

Komplett  
in Farbe

Kai Günster

Schrödinger lernt HTML5, CSS3 und JavaScript

Günster

2009-0



- Schreibe Webseiten für einfach alles, was einen Bildschirm hat
- Wappne dich mit AJAX und Objekten und sprich mit den Servern des weltweiten Webs

► Touch, Gesten, Video & Audio: HTML5 und CSS3 für alle Sinne

Galileo Computing



# Schrödinger lernt HTML5, CSS3 & JavaScript

Das etwas andere Fachbuch



MIT TALENT, KATZEN-  
PHOBIE UND LÄSSIGEM  
SCHÜTTWERK BESTACH  
SCHRÖDINGER DIE GALILEO-  
JURY, FÜR IHN GEHT  
JETZT EIN TRAUM IN  
ERFÜLLUNG.

# INHALTSVERZEICHNIS

Vorwort .....	21
---------------	----

## Kapitel 1: Fangen wir mit einem Gerüst an

### Aufbau einer Seite und die wichtigsten Elemente

#### Seite 23

Die drei ??? – HTML, CSS und JavaScript .....	24	Das Ziel im Auge – das Attribut target .....	43
Der Werkzeugkasten .....	26	Tinks und Largels .....	45
Webbrowser .....	27	Text war gestern – Bilder .....	47
Editor .....	28	Bevor das Bild geladen wurde ... .....	49
Das erste Dokument .....	29	... und hinterher .....	49
Markup und Tags .....	31	Das sollte man im Kopf haben –	
Struktur einer HTML-Seite .....	33	mehr vom <head> .....	52
Attribute, leere Tags und Links .....	35	Andere Länder, andere Zeichen:	
Links zwischen zwei Seiten –		Character Encoding .....	54
über den Gartenzaun .....	40	Denk noch mal drüber nach: Übungen .....	58

## Kapitel 2: Das World Wide Web, unendliche Weiten

### Serverkommunikation, Adressen, Standards

#### Seite 61

Wo finde ich denn nun meine Seite:		... röhre etwas Hypertext unter ... .....	82
Von Webservern und DNS .....	62	... und köchle alles, bis es bunt wird .....	84
URLs – alles an der richtigen Adresse .....	65	Das Ende von Mosaic und der erste	
Ferngespräch für Herrn Web Server – HTTP .....	69	Browserkrieg .....	85
Jetzt wird es ernst – unser eigener Webserver ...	73	Microsofts Monopol und der zweite	
Das obligatorische Geschichtskapitel –		Browserkrieg – der Rote Panda schlägt zurück ...	88
die Geschichte des World Wide Web .....	81	HTML ist nicht gleich HTML – eine Sprache,	
Man nehme ein ARPANET und lasse es reifen ...	81	verschiedene Dialekte .....	90

# Kapitel 3: Jetzt kommt Farbe ins Spiel

## Einführung in CSS

### Seite 93

Webseiten mit Stil – Inline Styles und Farben .....	94	Wohin damit? background-repeat, background-position und background-attachment .....	120
Welches Element hätten's denn gerne:		Hier war ich doch schon mal –	
Selektoren nach Tags, IDs und Klassen .....	100	Pseudoklassen für Links .....	127
Übungen mit dem Regenbogen .....	108	Farben und Selektoren:	
Drei Farben reichen völlig aus – das RGB-Modell	112	Übungen zum Abschluss .....	129
Durchschaut: rgba() und opacity .....	116		
Wir halten uns im Hintergrund – background-image .....	118		

# Kapitel 4: Kaskaden für Bossingen

## CSS-Selektoren und Typografie

### Seite 131

Was heißt jetzt eigentlich Cascading? .....	132	Seichte Kost: nur die direkten Kinder selektieren .....	151
CSS – den Tätern auf der Spur .....	136	Von Schriftgrößen und Selektoren: Übungen .....	152
Größe zeigen – mit font-size .....	140	Es muss nicht immer Times New Roman sein – Schriftarten .....	158
Ahnenforschung für Anfänger – Selektoren für Kinder und Nachfahren .....	145	Gutenbergs Erben – mehr von Schriften und Typografie .....	164
Für Fortgeschrittene: Nachfahren-Selektoren mit mehreren Ebenen .....	150	Die Schriftliche Prüfung: Übungen .....	168

# Kapitel 5: Ordnung in die Plattsammlung

## Listen und Tabellen

### Seite 171

Besser als Zeilenumbruch: Listen .....	172	Was steckt noch drin? Tabellen im Detail .....	191
Wer braucht da noch PowerPoint:		Auch Tabellen brauchen CSS-Liebe .....	197
CSS-Styles für Listen .....	178	Gefängnisreform für größere Zellen – rowspan und colspan .....	204
Definitionssache – Definition Lists mit <dl> .....	181	Tabellarische Übungen .....	206
Eine Liste von Übungen zu Listen .....	184		
Die Liste ist nicht genug – Tabellen .....	187		

# Kapitel 6: Von der Wiege bis zur Bahre – Formulare

## Formulare

### Seite 211

Mehr als nur anfragen: endlich mitreden.	212	Das muss ja nicht jeder sehen –	
Daten eingeben und zum Server schicken – einfaches		versteckte Felder	239
Formular	215	Jetzt kannst du doch noch Opern quatschen –	
Request ist nicht gleich Request – post und get	223	Textarea	240
Aber tippen ist anstrengend! Checkboxen		Formulare 2.0 – viel Neues in HTML5	243
und Radiobuttons	226	Formulare müssen nicht nach Behörde aussehen –	
Wer ist denn nun der Auserwählte?		CSS für Forms	246
Select-Boxen	230	Übungen! Neue Felder, neue Stile	251
Jetzt kommt endlich die Suche!	236	Alle Dateien laden hoooooch – File Upload	254

# Kapitel 7: Von Rändern und Schuhkartons

## Seitenlayout in HTML und CSS

### Seite 257

Die Grundlagen für alles – Block- und		Der Stapelfix™-Stapelplan	280
Inline-Elemente	258	Mehr zu Positionierung	285
Das Box-Model – stapelbares HTML	260	Elemente im Fluss – float und clear	287
Relativ und absolut	264	Floatende Layouts	291
Fünf kleine <div>-Container ...	266	Floatende Layouts	291
Das Gesetz des Kompasses	269	Von Boxen und Stapeln	292
Und weiter geht's mit den fünf <div>s	271	Und so sieht der Stylesheet am Ende aus:	297
Abstände aus der Nähe betrachtet	272	Semantik statt <div> – was gibt's Neues	
10 Liter HTML in einem 5-Liter-<div>:		in HTML5?	298
Overflow	274	Die CSS-Eigenschaft display – warum?	300
Schrödinger in seinem Element –		Wer verdeckt wen? z-index	303
Container schubsen	276	Das Fenster im Fenster	306
Genau dort – absolute Positionierung	278		

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

## Eine Website von Anfang an

### Seite 309

Eine Website von Anfang an .....	310	Für die Kunst – die Entengalerie .....	320
Die Seitenstruktur .....	313	Entengalerie plus – es geht noch cooler .....	326
Die Organisation des Stylesheets .....	318		

# Kapitel 9: Schöner wohnen mit CSS3

## CSS3

### Seite 329

Zum Schutz vor blauen Flecken – runde Ecken ...	330	Die Farbe des Kaffees .....	362
Rahmenbilder für Bilderrahmen .....	334	Gerade war gestern – CSS-Transformationen .....	364
Urlaubsfotos aus den 80ern .....	338	Jetzt bist du dran mit Drehen und Schieben .....	367
Licht und Schatten .....	341	Auf in die dritte Dimension! .....	370
Die Kiste im Licht – box-shadow .....	347	Gemeinsam sehen sie stark aus –	
Schlüsselmomente .....	350	Effekte mit CSS3 .....	372
Und es bewegt sich doch .....	355		

# Kapitel 10: Jetzt muss es sich aber endlich bewegen

## JavaScript

### Seite 379

JavaScript, was ist das eigentlich? .....	380	Zeichen, Zeichen, Zeichenkette .....	407
Und wie geht es jetzt? .....	383	Daten rein, Daten raus II: Eingabe .....	410
Zählen nach Zahlen .....	385	Übungen zu Strings und Ausgabe .....	414
Merk's dir für später – Variablen .....	389	Wenn ... dann ... .....	418
Übungen zu Variablen .....	394	Formulare – bitte geben Sie Ihre Adresse an .....	424
Zahlentheorie .....	397	Wenn die Praxis funktioniert, dann fehlt noch	
Daten rein, Daten raus I: Ausgabe .....	400	die Theorie .....	429
Woher weiß ich, wenn ein Fehler auftritt? .....	405	Was? Wie? Wenn? Dann? .....	432

# Kapitel 11: Programmieren mit Bausteinen

## Funktionen

### Seite 435

So funktioniert's mit Funktionen .....	<b>443</b>	Wie funktionieren meine Funktionen? .....	<b>464</b>
Mehr Werte als man zählen kann – Arrays .....	<b>447</b>	Manchmal geht alles schief – Fehler .....	<b>466</b>
Ein Übung für zwischendurch .....	<b>453</b>	Funktionen, Bürger erster Klasse .....	<b>471</b>
Von vorne bis hinten mit for .....	<b>455</b>	Funktionen in Funktionen in Funktionen .....	<b>476</b>
Von Dingen und Zeigern .....	<b>461</b>		

# Kapitel 12: Augen auf, du hast User!

## Eventhandler

### Seite 481

Reaktionsfreudiges JavaScript – Eventhandler ....	<b>482</b>	JavaScript im Schaumbad – blubbernde Events ...	<b>502</b>
Die Events mit der Maus .....	<b>489</b>	Keyboardevents .....	<b>505</b>
Mehr von der Maus .....	<b>492</b>	Timeout, Formevents und andere .....	<b>508</b>
Das Ziel im Auge – event.target .....	<b>495</b>	Übungen! .....	<b>510</b>
Gezieltes Mausen .....	<b>499</b>		

# Kapitel 13: Gerade stand das da noch nicht

## DOM-Manipulation

### Seite 513

Ein DOM für die HTML-Seite .....	<b>514</b>	Von einem Element zum anderen – navigieren im DOM .....	<b>532</b>
Gärtner für Webentwickler – das DOM als Baum .....	<b>518</b>	Rein, rauf, runter, raus – Elemente erzeugen, einfügen, entfernen und verschieben .....	<b>536</b>
Des Zauberlehrlings Hausaufgabe .....	<b>521</b>	Attribute und Styles .....	<b>542</b>
Mal wieder Wiederholungen – while-Schleifen ...	<b>530</b>	Die Meisterprüfung des DOM-Zauberlehrlings ...	<b>544</b>

# Kapitel 14: Schrödingers Welt der Programmierung

## Objekte und JSON

### Seite 549

Objektorientierung – was und warum? .....	550	Was steckt drin? for ... in .....	567
Objekte für Einsteiger .....	553	Übungen mit Objekten .....	571
Ran an die Eigenschaften .....	556	Konstruktoren und Prototypen .....	573
Und jetzt mit Methoden .....	561	Vererbung – und niemand muss dafür sterben ..	576
Das Schlüsselwort this und Function Binding ....	563	Übungen zu Prototypen und Vererbung .....	582

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

## Cookies, WebStorage und File-API

### Seite 587

Der Griff in die Keksdose .....	588	Heute das Dateisystem, morgen die Welt .....	611
Cookies ganz korrekt .....	590	Was du schon immer über eine Datei wissen wolltest .....	612
Cookies selbst gebacken .....	593	Dateien lesen - der FileReader .....	613
Jetzt wird gebacken .....	594	Dateien in der Praxis .....	618
Daten, so weit das Auge reicht – Web Storage ..	599	Das switch-Statement .....	623
Iterieren über Web Storage .....	601	Dateien und Bäckereien .....	627
Das Beispiel am Stück – und mit Objekt! .....	604	Dateiauswahl – wir können auch anders ..	632
Mehr zu Local Storage – Events und Limits .....	607	Und wir können auch noch anders – noch mal Dateiauswahl .....	634
Von Sandbox zu Sandbox .....	608		
Die große Datenhalde .....	610		

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

## Multimedia

### Seite 639

Bild und Ton im Browser .....	640	Die Fernbedienung für alles – <audio> und <video> mit JavaScript .....	646
Die MIME-Types .....	644	Was alles gehen und schiefgehen kann .....	651
Die Details .....	644		

Das ist nur ein Teil der möglichen Events: .....	<b>652</b>	Werkzeug zur Hand, das Diagramm wird transformiert .....	<b>668</b>
Was war und was kommt mit Multimedia .....	<b>653</b>	Und jetzt mit Tabellen-Daten .....	<b>669</b>
Schrödingers Terassenradio .....	<b>656</b>	Koordinatenballett .....	<b>672</b>
Picasso, Monet, Schrödinger – zeichnen auf dem <canvas> .....	<b>659</b>	Kunst und Text .....	<b>674</b>
Das JavaScript für die Grundausstattung .....	<b>661</b>	Auf dem rechten Pfad .....	<b>680</b>
Ein Beispiel macht alles klar – das erste Rechteck .....	<b>662</b>	Bild im Bild .....	<b>684</b>
Transformationen – die Leinwand drehen und strecken .....	<b>666</b>	Farbähnliche Dingsdas .....	<b>688</b>
		Übungen mit interessanter Überschrift .....	<b>693</b>
		Leinwand für Fortgeschrittene .....	<b>697</b>

## Kapitel 17: Schrödinger will's wissen

### Ajax

#### Seite 699

Was ist Ajax? .....	<b>700</b>	Der Rest ist wieder Geschichte – History-API .....	<b>722</b>
Hallo Server, bitte kommen .....	<b>704</b>	Die Sache mit dem Fragment .....	<b>726</b>
Hol dir die Antwort .....	<b>707</b>	Ich darf aber nicht mit Fremden sprechen – die Same Origin Policy .....	<b>729</b>
Die königliche POST ist da .....	<b>710</b>	Ja wo verbinden sie denn hin? .....	<b>734</b>
Wie Majestät wünschen .....	<b>713</b>	Jenseits von AJAX – Web Sockets .....	<b>736</b>
XmlHttpRequest Level 2 – jetzt mit Nutzlast .....	<b>719</b>		

## Kapitel 18: Jedem das Seine

### Responsive Webdesign und Mobile Devices

#### Seite 739

Was ist Responsive Design, und wozu ist es gut? .....	<b>740</b>	Das Kreuz mit den Bildern .....	<b>758</b>
Jedem seine Styles – Media Types in CSS2 .....	<b>743</b>	Sture Bilder .....	<b>762</b>
Media Features – CSS3 schafft neue Möglichkeiten .....	<b>746</b>	Größer ... größer ... größer ... zu groß! .....	<b>765</b>
Stapelfix Responsive .....	<b>747</b>	Sparsamer laden mit data-Attributen .....	<b>768</b>
Schritt 1: Zuerst wird die Sidebar umpositioniert	<b>750</b>	Was kann so ein Mobildings sonst noch? .....	<b>772</b>
Schritt 2: Jetzt mit handytuglicher Navigation .....	<b>752</b>	Fingergetatsche .....	<b>772</b>
All die vielen Bildschirme! .....	<b>756</b>	Wo zum Teufel bin ich? .....	<b>775</b>
		Schrödinger unterwegs .....	<b>782</b>
		Der Verfolger .....	<b>785</b>

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

## Was geht noch?

### Seite 787

CSS Bibliotheken und Frameworks .....	<b>789</b>	Programmieren geht nicht nur im Browser .....	<b>797</b>
JavaScript-Bibliotheken und neue APIs .....	<b>793</b>	Reine Handarbeit macht auch nicht glücklich .....	<b>801</b>
Aber es gibt auch noch andere Ansätze .....	<b>795</b>	Aber das Wichtigste .....	<b>802</b>

## Anhang A: Zeichencodes

Zeichencodes .....	<b>804</b>	Tabelle 2: Tastencodes für keyup und keydown .....	<b>806</b>
Tabelle 1: ASCII – Codes für keypress .....	<b>805</b>		

## Anhang B: Reguläre Ausdrücke

Muster für Zeichenketten .....	<b>808</b>	Die wichtigsten Elemente von regulären	
Reguläre Ausdrücke in JavaScript .....	<b>812</b>	Ausdrücken, kurz zusammengefasst .....	<b>817</b>

## Index .....

# —NEUN—

css3

# Schöner wohnen mit CSS3

Vor ein paar Jahren waren wir alle noch froh, dass es überhaupt CSS gab, auch wenn es nicht überall gleich funktionierte. Aber man wird anspruchsvoller und möchte irgendwann nicht mehr für jede runde Ecke im Design ein eigenes Bild erstellen. Oder man hätte gerne einen modernen 3D-Look, vielleicht sogar ein paar Animationen ohne JavaScript. Oder, oder, oder. Mit CSS3 kann man endlich (fast) alles machen, was man im Web gestalterisch möchte.

# Zum Schutz vor blauen Flecken – runde Ecken

Jetzt ist bald der Zeitpunkt gekommen, über die Grenzen von HTML und CSS hinauszutreten und uns JavaScript zuzuwenden. Noch vor ein paar Jahren hätten wir diese Grenze längst überschritten und würden allerspätestens jetzt über JavaScript sprechen.

Aber heute möchte ich dir erst noch zeigen, was CSS3 alles kann, denn das ist eine Menge. Durch CSS3 werden deine Webseiten wirklich **filmreif**.

Und dann gibt es noch einige Dinge, die es im Web schon immer gab, die aber jetzt endlich, endlich, endlich mit reinem CSS zu lösen sind.



**[Achtung]**

Leider ist alles, was wir in diesem Kapitel sehen werden, noch so neu, dass nicht alle verbreiteten Browserversionen es unterstützen. **Ja, IE 8, du bist gemeint!** In IE 9 sieht es zum Glück schon etwas besser aus.



Die erste Neuheit, die ich dir zeigen kann, ist etwas, das ziemlich jeder Webentwickler und -designer schon machen musste: **abgerundete Ecken**.

*Ja, Bossingen hatte neulich so was erwähnt ...*

Und in der Steinzeit der Webentwicklung, also noch bis vor ein bis zwei Jahren, hieß das immer genau eins: Bilder. Eins für jede Ecke, schließlich konnten wir auch noch nichts drehen in CSS.

**[Notiz]**

Außerdem sind wir damals bei tiefstem Schnee 5 Kilometer barfuß zum nächsten Computer gelaufen. Könnte man zumindest denken, so wie Webentwickler immer über die Vergangenheit reden.



Die Geschichte der runden Ecke ist eine Geschichte voller Bilder und jammernder Webentwickler.

## Die Ecken des Anstoßes

Langer Rede wenig Sinn, die neue CSS3-Eigenschaftsfamilie **border-radius** nimmt uns dieses Problem jetzt ab. Und der häufigste Fall, nämlich jede Ecke kreisrund zu machen, ist ausgesprochen einfach. Die Eigenschaft **border-radius** mit einer einzelnen Größenangabe als Wert für den Radius macht schöne runde Ecken in allen vier Ecken.



[Funktioniert in]

**border-radius** funktioniert im Internet Explorer ab der Version 9, in Chrome, Firefox und Safari.

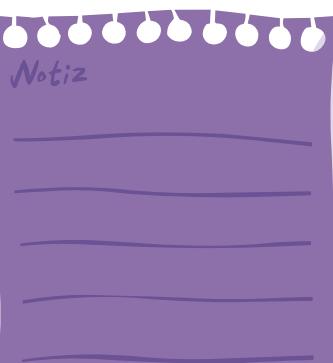
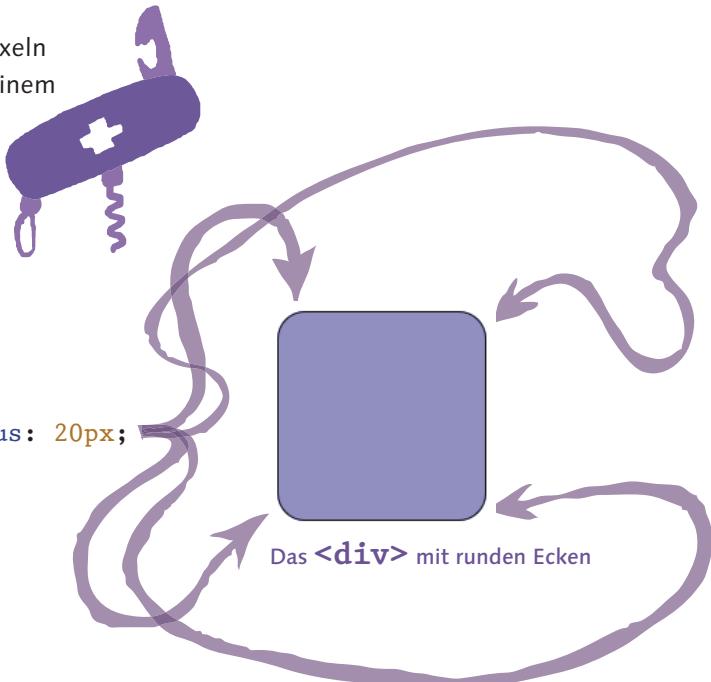
DUNKEL WARS,  
DER MOND  
SCHIEN HELLE...



[Einfache Aufgabe]

Setze an einem **<div>** mit 200 Pixeln Breite und Höhe runde Ecken mit einem Radius von 20 Pixeln.

```
div {  
    border-radius: 20px;  
}
```



[Notiz]

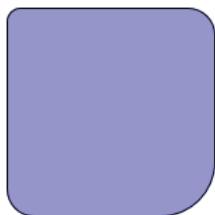
Um einen **border-radius** zu haben, muss übrigens nicht unbedingt eine **border** da sein. Auch wenn kein Rahmen da ist, bekommt der **Hintergrund** die richtige Form.

Auch der Eckradius lässt sich natürlich wieder für jede Ecke einzeln setzen, aber die Möglichkeiten dafür sind leider beide etwas unhandlich. Die CSS-Eigenschaften, um einzelne Ecken zu runden, heißen zum Beispiel **border-top-left-radius** oder **border-bottom-right-radius**. Ich sag's ja, unhandlich. Wie immer lassen sich auch mehrere Werte an die Kurzschreibweise **border-radius** übergeben.

*Also genau wie bei margin und padding?*

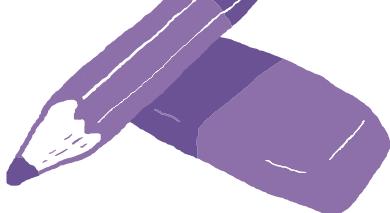
Genau, nur leider etwas unlogischer. Ein einzelner Wert, wie oben gesehen, wird auf alle vier Ecken angewandt. So weit, so gut, das war bei **margin** und **padding** auch so. Aber danach wird es merkwürdig. Gibt man zwei Werte an, gilt der erste für die Ecken links oben und rechts unten, der zweite für die beiden anderen. Und bei drei Werten wird es dann richtig komisch, da sieht es nämlich so aus wie im Bild. Mal ehrlich, das ist doch Humbug. Gib lieber alle vier Werte an, dann versteht es auch jeder.

```
div{  
    border-radius: 10px 20px 40px;  
}
```



Darf ich vorstellen –  
die Humbug-Ecken

Ansonsten ist **border-radius** aber ein weiterer, großer Sieg für die Faulheit. Früher war man 20 Minuten nur mit den **Eckbildern** beschäftigt, heute sind es 20 Sekunden CSS.



#### [Notiz]

Es gibt auch die Möglichkeit, die Rundung einer Ecke zu strecken oder zu stauchen, indem du an die Einzeleigenschaften für jede Ecke zwei Werte über gibst statt nur einen. Der erste gibt dann den horizontalen Radius an, der zweite den vertikalen. Aber man bekommt so nur selten ein schönes Ergebnis, kreisrund ist eben doch das Beste.

#### [Einfache Aufgabe]

Was passiert eigentlich, wenn für die Seiten der Box unterschiedliche Rahmen gesetzt sind?  
Setze für ein <div> unterschiedlich dicke und unterschiedlich gefärbte **borders** für alle vier Seiten, und schau es dir an.



Die Übergänge sehen doch richtig gut aus.  
Nicht von der Katze ablenken lassen!

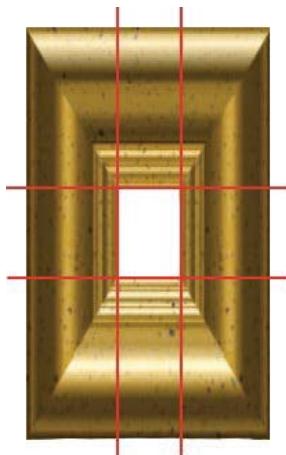


# Rahmenbilder für Bilderrahmen



Das mit den Rahmen sieht alles schon gut aus, du kannst deine Webseiten mit solchen Kleinigkeiten echt aufwerten. Wenn du die Fotos aus dem nächsten Urlaub mit einem schönen, abgerundeten Rahmen online stellst, dann kannst du deine Freundin vielleicht überreden, nicht alle Digitalfotos drucken zu lassen. Aber da können wir auch noch was Besseres: echte **Bilderrahmen**.

Wir brauchen aber diesmal etwas Vorbereitung. Für einen guten Bilderrahmen brauchen wir ein gutes Rahmenbild, und ein gutes Rahmenbild muss mehrere Dinge haben: vier Ecken und vier Seitenteile.



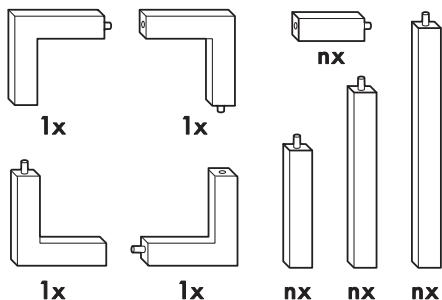
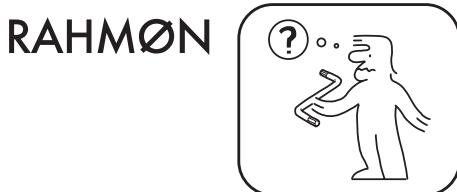
Vier Ecken, vier Seiten,  
mehr braucht man nicht.

Das ist ein ziemlich kleines Bild.  
Irgendwie glaub ich nicht,  
dass sie das übersetzen wird.

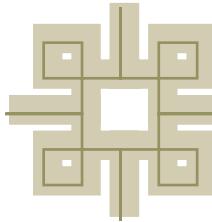


Glaub mir, der Rahmen ist groß genug für alles, was du rahmen möchtest. Der Bilderrahmen ist nämlich selbstwachsend, so was gibt es nicht beim großen schwedischen Möbelhaus. Aber stell dir mal vor, es gäbe dort den beliebig vergrößer- und verkleinerbaren Bilderrahmen: Er hätte einen Namen, wie zum Beispiel RAHMØN, bestünde aus Tausenden von Einzelteilen und würde nur mit einem **Sechskant-Innenschlüssel** zusammengebaut. RAHMØN hätte acht verschiedene Arten von Teilen: vier verschiedene Ecken, jeweils einmal, und vier verschiedene Arten von Seitenteilen, jeweils so oft, dass sich ein beliebig großes Bild damit rahmen ließe.

RAHMØN



*Den Rahmen lass ich dann mal lieber liefern ...*



[Funktioniert in]

**border-image** funktioniert in Chrome, in Firefox und in Safari ab der Version 6.



Genau diese Art Bausatz stellst du her, wenn du den Bilderrahmen oben an den roten Linien zerschneidest, nur ohne Sechskant-Innenschlüssel. Und mit CSS hast du auch nicht das Problem, dass die letzte Schraube nirgendwo zu finden ist. Und all das mit nur wenigen neuen Eigenschaften. Die wichtigsten heißen **border-image-source** und **border-image-slice**, mit der ersten gibst du an, **welches Bild überhaupt benutzt werden soll**, mit der zweiten, wie es zu **zerschneiden** ist.

```
img {  
    border-image-source: url("rahmen.png"); *1  
    border-image-slice: 116 65 116 65; *2  
    border-image-repeat: repeat; *3  
    border-width: 3em; *4  
    border-style: solid; *4  
}
```

\*1 Nicht stehen bleiben, es gibt hier nichts zu sehen. Rahmenbilder werden mit der **url**-Funktion geladen, genau wie andere Bilder auch.

\*2 Hier wird der Rahmen zersägt. Die erste Zahl gibt an, wie viele Pixel von oben der Schnitt für die beiden oberen Ecken und den oberen Mittelteil erfolgt. Die weiteren machen dieselbe Angabe von rechts, unten und links, also die Reihenfolge, die auch bei **margin** und **padding** verwendet wird. Wenn du nur einen, zwei oder drei Werte angibst, werden diese auch so verwendet, wie von den anderen Eigenschaften gewohnt.

\*3 Mit dieser Eigenschaft wird angegeben, wie sich die Seitenteile zwischen den Ecken verhalten sollen. Der Wert **repeat** bedeutet, dass das entsprechende Bildteil wiederholt wird, also genau wie bei RAHMØN.

\*4 Das Element muss auch überhaupt **einen Rahmen haben**, damit hier etwas zu sehen ist. Die **border-image**-Eigenschaften legen zwar fest, wie der Rahmen aussehen soll, aber sie sorgen nicht selbst dafür, dass es auch wirklich einen gibt.

Die Dicke des Rahmens ist auch sehr wichtig, die Einzelteile des Rahmenbildes werden nämlich so vergrößert oder verkleinert, dass sie genau diese Größe auch erfüllen.

#### Achtung!

Die Werte von **border-image-slice** sind zwar Pixelangaben, aber diese dürfen auf keinen Fall die Einheit **px** haben, hier werden einfach Zahlen angegeben. Andere Einheiten wie **pt** oder **em** können sowieso nicht verwendet werden, die einzige Alternative zu Pixeln sind Prozentangaben. Die haben dann auch ein Prozentzeichen.

[Achtung]

Wenn du die Kurzschrifweise **border: 3em solid black;** benutzen willst, dann muss diese Angabe unbedingt vor den **border-image**-Angaben stehen, sonst überschreibt sie diese wieder.



Für **border-image-repeat** ist **repeat** natürlich nicht der einzige Wert, das wäre ja sinnlos. **border-image-repeat: repeat;** kann dazu führen, dass irgendwo ein unvollständiges Seitenteil auftaucht, weil nun mal die Gesamtbreite nicht durch die Breite des Seitenteils teilbar ist. Das ist bei dem Bilderrahmen im Bild oben schnuppe, der passt schon zusammen, aber bei anderen Rahmenbildern kann das hässlich aussehen.



Von links nach rechts: **border-image-repeat: repeat, round, space und stretch**

Das linke Bild zeigt das Problem mit **repeat**, wenn die Zahlen nicht aufgehen.

**border-image-repeat: round** staucht die einzelnen Teile ein wenig, so dass sie passen, **space** fügt ein wenig Platz zwischen den Kacheln ein, und **stretch** streckt **ein Seitenteil**, so dass es die ganze Seite des Rahmens einnimmt. Mit dieser Art Rahmen sehen alle Varianten besser aus als **repeat**. Du kannst auch verschiedene Werte für die horizontalen und vertikalen Rahmenteile angeben, aber oben/unten und links/rechts lassen sich nicht trennen, für die gilt jeweils immer der gleiche Wert.

```
#beispiel{  
    border-image-repeat: round*1 stretch*2;  
}
```

<sup>\*1</sup> Oben und unten wird das Rahmenbild gestaucht und wiederholt.

<sup>\*2</sup> Links und rechts wird ein einzelnes Seitenteil gestreckt.

[Zettel]

Zurzeit kann kein Browser **border-image-repeat: space;** umsetzen. Firefox kennt zumindest die anderen drei Arten, Chrome und Safari können auch mit **round** nichts anfangen.

Du hast dir bestimmt schon gedacht,  
dass es auch hierfür eine **Kurzschreibweise** gibt.

Aber sicher,  
famles Entwicklerpack-

\*1 das Rahmenbild

`border-image: url(rahmen.png)*1 20 50*2 repeat stretch*3;`

\*2 Die Schnittkanten, genau wie für  
**border-image-slice**. Es  
funktionieren auch einer bis vier Werte.

\*3 ein oder zwei Werte für  
**border-image-repeat**

Bei so schönen Bilderrahmen kann sich deine  
Freundin doch gar nicht mehr beschweren, dass  
du nicht alle Bilder drucken lässt. Sie werden  
nie so schön, wie sie jetzt am Bildschirm sind.

# Urlaubsfotos aus den 80ern

Rahmenbilder sind nicht unbedingt das einfachste Thema in CSS, sollen wir das noch mal Schritt für Schritt durchgehen?

*Das wäre total gut, ich bin noch ein bisschen  
verwirrt von den vielen Eigenschaften.*

Das kenn ich, ging mir am Anfang auch so. Wir bauen noch ein Beispiel, und schon wird es klarer. Es soll wieder ein Bilderrahmen werden, aber dieses Mal nicht klassisch und edel in Holz, sondern so, wie du bestimmt auch noch alte Fotos hast: als Polaroid. Du brauchst wieder zuerst eine Seite, die das Bild anzeigt:

```

```

So weit, so einfach. Dazu gehört dann noch ein zweites Bild, nämlich der Rahmen. Einen Polaroid-Rahmen hab ich schon für dich vorbereitet.



Der Fotorahmen



## [Einfache Aufgabe]

Das wichtigste am Rahmenbild ist, es richtig zu schneiden. Öffne den Polaroid-Rahmen (**polaroid.png**) im Bildbearbeitungsprogramm deiner Wahl. Windows Paint oder Ähnliches reicht aus. Es muss nur Bildkoordinaten anzeigen, denn nach denen suchst du. Finde jetzt die vier Werte für **border-image-slice**.

*Oh je, da muss ich ja auch noch rechnen!*

*Zumindest für den rechten und unteren Rand.*

Ja, musst du wohl. Aber ganzzahlige Subtraktion kriegst du noch im Kopf hin, oder? Die Schnittkanten oben und links sind einfach zu finden, du musst nur die Pixelkoordinaten ablesen: 18 Pixel von oben, 24 Pixel von links. Von rechts und unten ist es dann ein wenig, aber wirklich nur ganz wenig, schwieriger. Wie du dich ja schon erinnert hast, wird nämlich die rechte Schnittkante als Abstand zum **rechten Rand** angegeben, und unten ebenso. Also musst du die Schnittkante finden und von der Bildbreite bzw. -höhe abziehen. Rechts ist das  $347 - 323 = 24$  Pixel, unten  $382 - 317 = 65$  Pixel.



[Code bearbeiten]

Setze für dein Foto ein Rahmenbild, und schneide es an der Linie, die du oben ermittelt hast.

### Vom Bild zum Polaroid

```
img {  
    border-image-source: url("polaroid.png"); *1  
    border-image-slice: 18 24 65 24; *2  
}
```

\*1 Hier setzt du das Rahmenbild, soweit kein Problem.

\*2 Und an den Kanten schneidest du es auseinander. Die Reihenfolge der Schnitte ist oben, rechts, unten und links. Denk daran, keine Einheiten für die Zahlen anzugeben!

Damit bist du auch schon fast fertig, allerdings ist noch kein Rahmen zu sehen. Es fehlen noch die Eigenschaften, mit denen das Element überhaupt einen Rahmen bekommt.



[Code bearbeiten]

Füge noch die CSS-Eigenschaften hinzu, mit denen das Element einen Rahmen in der richtigen Dicke bekommt.

## Das ist ein alter Hut

```
img {  
    border-image-source: url("polaroid.png");  
    border-image-slice: 18 24 65 24; *1  
    border-width: 18px 24px 65px 24px; *2  
    border-style: solid; *3  
}
```

\*1 Diese beiden bleiben unverändert.

\*2 Die Rahmendicke wird genau passend zu den Schnitten im Rahmenbild gesetzt. Die Werte stehen auch in der gleichen Reihenfolge, aber jetzt **brauchen** sie die Einheit **px**.

\*3 Und damit überhaupt ein Rahmen zu sehen ist, muss auch diese Eigenschaft noch sein.

Und das war's schon, fertig sind die Urlaubs-Polaroids – einfach, und sieht gut aus.



Vom letzten Schottlandurlaub

Sch... chic, ich mag den Polaroid-Look.  
Schade, dass es das wegen der Digitalkameras  
gar nicht mehr gibt.

# Licht und Schatten

Jetzt kannst du mit Rahmenbildern schon mal einen Fernseher für filmreifes CSS bauen, aber das reicht vorne und hinten nicht, damit es wie eine professionelle Filmproduktion aussieht. Was du brauchst, ist ein Titelschriftzug!

## Browser Wars

Browser Wars: Der schwarze Text



*Damit lockst du aber noch keinen  
vor den Bildschirm.*

Leider wahr. Nur eine **sans-serif**-Schrift zu benutzen und die **line-height** zu verkleinern, macht noch keinen Filmtitel. Hier fehlt noch ein toller Effekt, irgendwas, das vor ein paar Jahren im Web noch niemand konnte. Glühende Buchstaben. Oder vielleicht Schatten. Schatten wären ein guter Anfang.

[Notiz]



Die Eigenschaft **line-height** setzt den Abstand zwischen zwei Zeilen.

[Funktioniert in]



**text-shadow** funktioniert im Internet Explorer ab der Version 10, in Chrome, Firefox und Safari.

[Einfache Aufgabe]

Verpass dem langweiligen Titel einen Schatten: Die Eigenschaft **text-shadow** bekommt in der einfachsten Form drei Parameter, nämlich um wie weit der Schatten nach rechts verschoben ist, um wie weit nach unten und welche Farbe er hat. Für die Verschiebungen funktionieren die üblichen Längeneinheiten, und über Farben weißt du ja auch Bescheid. Weißt du doch noch, oder?



Klaro.

Ich kann einen Farbnamen benutzen,  
einen Hexadezimalen Wert mit der Rante  
davon oder die Funktionsschreibweise `rgb()`.

Hier ist der gesamte Style für den Titel bisher.  
Ist doch schön einfach, oder?

\*1 Hier gibt es nichts zu  
sehen, alles nur altes CSS.

\*2 Eine `line-height`  
kleiner als 1 funktioniert nur,  
weil kein Zeichen unter die  
Grundlinie geht: kein g, kein p  
und so weiter.

```
h1{  
    margin: 20px; *1  
    font-family: sans-serif; *1  
    font-weight: bold; *1  
    font-size: 32px; *1  
    text-align: center; *1  
    line-height: 0.7; *1 *2  
    text-shadow: *3 -0.2em *4 0.2em *5 gray *6;  
}
```

\*3 die CSS3-Eigenschaft  
für Textschatten:  
`text-shadow`

\*4 Um **0.2em** nach links verschoben.  
Da der erste Parameter die Verschiebung  
nach rechts angibt, nimmst du einfach  
negative Zahlen, um nach links zu  
verschieben.

\*5 um **0.2em** nach  
unten verschoben

# Browser Wars

Browser Wars 2:  
Der nächste Versuch

\*6 und in Grau, immer eine  
gute Farbe für Schatten

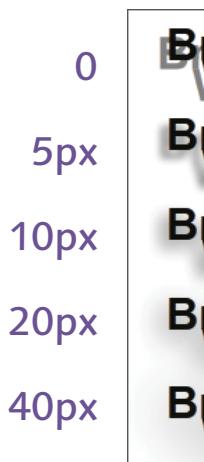
So richtig gefällt mir das aber nicht.

Erstens haben Schatten keine so klaren Umrisse, und zweitens findet ich es schwer zu lesen.

Alter Nörgler. Aber na gut, nichts leichter als das. Zum Glück kennt **text-shadow** einen weiteren Parameter, der genau das korrigiert. Zwischen der Oben-unten-Verschiebung und der Farbe kann man einen zusätzlichen Parameter angeben, der vorgibt, wie **unscharf** der Schatten ist. Um genau zu sein, gibst du eine Entfernung an, über die sich der Schatten ausbreiten darf. Je größer diese Entfernung, desto weiter erstreckt sich der Schatten, aber desto blasser ist der Schatten und desto aufgeweichter seine Kanten.

[Notiz]

Stell dir die Unschärfe so vor: Der Schatten wird immer mit der gleichen Menge Farbe gezeichnet. Wenn mit der gleichen Menge Farbe eine größere Fläche bedeckt werden soll, dann lässt die Deckkraft nach.



[Code bearbeiten]

Mache den Schatten etwas weicher, eine Unschärfe von 5 Pixeln sieht recht gut aus.



```
text-shadow: -0.2em 0.2em 5px*1 gray;
```

<sup>\*1</sup> Hier gehört der Parameter hin.  
5 Pixel Unschärfe sehen recht gut  
aus, weil man die Zeichen noch  
erkennen kann.

## Browser Wars

Browser Wars 3:  
Weicher als weich

Auf jeden Fall besser, aber so richtig an den Bildschirm fesselt es auch noch nicht. Du hast vorhin doch was von Glühen gesagt, vielleicht macht das mehr Eindruck.



[Code bearbeiten]

Ändere den vorhandenen Textschatten so, dass er gegenüber dem Text nicht mehr verschoben wird. Mache außerdem den unscharfen Bereich auch etwas größer, sagen wir 10 Pixel. Und mache das Ganze orange, wer möchte denn bitte graues Glühen sehen?

Versuchen wir's. Einen Text so richtig zum **Glühen** bringen, das kriegen wir auch mit **text-shadow** hin. Dazu wird das schattige Grau durch eine leuchtende Farbe ersetzt und der Schatten auf den Text zentriert.

## Browser Wars

Browser Wars 4:  
Jetzt sieht's heiß aus.

```
color: orange;
```

```
text-shadow: 0 0 10px orange;*2
```

<sup>\*1</sup> Ich hab dann auch mal die Textfarbe geändert ...

<sup>\*2</sup> ... in ein schönes,  
oranges Glühen.

*Es könnte immer noch etwas mehr Umph haben,  
findest du nicht?*

*Ins Kino würde ich für den Titel noch nicht gehen.*

Na gut, dann muss ich eben schwerere Geschütze auffahren. Wenn ein Glüheffekt nicht reicht, um dich zu beeindrucken, dann bleibt mir nur noch eine Möglichkeit: mehrere Glüheffekte! **text-shadow** kann nämlich nicht nur einen, sondern auch **mehrere Schatten** darstellen. Dafür gibst du, durch Kommas getrennt, mehrere Schatten an und fertig. Du kannst auch mehrmals denselben Schatten angeben, bei unscharfen Schatten führt das dazu, dass sie kräftiger werden.

**text-shadow:** 5px 5px 40px yellow<sup>\*1</sup>,  
10px 10px 30px orange<sup>\*2</sup>,  
20px 20px 20px red<sup>\*3</sup>...

\*1 Ein Schatten, ...

\*2 ... zwei Schatten, ...

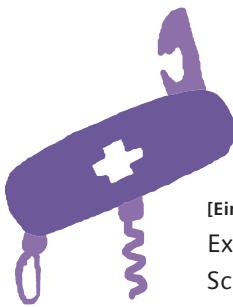
\*3 ... drei Schatten, und  
es gehen noch mehr.

[Achtung]

Mehrere Schatten funktionieren nur so.  
Die **text-shadow**-Eigenschaft darf  
nur einmal vorkommen und hat dann  
mehrere Werte. Wenn du stattdessen  
**text-shadow** mehrmals angibst,  
dann zieht nur die letzte Einstellung.

[Achtung]

Bei mehreren Schatten kann die  
Reihenfolge wichtig sein: Der erste  
Schatten wird ganz oben gezeich-  
net, spätere Schatten können des-  
halb dahinter **verschwinden**.



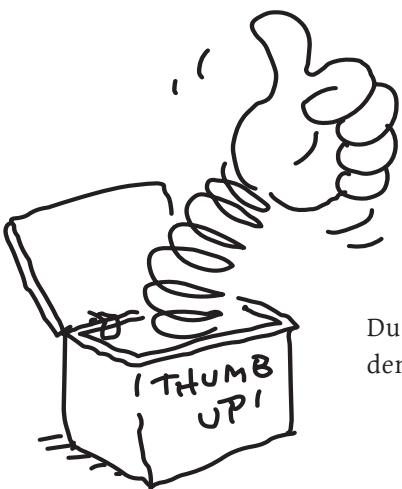
[Einfache Aufgabe]

Experimentiere ein wenig mit mehreren Schatten für den Glüheffekt. Mach einen wirklich eindrucksvollen Filmtitel!



## Browser Wars

Browsers Wars 5:  
Endlich Action!



*So wird doch endlich was daraus!*

Du hast bestimmt noch ein besseres Glühen hinbekommen, aber ich war mit dem schon ganz zufrieden. Das ist das CSS zu meinem Glühen:

```
color: orange;  
text-shadow: 0 0 40px yellow,  
0 0 30px yellow,  
0 0 20px yellow,  
0 0 10px orange, *1  
0 0 5px orange, *1  
0 0 2px red; *1
```

\*1 Nach innen wird der Schatten röter, so sieht das Ganze etwas satter aus.

# Die Kiste im Licht – box-shadow

Aber Text ist nicht alles, auch Schatten an Boxen würden doch ganz gut aussehen. Ein wenig Schatten an einer Layout-Box, und schon sieht sie aus, als würde sie über dem Hintergrund schweben, sehr edel. Wäre das nicht cool? Wäre es nicht nur, ist es, denn so etwas gibt es schon. Und das Beste: Was du gerade über Textschatten gelesen hast, funktioniert bei Boxschatten genauso.



[Funktioniert in]

**box-shadow** funktioniert  
im Internet Explorer ab  
der Version 9, in Chrome,  
in Firefox und in Safari ab  
der Version 5.1.

[Hintergrundinfo]

Mit Boxen sind nicht nur Elemente mit **display: block;** gemeint, um die es oben beim Box-Model hauptsächlich ging. Auch Inline-Elemente haben mindestens eine Box. Kommen innerhalb des Inline-Elements Zeilenumbrüche vor, dann besteht es sogar aus einer Box je Zeile – nur so können die Boxen auch rechteckig sein.



## Die beiden Schatten im Vergleich:

Dies ist ein Text, in dem gleich ein <span> mit einem Textschatten vorkommt.

Der Textschatten, wie wir ihn kennen und lieben

Dies ist ein Text, in dem gleich ein <span> mit einem Boxschatten vorkommt.

Und der neue Boxschatten

Klar, wenn eine Kiste Schatten wirft,  
dann sind die auch kistenförmig.  
Da kann ich mich nun mal ans.

Stimmt, und damit wäre auch schon alles Interessante gesagt zu **box-shadow**, gäbe es nicht zwei zusätzliche Optionen für die CSS-Eigenschaft. Zunächst mal sieht alles genauso aus wie bei einem Textschatten:

The diagram illustrates the four parameters of the `box-shadow` property. It shows a large brown rectangular box with a shadow. The shadow is composed of four segments: a horizontal segment extending to the right, a vertical segment extending downwards, a diagonal segment extending bottom-right, and a small vertical segment at the bottom edge. Callouts point to each segment with numbered labels:

- \*1 Die neue Eigenschaft heißt **box-shadow**, keine Überraschung.
- \*2 Als erste und zweite Parameter kommen immer noch die horizontale und vertikale Verschiebung, ...
- \*3 ... als Drittes die Unschärfe ...
- \*4 ... und als Viertes die Farbe. So einfach ist das.

Below the main box, a smaller box contains sample text:

```
box-shadow: 10px 10px 5px #AAAACCO;
```

A caption below the text box reads: "Eine Box mit Schatten". To the right of the text box, there is a stylized illustration of two figures in dynamic poses, possibly representing the concept of 'shadow' or 'movement'. At the bottom left, a note labeled "[Zettel]" provides additional information about the shadow's behavior with rounded corners:

[Zettel]  
Der Schatten folgt der Form des Rahmens. Runde Ecken am Rahmen bedeuten runde Ecken am Schatten. Alles andere sähe auch sehr, sehr **strange** aus.

At the bottom right, a large yellow arrow points towards a yellow rounded square, which is a visual representation of a box with rounded corners and a shadow.

Aber jetzt geht es noch weiter, Boxschatten können noch einige Dinge, die bei Textschatten überflüssig waren. Das Schlüsselwort **inset**, ganz am Ende der Deklaration angegeben, sorgt dafür, dass der Schatten nicht außerhalb der Box steht, sondern innerhalb – anders als beim Text ist da ja Platz.

```
box-shadow: 10px 10px 5px #AAAACC inset;
```

Lorem ipsum dolor sit amet,  
consectetur adipiscing elit.  
Nullam fermentum, mauris vel  
venenatis porttitor, arcu libero  
tempor tortor, non posuere  
lacus elit nec ligula.  
Vestibulum dapibus sapien ut  
diam placerat id pulvinar diam  
lobortis. Aliquam tempus risus  
vitae ligula accumsan eleifend.  
Ut danibus magna eu

Schatten im Inneren

Außerdem, und dann reicht es auch mit Schatten, kann man für **box-shadow** die Ausbreitung (spread) angeben, eine weitere Größenangabe zwischen Unschärfe und Farbe. Der Schatten breitet sich dadurch in alle Richtungen weiter aus.

*Warum mache ich dann nicht einfach den Schatten größer, anstatt noch einen Parameter anzugeben? Dann wird der Schatten auch größer, meine Verwirrung aber nicht ...*

Nein, den Schatten zu vergrößern, wäre nicht ganz das Gleiche. Machst du den Schatten größer, wächst er nur nach rechts und nach unten – oder nach links und nach oben, wenn du negative Werte angibst –, aber nie in alle vier Richtungen.



**Nullam fermentum, mauris vel  
venenatis porttitor, arcu libero  
tempor tortor, non posuere  
lacus elit nec ligula.  
Vestibulum dapibus sapien ut  
diam placerat id pulvinar diam  
lobortis. Aliquam tempus risus  
vitae ligula accumsan eleifend.  
Ut danibus magna eu**

**Ut danibus magna eu**

Nullam fermentum, mauris vel  
venenatis porttitor, arcu libero  
tempor tortor, non posuere  
lacus elit nec ligula.  
Vestibulum dapibus sapien ut  
diam placerat id pulvinar diam  
lobortis. Aliquam tempus risus  
vitae ligula accumsan eleifend.

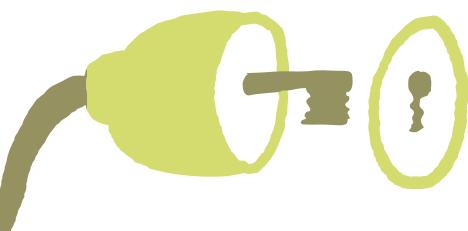
**Nullam fermentum, mauris vel  
venenatis porttitor, arcu libero  
tempor tortor, non posuere  
lacus elit nec ligula.  
Vestibulum dapibus sapien ut  
diam placerat id pulvinar diam  
lobortis. Aliquam tempus risus  
vitae ligula accumsan eleifend.  
Ut danibus magna eu**

Von links nach rechts: 10px Schatten, 20px Schatten und 10px Schatten mit 10px Spread

Achte auf die rechte, obere und die linke, untere Ecke, da liegt der Unterschied.  
Eher ein selten gebrauchtes Detail, aber es sieht schon ganz nett aus.

# Schlüsselmomente

Wir haben jetzt schon einiges für filmreifes HTML und CSS getan, aber es fehlt noch etwas, das meistens als wichtig angesehen wird für Filme: **Bewegung**. Vor gar nicht langer Zeit brauchte man noch für jede Art von Bewegung auf der Webseite JavaScript. Nur hat JavaScript damals auch nicht überall gleich funktioniert, also musste immer Flash her; oder auch mal ein animiertes GIF, aber das erlaubt keine Interaktion. Düstere Zeiten, aber zum Glück sind sie vorbei. **Animationen funktionieren seit Neuestem auch mit reinem CSS**, nicht mal JavaScript brauchen wir mehr.



### [Funktioniert in]

**Keyframe-Animationen** funktionieren im Internet Explorer ab der Version 10, in Chrome, Firefox und Safari.

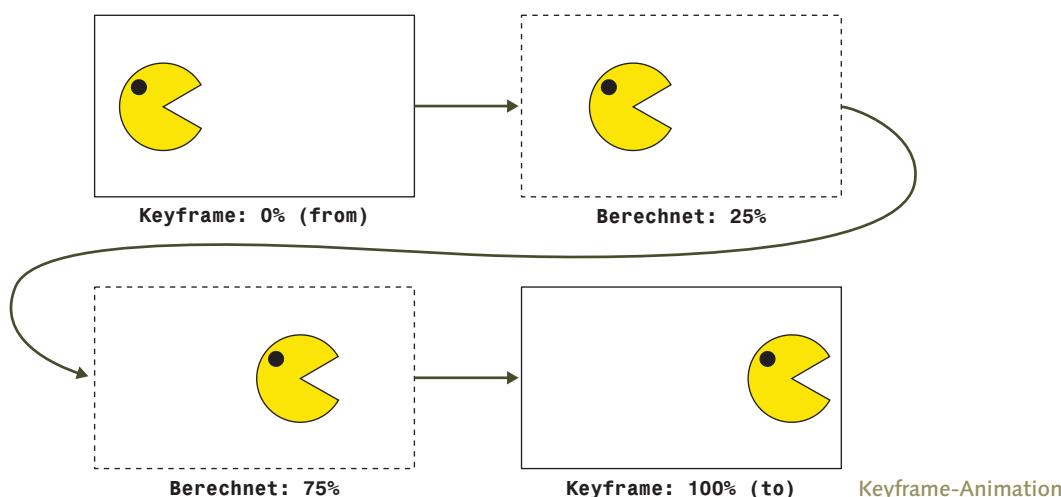
### [Zettel]

Animationen funktionieren auch mit CSS, vorausgesetzt, du benutzt nicht den Internet Explorer in einer Version kleiner als 10. Und vorausgesetzt, dass du alle relevanten Eigenschaften doppelt angibst, einmal so wie im Standard beschrieben und einmal mit **-webkit-** davor, für Chrome und Safari. Aber Animationen funktionieren!



### [Begriffsdefinition]

Ein Einzelbild aus einer Animation heißt ein **Frame**. Die Animationstechnik, die wir in CSS benutzen können, ist die **Keyframe-Animation**. Dabei werden Schlüsselframes der Animation vorgegeben, vor allem natürlich Start und Ende, und alle Frames dazwischen werden vom Computer berechnet. Für komplexe Animationen ist Keyframe-Animation recht aufwendig, aber einfache Animationen lassen sich sehr schnell und einfach umsetzen.



Eine Animation in CSS besteht aus zwei Teilen: zum einen der Definition der Animation, zum anderen der Zuweisung zu einem Element. In der Definition wird der Animation ein **Name** zugewiesen, und die Keyframes werden festgelegt. In etwa so:

\*1 Die neue @-Regel **@keyframes** beginnt eine Animationsdefinition. Browser, die **@keyframes** nicht kennen, ignorieren den gesamten Block. Für Chrome und Safari muss hier das reichlich hässliche **@-webkit-keyframes** stehen – und das vollkommen grundlos, alles andere bleibt gleich.

\*2 Als Nächstes bekommt die Animation einen Namen. Den brauchen wir gleich, um die Animation anzuwenden.

```
@keyframes*1 colors*2 {  
    from*3 {background-color: yellow; *4}  
    33%*3 {background-color: red; *4}  
    66%*3 {background-color: blue; *4}  
    to*3 {background-color: yellow; *4}  
}
```

\*3 Der **Keyframe-Selektor** gibt an, an welche Stelle der Animation dieser Keyframe gehört. **from** (oder 0 %) ist der erste Frame der Animation, **to** (oder 100 %) der letzte. Die Prozentangaben dazwischen kommen an die passende Stelle. Es gibt **keine** Angabe, wie lang die Animation insgesamt laufen soll, das kommt erst später.

\*4 In einem Keyframe können die meisten CSS-Eigenschaften stehen, die in normalen CSS-Regeln auch vorkommen können, alle Eigenschaften, deren Wert sich **interpolieren** lässt, um genau zu sein. Dazu gehören die offensichtlichen Dinge wie Größe, Position und Farbe, aber auch **margin**, **padding**, **border-radius** und viele andere. Nicht animieren lassen sich zum Beispiel **background-image** oder **border-style**: Den Wert zwischen **solid** und **dashed** kann der Browser nicht berechnen; diese Eigenschaften wechseln **plötzlich**, wenn ihr Keyframe an die Reihe kommt. Um Pac-Man über den Bildschirm rennen zu lassen, würdest du einfach die Eigenschaften **left** und/oder **top** animieren, aber dass er dabei auch noch kraftvoll zubeißt, wird mit CSS schwierig.



[Achtung]

Das heißt auch, dass man Animationen immer zweimal definieren darf, einmal mit **@keyframes** und einmal mit **@-webkit-keyframes**, und dass man in beiden Definitionen dieselben Keyframes wiederholen darf. Toll, oder?



[Hintergrundinfo]

Schwierig, aber nicht unmöglich: Du kannst Pac-Man aus mehreren Kreissegmenten zusammensetzen und diese mit Transformationen (die erkläre ich dir gleich) rotieren lassen. Siehe zum Beispiel hier: <http://veli.ee/lab/csspacman/>

[Zettel]

Man kann beliebig viele Keyframes angeben, aber wenn es sich um eine gleichförmige Änderung handelt, zum Beispiel eine Bewegung von A nach B bei gleichbleibender Geschwindigkeit, reichen **from** und **to** völlig aus.

*Das hätte ich mal als Kind gebraucht,  
als wir wochenlang Damenkino gemalt haben.*



Damit ist die Animation definiert, aber es bewegt sich noch nichts, es wird schließlich nirgends angegeben, welche Elemente animiert werden sollen. Aber dafür brauchen wir keine komplexe, neue Syntax mehr, du musst nur drei neue CSS-Eigenschaften lernen:

```
#move_me {  
    animation-name: colors; *1  
    animation-duration: 10s; *2  
    animation-iteration-count: infinite; *3  
    -webkit-animation-name: colors; *4  
    -webkit-animation-duration: 10s; *4  
    -webkit-animation-iteration-count: infinite; *4  
}
```

\*1 Als Erstes muss natürlich eine Animation angegeben werden. Hier gehört der Name hin, der in der **@keyframes**-Regel angegeben wurde.

\*2 Nun kommt auch endlich die Länge der Animation.

\*3 Wie oft soll die Animation wiederholt werden? **infinite** heißt beliebig oft, ansonsten sind Zahlenwerte angesagt.

\*4 Und dann wiederholen wir alles noch mal mit dem **-webkit-**Präfix. Ich hoffe, die Webkit-Entwickler kriegen das bald mal hin ...

*Warum gebe ich das erst hier an und nicht da, wo ich die Animation definiere?*

Ganz sicher bin ich da auch nicht, was sich jemand dabei gedacht hat. Vielleicht weil du so eine Animation mit verschiedenen Geschwindigkeiten abspielen kannst und dafür nicht die ganze Animation noch mal schreiben musst.

*Das mit dem **-webkit-** resst mich jetzt schon.  
Was ist aus der berüchteten Schreibfunkelheit der Webentwickler geworden?*

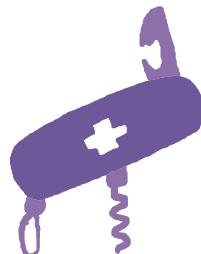
# Und es bewegt sich doch

Grau ist alle Theorie. Also, in unserem Fall natürlich nicht, wir haben ja die Farbe animiert. Aber alle Theorie ist doch theoretisch und oft ein wenig langweilig. Lass uns etwas animieren!

*Lass uns etwas mit Pac-Man animieren!  
Meine Freundin findet die Geister so süß, da kann ich doch  
was Schönes für ihre Website machen.*

[Einfache Aufgabe]

Alles klar, du sollst deine Geister haben. Setze auf eine neue Seite das Bild von Pinky (aus den Beispieldownloads). Es soll sich bis in alle Ewigkeit von der linken oberen Ecke 500 Pixel nach links und wieder zurück bewegen.



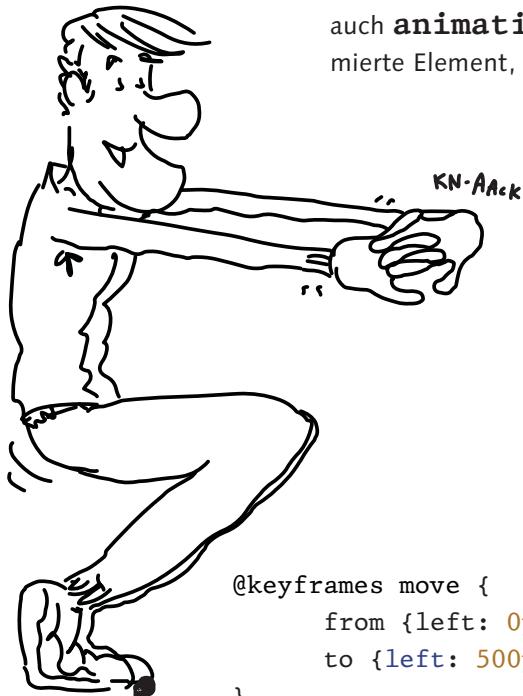
*Hin UND zurück? Wie mach ich denn das?  
Lass ich ihn von 0–50 % nach links laufen und dann von 50–100 %  
wieder zurück?*

Das kannst du machen, aber einfacher geht es mit einer zusätzlichen Eigenschaft (und schreibfauler auch, wir sind ja schließlich Entwickler).

<code>animation-direction: normal;</code>	Spielt die Animation ganz normal vorwärts ab.
<code>animation-direction: reverse;</code>	Die Animation läuft rückwärts, von 100 % bis 0 %.
<code>animation-direction: alternate;</code>	Abwechselnd zuerst vorwärts dann rückwärts. Dadurch ändert sich aber nicht die Dauer: Wenn <code>animation-duration: 10s</code> eingestellt ist, dann dauert es vorwärts 10 Sekunden und rückwärts wieder 10 Sekunden.
<code>animation-direction: alternate-reverse;</code>	wie <code>alternate</code> , aber zuerst rückwärts, dann vorwärts

[Achtung]

Genau wie **animation-duration** gehört auch **animation-direction** an das animierte Element, nicht an die Keyframes.



Das ist ja jetzt wirklich einfach.  
Pass auf, so wird's gemacht.



\*1 So läuft Pinky immer hin und her. Am Anfang kommt er nach 20 Sekunden wieder an: 10 Sekunden hin, 10 Sekunden zurück.

```
@keyframes move {  
    from {left: 0px;}  
    to {left: 500px;}  
}  
  
#move_me {  
    animation-name: move;  
    animation-duration: 10s;  
    animation-direction: alternate; *1  
    animation-iteration-count: infinite;  
    /* Und hier natürlich wieder dieser  
    ganze -webkit-Kram ... */  
}
```

[Achtung]

Die Animation von **0** bis **500px** ist nicht schwierig. Doch eine Animation vom linken Rand bis zum rechten Rand, aber nicht darüber hinaus laufen zu lassen, ist recht tricky. Die Lösung

```
from {left: 0;}  
to {right: 0;}
```

funktioniert nicht, weil die Animation nur einzelne Eigenschaften interpoliert. Obwohl left und right für uns zwar dasselbe bedeuten, nämlich die horizontale Position, sind es für CSS verschiedene Eigenschaften.



[Notiz]

Anstatt umzudrehen, läuft Pinky rückwärts wieder zum Anfang.  
Umdrehen würde einiges an weiterer Trickserei erfordern.



Die Animation muss übrigens nicht sofort anfangen, wenn die Seite geladen wird.  
Mit **animation-delay** lässt sich eine **Verzögerung** festlegen, nach der die Animation erst anfängt.

`animation-delay: 120s; *1`

\*1 Nach 2 Minuten startet die Animation.  
Der Besucher deiner Seite wird sich ganz  
schön erschrecken.



[Notiz]

Zeitangaben lassen sich außer in Sekunden (**s**)  
auch in Millisekunden (**ms**) angeben, andere  
Einheiten gibt es aber nicht.

Oder du kannst die Animation auch gar nicht abspielen, anstatt sie zu verzögern.  
Mit **animation-play-state: paused;** bewegt sich einfach nichts.

*Na super, da mach ich mir die ganze Arbeit,  
eine Animation zu definieren, und dann schaltest du sie wieder ab.  
Unverschämtheit. Das geht so mal gar nicht!*

Nur die Ruhe, auch die Eigenschaft hat einen Sinn. Man kann zum Beispiel  
später per JavaScript den Wert ändern in **animation-play-state: running;**, und schon geht es los.





#### [Schwierige Aufgabe]

Eigentlich brauchen wir nicht mal JavaScript. Du kannst auch mit der Pseudoklasse **:hover**, die du schon von Links her kennst, Pinky dazu bringen, sich nur so lange zu bewegen, bis er mit dem Mauszeiger eingefangen wird. Anders gesagt, Pinky bewegt sich immer, außer, wenn der Mauszeiger über ihm schwebt.

Das ist ja schon ziemlich cool.

```
div#moveme:hover*1 {  
    animation-play-state: paused;*2  
    -webkit-animation-play-state: paused;*3  
}
```

\*1 Etwas zu tun, sobald der Mauszeiger darüber schwebt, genau dafür wurde die **:hover**-Pseudoklasse gemacht.

\*2 Beim Hovern setzen wir **animation-play-state: paused;**. In der CSS-Regel ohne **:hover** musst du gar nichts tun, weil **running** der Defaultwert ist.

\*3 Und dazu will ich schon gar nichts mehr sagen ...

#### [Notiz]

Vielleicht ist dir aufgefallen, dass alle Animationen am Anfang und am Ende langsamer laufen als in der Mitte. Das liegt an einer weiteren CSS-Eigenschaft, die den Ablauf der Animation steuert: **animation-timing-function**. Wenn du nichts anderes einstellst, ist der Wert **ease**, dadurch werden Anfang und Ende der Animation verlangsamt. Willst du lieber eine konstante Geschwindigkeit, dann ist **animation-timing-function: linear**; richtig. Es gibt noch einen Stapel anderer Werte, die aber hier zu tief gingen.

# Und es bewegt sich noch etwas

Keyframe-Animationen sind schon tolle Dinger, oder? Und man kann auch echt viel damit machen. Aber bevor du jetzt „Pac-Man – der Film“ in reinem CSS produzierst, möchte ich dir noch etwas anderes zeigen. **Transitions** (Übergänge) können zwar nicht so komplexe Animationen darstellen wie Keyframes, aber dafür sind sie sehr viel einfacher umzusetzen.

Normalerweise ändert sich der Zustand einer CSS-Eigenschaft in dem Augenblick, in dem sie geändert wird, zum Beispiel durch die **:hover**-Pseudoklasse oder später durch JavaScript. Mit Transitions kannst du diese Änderung **über einen längeren Zeitraum ausdehnen**, es lassen sich Regeln erstellen wie „wenn der Mauszeiger über das Element fährt, dann soll die Farbe langsam von Blau nach Rot wechseln“.



[Funktioniert in]

**Transitions** funktionieren im Internet Explorer ab der Version 10, in Chrome, Firefox und Safari.

```
#hoverme {  
    background-color: red;*1  
    color: blue;*1  
    transition-properties: background-color, color;*2  
    transition-duration: 5s;*3  
    ...  
}  
#hoverme:hover {  
    background-color: blue;*4  
    color: red;*4  
}
```

\*1 Die Eigenschaften, die sich ändern sollen. Rot auf Blau ist vielleicht nicht schön, aber dafür auffällig. Wir machen ja hier keinen Kurs in Grafikdesign.

\*2 Bei **transition-properties** werden, durch Kommas getrennt, die Eigenschaften aufgelistet, für die ein langsamer Übergang stattfinden soll. Alles, was hier nicht drinsteht, ändert sich nach wie vor sofort.

\*3 Und so lange soll der Übergang dauern. Laut Spezifikation kannst du hier auch für jede Eigenschaft eine eigene Übergangszeit angeben, das wird aber noch von keinem Browser unterstützt.

\*4 Und das ist der Zielzustand. Sobald die Maus über dem Element schwebt, bewegen wir uns langsam in diese Richtung.



[Achtung]

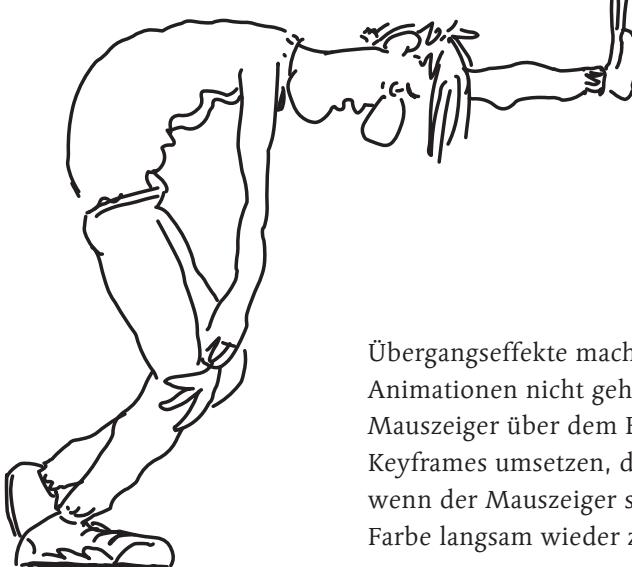
Rate doch mal, für welche Browser du an sämtliche  
**transition**-Eigenschaften **-webkit-** vorne  
dranschreiben musst.

Dann muss ich mir das HTML dafür wohl selbst bauen.

```
<html>... <body>  
  <div id="hoverme">  
    Ich kann nichts für diese Farben, die hat mein Kumpel ausgesucht.  
  </div>  
</body></html>
```

Damit hast du jetzt zwei mächtige Werkzeuge in deinem Animationswerkzeugkasten. Und wie immer, wenn du mehrere Werkzeuge hast, musst du auswählen, welches das richtige ist. Für Animationen mit mehreren Phasen musst du weiterhin Keyframes benutzen. Auch wenn eine Animation ohne Interaktion immer weiter

laufen soll, sind Keyframes die richtige Wahl, denn damit eine Transition endlos läuft, muss jemand immer wieder die CSS-Eigenschaft ändern. Sollen sich nur Eigenschaften langsam von A nach B ändern, entweder durch Benutzerinteraktion oder später, weil du mit JavaScript dran rumgefummelt hast, dann greif zu Transitions, denn sie sind viel einfacher anzuwenden.



Übergangseffekte machen auch Dinge möglich, die mit Keyframe-Animationen nicht gehen. Das Beispiel „Farbe ändern, wenn der Mauszeiger über dem Element schwebt“ lässt sich zwar auch mit Keyframes umsetzen, dann hört aber die Animation einfach auf, wenn der Mauszeiger sich weg bewegt. Mit Transitions kehrt die Farbe langsam wieder zum **Ursprungszustand** zurück.

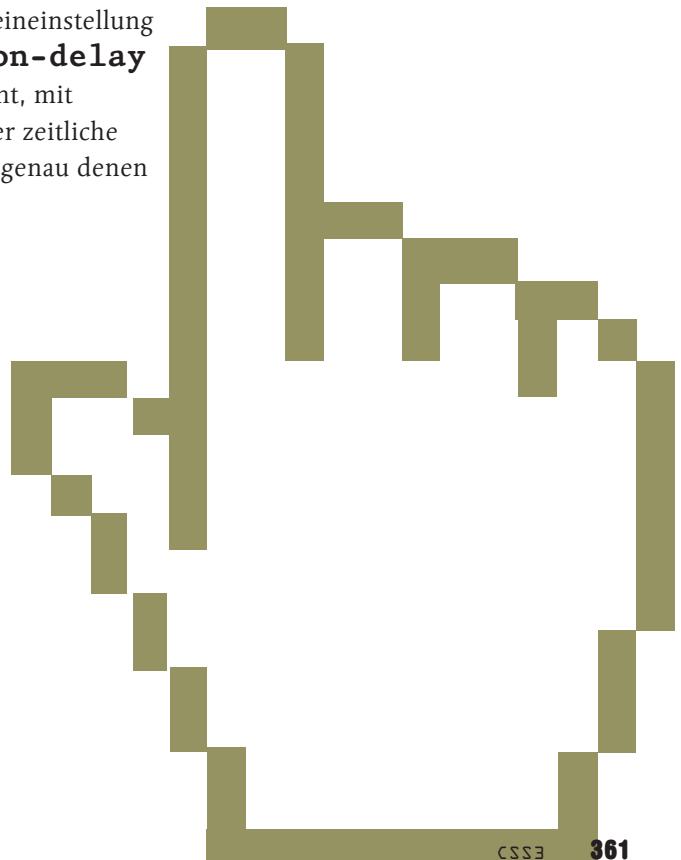
[Zettel]

Für **transition-properties** gibt es auch den speziellen Wert **all**, der alle Eigenschaften animiert, für die das möglich ist.

[Zettel]

Ich hätte hier jetzt gerne ein Bild gehabt, aber wie es aussieht, funktioniert das mit dem animierten Papier noch nicht ...

Für Transitions gibt es die gleichen Möglichkeiten zur Feineinstellung wie auch für Keyframe-Animationen. Mit **transition-delay** setzt man eine Verzögerung, bevor die Animation beginnt, mit **transition-timing-function** lässt sich der zeitliche Ablauf beeinflussen – die möglichen Werte entsprechen genau denen von **animation-timing-function**.



# Die Farbe des Kaffees

Niedliche Gespenster für deine Freundin animieren kannst du jetzt, nun möchte ich noch mal mein Lieblingsthema ins Spiel bringen: Kaffee. Ich finde es immer schwierig, jemandem Milch in den Kaffee zu schütten; es ist immer entweder zu viel oder zu wenig, nie mache ich es richtig. Da wirst du jetzt Abhilfe schaffen. Mit der Seite, die du jetzt erstellst, kannst du jederzeit genau zeigen, wie du deinen Kaffee möchtest.



[Schwierige Aufgabe]

Ein `<div>` soll, wenn die Maus darüber schwebt, seine Farbe langsam von schwarzem Kaffee (#423027) zu milchigem Kaffee (#A77F6B) ändern.



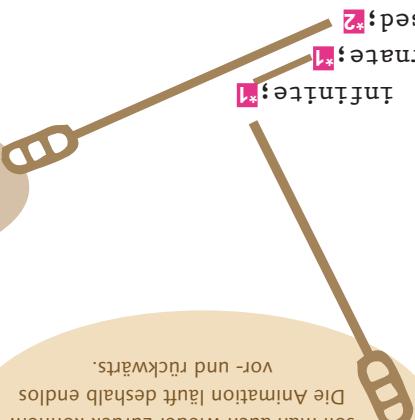
*Das ist ja echt eine Obsession von dir.  
Der viele Kaffee macht dich noch mal krank.  
Aber viel wichtiger: Soll ich das mit Keyframes oder  
Transition machen?*

Das zu entscheiden, ist Teil der Aufgabe. Aber als Hinweis: Die Seite ist viel nützlicher, wenn du den Mauszeiger wegziehen kannst und dadurch die Animation stoppt. Wenn das Element dann in den Ausgangszustand zurückkehrt, hast du nichts gewonnen.

Auch wenn es etwas mehr Schreibarbeit ist, ist Keyframe-Animation die bessere Lösung.  
Mit Transitionen geht alles wieder zurück auf Anfang, wenn du den Mauszeiger weg-  
ziehest. Um wirklich Zeigen zu können, wie dein Kaffee aussiehen soll, sollte die Farbe  
erhalten bleiben. Deshalb kommt die gleiche Technik zum Einsatz, die schon Pinky, das  
richtigen Kaffeefarb Vorbei ist, dann soll man auch wieder zurück können.  
Wenn man einmal an der Sollange niemand über das <div> die Animation.  
\*2

\*1 Wenn man einmal an der Sollange niemand über das <div> die Animation.  
Die Animation läuft deshalb endlos vor- und rückwärts.  
Gespenst, eingefangen hat:

```
Auch wenn es etwas mehr Schreibarbeit ist, ist Keyframe-Animation die bessere Lösung.  
Mit Transitionen geht alles wieder zurück auf Anfang, wenn du den Mauszeiger weg-  
ziehest. Um wirklich Zeigen zu können, wie dein Kaffee aussiehen soll, sollte die Farbe  
erhalten bleiben. Deshalb kommt die gleiche Technik zum Einsatz, die schon Pinky, das  
richtigen Kaffeefarb Vorbei ist, dann soll man auch wieder zurück können.  
Wenn man einmal an der Sollange niemand über das <div> die Animation.  
*2
```



\*2 Sollange niemand über das <div> die Animation.

```
#coffee {  
    width: 200px;  
    height: 200px;  
    animation-name: coffee;  
    animation-duration: 5s;  
    animation-iteration-count: infinite;  
    animation-direction: alternate; *1  
    animation-play-state: paused; *2  
}  
@keyframes coffee {  
    from {background-color: #423027;}  
    to {background-color: #A77F6B;}  
}
```

\*3 Erst mit der :hover - Pseudoklasse geht es los.

\*3 Erst mit der :hover -

[Zettel]

dem -webkit- Präfix auftauchen.  
Listning nicht gezeigt, aber du kennst  
es ja inzwischen: Alles, was mit Animati-  
on zu tun hat, muss zusätzlich mit  
-webkit-Präfixen versehen werden.

\*4 Die Animation hat nur Anfang  
und Ende, kleine Zwischenabschritte,  
und ändert die Hinteregrundfarbe.

\*4 Die Animation hat nur Anfang  
und Ende, kleine Zwischenabschritte,  
und ändert die Hinteregrundfarbe.

{\*4}

{\*3}

{\*2}

{\*1}

{\*3}

{\*2}

{\*1}

# Gerade war gestern – CSS-Transformationen

Bevor wir zur großen Eröffnung des CSS-Kinos kommen, hab ich noch ein letztes Werkzeug, das du an deinen CSS3-Werkzeuggürtel hängen kannst. Und auch dieses Werkzeug macht wirklich coole Effekte mit relativ einfachen Mitteln möglich: **Transformationen**.

Du hast zwar bisher schon vieles gesehen, das man mit CSS basteln kann, doch am Ende kamen immer gerade Elemente dabei heraus, mit Kanten parallel zum Fensterrand. Aber mit gerade ist jetzt Schluss!

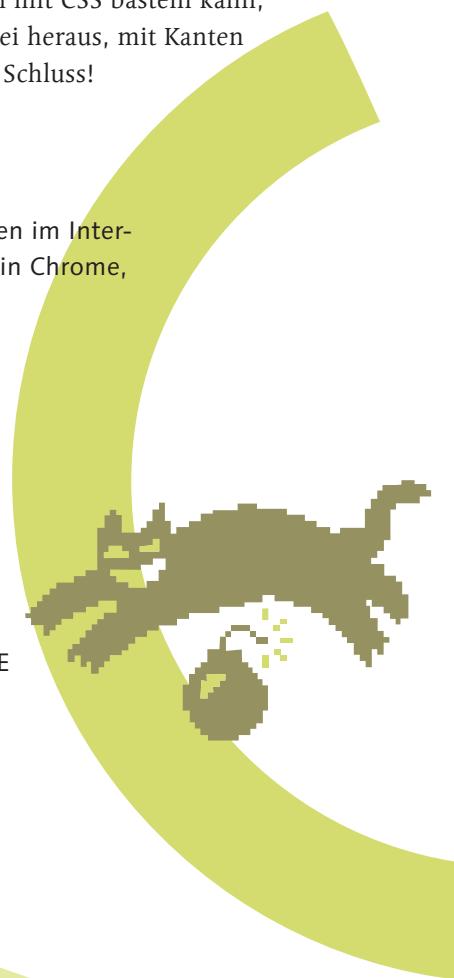


[Funktioniert in]

**Transformationen** funktionieren im Internet Explorer ab der Version 9, in Chrome, Firefox und Safari.

[Achtung]

In Chrome und Safari kommt mal wieder das gefürchtete **-webkit-**Präfix dazu. Außerdem machen die beiden bei einigen Transformationen, vor allem Rotationen, eine ziemliche Schweinerei, selbst in der neuesten Version. Jetzt möchte auch noch Internet Explorer mitspielen, in IE 9 gibt es diese Eigenschaften mit dem Präfix **-ms**, erst in IE 10 gibt es sie dann auch standardkonform.



Bevor ich auf Koordinatensysteme und Geometrie zu sprechen komme, zeige ich dir erst einmal, worum es geht.

## Effekt Nummer eins – gedrehter Text

Ich lege einen Textabsatz an und weise ihm die Style-Eigenschaft **transform: rotate(-2deg);** zu.

Lorem ipsum dolor sit amet, consectetur adipisciing elit. Suspendisse fringilla tempor consectetur. Nam eu bibendum magna. Curabitur orci tellus, congue ut consectetur vulputate, accumsan sit amet enim. Curabitur sit amet sapien nisi. Etiam porta laoreet tellus et varius. Cras sollicitudin dolor at nibh posuere venenatis lacinia lacus mattis. In dapibus velit vel erat sagittis dignissim. Etiam pretium arcu sit amet elit lacinia suscipit tempor quam scelerisque. Donec id pulvinar tortor. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Cras ac odio non urna sagittis ornare.

So sieht es sauber und ordentlich gedreht aus ...

Etiam ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse fringilla tempor consectetur. Nam eu bibendum magna. Curabitur orci tellus, congue ut consectetur vulputate, accumsan sit amet enim. Curabitur sit amet sapien nisi. Etiam porta laoreet tellus et varius. Cras sollicitudin dolor at nibh posuere venenatis lacinia lacus mattis. In dapibus velit vel erat sagittis dignissim. Etiam pretium arcu sit amet elit lacinia suscipit tempor quam scelerisque. Donec id pulvinar tortor. Class aptent taciti socio squ ad litora torquent per combodia nostra, per inceptos himenaeos. Cras ac odio non urna sagittis ornare.

... und so im aktuellen Chrome. Google, das geht besser!

Genau darum geht es bei Transformationen, den klassischen geometrischen Operationen, die du auch in GIMP, Photoshop oder jeder anderen Bildbearbeitungssoftware auf Bilder anwenden kannst:

## Verschieben, Skalieren, Rotieren und Scheren.



## [Achtung]

Bei allen Transformationen bist du selbst dafür verantwortlich, **Überlappungen** zu vermeiden!

**Text**  
Lorem ipsum  
dolor sit amet,  
consectetur  
adipiscing elit.

**Text**  
Lorem ipsum  
dolor sit amet,  
consectetur  
adipiscing elit.

**Text**  
Lorem ipsum  
dolor sit amet,  
consectetur  
adipiscing elit.

**Text**  
Lorem ipsum  
dolor sit amet,  
consectetur  
adipiscing elit

*Lorem ipsum  
dolor sit amet,  
consectetur  
adipiscing elit*

**1** Lorem ipsum  
dolor sit amet,  
consectetur  
adipiscing elit.

Von links nach rechts: das Original, verschoben nach unten, gleichmäßig vergrößert, horizontal vergrößert, aber vertikal verkleinert, rotiert und geschart

## Und so sehen die Transformationen in CSS aus:

<code>transform: translate(x, y)</code>	Verschieben	x und y geben an, wie weit das Element nach rechts bzw. nach unten verschoben werden soll.
<code>transform: scale(x)</code> <code>transform: scale(x, y)</code>	Skalieren	Die Größe des Elements wird mit dem angegebenen Faktor multipliziert, das heißt, $x > 1$ vergrößert das Element, $x < 1$ verkleinert es. Gibt man zwei Parameter an, wird das Element horizontal um Faktor x gestreckt, vertikal um Faktor y.
<code>transform: rotate(xdeg)</code>	Rotieren	Dreht das Element um x Grad im Uhrzeigersinn. Dabei muss deg als Einheit immer angegeben werden.
<code>transform: skewX(xdeg)</code> <code>transform: skewY(xdeg)</code>	Scheren	Das Element wird geschert, das heißt, eine Kante wird verschoben, so dass ein Parallelogramm entsteht. Bei <code>skewX</code> wird die untere Kante verschoben, bei <code>skewY</code> die rechte.

Das Original  
Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Skalieren  
Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Rotieren  
Lorem ipsum dolor sit amet, consectetur adipiscing elit.

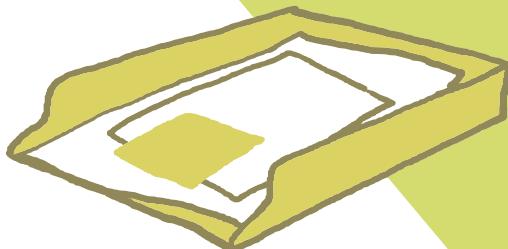
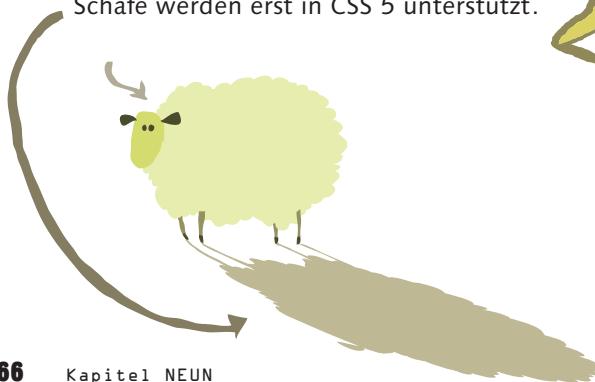
Scheren  
Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Das Original

Skalieren  
Lorem ipsum dolor sit amet, consectetur adipiscing elit.

[Ablage]

„Geschert“, nicht „geschorfen“. Schafe werden erst in CSS 5 unterstützt.



# Jetzt bist du dran mit Drehen und Schieben



## [Einfache Aufgabe]

Genug zugeschaut. Lege vier Textabsätze an, und wende jede Art von Transformation einmal an.

Es können auch **mehrere Transformationen nacheinander** angewendet werden, um genau den Effekt zu erreichen, den man möchte. Dazu gibst du alle Transformationen nacheinander an, zum Beispiel so:

\*1 Zuerst wird das Element um 45 Grad gedreht, ...

```
transform: rotate(45deg)*1 translate(100px, 100px)*2 scale(2)*3;
```

\*2 ... danach um 100 Pixel nach rechts und nach unten verschoben ...

\*3 ... und dann noch auf das Doppelte vergrößert.

## [Einfache Aufgabe]

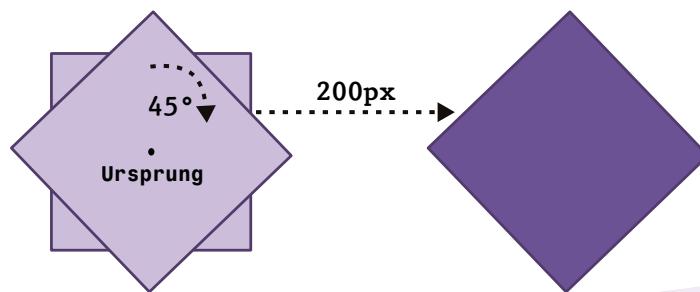
Bei mehreren Transformationen ist die Reihenfolge extrem wichtig. Schau dir einfach mal den Unterschied an zwischen `transform: rotate(45deg) translateX(200px);` und der Transformation in umgekehrter Reihenfolge.



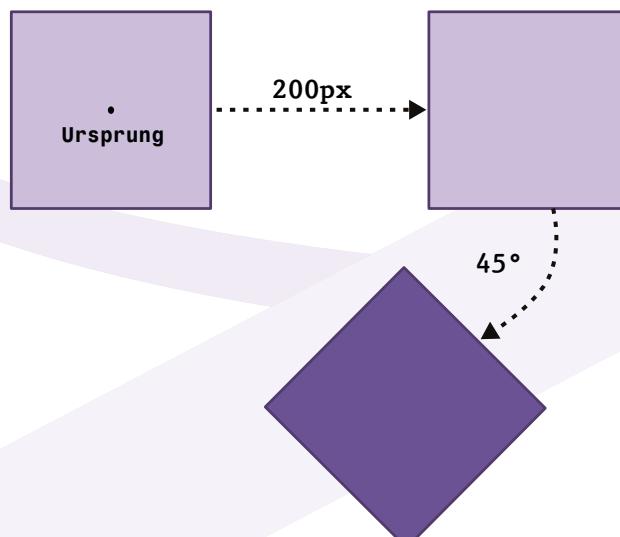
Aber warum ist die Reihenfolge so wichtig?



Das ist die 500€-Frage, aber sie hat eine einleuchtende Antwort: Der Punkt, um den alle Transformationen ausgeführt werden, der **Ursprungspunkt des Koordinatensystems**, ist fix. Ohne andere Angabe liegt der Ursprungspunkt in der Mitte des Elements, aber er bewegt sich nicht, wenn das Element sich bewegt. Er bleibt genau da, wo er angefangen hat. Und deswegen ändert die Reihenfolge der Operationen das Ergebnis.



Zuerst um 45° drehen, dann um 200 Pixel verschieben



Zuerst um 200 Pixel verschieben, dann um 45° drehen

Auch der Ursprungspunkt der Transformationen lässt sich verschieben. Falls sich mal etwas nicht um den Mittelpunkt drehen soll, kriegst du so auch das ganz einfach hin. Du musst nur die Eigenschaft **transform-origin** setzen, und zwar mit zwei Werten: Der erste gibt an, wo der Ursprung in Links-rechts-Richtung liegen soll, der zweite für oben und unten.

```
transform-origin: center*1 0px*2;
```

\*1 Zwischen links und rechts soll der Ursprung in der Mitte liegen ...

\*2 ... und ganz am oberen Rand.

Aber ... und was heißt das?

Das heißt, dass alle Transformationen jetzt um einen Punkt ausgeführt werden, der mitten auf der Oberkante liegt.

Ursprung

Der verschobene Ursprung

Beide Angaben können in Pixeln, Prozent oder mit den Schlüsselwörtern **left/center/right** bzw. **top/center/bottom** angegeben werden, und der Ursprungspunkt kann auch außerhalb des Elements liegen. Mit etwas Übung und diesen Werkzeugen lässt sich jede vorstellbare Transformation zusammenbauen.

[Notiz]

Für den mathematisch begabten Webentwickler lassen sich auch alle Transformationen als eine  $3 \times 2$ -Matrix mit der Eigenschaft **matrix(a, b, c, d, e, f)** anwenden, aber die Matrizendarstellung von Transformationen macht nicht jedem Spaß und würde hier auch zu tief in die Mathematik führen.

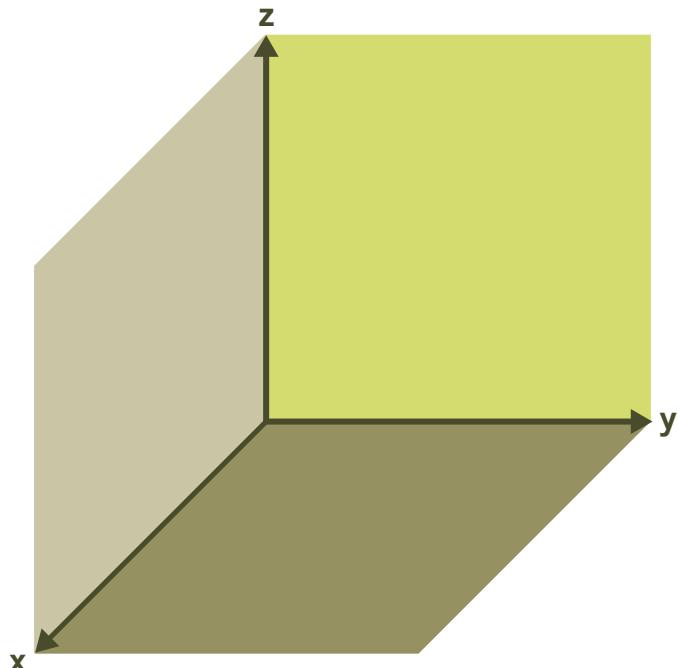


# Auf in die dritte Dimension!

Und auch das ist noch nicht alles! Wenn du dachtest, mehrere Transformationen zu kombinieren, wäre schon das Höchste, dann schnürst du dir jetzt besser die Schuhe etwas fester, denn da kann ich noch einen draufsetzen.

Es gibt Transformationen auch noch in **3D**. 3D-Grafik war vor ein paar Jahren noch die Domäne von Pixarfilmen und aufwendigen Spielen, heute geht es einfach mit CSS. Und (fast) alles, was wir dafür brauchen, sind ein paar neue Transformationsfunktionen. Drehung funktioniert nicht mehr nur um eine Achse, sondern um alle drei Raumachsen mit den drei Transformationen **rotateX**, **rotateY** und **rotateZ**.

Auch Translation entlang der z-Achse ist möglich, sie bringt ein Element näher an den Betrachter oder weiter von ihm weg. Mit nur einem Element hat das den gleichen Effekt wie eine Vergrößerung bzw. Verkleinerung, aber **translateZ** beeinflusst auch, welches Element vor oder hinter einem anderen liegt.



X-Achse, Y-Achse, Z-Achse – mehr Dimensionen wirst Du eher nicht brauchen

Damit 3D-Transformationen richtig funktionieren, muss aber eine zusätzliche Eigenschaft gesetzt werden: **perspective**. Damit wird angegeben, wie weit der Betrachter vom Bildschirm entfernt ist. Natürlich nicht wirklich, sondern nur virtuell. Je näher der virtuelle Betrachter am Bildschirm sitzt, desto stärker ist die perspektivische Verzerrung.

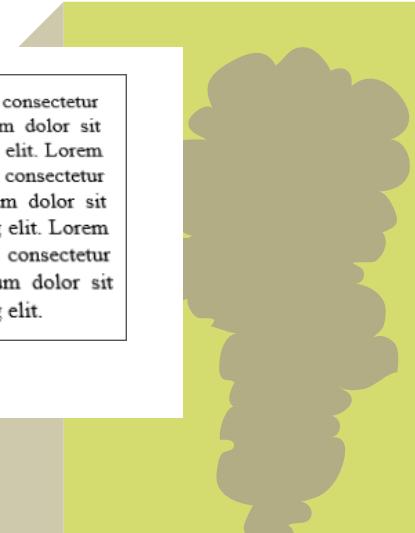
## Rotation um die x-Achse

*Lorem ipsum dolor sit amet, consectetur adipiscing elit. Lorem ipsum dolor sit amet, consectetur adipiscing elit.*

`perspective: 200px;`

*Lorem ipsum dolor sit amet, consectetur adipiscing elit. Lorem ipsum dolor sit amet, consectetur adipiscing elit.*

`perspective: 600px;`



Die **perspective**-Eigenschaft muss immer an einem umgebenden Element gesetzt werden. Stell dir dieses Element als die **Fensterscheibe** vor, hinter der die 3D-Welt liegt.

*Okay, ich glaube ich brauche jetzt eine Pause.  
Das war alles ziemlich viel in diesem Kapitel,  
das muss ich erst mal verdauen.*

Ja, das gebe ich zu, und man hätte zu allem noch viel mehr sagen können, aber ich hoffe, die Übersicht hilft dir auch schon weiter. Bevor du aber jetzt ins CSS-Koma fällst, lass uns noch eine Demo bauen, damit du auch siehst, was für coole Sachen mit den CSS3-Features möglich sind.



# Gemeinsam sehen sie stark aus – Effekte mit CSS3

Jetzt hab ich die ganze Zeit von **filmreifem CSS** gesprochen, es wird Zeit, dass auch wirklich mal ein Film daraus wird. Zuerst brauchen wir die Leinwand.

[Achtung/Vorsicht]

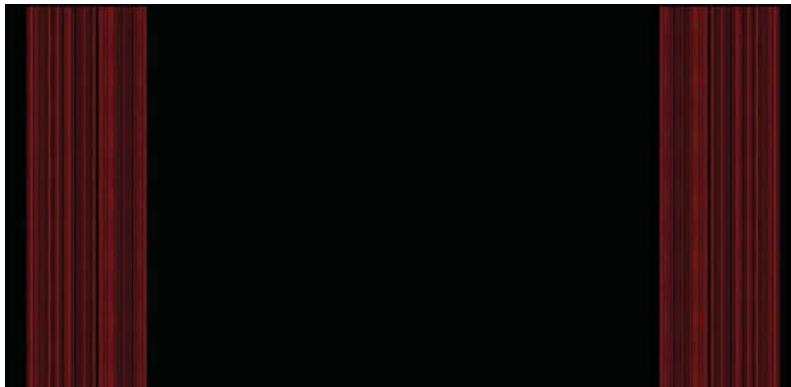
Diese Übung funktioniert in IE 9 noch nicht so richtig, benutze lieber den neuesten Firefox oder Chrome.



[Einfache Aufgabe]

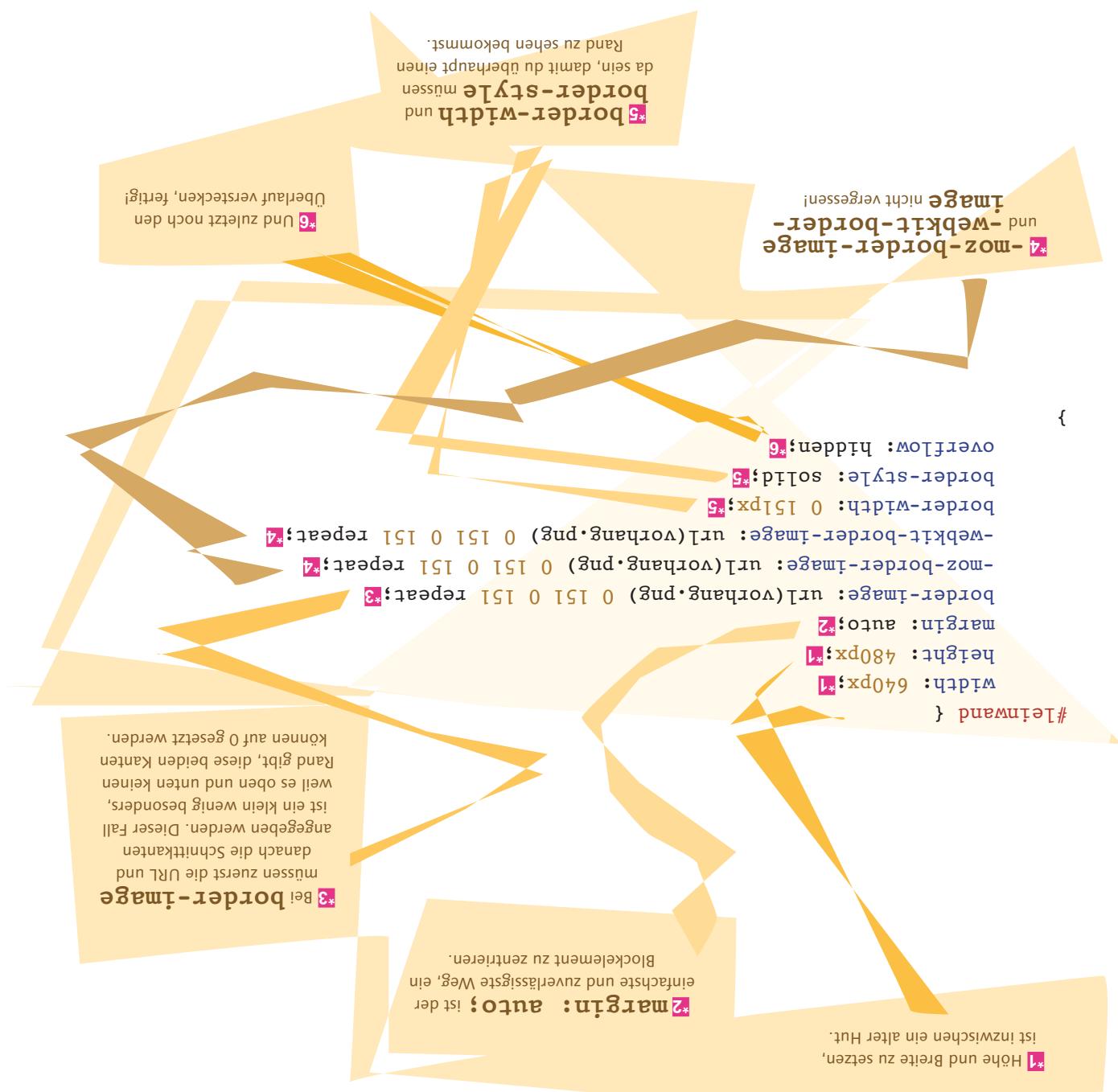
Lege eine neue Seite mit schwarzem Hintergrund an, auf dieser Seite ein **<div>** mit der ID **leinwand**. Die Leinwand soll 640 x 480 Pixel groß sein und horizontal zentriert. Dann wird der Vorhang aufgehängt. Benutze das Bild **vorhang.png** aus den Beispieldownloads als **border-image**. Es gibt keinen oberen oder unteren Rand, die Ränder links und rechts sind jeweils 151 Pixel breit. Da neben der Leinwand kein Film läuft, kannst du auch schon mal den Überlauf verstecken.

Wenn alles stimmt, sollte es so aussehen wie im Bild. Denk vor allem daran, **border-image** auch mit den Herstellerpräfixen anzugeben.



Der Vorhang ist offen, das Publikum hält den Atem an.





Falls deine Vorhang einfarbig nicht zu sehen sind, hast du vielleicht vergessen, **border-style** und **border-width** zu setzen. Keine Sorge, so ginge es mir auch gerade. Beide Eigenschaften sind notwendig, damit die Ränder angezeigt werden.

und **border-width** ist der Blockelement zu zentrieren. einfache und zuverlässige Wege, ein #Leinwand {



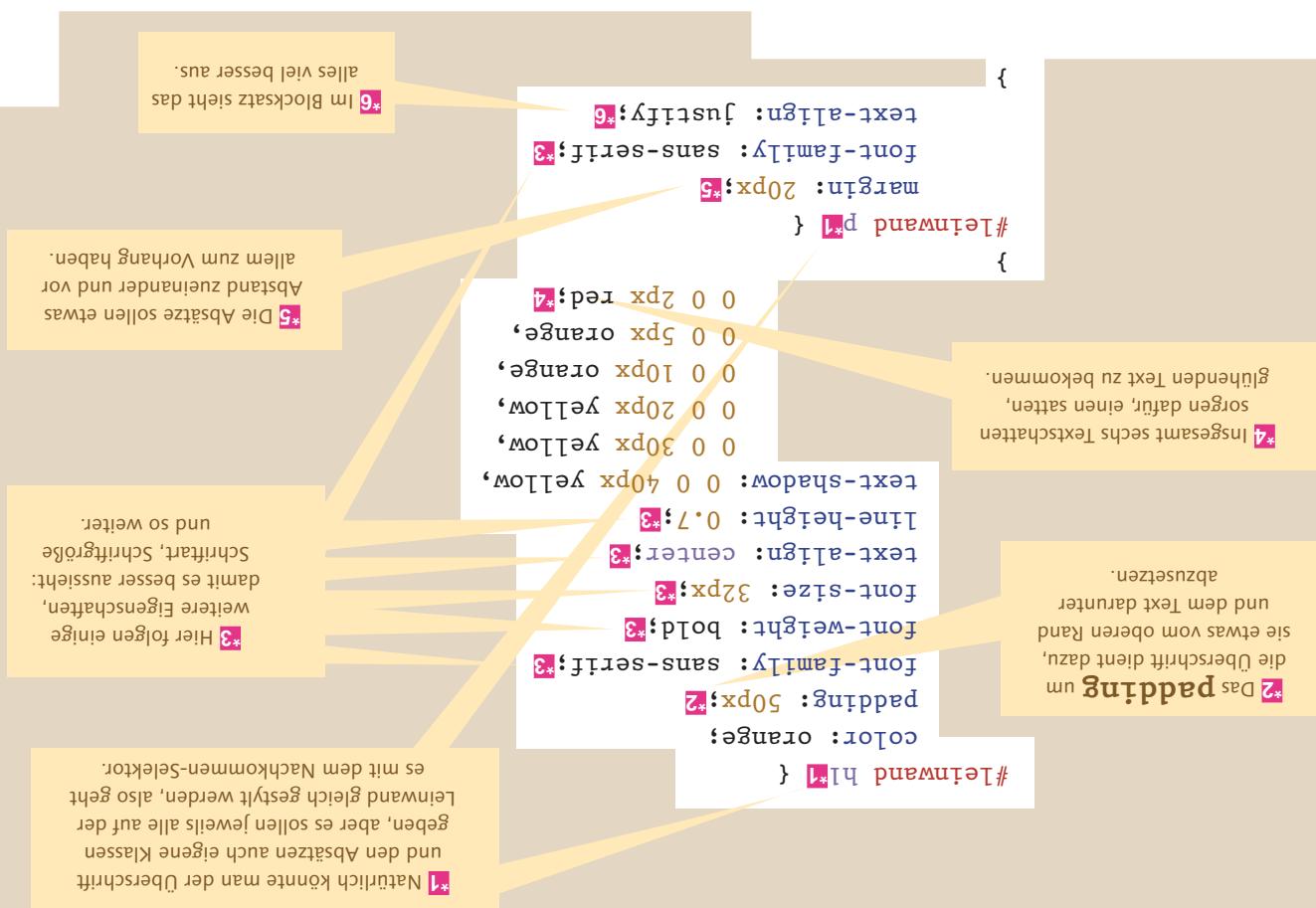
### [Einfache Aufgabe]

Als Nächstes dann der Titel: Pack den Titel „Browser Wars“ auf die Leinwand, mit einem schönen Actionfilm-Textschatten in Gelb. Schreib unter den Titel 10–20 Absätze Text – Lorem Ipsum ist mal wieder dein Freund – in gelber Schrift und im Blocksatz.

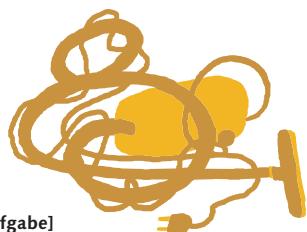


Mit Titel und Text

Diesmal gab es keine Sonderfälle und keine Tricks, es braucht einfach nur **Textschatten**.



Ich kann so langsam schon erraten, wo das hinführen soll.



[Schwierige Aufgabe]

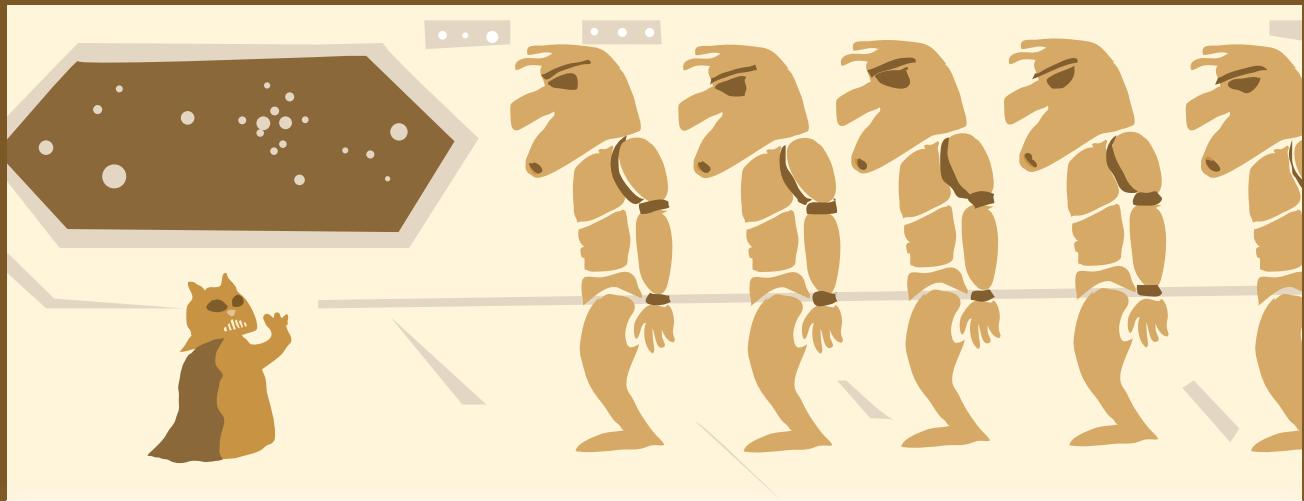
Wenn wir schon einen Film drehen, dann soll er sich auch bewegen. Füge zunächst innerhalb der Leinwand um die Überschrift und alle Absätze ein weiteres `<div>` mit der ID **film** hinzu. Dieses neue `<div>` soll per Keyframe-Animation einmal nach oben scrollen, bis es komplett verschwunden ist.

Dann wird dich der nächste Schritt ja auch kaum überraschen. Film ab!



[Zettel]

Wie hoch das Filmelement ist, kannst du in den Entwicklertools deines Browsers herausfinden. Wie lange die Animation laufen soll, musst du ausprobieren – langsam genug, um den Text zu lesen, aber nicht zu langsam.



\*1 Nach oben zu scrollen, ist ganz einfach, man verschiebt die Oberkante des Elements ins Negative. 2330 Pixel ist die Höhe meines Elements, deins kann natürlich anders sein.

\*2 Die Eigenschaft **top** wirkt nur bei positionierten Elementen.

```
@keyframes scroll {  
    from {top: 0;};*1  
    to {top: -2330px;};*1  
}  
  
#film {  
    position: relative;*2  
    animation-name: scroll;*3  
    animation-duration: 45s;*4  
    animation-iteration-count: 1;*5  
    animation-timing-function: linear;*6  
    animation-fill-mode: forwards;*7  
}
```

\*3 Hier wird die Animation auf das Element angewendet.

\*4 45 Sekunden war für die Textlänge eine gute Zeit.

\*5 Die Animation soll nur einmal ablaufen.

\*7 Noch eine nützliche Kleinigkeit: Wenn **animation-fill-mode: forwards** gesetzt ist, dann bleibt die Animation im Endzustand stehen, anstatt in den Anfangszustand zurückzuspringen.

\*6 Die Timing-Funktion **linear** lässt die Animation mit konstanter Geschwindigkeit laufen, anstatt sie am Anfang zu beschleunigen und am Ende zu verlangsamen.

#### [Achtung/Vorsicht]

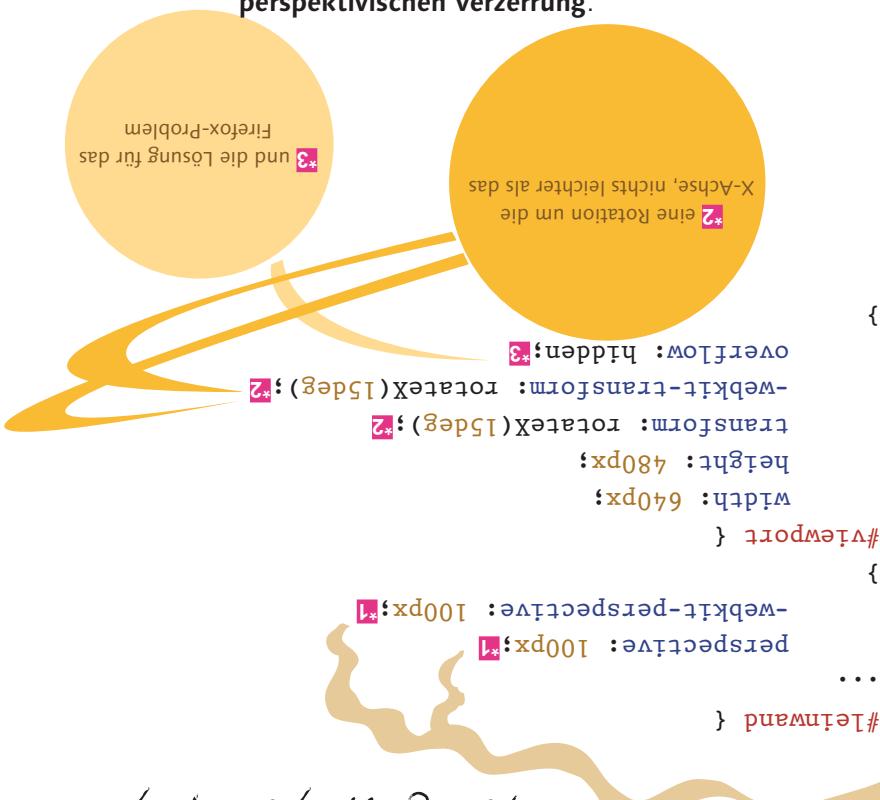
Denke daran, sowohl die **@keyframes**-Regel als auch die Animationseigenschaften mit dem Präfix **-webkit-** zu wiederholen.

## Und jetzt das große Finale: eine Transformation



[Schwierige Aufgabe]

Setze zunächst um das **film-<div>** noch ein weiteres **<div>** mit der ID **viewport**. Gib diesem **<div>** dieselbe Größe wie der Leinwand. Drehe es dann mit einer Transformation um 15° um die X-Achse. Fehlt noch die Perspektive: Setze **perspective: 100px**, eine so nahe Perspektive führt zu einer **starken perspektivischen Verzerrung**.



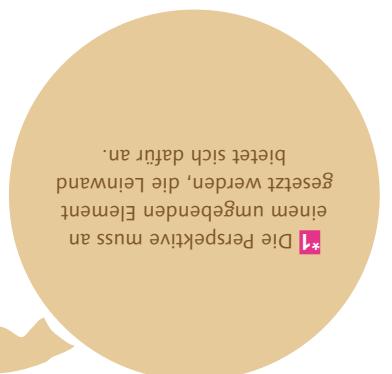
Haha! Ich hatte Recht.

Ein echter Klassiker, dieser Browser-Wars-Film. Aber warum das neue <div> für die Transformation? Dieses **viewport**-Ding meine ich.



[Achtung/Vorsicht]

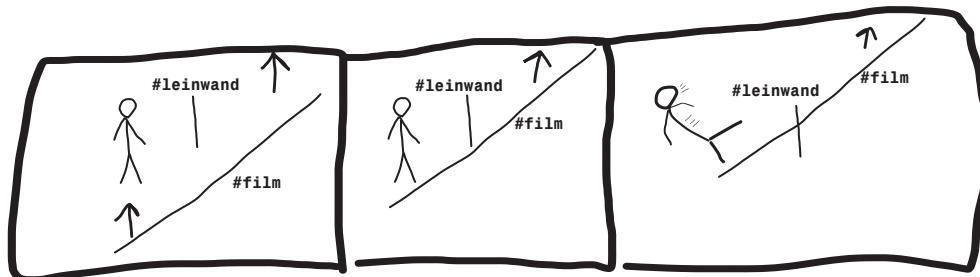
In aktuellen Versionen von Firefox gibt es einen Fehler, der bei dieser Transformation den Text **verschwinden** lässt. Um das zu umgehen, muss an **viewport** auch die Eigenschaft **overflow: hidden** gesetzt werden.



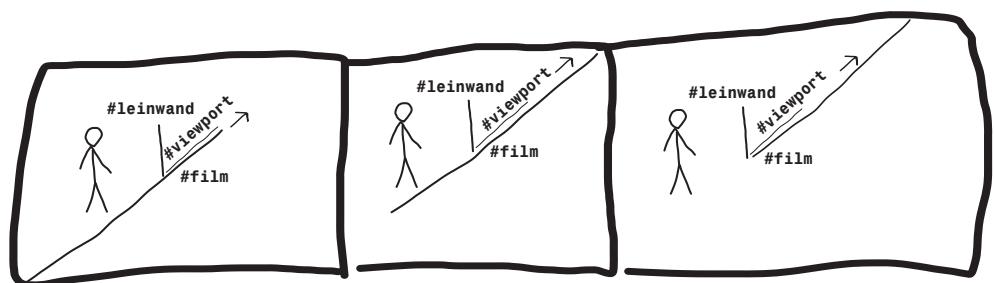
#1 Die Perspektive muss an gesetzt werden, die Leinwand bietet sich dafür an.

Das neue **<div>** ist notwendig, damit die Drehung so aussieht, wie wir es haben möchten. Das Element wird um seinen Mittelpunkt gedreht, und zwar um den des gesamten Elements, nicht nur um den des sichtbaren Teils. Das würde dazu führen, dass der Text am Anfang der Animation sehr klein wäre und gegen Ende immer größer würde. Mit dem zusätzlichen **<div>** wird die Drehung nur auf dieses angewandt, und der Text läuft in gleichbleibender Größe durch.

Warte, ich mal das mal auf einen Schmierzettel



Ohne `viewport`: `#film` bewegt sich nach oben im Koordinatensystem von `#leinwand`.

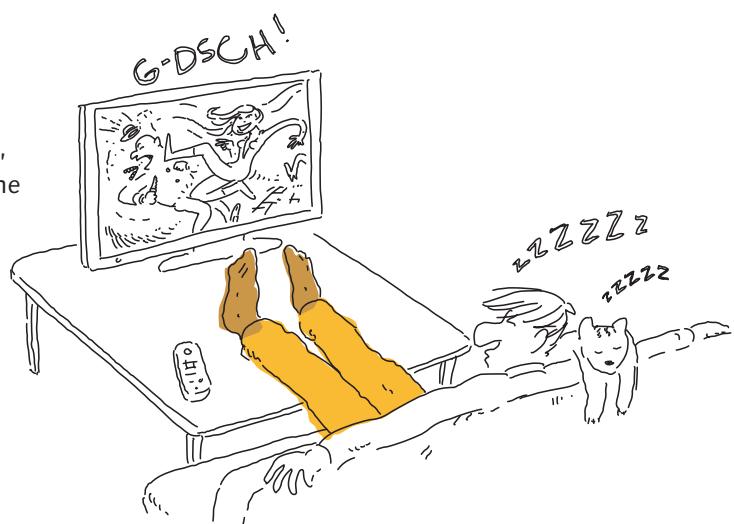
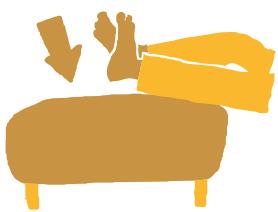


Mit `viewport`: `#film` bewegt sich nach oben im Koordinatensystem von `viewport`.

Darum brauchst Du das `<div>`.

[Belohnung/Lösung]

Jetzt hab ich aber wirklich mal wieder Lust, Star Wars zu sehen. Du solltest dir auch eine Pause gönnen und einen Film anschauen.



# Symbolle

[ ] → Array  
# 37  
&#x 56  
@font-face 160  
@import 319  
@keyframes 351  
@media 743  
!DOCTYPE → Doctype  
.htm 29  
.html 29  
3D (CSS) 370

## A

a (Tag) 36  
  *href* 36  
  *name* 37  
  *target* 43  
Absendeknopf → button 217  
Absolute Positionierung → position 278  
Abstand → margin 263  
Ajax 700  
  *Lesezeichen* 726  
alert 383  
Animation 51, 350  
animation-delay 357  
animation-direction 356  
animation-duration 354  
animation-iteration-count 354  
animation-name 354  
animation-play-state 357  
animation-timing-function 358  
arguments 460  
Array 447  
  *join()* 451  
  *length* 450  
  *pop()* 450  
  *push()* 450  
  *reverse()* 451  
  *shift()* 450  
  *sort()* 472  
  *unshift()* 450  
Artefakt (Bild) 51  
article (Tag) 299

aside (Tag) 299  
Attribut 35  
  *Wert* 36  
Attribut-Selektor 249  
audio (Tag) 640

## B

background-attachment 120  
background-color 96  
  *transparent* 118  
background-image 118  
background-position 120  
background-repeat 120  
background-size 758  
base (Tag) 52  
Bild 47  
  *Höhe und Breite* 48  
Bildergalerie 320  
Bildformat 50  
bind 566  
Blob 719  
Blockelement 258  
  *zentrieren* 272  
blockquote (Tag) 288  
Blocksatz → text-align 167  
body (Tag) 31  
border 197, 263  
border-collapse 200  
border-color 199  
border-image-repeat 336  
border-image-slice 335  
border-image-source 335  
border-radius 331  
border-style 199  
border-width 199  
bottom 278  
Box-Model 260  
Boxschatten 347  
box-shadow 347  
br (Tag) 172  
Breite → width 261  
Browser 24, 27  
button (Tag) 217

## C

canvas (Tag) 659  
caption (Tag) 191  
caption-side 195  
case → switch 623  
Character Encoding 54, 213  
Character Entity 56  
Checkbox 226  
clear 290  
Codec 641  
col (Tag) 190  
colgroup (Tag) 191  
color → Farben 96  
const 392  
Content Area 261  
Cookie 588  
CORS 733  
CSS 93  
cursive 159

## D

data-Attribute 769  
datalist (Tag) 234  
Dateiauswahl  
    per Drag & Drop 634  
    schön mit JavaScript 632  
    schön ohne JavaScript 255  
Datei  
    Informationen 612  
    lesen 613  
Dateiendung 29  
Dateiupload 254  
dd (Tag) 181  
Definitionlist → dl 181  
dfn (Tag) 181  
disabled (Pseudoklasse) 248  
display 300  
    block 300  
    inline 300  
    none 301  
div (Tag) 260  
dl (Tag) 181  
DNS 63  
Doctype 33

document  
    cookie 593  
    createCommentNode 540  
    createElement 540  
    createTextNode 540  
    getElementById 404  
    getElementsByClassName 518  
    getElementsByTagName 518  
Document Object Model → DOM 514  
DOM 514  
Domain Name System → DNS 63  
Drag & Drop 634  
Druck-Stylesheet 754  
dt (Tag) 181  
Durchgestrichen → text-decoration 166  
Durchsichtigkeit → opacity 116

## E

Editor 28  
Eingabefeld → input 214  
Einrückung 29  
Element 518  
    addEventListener 484  
    attachEvent 486  
    classList 523  
    className 522  
    getElementsByClassName 518  
    getElementsByTagName 518  
    removeChild 536  
    style 543  
    textContent 546  
em (Einheit) 142  
em (Tag) 168  
embed (Tag) 654  
Elemente erzeugen 536  
enabled (Pseudoklasse) 248  
Entity  
    benannte 57  
Event 482  
    blur 508  
    click 489  
    dblclick 493  
    focus 508  
    key 505  
    keycode 506

*keydown* 505  
*keypress* 505  
*keyup* 505  
*load* 488  
*mousedown* 493  
*mouseenter* 503  
*mouseleave* 503  
*mousemove* 492  
*mouseout* 492  
*mouseover* 492  
*mouseup* 493  
*preventDefault()* 504  
*resize* 508  
*srcElement* 495  
*scroll* 508  
*stopPropagation()* 502  
*storage* 608  
    *target* 495  
Event Bubbling 502  
Eventhandler 401, 482  
Event Propagation 502  
Exception 466

## F

falsy 430  
fantasy 159  
Farben 96, 112, 116  
Fehler → Exception 466  
Feste Positionierung → position 286  
Fett gedruckt → font-weight 164  
fieldset (Tag) 237  
File-API 611  
FileReader 613  
    *Events* 616  
float 287  
Fließkommazahlen 397  
Floating Layout 748  
focus (Pseudoklasse) 248  
Font 56, 158  
font-family 158  
font-size 140  
font-style 145  
font-variant 165  
font-weight 164

footer (Tag) 299  
for ... in 567  
form (Tag) 212  
    *accept-encoding* 213  
    *action* 213  
    *method* 213, 223  
    *novalidate* 244  
Formular → form 212  
Formularevent  
    *blur* 508  
    *focus* 508  
for-Schleife 455  
function 400, 436  
Funktion → function 400, 436  
Funktionsparameter → Parameter 440  
Funktionsreferenz 471

## G

Geschichte des WWW 81  
Gestensteuerung 775  
get 223  
GIF 51  
Globale Variable → Scope 441

## H

h1, h2, h3, h4, h5, h6 (Tags) 34  
Hash 37  
head (Tag) 33, 52  
header (Tag) 299  
height 48, 261  
Hexadezimal 54  
Hintergrundbild → background-image 118  
Hintergrundfarbe → background-color 96  
History → window:history 722  
Höhe → height 261  
hover (Pseudoklasse) 128  
hsl 115  
html (Tag) 33  
HTML 24  
    *Dialekte* 90  
HTTP 69  
Hypertext 31, 35  
Hypertext Transfer Protocol → HTTP 69

## I

id 37  
if... else... 418  
iframe (Tag) 306  
img (Tag) 47  
    *alt (Attribut)* 48  
    *src* 48  
inherit 135  
Inline-Element 258  
Inline Styles 94  
input (Tag) 214  
    *Bild* 219  
    *Checkbox* 226  
    *datalist* 234  
    *Datum* 243  
    *disabled* 248  
    *email* 243  
    *Farbe* 243  
    *hidden* 239  
    *maxlength* 221  
    *pattern* 243  
    *placeholder* 220  
    *Radiobutton* 226  
    *readonly* 248  
    *required* 243  
    *Textfeld* 214  
    *url* 243  
    *value* 221  
    *Zahl* 243  
Integer 397  
invalid (Pseudoklasse) 247  
italic → font-style 145

## J

JavaScript 24, 379  
JPEG 51  
JSON 559  
JSONP 731

## K

Keyframe-Animation 350  
Kinder 148  
Kindselektor 151

Kompression (Bild) 51  
Konstante 392  
Konstruktor 573  
kursiv → font-style 145

## L

label (Tag) 218  
    *for* 218  
left 278  
legend (Tag) 237  
li (Tag) 174  
Lightbox 326  
link (Tag) 98  
Link 35  
    *absoluter* 42  
    *relativer* 41  
Linksbündig → text-align 167  
list-style-image 179  
list-style-type 178  
Listen 171  
localStorage → Web Storage 599  
Lokale Variable → Scope 441

## M

margin 263  
Markup 31  
Markupsprache 31  
Mausevent 492  
    *buttons* 493  
    *Koordinaten* 492  
Media Queries 743  
meta (Tag) 53  
Metadaten 53  
Mime-Type 613  
monospace 159  
MP3 644  
MP4 644  
Multimedia 639

## N

Nachfahren 148  
Neuer Tab 44  
Neues Fenster 44

Node 532  
appendChild 537  
firstChild 534  
getAttribute 542  
insertBefore 539  
lastChild 534  
nextSibling 534  
nodeName 533  
nodeType 533  
nodeValue 533  
previousSibling 534  
setAttribute 542  
NodeList 519, 529  
noscript (Tag) 383  
nth-child (Pseudoklasse) 200  
Number 397  
toFixed 399

## 0

Object Initializer 559  
object (Tag) 654  
Objekt 550  
  *Eigenschaften* 556  
  *Methoden* 561  
Objektliteral → Object Initializer 559  
Objekttypen erzeugen 573  
ol (Tag) 174  
opacity 116  
optional (Pseudoklasse) 246  
overflow 274  
  *auto* 274  
  *hidden* 274  
  *scroll* 274  
  *visible* 274

## P

p (Tag) 31, 33  
padding 263  
Parameter 440  
parseFloat 413  
parseInt 413  
Pass by reference 461  
perspective 371  
Pfad 680  
picture (Tag) 763

Plug-in 653  
PNG 51  
position 283  
  *absolute* 283  
  *fixed* 286  
  *relative* 285  
  *static* 285  
Positionierung → position 285  
post 223  
Prototyp → prototype 574  
prototype 574  
Pseudoklasse 127  
pt (Einheit) 141  
px (Einheit) 141

## R

Radiobutton 226  
Rahmen → border 197  
Rahmenbild → border-image 334  
Raute → Hash 37  
read-only (Pseudoklasse) 248  
read-write (Pseudoklasse) 248  
Rechnen 388  
Rechtsbündig → text-align 167  
Relative Positionierung → position 285  
Requesttypen 223  
required (Pseudoklasse) 246  
Resizeevent 508  
Responsive Design 740  
  *für Hintergründe* 758  
  *für Bilder* 762  
return 440  
rgba → Farben 116  
RGB → Farben 112  
Rotieren → transform:rotate 364  
Runde Ecken → border-radius 330

## S

Same Origin Policy 729  
sans-serif 159  
Scheren → transform:skew 364  
Schleife  
  *for* 455  
  *while* 530  
Schriftart → font-family 158

Schriftgröße → font-size 140  
Scope 441  
script (Tag) 383  
Scrollevent 508  
section (Tag) 299  
Sectioning Content Elements 299  
select (Tag) 230  
    multiple 234  
    optgroup 233  
    option 231  
    size 234  
Select-Box → select 230  
Selektor  
    Nachfahren 145  
    nach ID 102  
    nach Klasse 104  
    nach Tag 100  
    serif 159  
sessionStorage → Web Storage 609  
Skalieren → transform:scale 364  
small-caps → font-variant 165  
source (Tag) 642  
span (Tag) 94  
Sprungmarke 37  
Sprungziel 38  
Stacking Context 304  
Statische Positionierung → position 285  
String 407  
    charAt 408  
    indexOf 409  
    lastIndexOf 409  
    length 408  
style (Attribut) 94  
style (Tag) 98  
Stylesheet → CSS 93  
Submit Button → button 217  
switch 623

## T

Tabelle → table 187  
Tabellenkopf (Tag) 191  
Tabellenspalte → col 190  
Tabellenzeile (Tag) 188  
Tabellenzelle (Tag) 188  
table (Tag) 187

Tag 31  
    leeres 35  
    öffnendes 31  
    schließendes 31  
    verschachteln 32  
Tastaturevent  
    keydown 505  
    keypress 505  
    keyup 505  
tbody (Tag) 188  
td (Tag) 188  
    colspan 204  
    rowspan 204  
text-align 167  
textarea (Tag) 240  
text-decoration 166  
Texteditor 28  
Textschatten 341  
text-shadow 341  
text-transform 165  
tfoot (Tag) 191  
th (Tag) 191  
thead (Tag) 191  
this 563  
throw → Exception 466  
toFixed 399  
Touch-Steuerung 772  
tr (Tag) 188  
transform 364  
    rotate 366  
    rotateX 370  
    rotateY 370  
    rotateZ 370  
    scale 366  
    skew 366  
    translate 366  
    translateZ 370  
transform-origin 368  
transition-delay 361  
Transformation (CSS) 364  
Transformation (Canvas) 666  
Transitions 359  
transition-timing-function 361  
Transparenz (Bild) 51  
Truthy 430  
try ... catch → Exception 466

# U

Überschrift 34  
  *Ebene* 34  
ul (Tag) 174  
Umfließen → float 287  
undefined 390  
Unicode 54  
Uniform Resource Locator → URL 65  
Unterstrichen → text-decoration 166  
Unterverzeichnis 41  
URL 65  
  *absolute* 42  
  *Fragment* 67  
  *Port* 68  
  *Protokoll* 65  
  *Query String* 67  
  *relative* 52  
UTF-8 → Unicode 55

# V

valid (Pseudoklasse) 247  
var 389  
Variable 389  
  *globale* 441  
  *lokale* 441  
Vererbung 576  
Vergleichsoperatoren 420  
Verschieben → transform:translate 364  
Versteckte Felder 239  
Verzeichnis 40  
Verzweigung  
  *if...else* 418  
  *switch* 623  
video (Tag) 640  
visited (Pseudoklasse) 127  
Void-Element 48

# W

WAP 742  
Webbrowser → Browser 24, 27  
Webfonts → @font-face 160  
WebM 644

Webserver 73  
  *für Linux* 76  
  *für MacOS* 78  
  *unter Windows* 74  
Web Sockets 736  
Web Storage 599  
  *Größenlimit* 607  
while 530  
width 48, 261  
window  
  *addEventListener* 488  
  *event* 490  
  *history* 722  
  *innerHeight* 508  
  *innerWidth* 508  
  *pageXOffset* 508  
  *pageYOffset* 508  
  *setTimeout* 509  
  *URL* 720  
WML 742  
Wohlgeformt 32  
WYSIWIG 28

# X

XMLHTTP 701  
XMLHttpRequest 702  
  *Datei-Upload* 721  
  *Events* 704  
  *Level 2* 712  
  *open* 705  
  *send* 705

# Z

z-index 303  
Zeichenkette → String 407  
Zeitgesteuerte Ereignisse → window:setTimeout 509  
Zentriert → text-align 167