
Kai Günster
Mit Syntax-Highlighting!
Inkl. Downloads
Schrödinger lernt HTML5, CSS & JavaScript
Das etwas andere Fachbuch
Lerne die Sprachen des Webs von Grund auf
Mobile Layouts, Geolocation, Touchevents, Audio & Video... alles drin
Durchblicken, mitmachen und genießen!
VIERTE AUFLAGE
Rheinwerk Computing

# —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.

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 einigen 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.

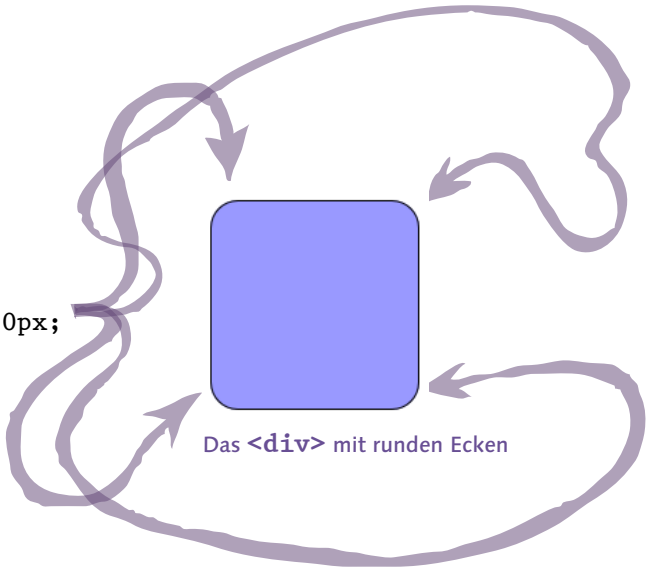
Die Geschichte der runden Ecke ist eine Geschichte voller Bilder und jammernder Webentwickler.

Die Ecken des Anstoßes

Langer Rede wenig Sinn, die 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.

**[Einfache Aufgabe]**
Setze an einem **`<div>`** mit 200 Pixeln Breite und Höhe runde Ecken mit einem Radius von 20 Pixeln.


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



Das **`<div>`** mit runden Ecken

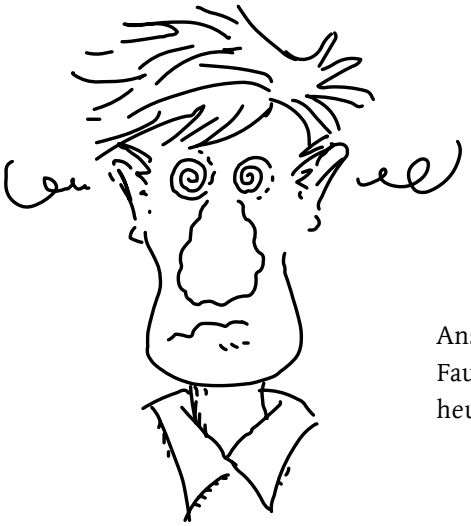

**[Notiz]**
Um einen **`border-radius`** zu haben, muss übrigens nicht unbedingt eine **`border`** da sein. Auch wenn kein Rahmen da ist, bekommt der **Hintergrund** die richtige Form.

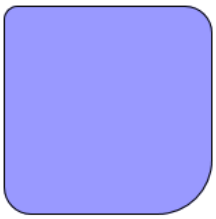
Auch der Eckradius lässt sich natürlich wieder für jede Ecke einzeln setzen, aber die Möglichkeiten dafür sind leider beide etwas unhandlich. Die CSS-Eigenschaften, um einzelne Ecken zu runden, heißen zum Beispiel **`border-top-left-radius`** oder **`border-bottom-right-radius`**. Ich sag's ja, unhandlich.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

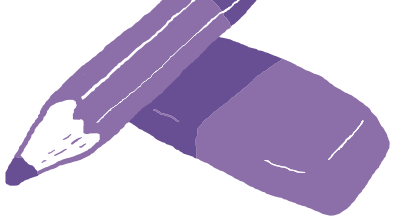
```
div {
      border-radius: 10px 20px 40px;
}
```





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.

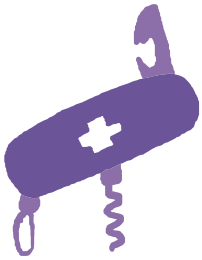


**[Notiz]**

Es gibt auch die Möglichkeit, die Rundung einer Ecke zu strecken oder zu stauchen, indem du an die Einzeleigenschaften für jede Ecke zwei Werte übergibst 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 **`border`**s für alle vier Seiten, und schau es dir an.

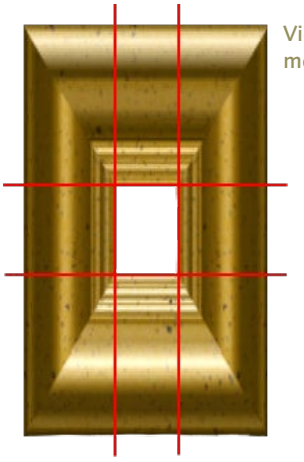


Die Übergänge sehen doch richtig gut aus.
Nicht von der Katze ablenken lassen!

# Rahmenbilder für Bilderrahmen

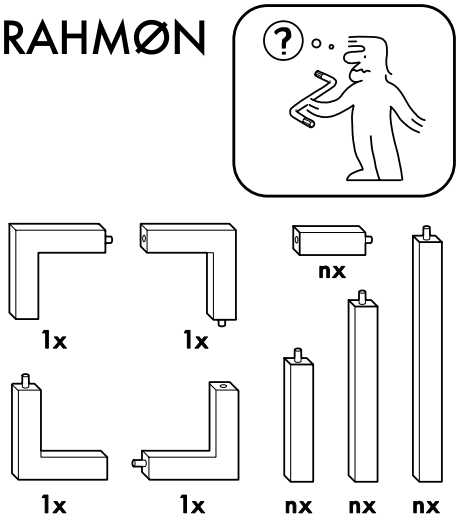
Das mit den Rahmen sieht alles schon gut aus, du kannst deine Webseiten mit solchen Kleinigkeiten echt aufwerten. Wenn du die Fotos aus dem nächsten Urlaub mit einem schönen, abgerundeten Rahmen online stellst, dann kannst du deine Freundin vielleicht überreden, nicht alle Digitalfotos drucken zu lassen. 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.


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.



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

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 Kurzschreibweise **`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;
}
```

*1 Oben und unten wird das Rahmenbild gestaucht und wiederholt.

*2 Links und rechts wird ein einzelnes Seitenteil gestreckt.

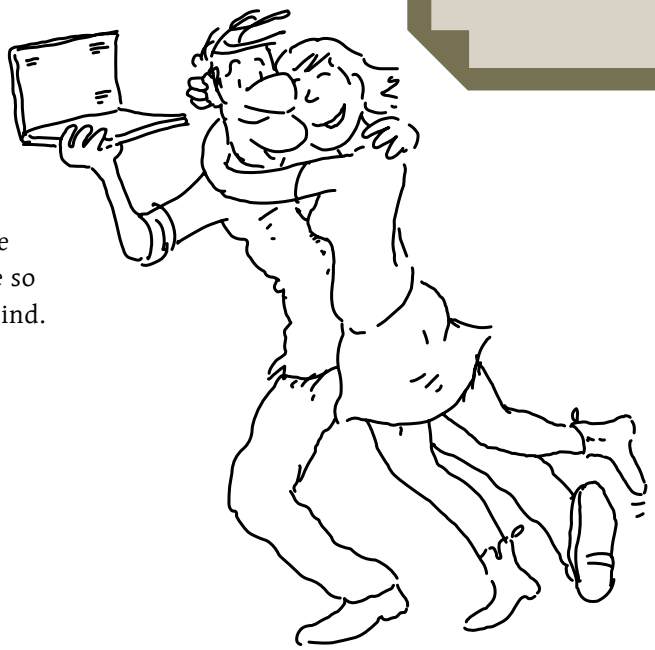
Du hast dir bestimmt schon gedacht,
dass es auch hierfür eine **Kurzschreibweise** gibt.


*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

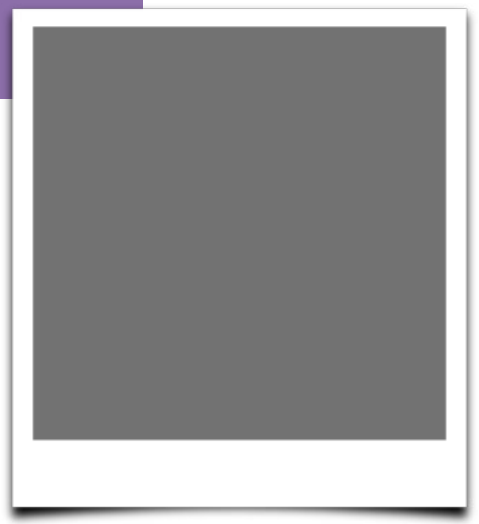
Rahmenbilder sind nicht unbedingt das einfachste Thema in CSS, sollen wir das noch mal Schritt für Schritt durchgehen?

*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:

```
<img src="bg2.png" alt="Schottland">
```

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.

## Vom Bild zum Polaroid

**[Code bearbeiten]**
Setze für dein Foto ein Rahmenbild, und schneide es an der Linie, die du oben ermittelt hast.

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

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

*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**

*1 Diese beiden bleiben unverändert.

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

*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

Sehr 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.

**[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 Raute
davor 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.

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

```
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;
}
```

*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

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

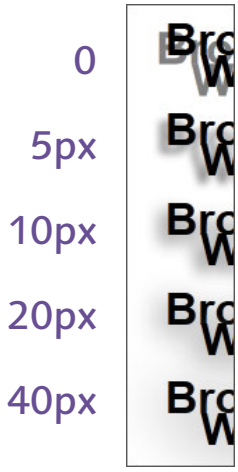
Browser Wars

Browser Wars 2:
Der nächste Versuch

So richtig gefällt mir das aber nicht.
Erstens haben Schatten keine so klaren Umrisse,
und zweitens find 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.




Von oben nach unten
immer weicher gezeichnet

**[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;
```

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


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.**

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.

**[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?


Browser Wars 4:
Jetzt sieht's heiß aus.

*1 Ich hab dann auch mal die Textfarbe geändert …

```
color: orange;*1
text-shadow: 0 0 10px orange;*2
```

*2 … 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.

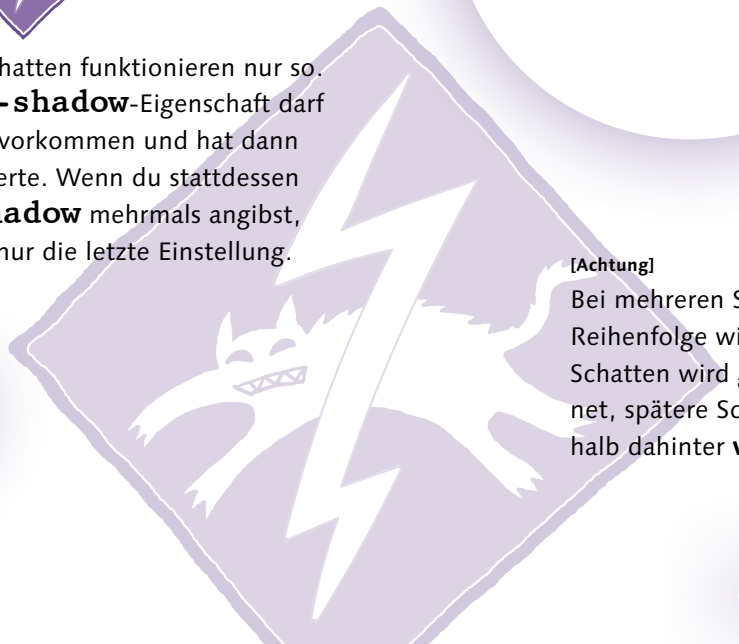
```
text-shadow: 5px 5px 40px yellow*1,
             10px 10px 30px orange*2,
             20px 20px 20px red*3...
```

*1 Ein Schatten, ...

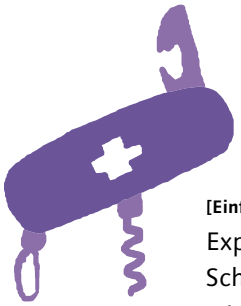
*2 ... zwei Schatten, ...

*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.



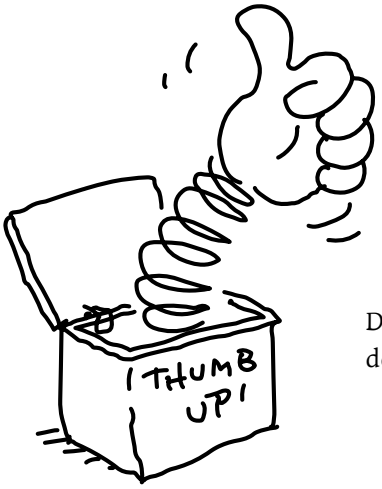
**[Achtung]**
Bei mehreren Schatten kann die Reihenfolge wichtig sein: Der erste Schatten wird ganz oben gezeichnet, spätere Schatten können deshalb dahinter **verschwinden**.



**[Einfache Aufgabe]**

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


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.

**[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  mit einem Textschatten vorkommt.

Der Textschatten, wie wir ihn kennen und lieben

Dies ist ein Text, in dem gleich ein  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:

*1 Die neue Eigenschaft heißt **box-shadow**, keine Überraschung.

*2 Als erste und zweite Parameter kommen immer noch die horizontale und vertikale Verschiebung, ...

```
box-shadow*1: 10px*2 10px*2 5px*3 #AAAACC*4;
```

*3 ... als Drittes die Unschärfe ...

*4 ... und als Viertes die Farbe. So einfach ist das.

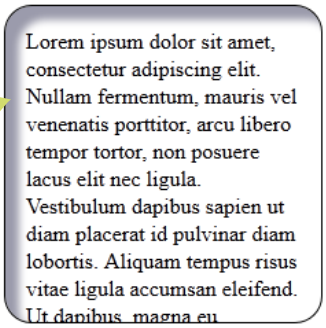
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam fermentum, mauris vel venenatis porttitor, arcu libero tempor tortor, non posuere lacus elit nec ligula. Vestibulum dapibus sapien ut diam placerat id pulvinar diam lobortis. Aliquam tempus risus vitae ligula accumsan eleifend. Ut danibus,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

Eine Box mit Schatten

**[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.

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;
```

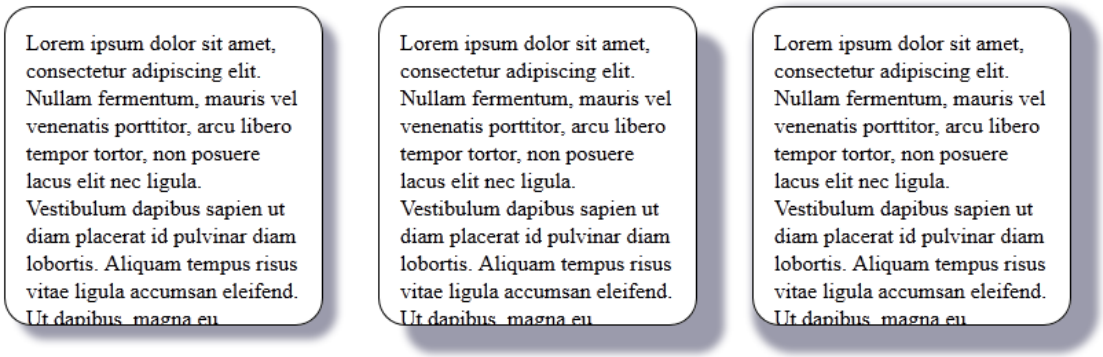



Schatten im Inneren

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 mach 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.

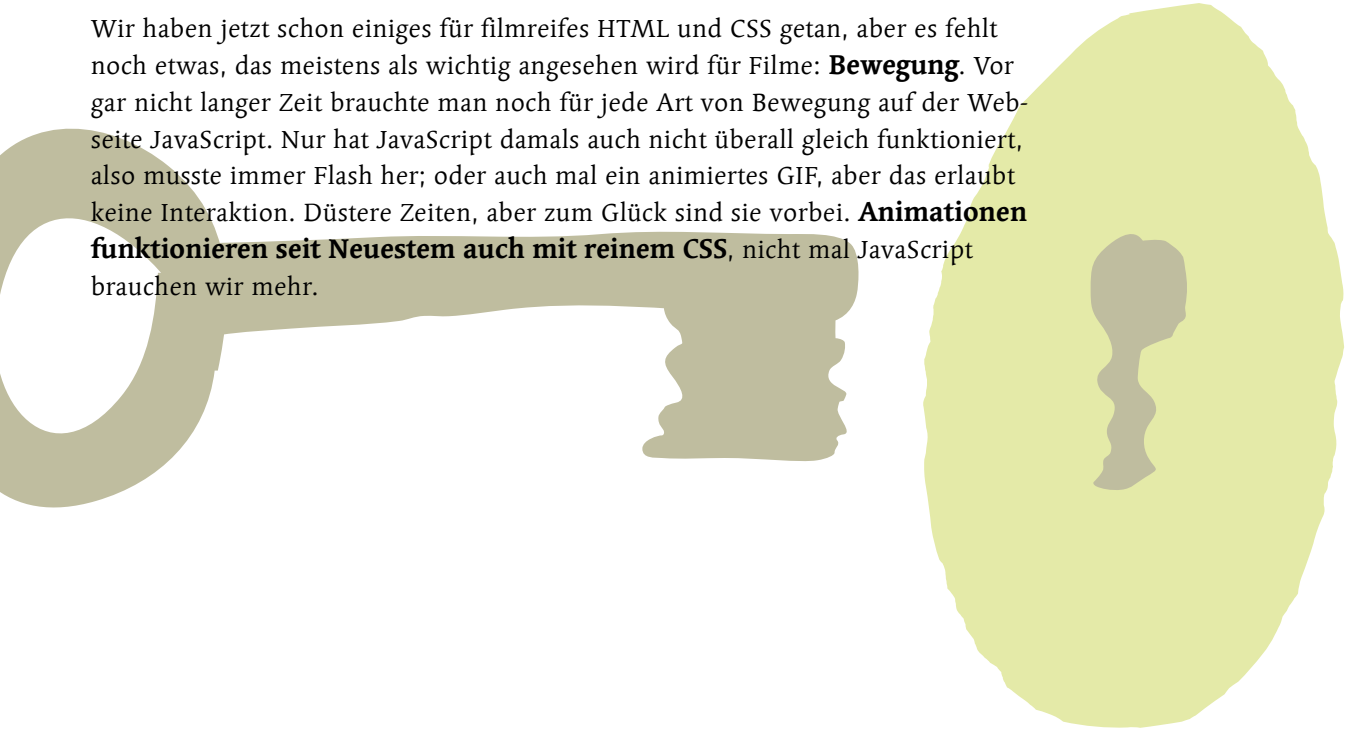



Von links nach rechts: 10px Schatten, 20px Schatten und 10px Schatten mit 10px Spread

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.

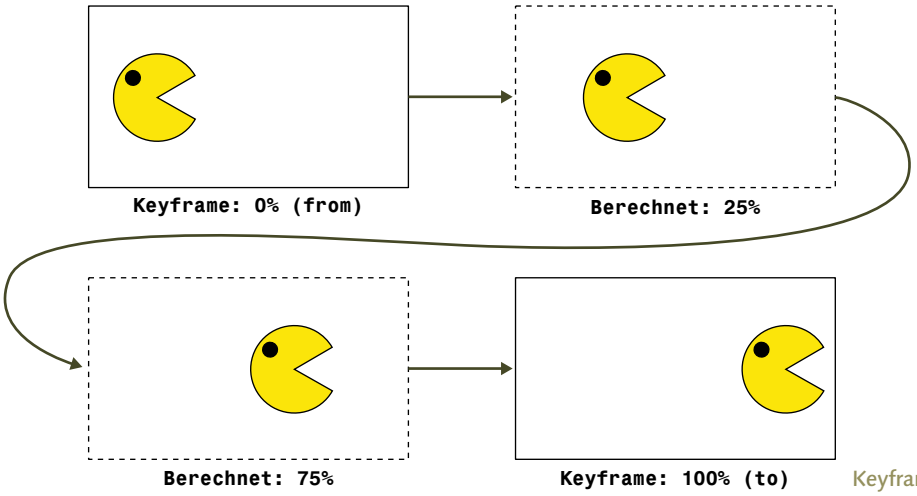
## Schlüsselmomente

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.



**[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.




Keyframe-Animation

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.

*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}
}
```

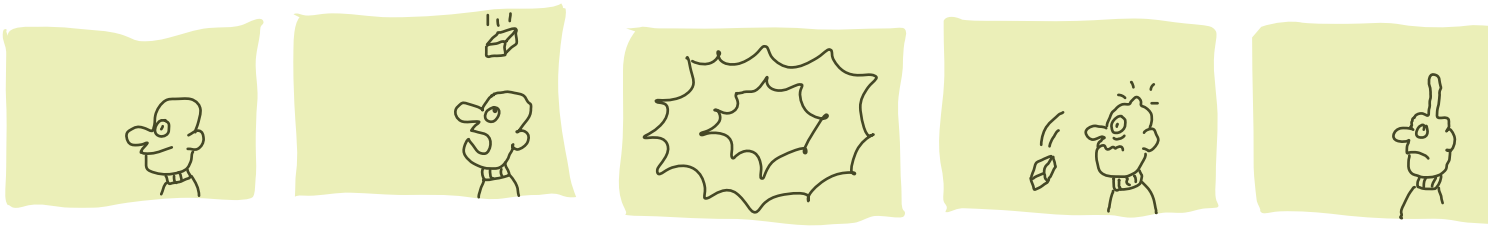


*3 Der **Keyframe-Selektor** gibt an, an welche Stelle der Animation dieser Keyframe gehört. **`from`** (oder 0 %) ist der erste Frame der Animation, **`to`** (oder 100 %) der letzte. Die Prozentangaben dazwischen kommen an die passende Stelle. Es gibt **keine** Angabe, wie lang die Animation insgesamt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

**[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 Daumenkinos gemalt haben.



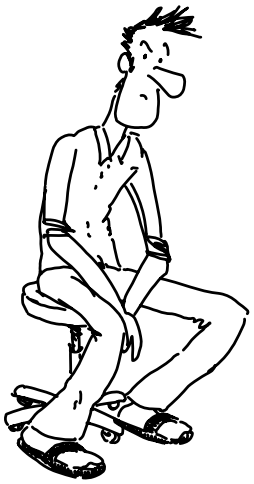
Damit ist die Animation definiert, aber es bewegt sich noch nichts, es wird schließlich nirgends angegeben, welche Elemente animiert werden sollen. Aber dafür brauchen wir keine komplexe, neue Syntax mehr, du musst nur drei neue CSS-Eigenschaften lernen:

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

*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.



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.

Jetzt macht eure Schreibfaulheit es aber etwas kompliziert, Leute ...

# Und es bewegt sich doch

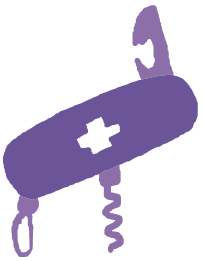
Grau ist alle Theorie. Also, in unserem Fall natürlich nicht, wir haben ja die Farbe animiert. Aber alle Theorie ist doch theoretisch und oft ein wenig langweilig. Lass uns etwas animieren!

*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 rechts und wieder zurück bewegen.

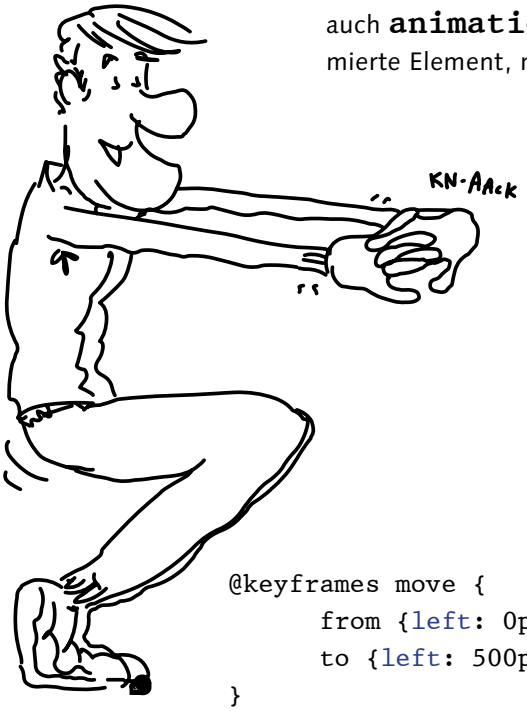


*Hin UND zurück? Wie mach ich denn das?*

**Lass ich ihn von 0–50 % nach links laufen und dann von 50–100 % wieder zurück?**

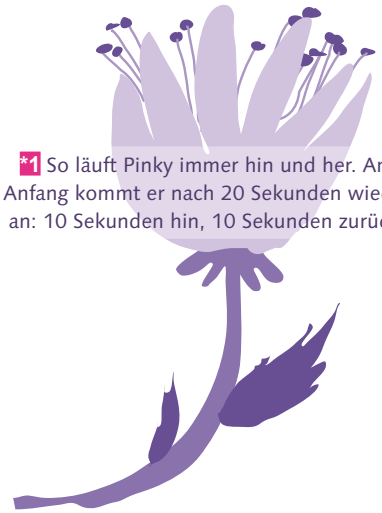
Das kannst du machen, aber einfacher geht es mit einer zusätzlichen Eigenschaft (und schreibfauler auch, wir sind ja schließlich Entwickler).

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



```
@keyframes move {
      from {left: 0px;}
      to {left: 500px;}
}
#move_me {
      position: absolute;
      animation-name: move;
      animation-duration: 10s;
      animation-direction: alternate;*1
      animation-iteration-count: infinite;
}
```

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

**[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

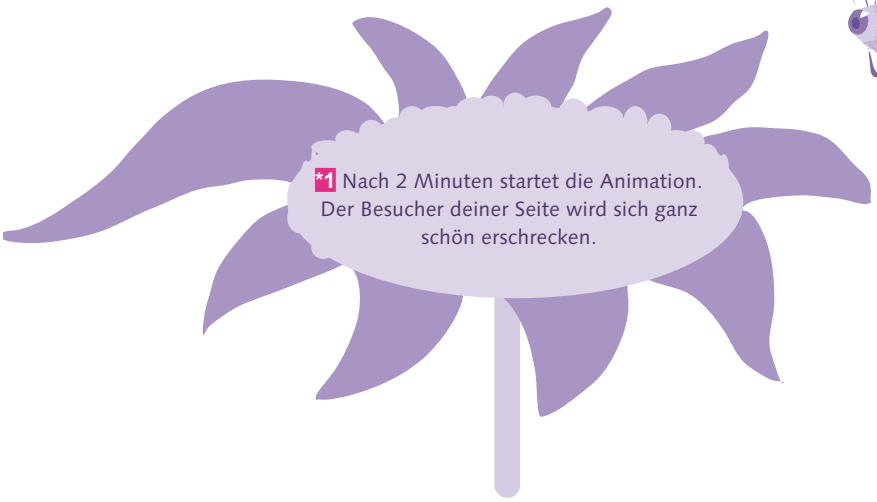
```
from {left: 0;}
to {right: 0;}
```

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 verschieden Eigenschaften.

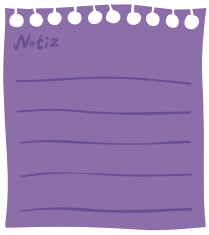
**[Notiz]**
Anstatt umzudrehen, läuft Pinky rückwärts wieder zum Anfang. Umdrehen würde einiges an weiterer Trickserei erfordern.

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.

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

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

*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.


**[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“.

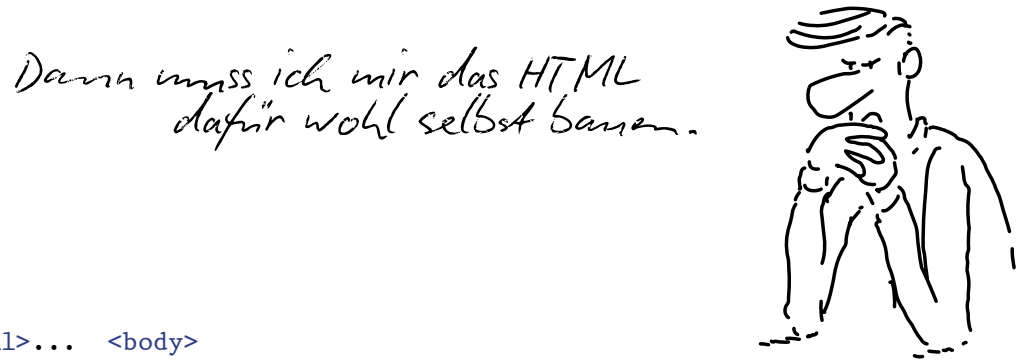
```
#hoverme {
      background-color: red;*1
      color: blue;*1
      transition-properties: background-color, color;*2
      transition-duration: 5s;*3
      ...
}
#hoverme:hover {
      background-color: blue;*4
      color: red;*4
}
```

*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.

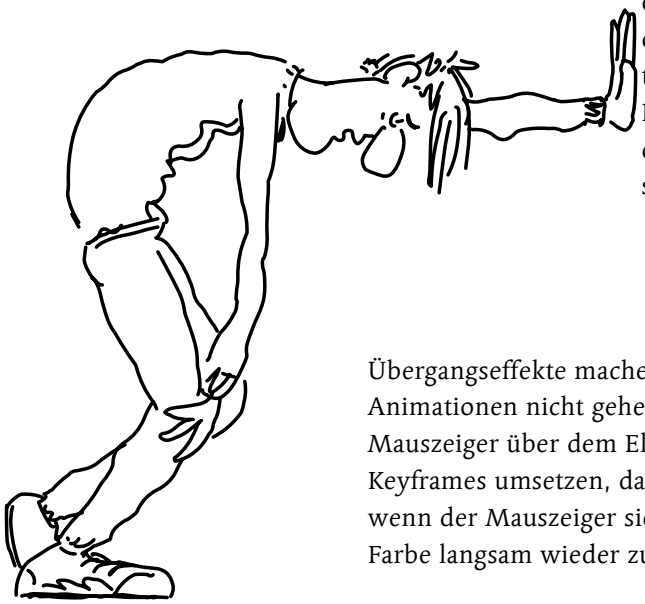
*3 Und so lange soll der Übergang dauern. Laut Spezifikation kannst du hier auch für jede Eigenschaft eine eigene Übergangszeit angeben, das wird aber noch von keinem Browser unterstützt.

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




```
<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 Interkation 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 wegbewegt. 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 auch in dieser Auflage immer 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 Transitions geht alles wieder zurück auf Anfang, wenn du den Mauszeiger wegziehst. Um wirklich zeigen zu können, wie dein Kaffee aussehen soll, sollte die Farbe erhalten bleiben. Deshalb kommt die gleiche Technik zum Einsatz, die schon Pinky, das Gespenst, eingefangen hat:

```
#coffee {
      width: 200px;
      height: 200px;
      animation-name: coffee;
      animation-duration: 5s;
      animation-iteration-count: infinite;*1
      animation-direction: alternate;*1
      animation-play-state: paused;*2
}
#coffee:hover {
      animation-play-state: running;*3
}
@keyframes coffee {
      from {background-color: #423027;}
      to {background-color: #A77F6B;}
}*4
```

*1 Wenn man einmal an der richtigen Kaffeefarbe vorbei ist, dann soll man auch wieder zurück können. Die Animation läuft deshalb endlos vor- und rückwärts.

*2 Solange niemand über das `<div>` rübermaust, pausiert die Animation.

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

*4 Die Animation hat nur Anfang und Ende, keine Zwischenschritte, und ändert die Hintergrundfarbe.

# 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!

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

### Effekt Nummer eins – gedrehter Text

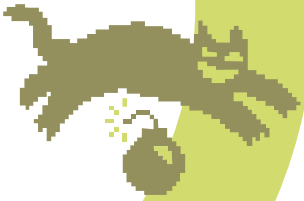
Ich lege einen Textabsatz an und weise ihm die Style-Eigenschaft `transform: rotate(-2deg);` zu.

Lorem ipsum dolor sit amet, consectetur 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 sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Cras ac odio non urna sagittis ornare.

Textabsatz um zwei Grad gegen den Uhrzeigersinn gedreht.

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!

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

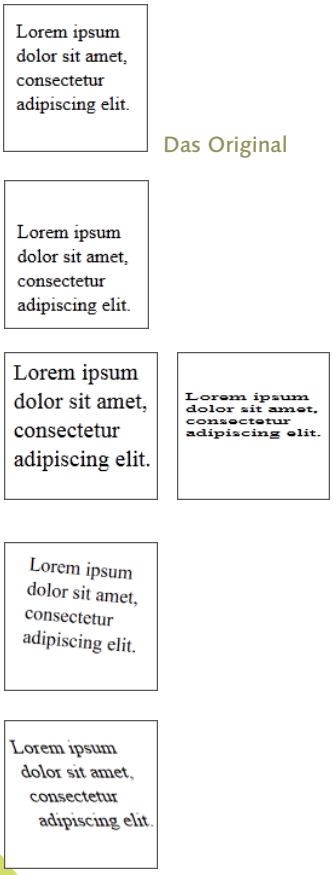
Lorem ipsum dolor sit amet, consectetur adipiscing elit.

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 geschert

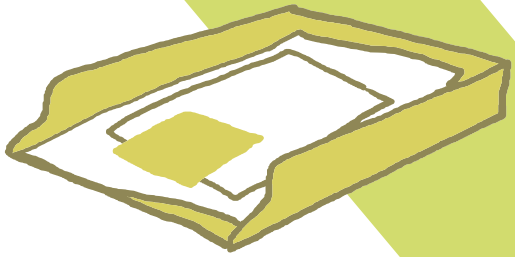
## Und so sehen die Transformationen in CSS aus:

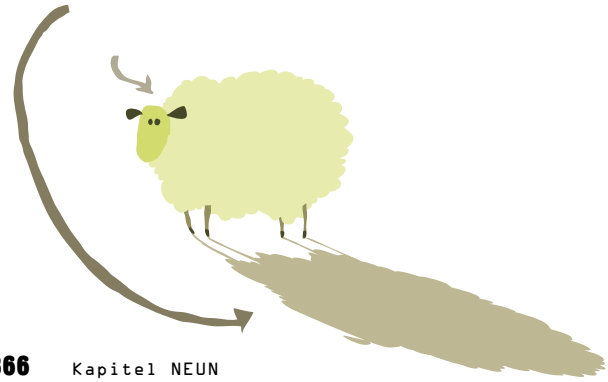
| | | |
|---|---|---|
| `transform: translate(x, y)` | Verschieben | x und y geben an, wie weit das Element nach rechts bzw. nach unten verschoben werden soll. |
| `transform: scale(x)`<br>`transform: scale(x, y)` | Skalieren | Die Größe des Elements wird mit dem angegebenen Faktor multipliziert, das 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. |
| `transform: rotate(xdeg)` | Rotieren | Dreht das Element um x Grad im Uhrzeigersinn. Dabei muss **`deg`** als Einheit immer angegeben werden. |
| `transform: skewX(xdeg)`<br>`transform: skewY(xdeg)` | Scheren | Das Element wird geschert, das heißt, eine Kante wird verschoben, so dass ein Parallelogramm entsteht. Bei **`skewX`** wird die untere Kante verschoben, bei **`skewY`** die rechte. |




**[Ablage]**

„Geschert“, nicht „geschoren“.
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 ...

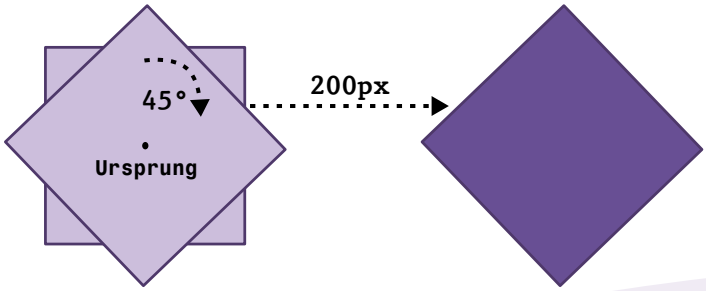
*3 ... und dann noch auf das Doppelte vergrößert.

**[Einfache Aufgabe]**

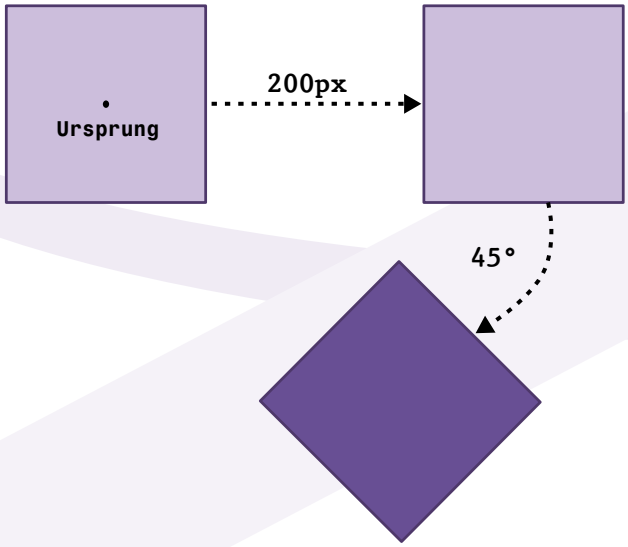
Bei mehreren Transformationen ist die Reihenfolge extrem wichtig. Schau dir einfach mal den Unterschied an zwischen `transform: rotate(45deg) translate(200px, 0px);` und der Transformation in umgekehrter Reihenfolge.

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.

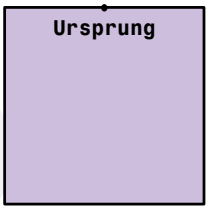
*1 Zwischen links und rechts soll der Ursprung in der Mitte liegen ...

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

*2 ... und ganz am oberen Rand.

Ähm ... und was heißt das?

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




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]**

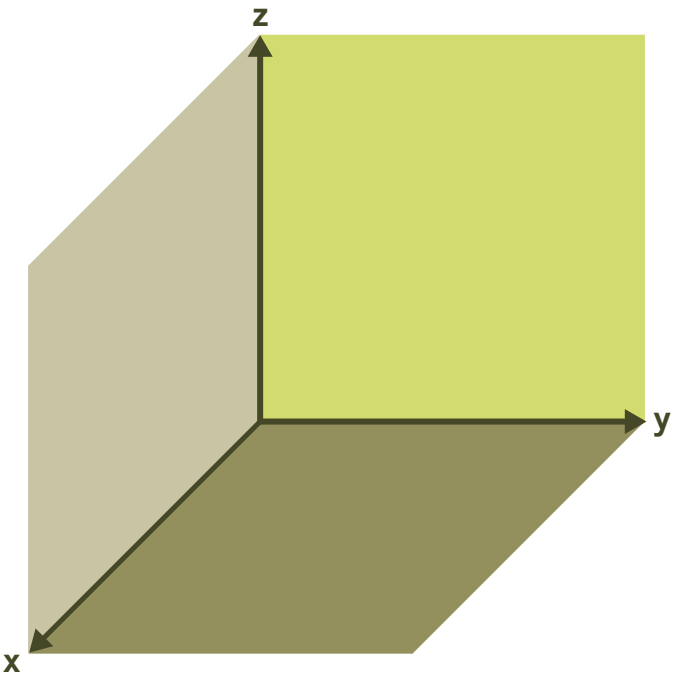
Für den mathematisch begabten Webentwickler lassen sich auch alle Transformationen als eine 3x2-Matrix mit der Eigenschaft **`matrix(a, b, c, d, e, f)`** anwenden, aber die Matrizendarstellung von Transformationen macht nicht jedem Spaß und würde hier auch zu tief in die Mathematik führen.

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

**[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.

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

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

*1 Höhe und Breite zu setzen, ist inzwischen ein alter Hut.

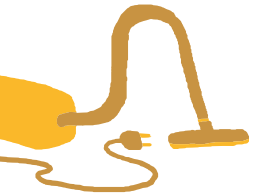
*2 **`margin: auto;`** ist der einfachste und zuverlässigste Weg, ein Blockelement zu zentrieren.

*3 Bei **`border-image`** müssen zuerst die URL und danach die Schnittkanten angegeben werden. Dieser Fall ist ein klein wenig besonders, weil es oben und unten keinen Rand gibt, diese beiden Kanten können auf 0 gesetzt werden.

```
#leinwand {
      width: 640px;*1
      height: 480px;*1
      margin: auto;*2
      border-image: url(vorhang.png) 0 151 0 151 repeat;*3
      border-width: 0 151px;*4
      border-style: solid;*4
      overflow: hidden;*5
}
```

*5 Und zuletzt noch den Überlauf verstecken, fertig!

*4 **`border-width`** und **`border-style`** müssen da sein, damit du überhaupt einen Rand zu sehen bekommst.



**[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**.

```
#leinwand h1*1 {
    color: orange;
    padding: 50px;*2
    font-family: sans-serif;*3
    font-weight: bold;*3
    font-size: 32px;*3
    text-align: center;*3
    line-height: 0.7;*3
    text-shadow: 0 0 40px yellow,
                 0 0 30px yellow,
                 0 0 20px yellow,
                 0 0 10px orange,
                 0 0 5px orange,
                 0 0 2px red;*4
}

#leinwand p*1 {
    margin: 20px;*5
    font-family: sans-serif;*3
    text-align: justify;*6
}
```

*1 Natürlich könnte man der Überschrift und den Absätzen auch eigene Klassen geben, aber es sollen jeweils alle auf der Leinwand gleich gestylt werden, also geht es mit dem Nachkommen-Selektor.

*2 Das **padding** um die Überschrift dient dazu, sie etwas vom oberen Rand und dem Text darunter abzusetzen.

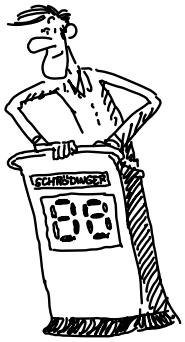
*3 Hier folgen einige weitere Eigenschaften, damit es besser aussieht: Schriftart, Schriftgröße und so weiter.

*4 Insgesamt sechs Textschatten sorgen dafür, einen satten, glühenden Text zu bekommen.

*5 Die Absätze sollen etwas Abstand zueinander und vor allem zum Vorhang haben.

*6 Im Blocksatz sieht das alles viel besser aus.

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



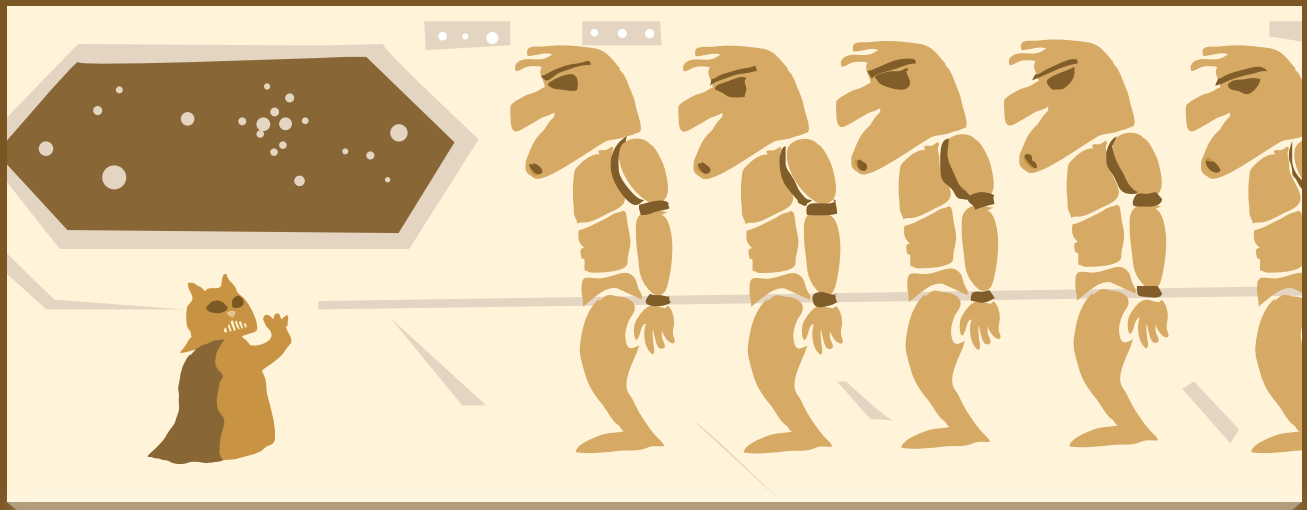

Dann wird dich der nächste Schritt ja auch kaum überraschen. Film ab!

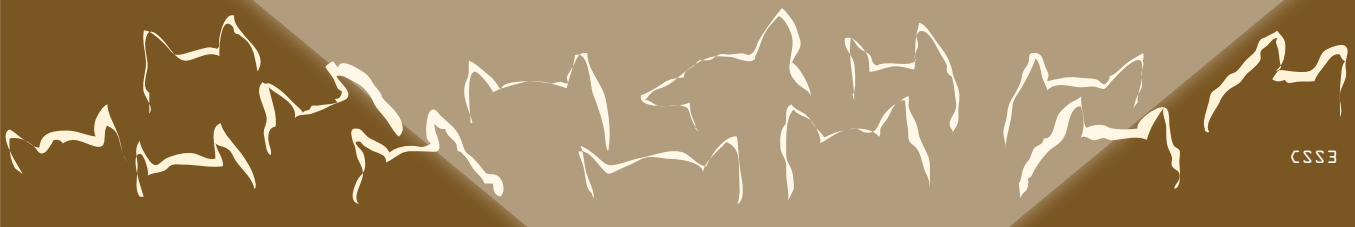
**[Schwierige Aufgabe]**

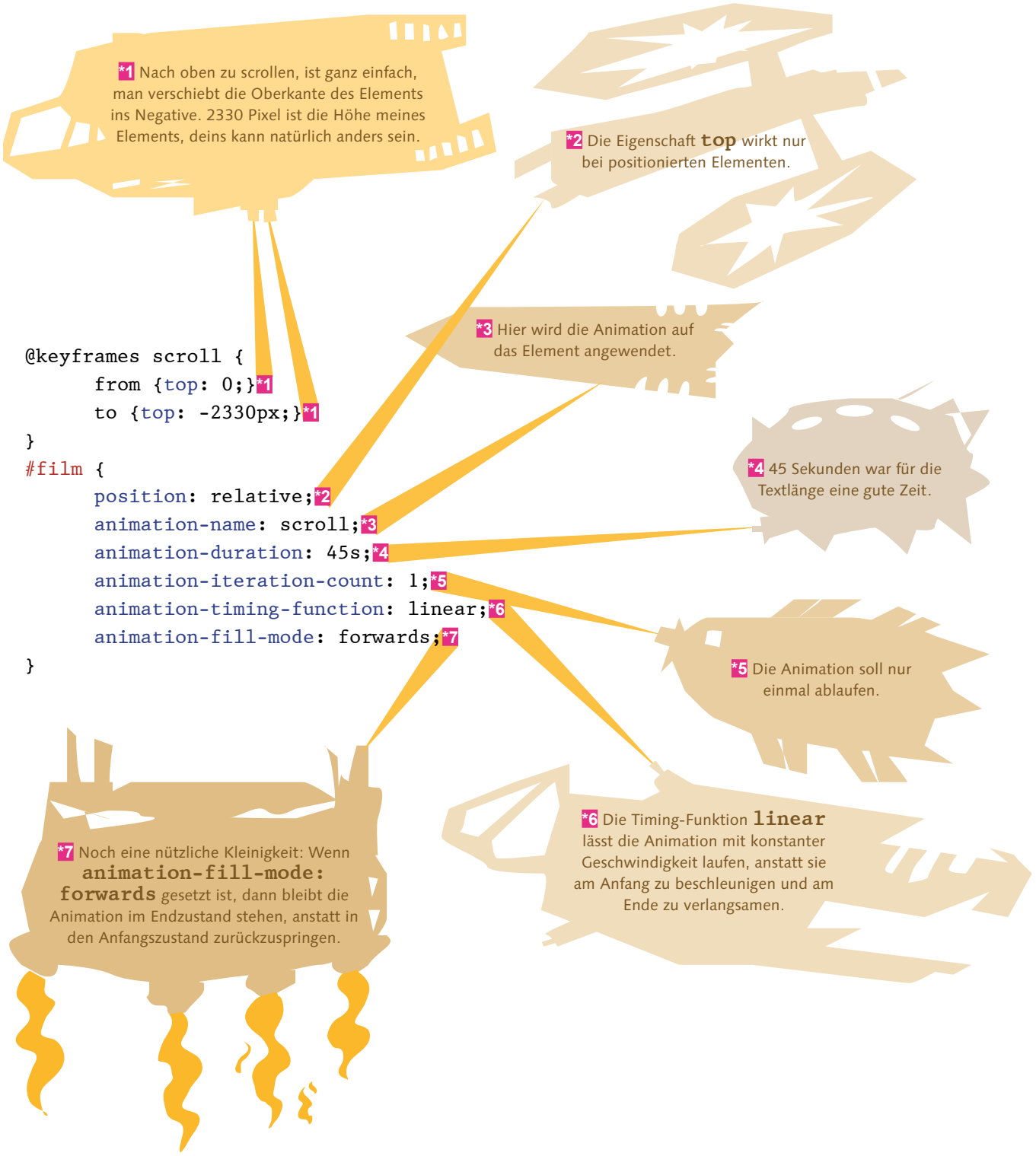
Wenn wir schon einen Film drehen, dann soll er sich auch bewegen. Füge zunächst innerhalb der Leinwand um die Überschrift und alle Absätze ein weiteres **`<div>`** mit der ID **`film`** hinzu. Dieses neue **`<div>`** soll per Keyframe-Animation einmal nach oben scrollen, bis es komplett verschwunden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

**[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.
*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.
*6 Die Timing-Funktion linear lässt die Animation mit konstanter Geschwindigkeit laufen, anstatt sie am Anfang zu beschleunigen und am Ende zu verlangsamen.
*7 Noch eine nützliche Kleinigkeit: Wenn animation-fill-mode: forwards gesetzt ist, dann bleibt die Animation im Endzustand stehen, anstatt in den Anfangszustand zurückzuspringen.
@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
}

## 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**.

```
#leinwand {
  ...
  perspective: 100px; *1
}

#viewport {
  width: 640px;
  height: 480px;
  transform: rotateX(15deg); *2
  overflow: hidden; *3
}
```

*1 Die Perspektive muss an einem umgebenden Element gesetzt werden, die Leinwand bietet sich dafür an.

*2 eine Rotation um die X-Achse, nichts leichter als das

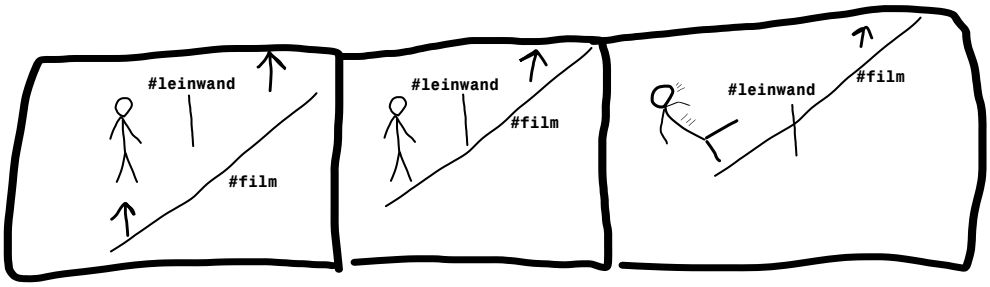
*3 und die Lösung für das Firefox-Problem

Haha! Ich hatte Recht.

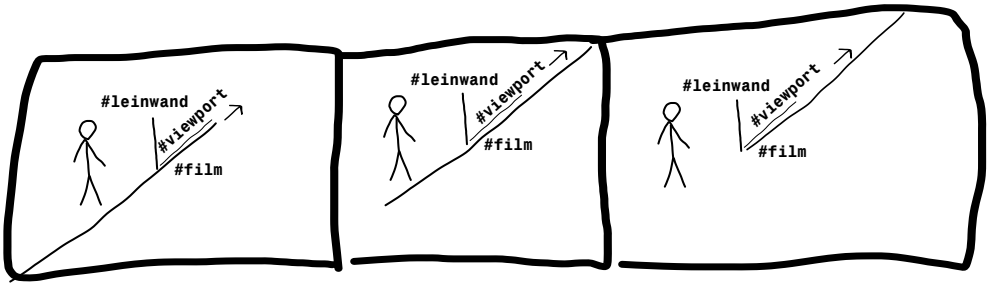
**Ein echter Klassiker, dieser Browser-Wars-Film. Aber warum das neue `<div>` für die Transformation? Dieses `viewport`-Ding meine ich.**

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 .

**[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.

# Wie in der Zeitung – mehrspaltiges Layout

Oho, Schrödinger. Mit der Tageszeitung beim Frühstück. Ich bin direkt ein wenig überrascht, dass du noch auf toten Bäumen liest. Ich krieg das alles online.

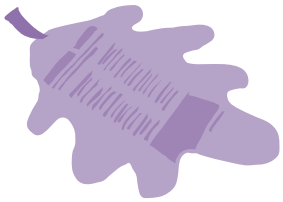
Ich find die Zeitung morgens **immer noch super. Viel gemütlicher zu lesen als auf einem Tablet … und ja, nimm dir ruhig den letzten Kaffee, den hast du ja jetzt sowieso schon. Aber grad hab ich nicht wirklich gelesen. Ich habe darüber nachgedacht, ob ich so ein typisches Zeitungslayout in HTML umsetzen könnte, bei dem vielen Webdesign, das ich jetzt schon gelernt habe.**

**Oh, sorry,** ich dachte, du bist fertig mit dem Kaffee. Ich mach gleich neuen. Und ich finde es richtig cool, dass du so viel Spaß am Webdesign gefunden hast, dass du freiwillig über so was nachdenkst. Und was ist deine inzwischen schon recht professionelle Meinung dazu?

Die Artikel und Bilder anzuordnen **ist eigentlich nur Fleißarbeit. Das sind positionierte Blockelemente, wie du es mir neulich erklärt hast. Aber ich weiß nicht, wie ich den Text von einem Artikel in mehrere Spalten packen kann.**

Ja, ich sehe dein Problem. Du kannst zwei Spalten für einen Text nicht einfach als zwei ** **s anlegen, denn dann müsstest du ja den Text von Hand zwischen den beiden verteilen. Änderst du dann etwas am Text, musst du Worte umverteilen, bis die Spalten wieder gleich sind. Keine schöne Arbeit, und du wirst es kaum so hinkriegen, dass es bei allen Lesern gut aussieht, mit verschiedenen Bildschirmgrößen und Zoomstufen.

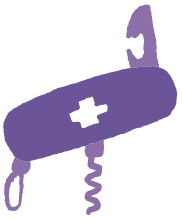
Ja, das sind genau die Probleme, über die ich auch nachdenke.



Und du bist zum Glück nicht der Erste, der sich darüber Gedanken macht. Mit den Mitteln, die du bisher kennst, ist es wirklich kaum möglich, einen **Text auf mehrere Spalten** zu **verteilen**. Und weil sich das schon andere Designer gewünscht haben, gibt es inzwischen spezielle CSS-Eigenschaften dafür. Und wie so vieles in HTML und CSS ist es viel einfacher, als du glaubst.

**[Funktioniert in]**

Einige fortgeschrittene Eigenschaften für mehrspaltige Layouts funktionieren auch heute noch nicht zuverlässig. Was wir uns hier anschauen, wird aber von allen Browsern unterstützt.



**[Einfache Aufgabe]**

Lege eine Seite mit einigen Absätzen Lorem-Ipsum-Text in einem ** ** an. Wende auf das ** ** die unten gezeigten CSS-Regeln an.

Im Markup nichts Neues…

```
<div class="spalten">
    Lorem ipsum…
</div>
```

…und im CSS nichts Schwieriges

```
.spalten {
    width: 100%;
    column-count: 4;*1
}
```

*1 Ja, so einfach könnte es sein.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur egestas odio fringilla dictum dictum. Suspendisse potenti. Etiam at aliquam diam, sit amet semper felis. Ut vel justo sit amet dolor blandit fermentum. Curabitur tellus mi, dapibus eget mattis et, eleifend maximus lacus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Maecenas augue felis, scelerisque semper tempus facilisis, tincidunt eget nulla. Nunc efficitur imperdiet sem, quis euismod magna finibus at. Quisque dictum nec nunc ac consectetur. Aenean odio nisl, pharetra non consectetur quis, ullamcorper sit amet neque.

Vestibulum odio erat, bibendum non justo ac, vulputate fermentum lacus. Duis risus dolor, suscipit eget mi id, dapibus consequat purus. Phasellus egestas lacus non aliquet dictum. Duis tortor ligula, ornare nec diam sit amet, viverra eleifend erat. Vestibulum libero nisi, porttitor vel pharetra at, pharetra sed lorem. Aliquam non ante nunc. Pellentesque dui erat, lobortis in elementum non, tincidunt vitae ex. Sed ligula sapien, malesuada non justo sed, iaculis tempus lorem. Integer posuere et ante quis lacinia. Aenean a felis non est posuere maximus. Proin vitae pulvinar erat. Sed luctus accumsan tristique.

In euismod quam ut tellus pretium, in scelerisque lectus faucibus. Sed ante velit, eleifend vitae iaculis ut, aliquam eu tellus. Vivamus pellentesque tincidunt varius. Fusce lobortis velit lacus, sit amet sollicitudin tellus facilisis a. Nunc feugiat bibendum dui eu tristique. Cras sollicitudin viverra mi, non blandit dolor tincidunt non. Quisque varius

Ein Text in vier Spalten – für den Anfang nicht schlecht

Das war ja schon wieder zu einfach.

Ja, oder? Natürlich gibt es da mal wieder viel mehr hinten dran, aber dein Hauptproblem ist gelöst: Text wird automatisch auf die Spalten verteilt, du musst das nicht von Hand machen. Es wird jetzt allerdings noch nicht überall gut aussehen. Dadurch, dass du eine feste Spaltenzahl vorgibst, werden die Spalten zu schmal, wenn das Browserfenster schmal ist. Oder wenn jemand sein Tablet im Hochformat hält statt im Querformat. Deswegen solltest du immer auch eine **Mindestbreite** für die Spalten angeben.

**[Einfache Aufgabe]**

Zieh das Browserfenster mit der oben angelegten Seite breiter und schmaler. Füge dann die CSS-Regeln unten hinzu, und vergleiche das Verhalten vorher und nachher.

Sorgt für Mindestbreiten

```
.spalten {
    …
    column-width: 12em;*1
}
```

*1 Auch wenn die Eigenschaft nicht explizit so heißt, handelt es sich um eine Mindestbreite.

Jetzt wird der Browser dafür sorgen, dass Spalten immer mindestens diese Breite haben. Wenn dadurch nicht so viele Spalten passen, wie du in `column-count` angibst, stellt der Browser weniger Spalten dar, verteilt aber den ganzen verfügbaren Platz auf sie. So sieht es auch in schmalen Browsern gut aus.

**[Notiz]**

`column-count` ist übrigens auch nur ein Mindestwert. Wenn der Text einfach nicht in den zur Verfügung stehenden Platz passt und das Element nicht höher werden kann, dann kann der Browser es breiter machen und weitere Spalten hinzufügen. Du kannst dieses Verhalten mit der schon bekannten `overflow`-Eigenschaft beeinflussen.

Um jetzt noch ein wenig mehr Einfluss auf das Aussehen deiner Textspalten zu nehmen, kannst du zusätzlich einstellen, wie viel Platz zwischen zwei Spalten bleiben soll und ob zwischen ihnen eine Trennlinie steht.

## Abstände und Trennlinien

```
.spalten {
    column-gap: 3em;*1
    column-rule: 1px solid #00F;*2
}
```

*1 So stellst du den Raum zwischen zwei Spalten ein …

*2 … und so die Trennlinie. Das Format entspricht dem von **border**: Rahmendicke, Linienstil und Farbe.

**[Achtung]**

Anders als **border** im Box-Model nimmt **column-rule** keinen eigenen Platz ein, sondern ist Teil des **column-gap**.

Damit hast du fast erreicht, was du möchtest. Einen wichtigen Unterschied gibt es noch zwischen deinem Layout und dem einer Zeitung: Dein mehrspaltiges Layout wird auf den gesamten Inhalt des Elements angewendet, inklusive der Überschrift. Die sollte aber über allen Spalten stehen, wie es sich für eine Überschrift eben gehört.

```
.spalten h1 {
    column-span: all;
}
```

Eine Überschrift über alle Spalten

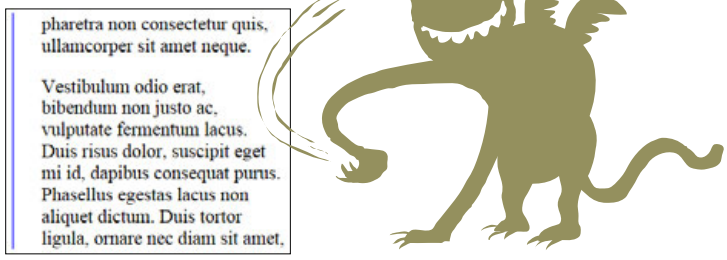
So werden **h1**-Überschriften über alle Spalten verteilt. Oder zumindest fast. Sie werden über so viele Spalten verteilt, wie du als **column-count** angegeben hast. Wenn der Browser aus Platzgründen mehr Spalten anlegt, dann erstreckt sich die Überschrift nicht auch über diese, sondern bricht bei Bedarf eher um. Das ist insgesamt noch unbefriedigend. Genauso unbefriedigend ist, dass die einzigen Werte für **column-span** **all** und **none** sind. Es gibt noch keine Möglichkeit, zum Beispiel ein Bild über zwei Spalten zu strecken. Aber der Standard entwickelt sich ja noch, diese Details kommen vielleicht in Zukunft.

**[Notiz]**

Du kannst zwar jetzt Layouts mit mehreren Textspalten erstellen, aber wenn du diese Möglichkeit nutzt, solltest du auch darüber nachdenken, ob es eine gute Idee ist. Ist zum Beispiel der Text so lang, dass der Leser hoch- und runterscrollen muss, sind mehrere Spalten nicht sehr angenehm. Lesen und runterscrollen, zurück nach oben, lesen und runterscrollen … Das ist nicht so toll. Aber richtig eingesetzt, sind mehrspaltige Layouts sehr hübsch, sogar noch hübscher mit den Details aus dem nächsten Abschnitt.

# Die richtige Textverteilung

Damit funktionieren die Spalten schon gut, aber kosmetisch glänzt noch nicht alles. Es fällt zum Beispiel auf, dass der rechte Rand jeder Spalte aussieht, als hätte die Katze ihre Krallen daran gewetzt.




Da hat die Katze wieder zugeschlagen.

Du kennst bereits ein Mittel dagegen. Wenn du mit **`text-align: justify`** das Element auf Blocksatz einstellst, dann funktioniert das auch in Spalten. Die Ränder werden geglättet. Dafür hast du jetzt viel weißen Zwischenraum in manchen Zeilen, viel besser sieht das auch nicht aus. Aber die meisten Browser haben noch einen neuen Trick gelernt, mit dem gerade schmale Spalten besser aussehen: **Silbentrennung**!

**[Funktioniert in]**
Silbentrennung mit **`hyphens`** funktioniert in allen Browsern, aber Safari kennt die Eigenschaft aktuell nur mit dem **`-webkit`**-Präfix.

## Silbentrennung einrichten

```
.spalten {
    text-align: justify;*1
    hyphens: auto;*2
    -webkit-hyphens: auto;*3
    …
}
```

*1 Den kennst du schon: Blocksatz.

*2 Silbentrennung heißt im Englischen Hyphenation, deswegen heißt die CSS-Eigenschaft **`hyphens`**. Mit **`auto`** schaltest du automatische Silbentrennung an.

*3 Wir sind ganz nah dran, dass Webseiten einfach überall gleich funktionieren. Ganz da sind wir leider noch nicht.

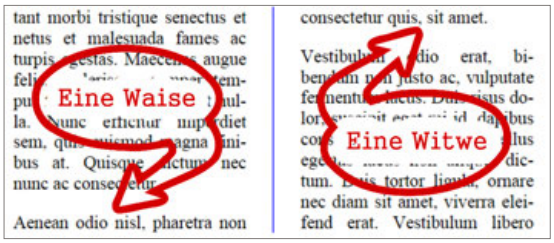
**[Zettel]**
Du kannst an einem HTML-Element angeben, in welcher Sprache sein Inhalt geschrieben ist, indem du das **`lang`**-Attribut setzt, zum Beispiel **`lang="de"`**. Die Silbentrennung kann diese Angabe verwenden, um **sprachspezifische Trennregeln** korrekt anzuwenden.

*Das sieht jetzt echt besser aus. Richtig professionell!*

**Ja, oder?** Aber für richtig professionelles Aussehen gibt es noch etwas, und dafür zieh ich dich noch mal in die abstrakten Tiefen der Typografie. Kennst du **Witwen und Waisen**?

*Das ist eine sehr düstere Frage. Aber ja, ich bin schon in einem Alter, in dem manche meiner Bekannten keine Eltern mehr haben …*

Ah, nein, sorry, nicht so düster. Ich meinte typografische Witwen und Waisen. Für Typografen ist eine Witwe die letzte Zeile eines Absatzes, wenn sie allein auf einer neuen Seite oder in einer neuen Spalte steht. Eine Waise ist die erste Zeile eines Absatzes, wenn sie allein am Ende einer Seite oder einer Spalte steht. Für Seiten ist uns das meistens egal, damit haben wir nur in einem Stylesheet für den Druck etwas zu tun. Aber für Spalten ist das jetzt wichtig.




Witwen und Waisen

**[Hintergrundinfo]**

Im deutschen Sprachgebrauch hießen Witwen früher auch Hurenkinder und Waisen Schusterjungen. Ein Merksatz, der erläutert, was von den beiden was ist, lautet: „Schusterjungen wissen nicht, wo sie hingehen, Hurenkinder nicht, wo sie herkommen." Typografie ist eine schwarzhumorige Angelegenheit …

## CSS10

Egal, wie man diese Zeilen nennt, sie sehen nicht besonders schön aus. Aber auch dafür gibt es selbstverständlich CSS. Mit zwei Eigenschaften kannst du steuern, wie viele Zeilen mindestens in der alten Spalte und wie viele mindestens in der neuen Spalte stehen müssen. Natürlich geht das nur, wenn der Absatz überhaupt so viele Zeilen hat – Text ausdenken kann sich der Browser noch keinen.

**[Funktioniert in]**

**widows** und **orphans** funktionieren noch nicht in Firefox.

## Witwen und Waisen vermeiden

```
.spalten {
    widows: 3;*1
    orphans: 3;*2
    …
}
```

*1 Mindestens drei Zeilen sollen in der neuen Spalte stehen …

*2 … und mindestens drei in der alten verbleiben. Wenn der Absatz insgesamt weniger Zeilen hat, ist es Sache des Browsers, was er mit denen macht.

Und schon kannst du Text auf mehrere Spalten ausbreiten und hast auch einige Werkzeuge, damit es dann auch wirklich gut aussieht. Das war vielleicht nicht ganz so spannend, wie vorher einen Film zu drehen, aber ich finde es auch ziemlich cool. Und, ganz ehrlich, so chic CSS-Filme auch sein mögen, wirst du für mehrspaltigen Text wahrscheinlich mehr Anwendungen finden.

# INHALTSVERZEICHNIS

# Kapitel 3: Jetzt kommt Farbe ins Spiel

## Einführung in CSS



# Kapitel 4: Kaskaden für Bossingen

## CSS-Selektoren und Typografie



# Kapitel 5: Ordnung in die Plattensammlung

## Listen und Tabellen

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

## Formulare



# Kapitel 7: Von Rändern und Schuhkartons

## Seitenlayout in HTML und CSS

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

## Eine Website von Anfang an



# Kapitel 9: Schöner wohnen mit CSS3

## CSS3



# Kapitel 10: Jetzt muss es sich aber endlich bewegen

## JavaScript

# Kapitel 11: Programmieren mit Bausteinen

## Funktionen



# Kapitel 12: Augen auf, du hast User!

## Eventhandler



# Kapitel 13: Gerade stand das da noch nicht

## DOM-Manipulation

# Kapitel 14: Schrödingers Welt der Programmierung

## Objekte und JSON



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

## Cookies, WebStorage und File-API

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

## Multimedia



# Kapitel 17: Schrödinger will's wissen

## Ajax

# Kapitel 18: Verwandlungskunst

## Responsive Webdesign und Mobile Devices

**Seite 759**



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

## Was geht noch?

**Seite 821**



# Anhang: Reguläre Ausdrücke und Zeichencodes

# Schrödinger lernt HTML5, CSS und JavaScript

Design & Elektronik

„HÄTTE ES DOCH SOLCHE BÜCHER VOR 20 JAHREN SCHON GEGEBEN!“

Amazon-Leserstimme:

„Das Layout der Seiten ist genial.“

Amazon-Leserstimme:

„Jetzt bin ich platt. Ich kann nicht aufhören zu lesen.“

Christian Mantey, IT-Dozent

„Überraschend gut! Sehr zu empfehlen!“

c't zur Schrödinger-Reihe

„Ein neuer Weg bei der Vermittlung von Entwickler-Fachwissen.“

## DAS ALLES UND NOCH VIEL MEHR:

- Werkzeuge und Webserver installieren
- Ausführlicher Einstieg in HTML, CSS und JavaScript
- Benutzerfreundliche GUIs erstellen
- Mit dem Server kommunizieren
- Mobile Endgeräte und Responsive Web Design
- Video- und Audiomaterial einbinden
- CSS-Animationen
- Touchevents für Tablets und Handys

Schrödinger ist unser Mann fürs Programmieren. Er kann schon was, aber noch keine Webentwicklung. Zum Glück hat er einen Kumpel, der auf jede Frage eine Antwort weiß, wenn es nur genug Kaffee gibt. So holt er sich bergeweise Tipps und lässt nicht locker, bis er alles kapiert hat.

## Eine runde Sache!

Lerne **HTML**, **CSS** und **JavaScript**, ohne das Buch zu wechseln. Der Durchmarsch von der ersten HTML-Seite bis zur Standortverfolgung mit dem Smartphone. Alles auf dem neuesten Stand und wenn Du willst, mit Deinem **eigenen Webserver**.

## SCHRÖDINGER GARANTIERT:

- Vollen Durchblick durch **moderne Webstandards**
- Unmengen an **Beispielen** und **Übungen**
- Für Einsteiger und Umsteiger **perfekt**
- Alle **Beispielprojekte** zum **Download**

## Ein echtes Fachbuch, nur eben ganz anders!

Unser Autor: Kai Günster ist Web- und Softwareentwickler aus Leidenschaft. Er hält HTML5 für die beste Erfindung seit der Laugenbrezel und den Webbrowser für das universelle GUI der Zukunft. Um euch alles beizubringen, hat er sogar seine eigene Katze verpixelt.

Rheinwerk Computing

Gedruckt in Deutschland
Papier aus nachhaltiger Waldwirtschaft
Mineralölfreie Druckfarben

€ 49,90 [D] € 51,30 [A]

Für Windows, Mac und Linux
Programmierung
ISBN 978-3-8362-9596-3