gar nicht brauchst. Dadurch wird erstens dein System langsamer, zweitens ist jeder unnötig gestartete Dienst, egal, wie gut er programmiert ist, ein Sicherheitsrisiko.
Am Ende der Installation bietet der Wizard an, das **XAMPP Control Panel** zu starten. Du kannst es aber auch jederzeit aus dem Startmenü heraus aufrufen.



XAMPP für Windows: Installationsdialog



XAMPP für Windows starten



Das XAMPP Control Panel

Im Control Panel müssen wir eine Einstellung ändern: den Port, auf dem der Apache auf Verbindungen wartet. Das hat nichts damit zu tun, dass wir den Server verstecken wollen oder mit Sicherheit – dafür solltest du einen Virenscanner und eine Firewall einsetzen –, sondern hat einen pragmatischeren Grund: **Einige verbreitete Programme, zum Beispiel Skype, belegen Port 80**. Auf einem Server ist das kein Problem, da läuft – hoffentlich – kein Skype, aber auf einem Computer, an dem du arbeitest, könnte eines dieser Programme installiert sein. Apache könnte diesen Port dann nicht belegen und deshalb auch nicht starten.

Also stellen wir den Port gleich um, das spart uns die Probleme. Klicke dazu im Control Panel auf den Konfig-Button für Apache, und wähle aus den angebotenen Optionen „Apache (httpd.conf)“ aus. Die Konfigurationsdatei öffnet sich im Texteditor. Es gibt hier zwei Stellen zu ändern. Finde zuerst die Zeile

```
Listen 80
```

und ändere sie in

```
Listen 8080
```

Anschließend such die Zeile

```
ServerName localhost:80
```

und ändere auch hier den Port:

```
ServerName localhost:8080
```

Speichere die Datei, und beende den Texteditor. Jetzt ist nur noch dasselbe für HTTPS zu tun. Klicke nochmal den Konfig-Button, aber wähle diesmal „Apache (httpd-ssl.conf)“. Hier sind drei Zeilen zu ändern:

```
Listen 443
```

nach

```
Listen 8083,
```

```
<VirtualHost _default_:443>
```

nach

```
<VirtualHost _default_:8083>
```

und

```
ServerName localhost:443
```

nach

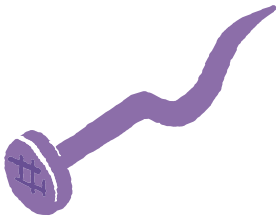
```
ServerName localhost:8083
```

[Achtung]
Ändere sonst nichts an der Konfiguration, sonst kann es sein, dass Apache nicht startet.

Wieder speichern und beenden.

Und dann ist es Zeit, den Server zu **testen**. Klicke den Start-Knopf für Apache. Im unteren Teil des Fensters sollte jetzt die Meldung „Statusänderung erkannt: gestartet“ erscheinen.

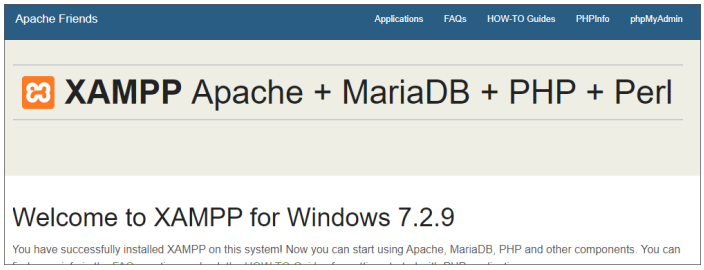
So sieht es nun hoffentlich aus ...



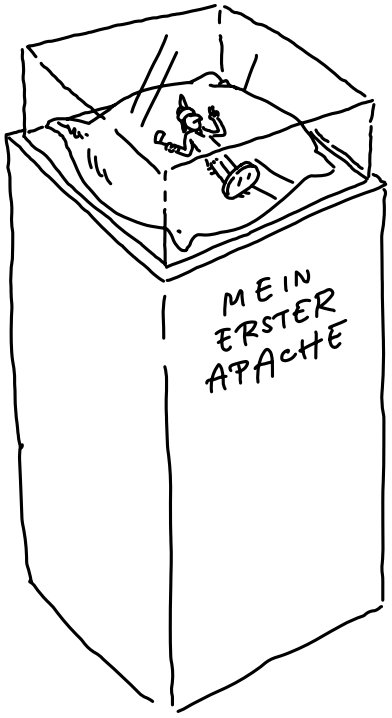
[Fehler/Müll]

Falls sofort anschließend die Meldung „Statusänderung erkannt: gestoppt“ erscheint, sind die gewählten Ports belegt. Versuche in dem Fall, in den beiden Konfigurationsdateien andere Ports einzutragen. Alles oberhalb von 8000 hat eine gute Chance, frei zu sein. Dann musst du in allen Beispiel-URLs statt 8080 und 8083 deine Ports verwenden, sonst ändert sich nix.

Wenn du jetzt im Browser die URL **http://localhost:8080** aufrufst, solltest du eine Seite sehen, die ungefähr so aussieht:



XAMPP erfolgreich installiert



Gratuliere, du hast deinen ersten eigenen Webserver installiert!

Damit hast du den schwierigsten Teil des Buches hinter dir.

Linux

Bei aller Liebe zu Linux, einer der Nachteile des Systems ist, dass man nur selten einen schicken Installations-Wizard bekommt. Und auch kein tolles Control Panel. Deshalb müssen wir jetzt ein wenig im **Terminal** arbeiten. Aber keine Sorge, auch das ist nicht schwierig.

Als Erstes brauchst du ein Terminalfenster. In den meisten Distributionen findest du es im Launcher als Terminal, Terminalemulation oder Xterm.

Für die Installation – und auch zum Ausführen – von XAMPP brauchst du Root-Rechte, deshalb lautet der erste Befehl:

```
sudo su
```

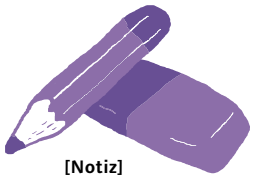
[Achtung]
In den Anweisungen gehen wir davon aus, dass du Linux auf deinem eigenen Computer zu Hause benutzt. Wenn du auf einem Mehrbenutzersystem arbeitest, zum Beispiel an der Universität, kannst du die Installation nicht wie beschrieben ausführen. Frag in dem Fall deinen Administrator, wo du deine Beispiele in einen Webserver legen kannst: Viele Systeme haben ein Verzeichnis für User, um Webinhalte abzulegen.

Dann musst du das XAMPP-Paket entpacken. Der Pfad ist unter Linux festgelegt, es muss nach **`/opt/lampp`** entpackt werden.

```
tar -xvzf xampp-linux-1.8.0.tar.gz -C /opt
```

Jetzt müssen wir auch hier die Ports für den Apache ändern, im Gegensatz zu den Windows-Benutzern aber ohne Control Panel. Doch als Linux-User fürchtet man sich ja nicht vor ein paar Konfigurationsdateien. Außerdem haben wir eine Datei weniger zu ändern als die Windows-Leute. Editiere **`/opt/lampp/etc/httpd.conf`**:

```
geany /opt/lampp/etc/httpd.conf
```



[Notiz]
Du musst natürlich nicht geany benutzen: gedit, vi oder jeder andere Editor funktioniert auch.

Es gibt hier zwei Stellen zu ändern. Finde zuerst die Zeile

```
Listen 80
```

und ändere sie in

```
Listen 8080
```

Anschließend such die Zeile

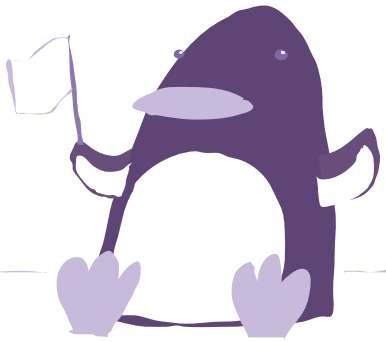
```
ServerName localhost
```

und füge den Port hinzu

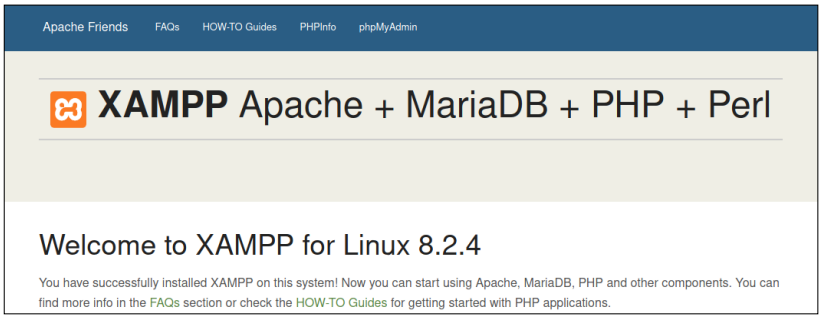
```
ServerName localhost:8080
```

Speichere die Datei, und schließe sie. Jetzt kannst du Apache starten mit dem Kommando

```
/opt/lampp/lampp startapache
```



Und das war's auch schon, du kannst den Server jetzt testen. Öffne im Browser die URL **http://localhost:8080/**, und du solltest in etwa dieses Bild sehen:

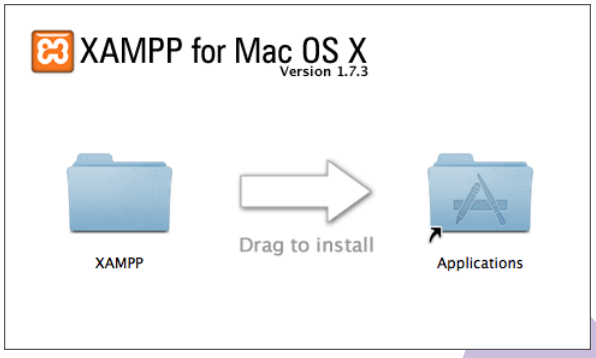


XAMPP erfolgreich installiert

Gratuliere, du hast gerade deinen ersten eigenen Webserver installiert!
Und gleichzeitig hast du auch den schwierigsten Teil des Buches überstanden,
der Rest ist ein Zuckerschlecken.

Mac

Ich bin ja selbst kein Mac-Benutzer, aber ich muss neidlos anerkennen, die Installation ist einfach. Man braucht nicht mal die Tastatur. Öffne die DMG-Datei, und zieh den XAMPP-Ordner nach **`Applications`**. Fertig installiert.



XAMPP installieren für den Mac

Für den nächsten Schritt, die Konfiguration, brauchen wir aber dann doch die Tastatur. **Es tut mir auch wirklich ein wenig leid**. Wir müssen die Datei **`/Applications/XAMPP/etc/httpd.conf`** editieren und zwei Stellen ändern:
zunächst die Zeile

```
Listen 80
```

in

```
Listen 8080
```

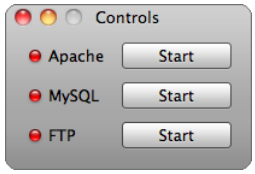
Und anschließend die Zeile

```
ServerName localhost
```

in

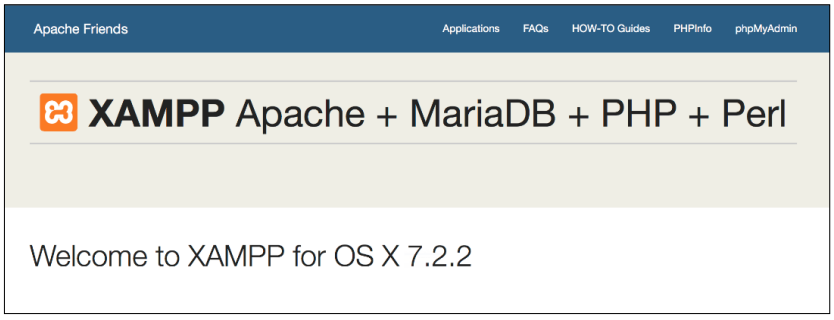
```
ServerName localhost:8080
```

Speichern, und das war's. Den Server zu starten, ist wieder wunderbar einfach und mit der Maus möglich. Starte XAMPP Control aus dem XAMPP-Ordner, und starte Apache per Knopfdruck:



XAMPP Control

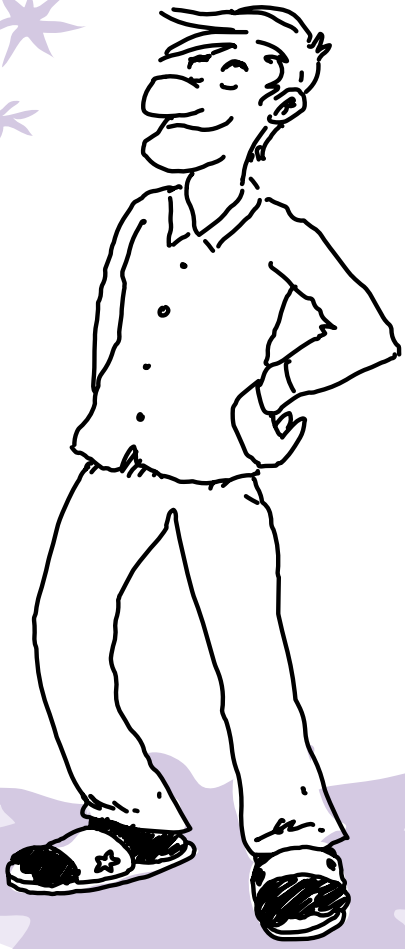
Jetzt kannst du den Server testen, indem du im Browser zu http://localhost:8080 gehst. Du solltest diese Seite sehen:



XAMPP erfolgreich installiert

Gratuliere, dein erster eigener Webserver!

Das war wirklich nicht schwierig, oder?



Hier geht's weiter für alle Systeme

Jetzt sind wir, egal mit welchem System, soweit, dass der Server funktioniert. Eine letzte Sache ist aber noch zu tun: Wir wollen auch unsere **eigenen Seiten** sehen, nicht nur die XAMPP Startseite.

Das ist jetzt leichter getan als gesagt: Im XAMPP-Verzeichnis gibt es, egal auf welchem System, den Ordner **htdocs**. Alles, was in diesem Ordner liegt, ist durch den Apache zugänglich. Du kannst Dateien direkt in htdocs ablegen und dann unter http://localhost:8080/<dateiname> darauf zugreifen. Oder du erstellst einen Ordner und legst die Dateien dort ab, dann musst du den Ordner auch in der URL angeben.

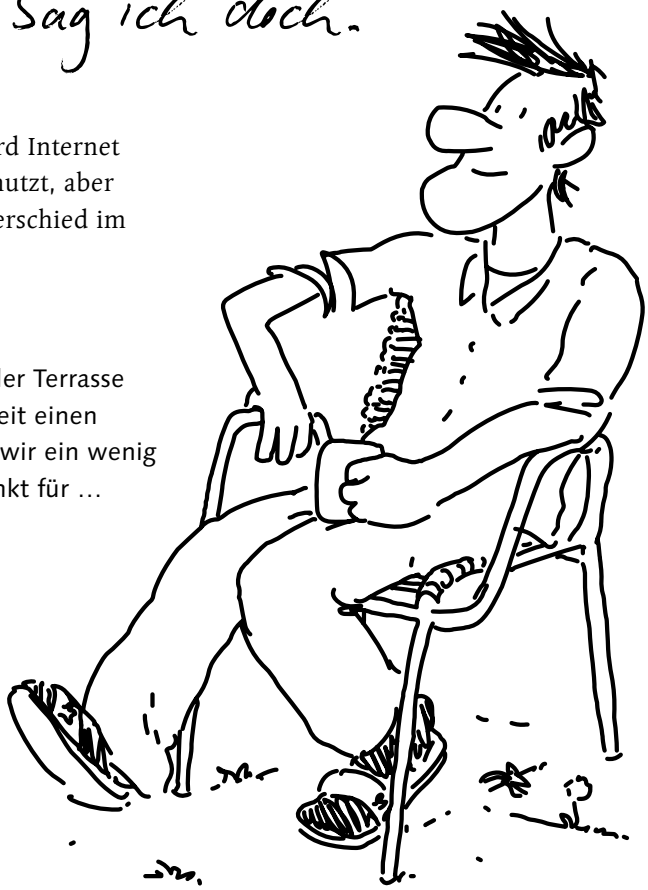
[Einfache Aufgabe]

Versuch mal, in htdocs einen Ordner anzulegen und dann auf den Ordner zuzugreifen, also ohne Dateinamen: http://localhost:8080/ordner/.
Du solltest eine praktische Verzeichnisübersicht sehen, mit der du zwischen den Seiten im Ordner navigieren kannst.

Coole Sache! Jetzt fühle ich mich wie ein richtiger Webentwickler. Meine erste Internetseite, von einem echten Webserver serviert.

Webseite.

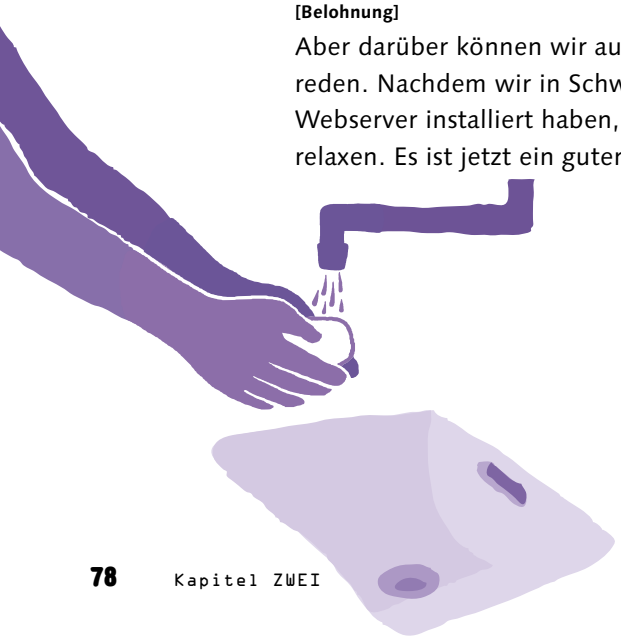
Sag ich doch.



Nein, du hast Internetseite gesagt. Im Alltag wird Internet und World Wide Web heute zwar synonym genutzt, aber als Webentwickler solltest du dir über den Unterschied im Klaren sein.

[Belohnung]

Aber darüber können wir auch auf der Terrasse reden. Nachdem wir in Schwerstarbeit einen Webserver installiert haben, sollten wir ein wenig relaxen. Es ist jetzt ein guter Zeitpunkt für …

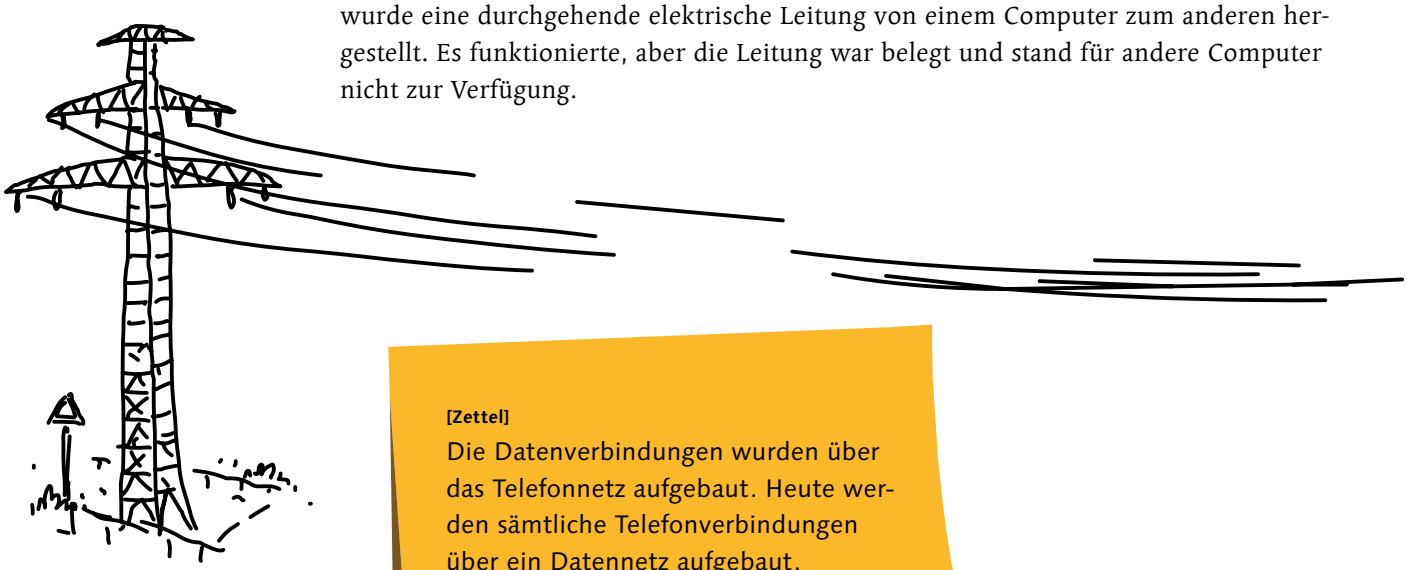


Das obligatorische Geschichtskapitel – die Geschichte des World Wide Web

[Zettel]
Wenn du dich für die Geschichte des World Wide Web nicht interessierst, kannst du diesen Abschnitt ruhig überspringen. Den Abschnitt „HTML ist nicht gleich HTML – eine Sprache, verschiedene Dialekte“ solltest du dann wieder lesen.

Man nehme ein ARPANET und lasse es reifen …

Am Anfang, lange bevor es Webseiten gab oder jemand seinen Computer zu Hause an ein Telefon angeschlossen hat, gab es das **ARPANET**. Das ARPANET wurde von der Advanced Research Projects Agency (ARPA) des US Verteidigungsministeriums entwickelt, und es war damals ein revolutionärer Schritt: Es war das erste **paketbasierte** (packet switched) Computernetzwerk. Es war auch vorher möglich, Computer miteinander zu verbinden, aber die Verbindungen waren leitungsbasiert (circuit switched): Es wurde eine durchgehende elektrische Leitung von einem Computer zum anderen hergestellt. Es funktionierte, aber die Leitung war belegt und stand für andere Computer nicht zur Verfügung.



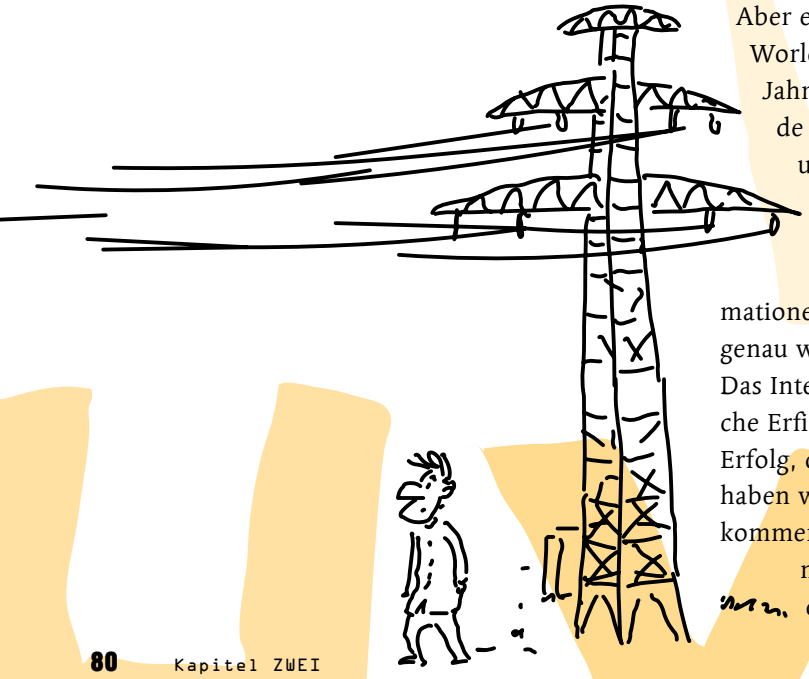
[Zettel]
Die Datenverbindungen wurden über das Telefonnetz aufgebaut. Heute werden sämtliche Telefonverbindungen über ein Datennetz aufgebaut.

Die revolutionäre Idee des ARPANET war es, die zu übertragenen Daten in Pakete zu zerlegen und jedes Paket mit einer Empfängeradresse zu versehen, **genau wie ein Postpaket**. Dadurch war es plötzlich nicht mehr nötig, eine Leitung zwischen genau zwei Computern zu haben: Eine Leitung konnte jetzt Pakete für verschiedene Computer transportieren, und anhand der Empfängeradresse konnte es zum richtigen geleitet werden. So wurden große Computernetzwerke überhaupt erst realisierbar. Das ist zwar nur die ganz grobe Erklärung eines paketbasierten Netzwerks, aber für unsere Zwecke reicht es. Es gibt ganze Bücher nur zu diesem Thema.

[Zettel]
Die erste Nachricht im ARPANET wurde am 29.10.1969 um 10:30 von einem UCLA Studenten gesendet. Die Nachricht lautete „lo“. Es hätte eigentlich „login“ werden sollen, aber nach dem zweiten Zeichen brach die Verbindung ab. Ich werde mich nicht mehr über Verbindungsabbrüche beschweren.

Nachdem anfangs nur vier Institutionen an das ARPANET angeschlossen waren, wurden es schnell mehr, vor allem Universitäten und Forschungsinstitute. Mit dem Wissen, das aus und mit ARPANET gewonnen wurde, wurde 1982 eine Sammlung von Protokollen mit dem Namen Internet Protocol Suite standardisiert. Diese Protokolle sind bis heute im Einsatz und halten das weltweite Computernetz *Internet* zusammen.

… rühre etwas Hypertext unter …



Aber es sollte immer noch einige Zeit dauern, bis das World Wide Web dazukam. Während der 1980er Jahre wurde das Internet weiter ausgebaut und wurde schnell seinem Anspruch gerecht, Computer und Menschen in aller Welt zu verbinden. Dienste wie E-Mail und Newsgroups gab es schon, auch Datentransfer von einem Computer zum anderen war natürlich möglich, aber Informationen waren noch immer unabhängig voneinander, genau wie einfache Textdateien im Dateisystem.
Das Internet war zweifellos schon alleine eine erfolgreiche Erfindung, aber es hatte nicht den explosiven Erfolg, den das World Wide Web kurze Zeit später haben würde. Wie so oft musste eine zweite Idee dazukommen: **Hypertext**. Die Idee, Dokumente durch Links miteinander zu verknüpfen, war schon lange vor dem World Wide Web aufgekommen. Das Wort

Hypertext wurde dafür zum ersten Mal 1963 vom amerikanischen Soziologen und Philosophen Ted Nelson verwendet. Der hatte das Konzept für sein Projekt Xanadu entworfen, ein Projekt, das nie wirklich vom Boden abhob und vom World Wide Web verdrängt wurde, noch bevor es vollständig umgesetzt war. Unter anderem sah Xanadu vor, dass alle Links bidirektional sein sollten, also Seiten immer in beide Richtungen verbinden – man hätte für jeden Link bei der Gegenseite anfragen müssen.

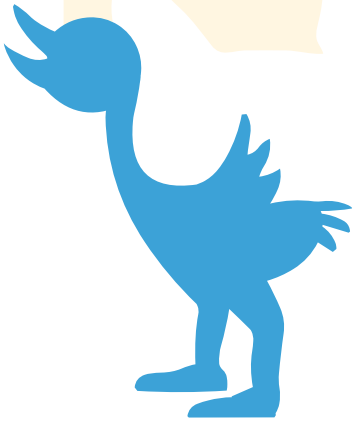
[Zettel]
Obwohl das WWW vieles von dem umsetzt, was das Projekt Xanadu schaffen wollte, sprach sich Ted Nelson des Öfteren gegen das WWW aus. Er sieht es als übermäßige Vereinfachung seiner Idee.

Zusammen brachte die beiden Ideen Tim Berners-Lee. Tim war zu der Zeit am Forschungszentrum **CERN** beschäftigt, dem damals größten Internetnode Europas.

Das CERN macht also mehr, als nur unschuldige Teilchen gegeneinanderzuschleudern.

Er schrieb den ersten Projektvorschlag für das World Wide Web 1989 und überarbeitete ihn 1990 zusammen mit Robert Cailliau bis zu einem Stand, der von seinem Chef, Mike Sendall, akzeptiert wurde. Bis August 1991 entwickelte Tim den ersten Webserver (CERN HTTPd), den ersten Webbrowser (der gleichzeitig auch der erste Editor für Webseiten war), die Sprache HTML, das Protokoll HTTP und die URL. Die Sommerabende in der Schweiz müssen ziemlich langweilig gewesen sein … Am 6. August 1991 ging die erste Website des World Wide Web online. Es ist nicht überliefert, wie er diese Neuigkeit verbreitete. Twitter gab es ja noch nicht.

„Hey zusammen, habe grade die erste Website der Welt online gestellt. URL ist zu lang. #WWW“



[Zettel]
Tim Berners-Lee ist noch heute Direktor des World Wide Web Consortium (W3C), der Organisation, die Standards für das WWW definiert. Er wurde auch von Queen Elizabeth II. zum Ritter geschlagen.

W3C

World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.
Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project,
Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

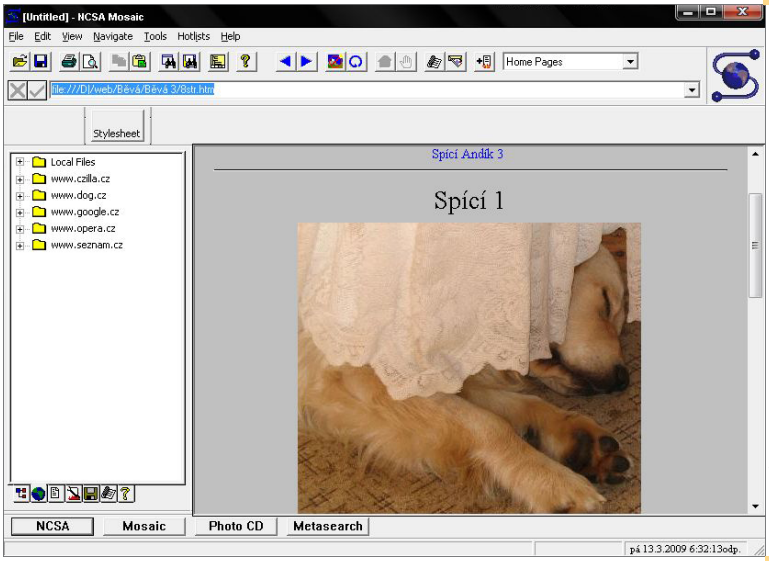
What's out there?
Pointers to the world's online information, subjects , W3 servers, etc.
Help
on the browser you are using
Software Products
A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library)
Technical
Details of protocols, formats, program internals etc
Bibliography
Paper documentation on W3 and references.
People
A list of some people involved in the project.
History
A summary of the history of the project.
How can I help ?
If you would like to support the web..
Getting code
Getting the code by anonymous FTP , etc.

Die erste Webseite, in einer Version von 1992. Die erste Version ist wohl für immer verloren.

… und köchle alles, bis es bunt wird

Damit war das World Wide Web erfunden. Zwei weitere große Schritte sollten noch folgen, bevor es zur Webexplosion kam. Am 30. April 1993 erklärte das CERN, dass die Nutzung der WWW-Dienste jedem offenstehen soll, ohne eine Gebühr zu erheben.
Damit ist zwar nur die Nutzung der Technologie gemeint, nicht der Zugang, aber trotzdem war dies ein großer und wichtiger Schritt für das Web.
Ebenfalls 1993 wurde an der University of Illinois der erste grafische Webbrowser entwickelt: **Mosaic**. Mosaic war der letzte Schritt zum World Wide Web, wie wir es heute kennen. Dinge wie eingebettete Bilder und anklickbare Links, die heute selbstverständlich sind, wurden in Mosaic zum ersten Mal umgesetzt.

Aber bunte Bilder und klickbare Links waren nicht der Hauptgrund, warum Mosaic das Inter-

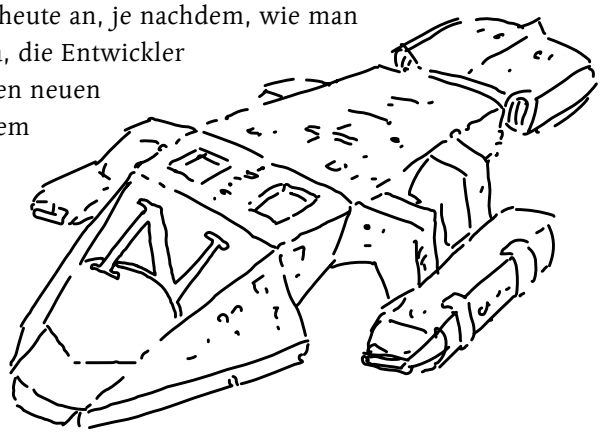


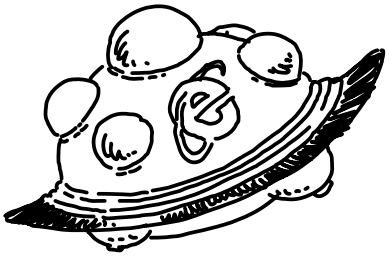
net voranbrachte. Mosaic war auch der erste verbreitete Browser für Windows. Frühere Browser waren nur für Unix-Systeme geschrieben, die zu der Zeit allein in Universitäten und großen Firmen existierten. Das einzige verbreitete Betriebssystem für Heimcomputer war Windows, damals in Version 3.1. Erst durch Mosaic wurde das World Wide Web für Windows-Computer zugänglich. Und erst durch das World Wide Web wurde das Internet für viele Benutzer zu Hause interessant. Das World Wide Web und Mosaic waren treibende Kräfte hinter dem Boom des Internets, dank dem wir heute in fast jedem Haushalt einen Internetanschluss haben.

[Zettel]
Mosaic war nicht der erste Browser für Windows, die Ehre gebührt Cello, der aber nicht dieselbe Verbreitung fand wie Mosaic.

Das Ende von Mosaic und der erste Browserkrieg

Mosaics Erfolg war nur von kurzer Dauer – oder er dauert bis heute an, je nachdem, wie man es sieht. Noch 1994 gründeten Marc Andreessen und Eric Bina, die Entwickler von Mosaic, die Mosaic Communications Corporation, die einen neuen Browser vorstellte: Mosaic Netscape. Nur einen Monat, nachdem Mosaic Netscape verfügbar war, änderten sie den Namen der Firma und des Browsers, um Konflikten um die Rechte am Namen Mosaic aus dem Weg zu gehen. Vom 14. November 1994 an produzierte die Netscape Communication Corporation den **Netscape Navigator**. Eine Seite des Browserkriegs war angetreten, auch wenn noch keiner ahnte, dass der Krieg kommen würde. Aber zu einem Krieg gehört mehr als ein Teilnehmer. Zumindest meistens, Fehler kommen vor.





Es gab zu der Zeit eine Flut von neuen Webbrowsern, aber die meisten waren nie verbreitet genug, um wichtig zu sein. Nur ein Browser wurde einflussreich genug, um auf den Schlachtfeldern des ersten Browserkriegs gegen Netscape Navigator anzutreten: **Microsoft Internet Explorer**. Die erste Version erschien 1995 als Teil des Windows 95-Plus!-Pakets.

[Zettel]
Ich fühle mich richtig alt, dass ich mich daran noch selbst erinnern kann …

Es folgte eine Zeit, in der beide Seiten durch neue Features in ihren Browsern versuchten, den Konkurrenten auszustechen. Einige dieser Neuerungen kann man heute nur noch als **Blödsinn** betrachten. Nein, eigentlich konnte man das auch damals schon. Es gab zum Beispiel die Tags `<blink>` und `<marquee>`, die Text blinken oder von links nach rechts durchlaufen ließen. So nützlich wie ein Lochstecher für Frischkäse.

Andererseits wurde in dieser Zeit auch JavaScript entwickelt, zunächst von Netscape, aber wenig später auch von Microsoft (unter dem Namen JScript). Andersherum war Internet Explorer der erste Browser, der 1996 CSS unterstütze. So ein bisschen. Nicht viel. Aber CSS.

[Zettel]

Die beiden großen Browser waren nur selten kompatibel zueinander und häufig auch nicht zum vom W3C gesetzten Standard. Tags wurden nicht unterstützt oder anders interpretiert, und Skripte, die in beiden Browsern funktionierten, waren so gut wie unmöglich. Fast jede Website hatte zu dieser Zeit eine Angabe, mit welchem Browser sie am besten betrachtet wird.

Dieses Buch wird am besten mit offenen Augen gelesen.

Es ist heute schwer zu glauben, aber während der Browserkriege war lange Zeit Microsoft die **Rebellentruppe**, die gegen Netscapes Übermacht kämpfte. Microsoft war zwar auch damals schon ein riesiger Konzern, während Netscape zu seinen größten Zeiten nur ca. 2.500 Mitarbeiter hatte, aber lange Zeit hatte der Netscape Navigator einen Marktanteil von mehr als 80 %. Also tat Microsoft das, was die Unterlegenen schon immer taten: suchen, wo sie einen Vorteil hatten, und diesen ausnutzen, so gut es nur geht.

Und Microsoft hatte einen riesigen Vorteil: **die Kontrolle über mehr als 90 % aller Heimcomputer**. Ab 1997 wurde mit dem Internet Explorer 4 dieser Vorteil ausgenutzt. Der Internet Explorer 4 wurde mit jeder Windows-Installation mitgeliefert. Niemand musste mehr die Entscheidung treffen, welchen Browser er benutzen wollte, es war schon einer da.

Für Netscape war das der Todesstoß. Microsoft hatte riesige Einnahmen aus anderen Quellen – Windows, Office usw. – und hatte keine Probleme damit, den Internet Explorer zu verschenken. Netscapes einzige Geldquelle war dagegen der Verkauf von Browsern an Firmenkunden – für Privatnutzer war auch Netscape immer kostenlos –, und diese Einnahmen versiegten nun schnell.

Na, das sind ja Geschäftspraktiken ...

Die Unterschiede zwischen den Browsern waren nicht so groß, dass Firmen bereit waren, für den Netscape-Browser zu bezahlen, anstatt den Internet Explorer kostenlos zu benutzen. Microsofts Strategie war erfolgreich, die Firma Netscape wurde 1998 von AOL aufgekauft. Die einzige neue Netscape-Version unter AOL erschien im Jahr 2000, nach zwei Jahren Entwicklungszeit, aber trotzdem bevor sie reif war. Netscape 6 war von Abstürzen und Fehlern geplagt, und selbst eingefleischte Netscape-Benutzer wendeten sich ab.

[Zettel]

Ein mehr als drei Jahre dauernder Prozess in den USA sollte zwischen 1998 und 2001 klären, ob es legal war, dass Microsoft seinen Browser mit dem Betriebssystem bündelte. Es kam nie zu einem Urteil. Man einigte sich darauf, dass alle von Internet Explorer genutzten Schnittstellen auch für andere Browser geöffnet würden, aber es gab keine Entscheidung, ob die Bündelung legal war oder nicht.

Microsofts Monopol und der zweite Browserkrieg – der Rote Panda schlägt zurück

Die Firma Netscape war tot, Microsoft hatte den ersten Browserkrieg gewonnen. **2002 hatte Internet Explorer 6 einen Marktanteil von 96%**. Eine düstere Zeit, denn ohne ernstzunehmende Konkurrenz hatte Microsoft keinen Grund, den Internet Explorer weiterzuentwickeln. Es sollte bis 2006 dauern, bevor eine neue Version den Internet Explorer 6 aus dem Jahre 2001 ablöste. **Die technische Entwicklung stand still**.

[Zettel]
IE 6 wird von den meisten Webentwicklern als schlimmster Fluch aller Zeiten angesehen: Obwohl er sämtliche Standards gekonnt ignorierte, war er sehr lange ein verbreiteter Browser. Wollte oder musste man ihn unterstützen, musste man mindestens den JavaScript-Code komplett zweimal schreiben.

Dass Microsofts Monopol nicht ewig bestehen konnte, ist keine Überraschung. Woher der Monopolbrecher kam, schon eher. Der Gegenschlag, der Auftakt zum zweiten Browserkrieg, erfolgte durch Netscape **von jenseits des Grabes**. Nach einigen Jahren Arbeit in fast volkommener Unbekanntheit veröffentlichten sie den Programmcode als Open-Source-Projekt und übertrugen die Pflege der eigens gegründeten Mozilla Foundation. Mozilla brachte die erste Version von ~~Phoenix Firebird~~ Firefox heraus.

Von seinem Debüt an bis 2010 hat Firefox stetig Marktanteile gewonnen, seit 2010 ist sein Anteil in etwa stabil. Der Aufstieg von Firefox hat auch Microsoft dazu gezwungen, den Internet Explorer wieder weiterzuentwickeln, wollten sie nicht komplett aus dem Markt gedrängt werden. Mit steigender Versionsnummer gelang es dem IE sogar, seinen Ruf als Folterinstrument für Webentwickler wieder abzulegen, den er sich mit dem IE6 hart erarbeitet hatte. Er wurde bis Version 11 weiterentwickelt, inzwischen ist er aber weitestgehend vom Markt verschwunden. Neuere Windows-Versionen haben Microsofts neuen Edge-Browser, ein für Webentwickler viel freundlicheres Stück Software.