

CS 50 Web Design

APCSP Module 2: Internet



Unit 2: CSS (Chapter 15-19)

LECTURE 6B: ANIMATION AND WEB PAGE INTEGRATION

DR. ERIC CHOU

IEEE SENIOR MEMBER



Objectives

- Creating smooth transitions
- Moving, rotating, and scaling elements
- Combining transitions and transforms
- A few words about 3-D transforms
- Keyframe animations



Objectives

- Styling forms
- Style properties for tables
- Using a CSS reset or normalizer
- Image replacement techniques
- CSS sprites
- CSS feature detection

Transition

SECTION 1



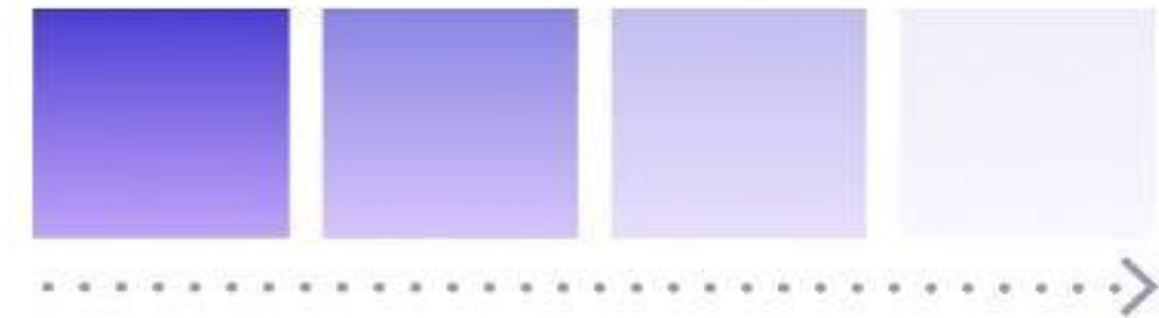
Transition Basics

Transitions are a lot of fun, so let's give them a whirl. When applying a transition, you have a few decisions to make, each of which is set with a CSS property:

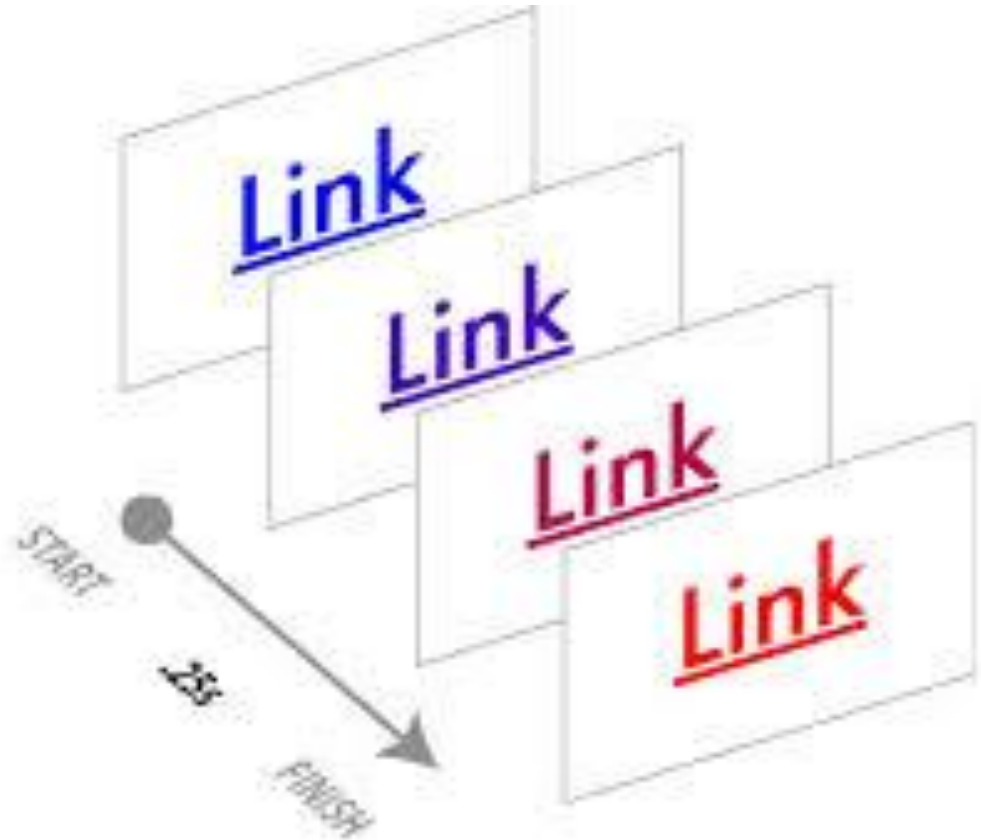
- Which CSS property to change (**transition-property**) (*Required*)
- How long it should take (**transition-duration**) (*Required*)
- The manner in which the transition accelerates (**transition-timingfunction**)
- Whether there should be a pause before it starts (**transition-delay**)

CSS Transitions

- Transition is used to change the status of some elements from one state to another.
- The CSS transition will smooth out the state changes from state to state over time by filling frames in between. Animators call that **tweening**.
- CSS transitions were originally developed by the **webkit** team for the safari browser, but they are now a Working Draft at the W3C.



CSS Transitions



CSS Transitions

Transition Basics Shorthand

transition: property duration
timing-function delay;

Which CSS property to change:

transition-property: property-name | all none

How long it should take:

transition-duration: time

The manner in which the transition accelerates:

transition-timing-function: ease | linear | ease-in |
ease-out | ease-in-out | step-start | step-end | step |
cubic-Bezier(#, #, #, #)

Whether there should be a pause before it starts:

transition-delay: time


The markup:

```
<a href="" class="smooth">awesomesauce</a>
```

The styles:

```
a.smooth { /* base condition */
  display: block;
  text-decoration: none;
  text-align: center;
  padding: 1em 2em ;
  width: 10em;
  border-radius: 1.5em;
  color: #fff;
  background-color: mediumblue;
  transition-property: background-color;
  transition-duration: 0.3s;
}

a.smooth:hover, a.smooth:focus {
  background-color: red;
}
```



actions for transition

Transition Property and Duration Example

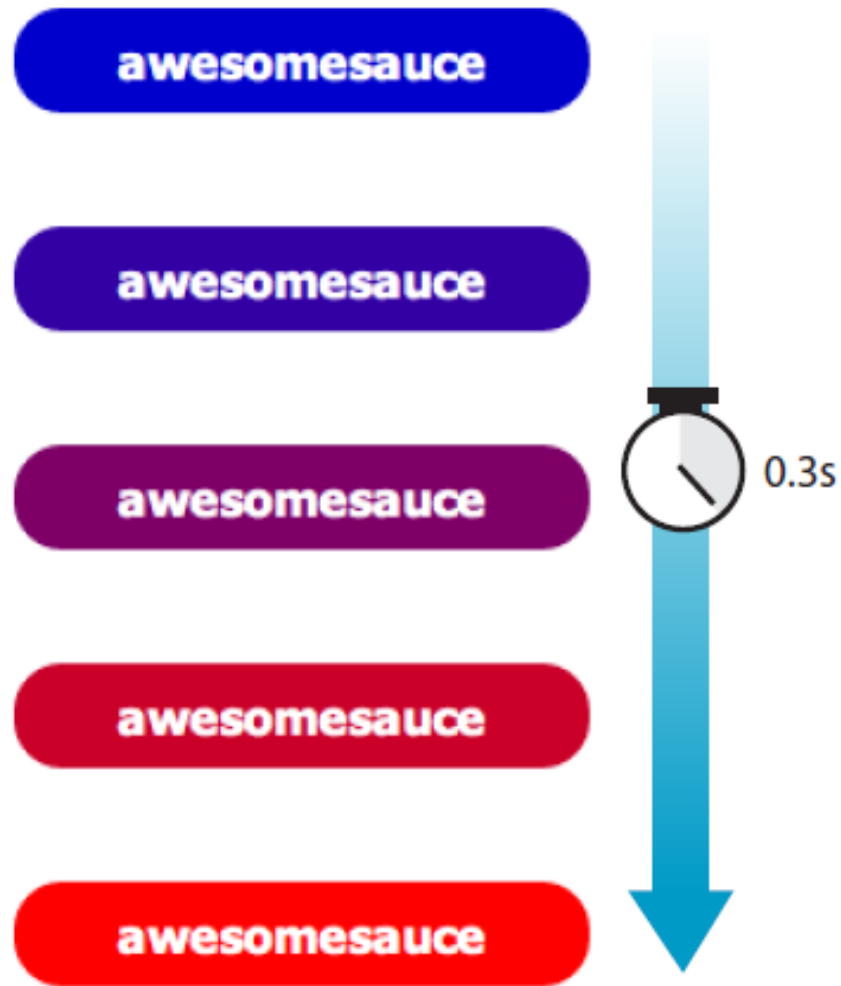


FIGURE 18-1. The background color of this link gradually fades from blue to red over .3 seconds when ~~awesomesauce~~ a transition is applied.

Transition Property and Duration Example

transition-duration

Values:	<i>time</i>
Default:	0s
Applies to:	all elements, :before and :after pseudo-elements
Inherits:	no

transition-duration sets the amount of time it takes for the animation to complete in seconds (**s**) or milliseconds (**ms**). I've chosen .3 seconds, which is just enough to notice something happened but not so long that the transition feels sluggish or slows the user down. There is no correct duration, of course, but I've found that .2s seems to be a popular transition time for UI elements. Experiment to find the duration that makes sense for your application.

How long should
it take?

Timing

SECTION 1

Timing Functions

transition-timing-function

Values:	ease linear ease-in ease-out ease-in-out step-start step-end steps cubic-bezier(##,##,##,##)
Default:	ease
Applies to:	all elements, :before and :after pseudo-elements
Inherits:	no

The property and the duration are required and form the foundation of a transition, but you can refine it further. There are a number of ways a transition can roll out over time. For example, it could start out fast and then slow down, start out slow and speed up, or stay the same speed all the way through, just to name a few possibilities. I think of it as the transition “style,” but in the spec, it is known as the [timing function](#) or [easing function](#).

Timing Functions

www.joelambert.co.uk/morf/

Source & Docs @ GitHub

Download morf.js

Download morf.min.js

Morf comes bundled with the fantastic Shifty.js for tweening regular CSS properties.

If you already use Shifty
heres a version of Morf
without Shifty pre-bundled.
Morf requires at least Shifty
0.1.3.

morf.noshifty.js

morf.noshifty.min.js

Timing Functions

www.joelambert.co.uk/morf/



Native

These are the natively supported easing functions, built into WebKit.

linear

ease

ease-in

ease-out

ease-in-out

Custom

These are custom easing functions (*thanks to Robert Penner & Thomas Fuchs*) that can produce much more interesting transitions.

sinusoidal

spring

easeTo

easeFrom

easeFromTo

bouncePast

bounce

swingTo

swingFrom

swingFromTo

elastic

easeInOutBack

easeOutBack

easeInBack

easeOutBounce

easeInOutCirc

easeOutCirc

easeInCirc

easeInOutExpo

easeOutExpo

easeInExpo

easeInOutSine

easeOutSine

easeInSine

easeInOutQuint

easeOutQuint

easeInQuint

easeInOutQuart

easeOutQuart

easeInQuart

easeInOutCubic

easeOutCubic

easeInCubic

easeInOutQuad

easeOutQuad

easeInQuad

linear

- **ease:** Starts slowly, accelerates quickly, and then slow down at the end. This is the default value and works just fine for most short transitions.
- **linear:** Speed stays consistent from the transition's beginning to end.
- **ease-in:** Start slowly, then speed up.
- **ease-out:** Start out fast, then slows down.
- **ease-in-out:** Starts slowly, speeds up, and then slow down again at the very end. It is similar to ease, but with less pronounced acceleration in the middle.

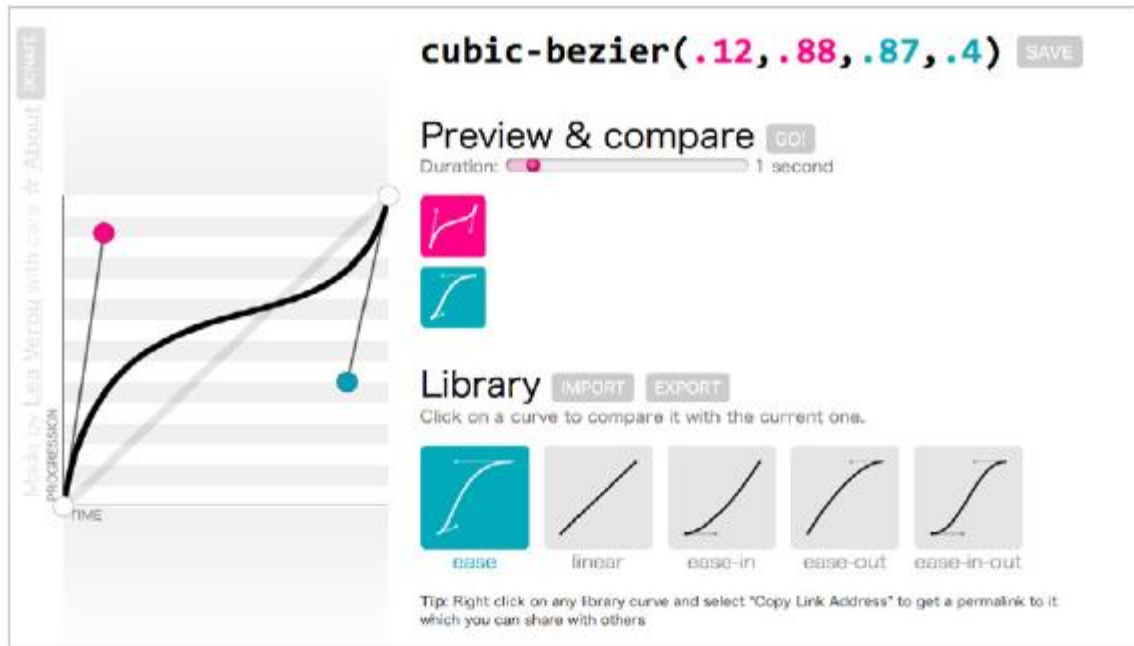
Timing Function Description

- ease-in-out:** Starts slowly, speeds up, and then slows down again at the very end. It is similar to ease, but with less pronounced acceleration in the middle.
- cubic-bezier(#, #, #, #):** This is a function for defining a Bezier curve that describes the transition acceleration. It's super math-y and I can't explain it all here.
- steps(#, start | end):** Divides the transitions into a number of steps as defined by stepping function. The first value is the number of steps, and the start and end keywords define whether the change in state happens at the beginning (start) or end of each step.

Timing Function Description

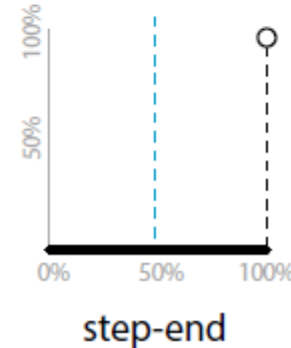
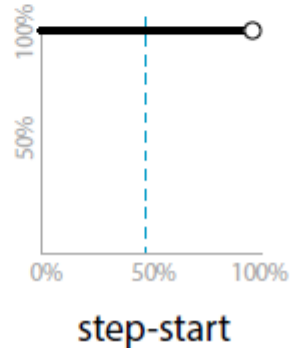
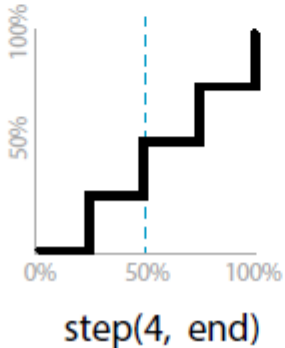
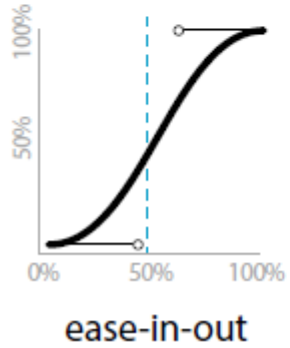
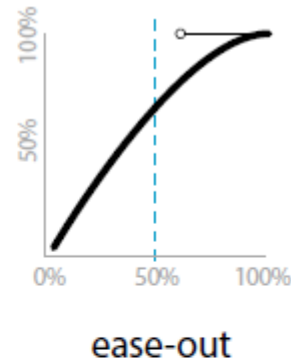
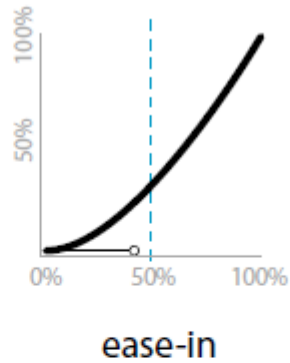
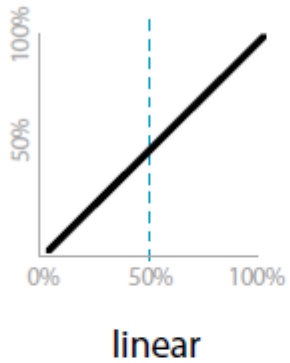
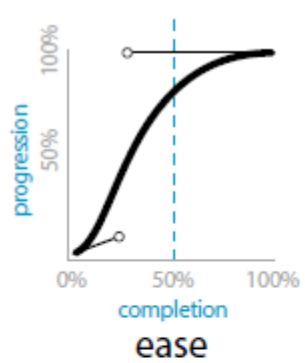
- step-start:** The changes states in one step, at the beginning of the duration time (the same as `steps(1, start)`). The result is a sudden state change, the same as if no transition had applied at all.
- step-end:** This changes states in one step, at the end the duration time (the same as `steps(1, end)`).

Timing Function Description

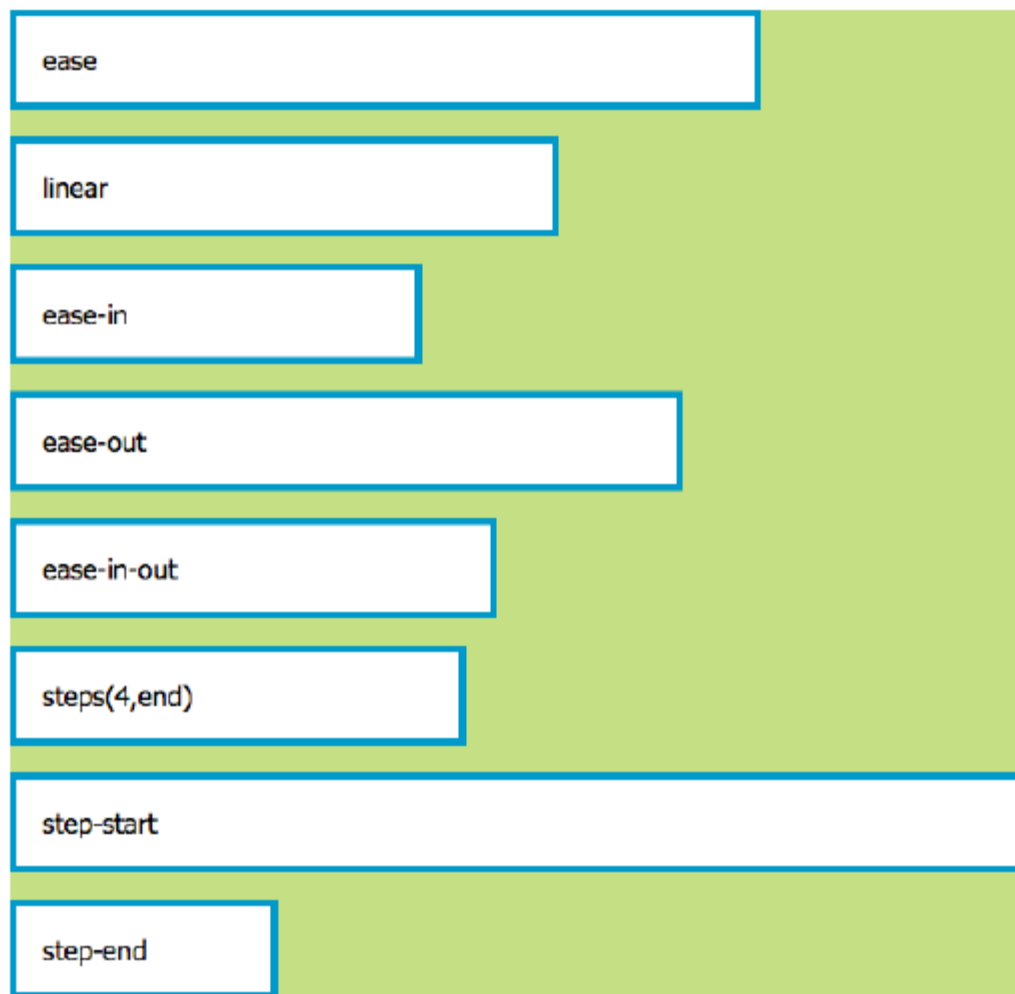


Timing Function Description

FIGURE 18-2. Examples of Bezier curves from Cubic-Bezier.com. On the left is my custom curve that starts fast, slows down, and ends fast.



Timing Function Description



Timing Function Description

The width of the white boxes is set to transition from 0 to 100% width over 4 seconds. This screenshot shows the progress after 2 seconds (50%) for each timing function.

FIGURE 18-3. In this **transition-timing-function** demo, the elements reach full width at the same time but vary in the manner in which they get there. If you'd like to see it in action, the *ch18_figures.html* file is available with the materials for this chapter.

Setting a Delay

transition-delay

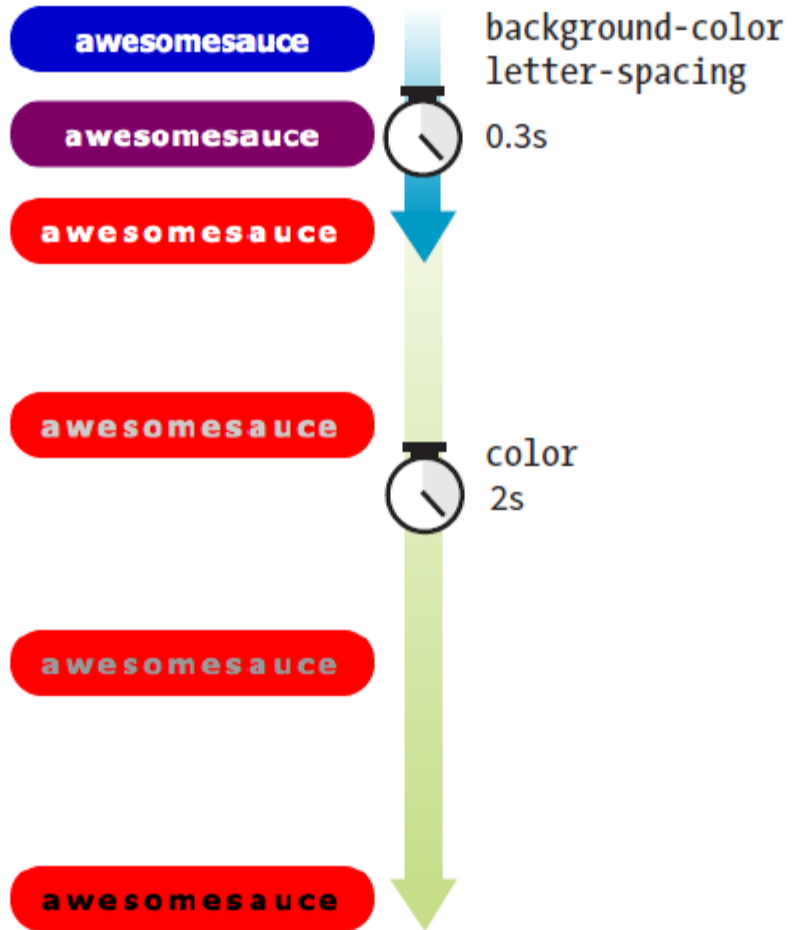
Values:	<i>time</i>
Default:	0s
Applies to:	all elements, :before and :after pseudo-elements
Inherits:	no

The **transition-delay** property, as you might guess, delays the start of the animation by a specified amount of time. In the following example, the background color transition starts .2 seconds after the pointer moves over the link.

```
.smooth {  
  ...  
  transition-property: background-color;  
  transition-duration: 0.3s;  
  transition-timing-function: ease-in-out;  
  transition-delay: 0.2s;  
}
```

```
.smooth {  
    ...  
    transition-property: background-color,  
    color, letter-spacing;  
    transition-duration: 0.3s, 2s, 0.3s;  
    transition-timing-function: ease-out,  
    ease-in, ease-out;  
}  
  
.smooth:hover, .smooth:focus {  
    background-color: red;  
    letter-spacing: 3px;  
    color: black;  
}
```

Applying Multiple Transitions



Applying Multiple Transitions

FIGURE 18-4. The **color**, **background-color**, and **letter-spacing** change at different paces.

```
.smooth {  
  ...  
  transition: background-color 0.3s ease-out,  
             color 2s ease-in,  
             letter-spacing 0.3s ease-out;  
}
```

Applying Multiple Transitions

Backgrounds	Border/Outlines	Color/Opacity	Font/text	Element Measurements	Position
background-color	border-bottom-color	Color	font-size	height	top
background-position	border-bottom-width	Opacity	font-weight	width	right
	border-left-color	visibility	letter-spacing	max-height	bottom
	border-left-width		line-height	max-width	left
	border-right-color		text-indent	min-height	z-index
	border-right-width		text-shadow	min-width	clip
	border-top-color		word-spacing	margin-bottom	
	border-top-width		vertical-align	margin-left	
	border-top=color			margin-right	
	border-top-width			margin-top	
	border-spacing			padding-bottom	
	outline-color			padding-left	
Animatable CSS Properties	outline-offset			padding-right	
	outline-width			padding-top	
				crop	

Transform

SECTION 1

CSS Transforms

transform

Values:	rotate() rotateX() rotateY() rotateZ() rotate3d() translate() translateX() translateY() scale() scaleX() scaleY() skew() skewX() skewY() none
Default:	none
Applies to:	transformable elements (see sidebar)
Inherits:	no

The CSS3 Transforms Module (www.w3.org/TR/css-transforms-1) gives authors a way to rotate, relocate, resize, and skew HTML elements in both two- and three-dimensional space. It is worth noting up front that transforms change how an element displays, but it is not motion- or time-based.

However, you can animate from one transform state to another using transitions or keyframe animations, so they are useful to learn about in the context of animation.

Transformable Elements

You can apply the **transform** property to most element types:

- HTML elements with replaced content, such as **img**, **canvas**, form inputs, and embedded media
- Elements with their display set to **block**, **inline-block**, **inlinetable** (or any of the **table-*** display types), **grid**, and **flex**

It may be easier to note the element types you *cannot* transform, which include:

- Non-replaced inline elements, like **em** or **span**
- Table columns and column groups (but who'd want to?)

FIGURE 18-6 shows a representation of four two-dimensional transform functions: **rotate()**, **translate()**, **scale()**, and **skew()** (see Note). The dashed outline shows the element's original position.

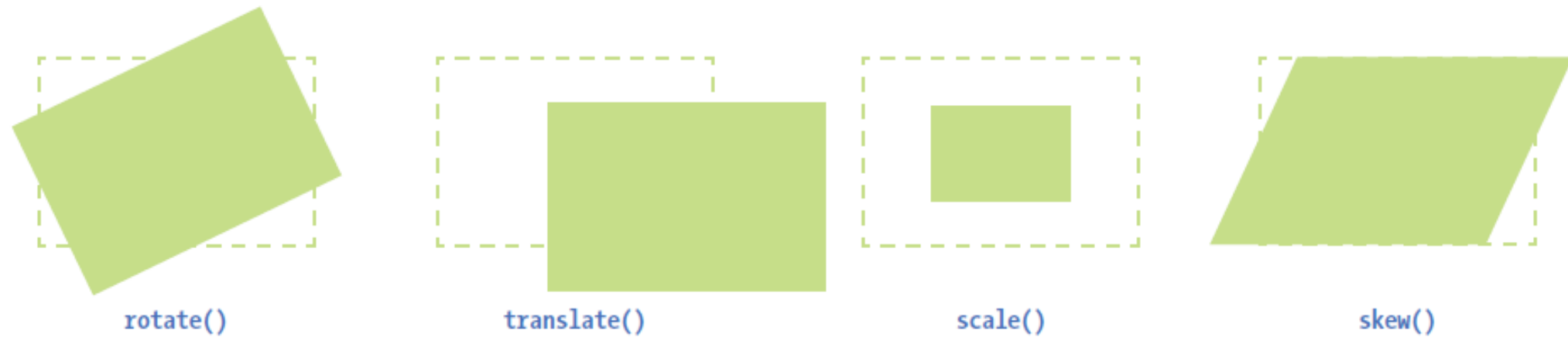
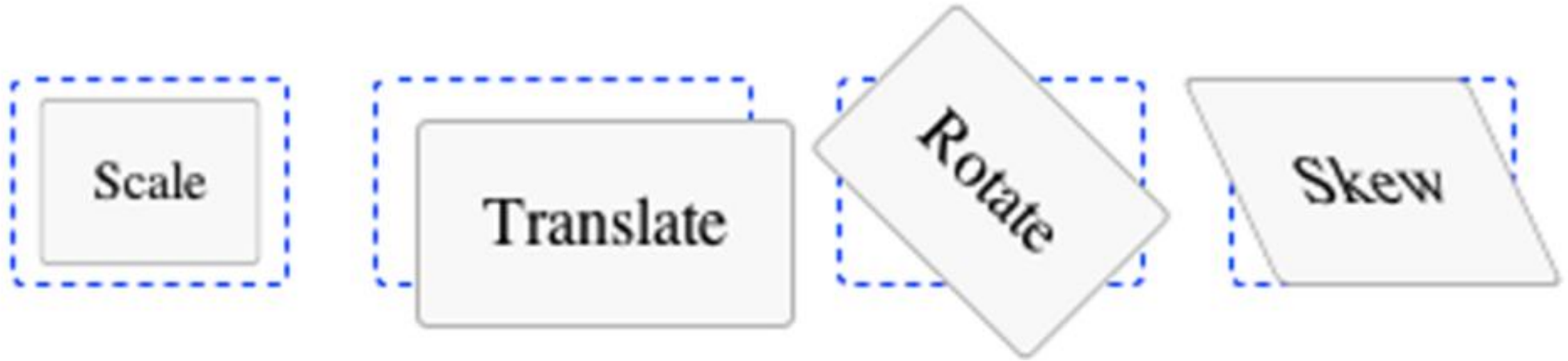


FIGURE 18-6. Four types of transforms: **rotate()**, **translate()**, **scale()**, and **skew()**.



CSS Transforms



skew



scale



translate



rotate



skew X



scale X



translate X



rotate X



skew Y



scale Y



translate Y



flip Y

CSS Transforms



Transform Origin (Rotate)

transform-origin: percentage | length | left
| center | right | top | bottom

transform: rotate(-10deg); /* in degree */

Transformable Elements:

- img, canvas, form input, and embedded media

```
img {  
  width: 400px;  
  height: 300px;  
  transform: rotate(-10deg);  
}
```

Transform Origin (Rotate)



```
transform: rotate(-10deg);
```

FIGURE 18-7. Rotating an `img` element by using `transform: rotate()`.

transform-origin

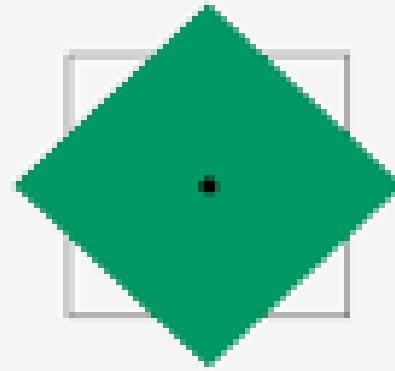
Values: *percentage* | *length* | left | center | right |
top | bottom
Default: 50% 50%
Applies to: transformable elements
Inherits: no

The value for **transform-origin** is either two keywords, length measurements, or percentage values. The first value is the horizontal offset, and the second is the vertical offset. If only one value is provided, it will be used for both. The syntax is the same as you learned for **background-position** back in Chapter 13, Colors and Backgrounds. If we wanted to rotate our image around a point at the center of its top edge, we could write it in any of the following ways:

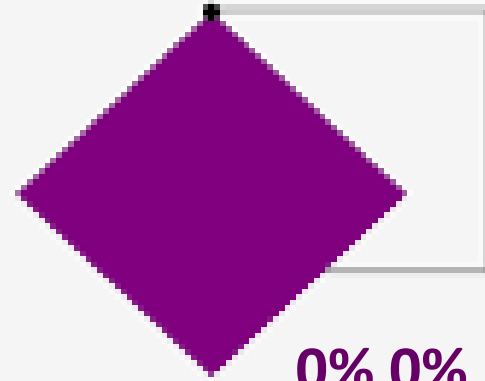
```
transform-origin: center top;  
transform-origin: 50%, 0%;  
transform-origin: 200px, 0;
```

Transform Origin (Rotate)

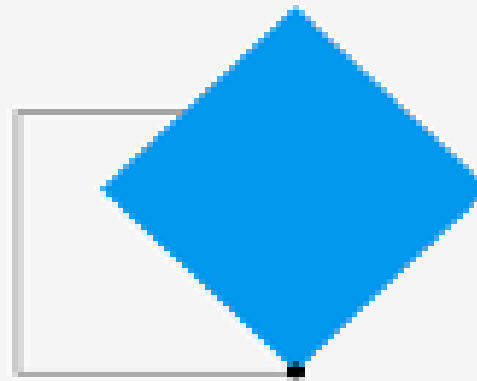
Transform Origin (Rotate)



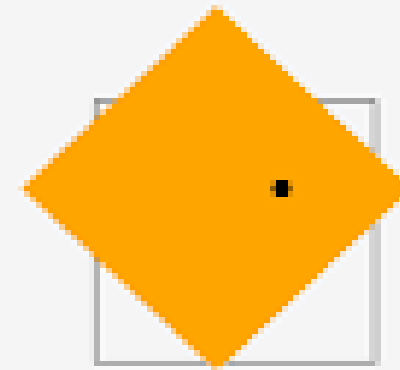
50% 50%



0% 0%



100% 100%



60% 30%

Transforming the position (translate)

The **translateX()** function allows you to move an element on a horizontal axis; **translateY()** is for moving along the vertical axis; and **translate()** combines both x and y values.

```
transform: translateX(50px);  
transform: translateY(25px);  
transform: translate(50px, 25px);  
/* (translateX, translateY) */
```



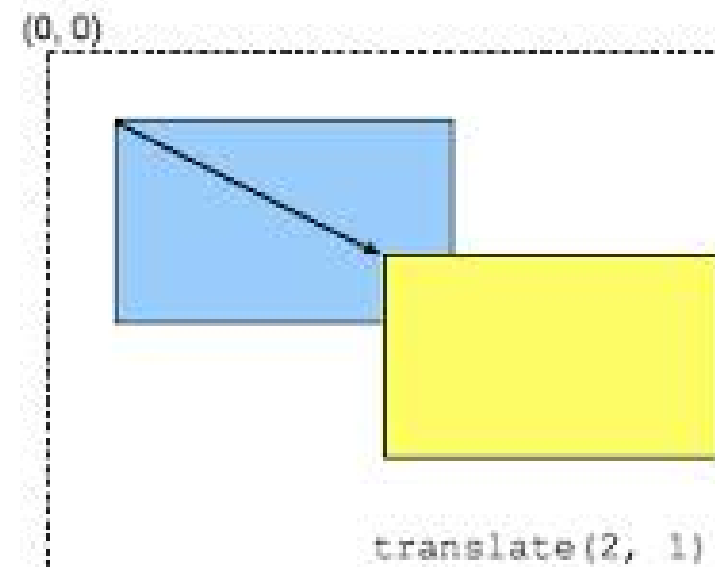
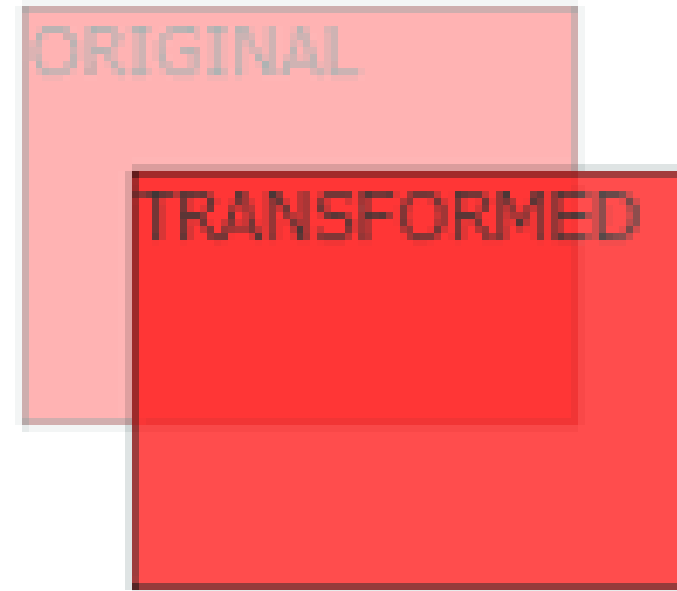
transform: translate(90px, 60px);



transform: translate(-5%, -25%);

FIGURE 18-9. Moving an element around with the **translate()** function.

Transforming the position (translate)



Transforming the Size (scale)

Make an element appear larger or smaller by using one of three scale functions: **scaleX()** (horizontal), **scaleY()** (vertical), and the shorthand **scale()**. The value is a unitless number that specifies a size ratio. This example makes an image 150% its original width:

```
a img {  
  transform: scaleX(1.5);  
}
```

The **scale()** shorthand lists a value for **scaleX** and a value for **scaleY**. This example makes an element twice as wide but half as tall as the original:

```
a img {  
  transform: scale(2, .5);  
}
```




`transform: scale(1.25);`



`transform: scale(.75);`



`transform: scale(1.5, .5);`

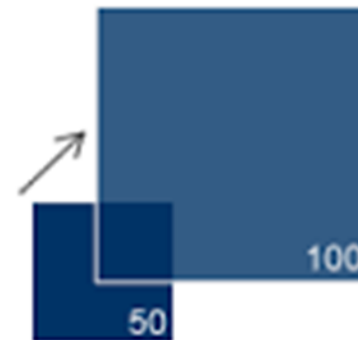
Transforming the Size (scale)

FIGURE 18-10. Changing the size of an element with the **scale()** function.

Transform the
size (scale)



transform:scale(2)



Making It Slanty (skew)

The quirky collection of skew properties—**skewX()**, **skewY()**, and the shorthand **skew()**—changes the angle of either the horizontal or vertical axis (or both axes) by a specified number of degrees. As for **translate()**, if you provide only one value, it is used for **skewX()**, and **skewY()** will be set to zero. The best way to get an idea of how skewing works is to take a look at some examples ([FIGURE 18-11](#)):

```
a img {  
    transform: skewX(15deg);  
}  
a img {  
    transform: skewY(30deg);  
}  
a img {  
    transform: skew(15deg, 30deg);  
}
```



`transform: skewX(15deg);`



`transform: skewY(30deg);`



`transform: skew(15deg, 30deg);`

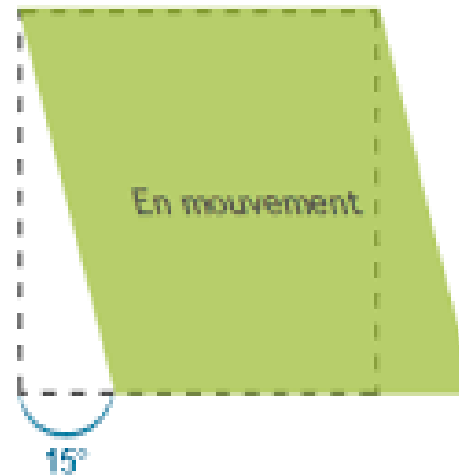
Making It Slanty (skew)

FIGURE 18-11. Slanting an element by using the **skew()** function.

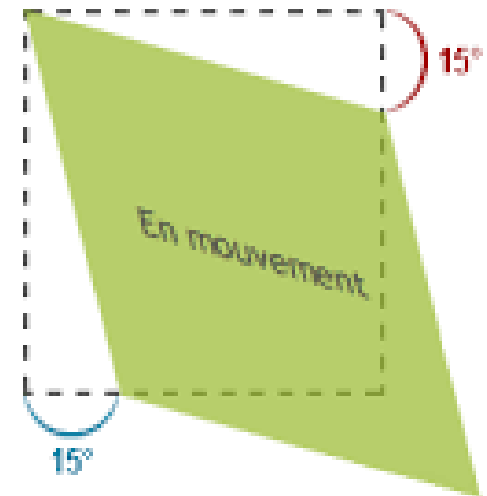
Making It Slanty (skew)



`skew(15deg, 0deg)`



`skew(15deg, 15deg)`



Applying Multiple Transforms

It is possible to apply more than one transform to a single element by listing out the functions and their values, separated by spaces, like this:

```
transform: function(value) function(value) ;
```

In the example in [FIGURE 18-12](#), I've made the forest image get larger, tilt a little, and move down and to the right when the mouse is over it or when it is in focus:

```
img:hover, img:focus {  
    transform: scale(1.5) rotate(-5deg)  
    translate(50px, 30px) ;  
}
```

Normal state



`:hover, :focus`
`rotate()`, `translate()`, and `scale()` applied



Applying Multiple Transforms

FIGURE 18-12. Applying `scale()`, `rotate()`, and `translate()` to a single element.

2D Transforms

<https://angrytools.com/css-generator/transform/>

ANGRY TOOLS

Search Tool

Home

Heart

Search owner of a phone number

Powered By BeenVerified

Enter Area Code: () -

CSS Generator - Matrix Transform

Border

Border Radius

Box Shadow

Background

Text-Shadow

Gradient

Matrix Transform

Transition

CSS Animation

Image Filter

1

save | reset

Transform

matrix (a,b,c,d,x,y)

100

010

001

Individual Transform

☐

scale

11

rotate

0

skew

00

translate XY

00

Code

```
transform:matrix(1,0,0,1,0,0);
-ms-transform:matrix(1,0,0,1,0,0);
-webkit-transform:matrix(1,0,0,1,0,0);
```


3D Transform

SECTION 1

3-D Transforms

- In addition to the two-dimensional transform functions we've just seen, the CSS Transforms spec also describes a system for creating a sense of three dimensional space and perspective. Combined with transitions, you can use 3-D transforms to create rich interactive interfaces, such as image carousels, flippable cards, or spinning cubes! [FIGURE 18-14](#) shows a few examples of interfaces created with 3-D transforms.
- It's worth noting that this method does not create 3-D objects with a sense of volume; it merely tilts the otherwise flat element box around on three axes (animation expert Val Head calls them "postcards in space"). The rotating cube example in the figure merely stitches together six element boxes at different angles. That said, 3-D transforms still add some interesting depth to an otherwise flat web page.

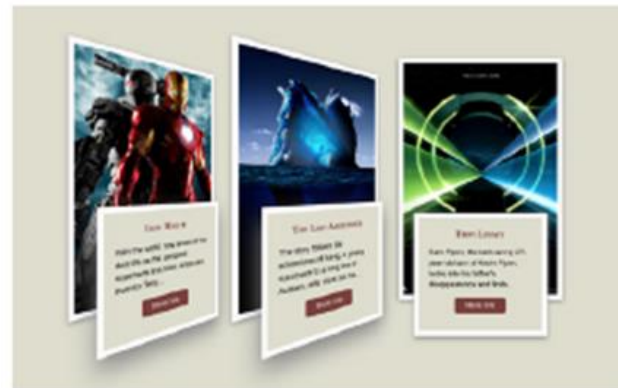
3-D Transforms



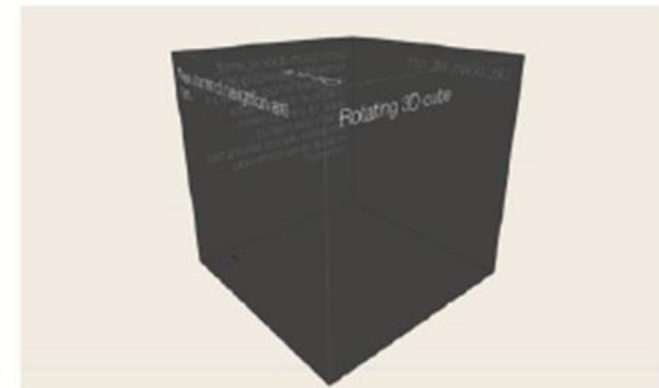
Animated book covers by Marco Barria
tympanus.net/Development/AnimatedBooks/



Webflow transform tools example
3d-transforms.webflow.com



Movie poster animation by Marco Kuiper
demo.marcofolio.net/3d_animation_css3/



3D CSS Rotating Cube by Paul Hayes
paulrhayes.com/experiments/cube-3d/

FIGURE 18-14. Some examples of 3-D transforms. The book covers, movie posters, and 3-D cube also have cool animation effects, so it's worth going to the links and checking them out. Webflow is a visual web design tool that includes the ability to create 3-D transformed elements.

CSS3 3D Transforms - WD

Method of transforming an element in the third dimension using the **transform** property. Includes support for the **perspective** property to set the perspective in z-space and the **backface-visibility** property to toggle display of the reverse side of a 3D-transformed element.

Global 81.84% + 6.92% = 88.77%
unprefixed: 65.56% + 6.92% = 72.48%

Current aligned Usage relative Show all									
IE	Edge	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Chrome for Android
			47					4.3	
8			48					4.4	
9		44	49	2 9		8.4		4.4.4	
1 11	13	45	50	2 9.1	36	9.2	8	47	49
	14	46	51	2 TP	37	9.3			
		47	52		38				
		48	53						

3D Transforms Browser Support
-ms- -moz- -webkit-



3D Transforms

FIGURE 18-15. Our aquarium images arranged in space...*space...space...*

3-D Transforms

THE MARKUP

```
<ul>
  <li><a href=""></a></li>
  <li><a href=""></a></li>
  <li><a href=""></a></li>
  <li><a href=""></a></li>
</ul>
```

THE STYLE

```
ul {
  width: 1000px;
  height: 100px;
  list-style-type: none;
  padding: 0;
  margin: 0;
  perspective: 600;
}
```

3-D Model

The **perspective-origin** property (not shown) describes the position of your eyes relative to the transformed items. The values are a horizontal position (**left**, **center**, **right**, or a length or percentage) and a vertical position (**top**, **bottom**, **center**, or a length or percentage value). The default ([FIGURE 18-15](#)) is centered vertically and horizontally (**perspective-origin: 50% 50%**). The final transform-related property is **backface-visibility**, which controls whether the reverse side of the element is visible when it spins around.

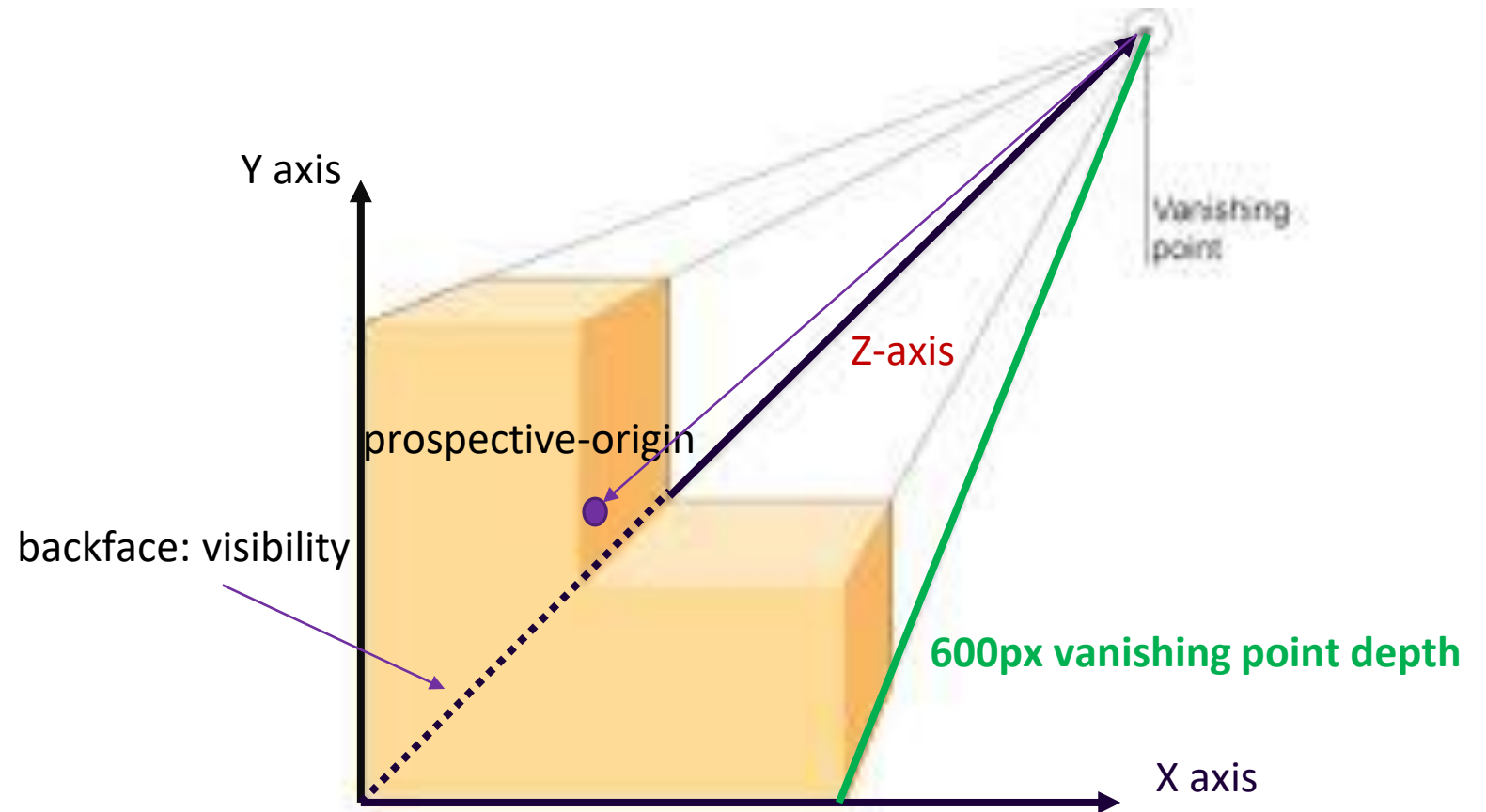
3-D Model

Vanishing Point Projection on X-Y Plane:

prospective-origin: 50% 50%

Vanishing Point Depth:

prospective: 600px;



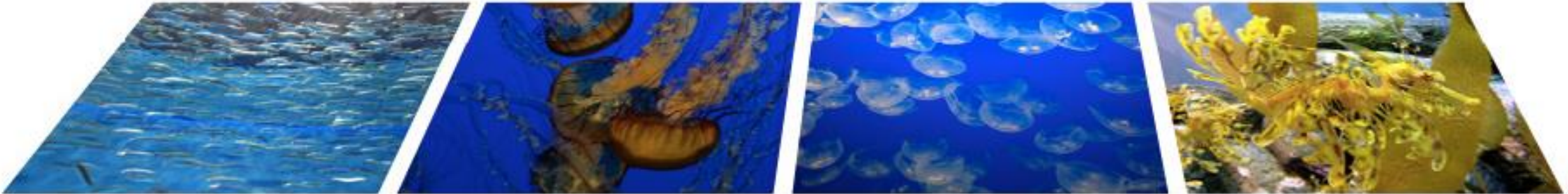
Set Up 3D Model

```
/* -webkit- browsers */
-webkit-perspective: 600px;
-webkit-perspective-origin: 50% 50%;
-webkit-backface-visibility:30%;
/* -moz- Firefox browser */
-moz-perspective: 600px;
-moz-perspective-origin: 50% 50%;
-moz-backface-visibility:30%;
/* -ms- IE browsers */
-ms-perspective: 600px;
-ms-perspective-origin: 50% 50%;
-ms-backface-visibility:30%;
/* general browsers (unknown) */
perspective: 600px;
perspective-origin: 50% 50%;
backface-visibility:30%;
```

3-D Model

With the 3-D space established, apply one of the 3-D transform functions to each child element—in this case, the `li` within the `ul`. The 3-D functions include `translate3d`, `translateZ`, `scale3d`, `scaleZ`, `rotate3d`, `rotateX`, `rotateY`, `rotateZ`, and `matrix3d`. You should recognize some terms in there. The `*Z` functions define the object's orientation relative to the z-axis (picture it running from your nose to this page, whereas the x- and y-axes lie flat on the page).

```
li {  
  float: left;  
  margin-right: 10px;  
  transform: rotateX(45deg);  
}
```



3D Transforms

FIGURE 18-16. The same list of images rotated on their horizontal axes with **rotateX()**.

<https://polypane.app/css-3d-transform-examples/>

The screenshot displays a web-based interface for visualizing 3D transformations using CSS3's `matrix3d()` function.

Top Bar:

- Left: "CSS3 Matrix3D"
- Right: "GitHub" link

Main Viewport:

A 3D coordinate system is shown with dashed axes. A unit cube is centered at the origin. The faces are colored: top (blue), bottom (red), left (green), right (yellow), front (cyan), and back (magenta). The number "1" is visible on the front face, indicating its initial position or scale.

Transform Controls:

A section titled "Toggle 3D view" (with a switch) contains a grid of sliders for various transformation parameters:

scale _x (1)	shear _{yx} (0)	shear _{zx} (0)	P _x (0)
shear _{xy} (0)	scale _y (1)	shear _{zy} (0)	P _y (0)
shear _{xz} (0)	shear _{yz} (0)	scale _z (1)	P _z (0)
translate _x (0)	translate _y (0)	translate _z (0)	P _w (0)

Additional Controls (not part of CSS3 Matrix3D):

Below the main controls are three sliders labeled:

- rotateX(0deg)
- rotateY(0deg)
- rotateZ(0deg)

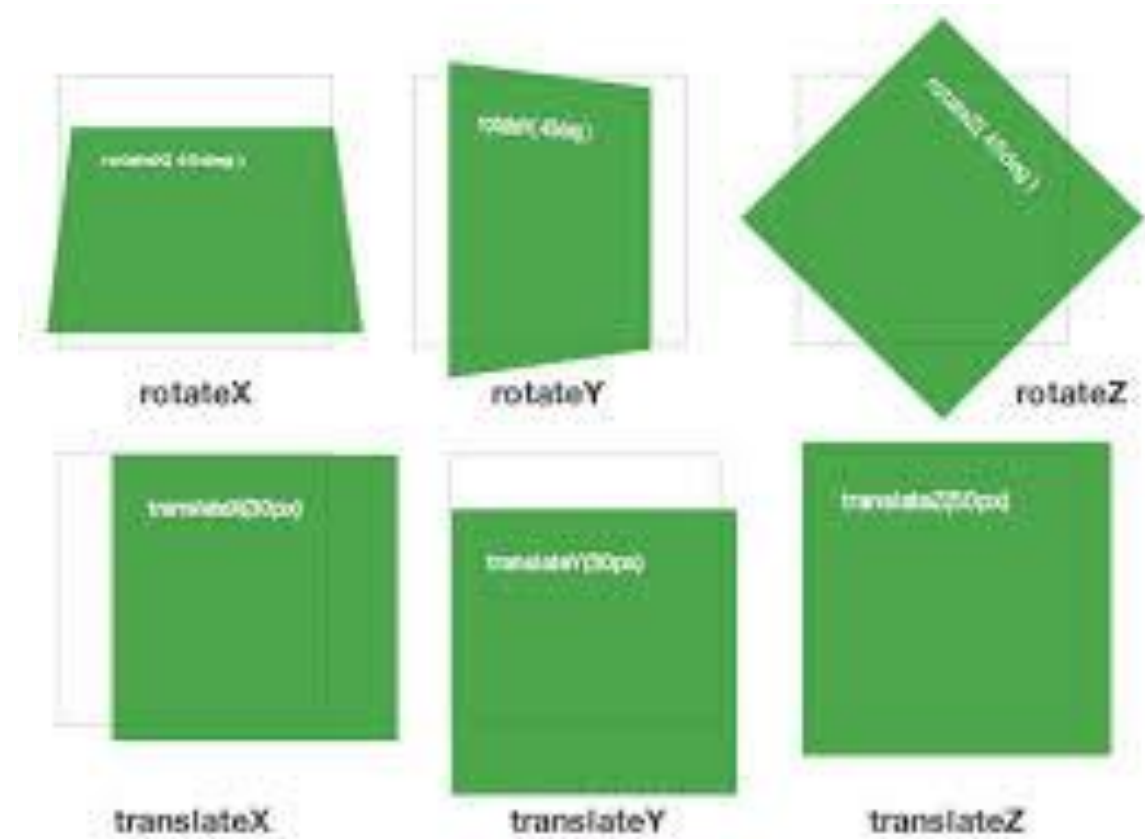
Notes:

- According to [CSS Transforms Module Level 1](#), `translate[XYZ]` are placed along column 4 of a matrix, but browsers implement by swapping column and row 4.
- [Interpolation of 3D Matrices](#)

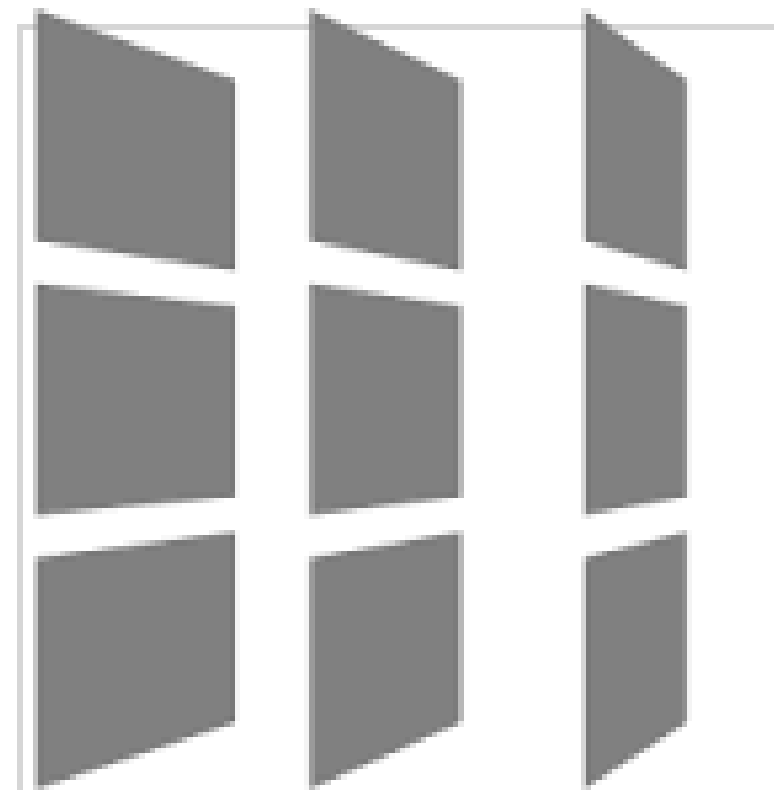
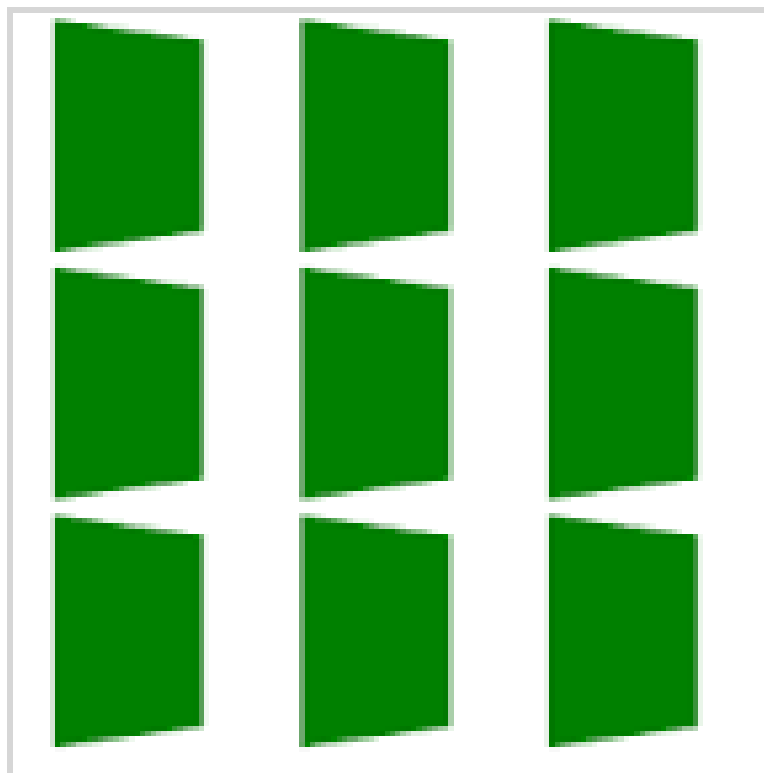
Code Area:

```
transform: matrix3d(  
  1, 0, 0, 0,  
  0, 1, 0, 0,  
  0, 0, 1, 0,  
  0, 0, 0, 1  
)
```

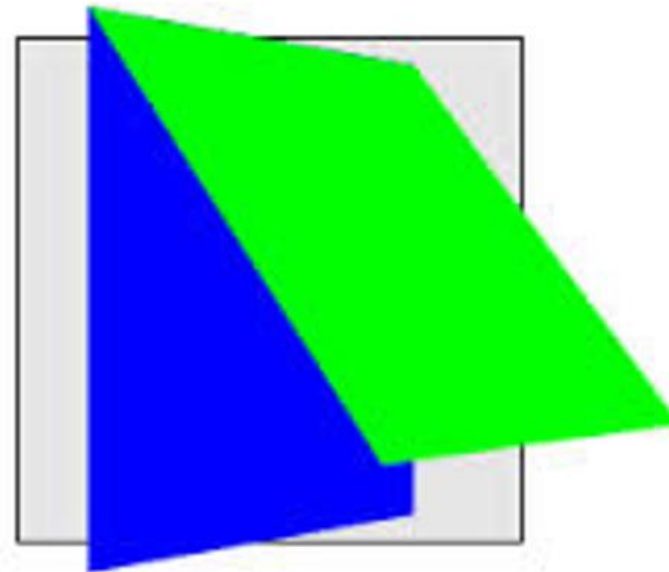
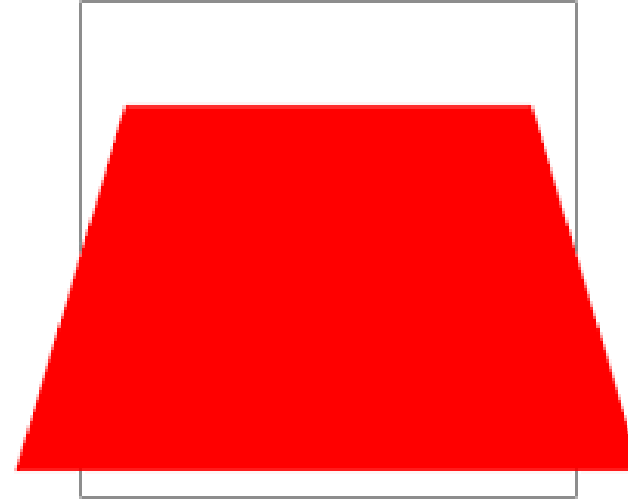
3D Rotation and Translate



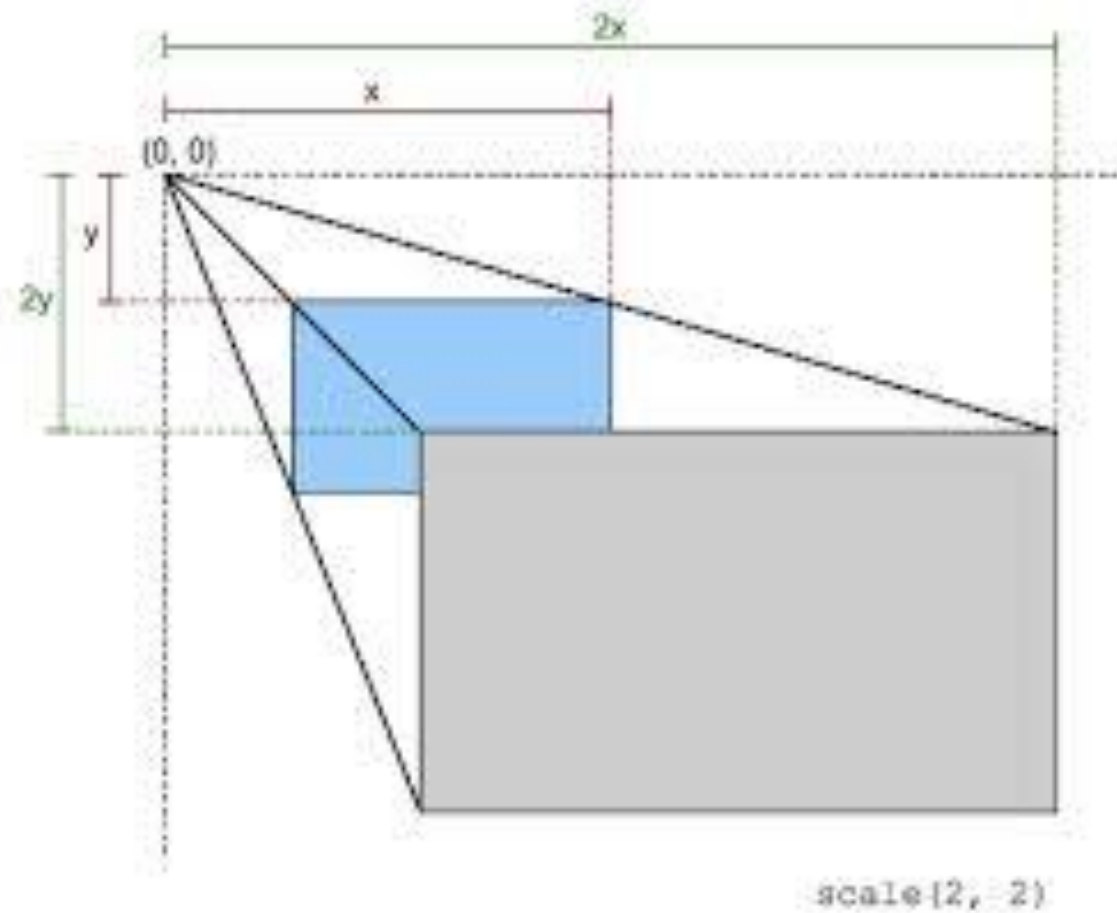
3D RotateY



3D RotateX and RotateZ



3D Scale



3D Transformed Image



Animation

SECTION 1

Keyframe Animation

- The CSS Animations Module allows authors to create real, honest-to-goodness keyframe animation. [FIGURE 18-17](#) shows just a few examples that you can see in action online. Unlike transitions that go from a beginning state to an end state, keyframe animation allows you to explicitly specify other states at points along the way, allowing for more granular control of the action.
- Those “points along the way” are established by [keyframes](#) that define the beginning or end of a segment of animation.

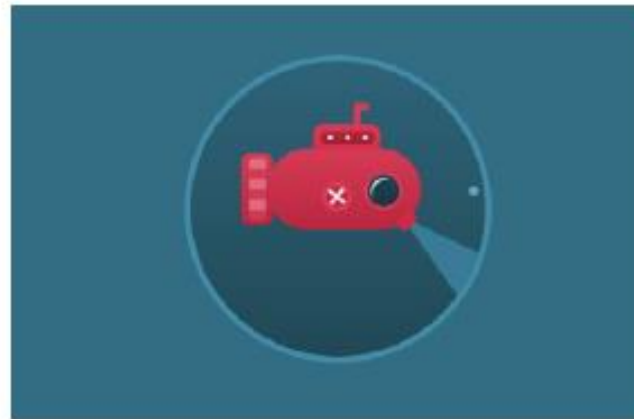
Keyframe Animation



MADMANIMATION
by Anthony Calzadilla and Andy Clarke
stuffandnonsense.co.uk/content/demo/madmanimation/—hint: click WATCH



How I Learned to Walk
by Andrew Wang-Hoyer
andrew.wang-hoyer.com/experiments/walking/



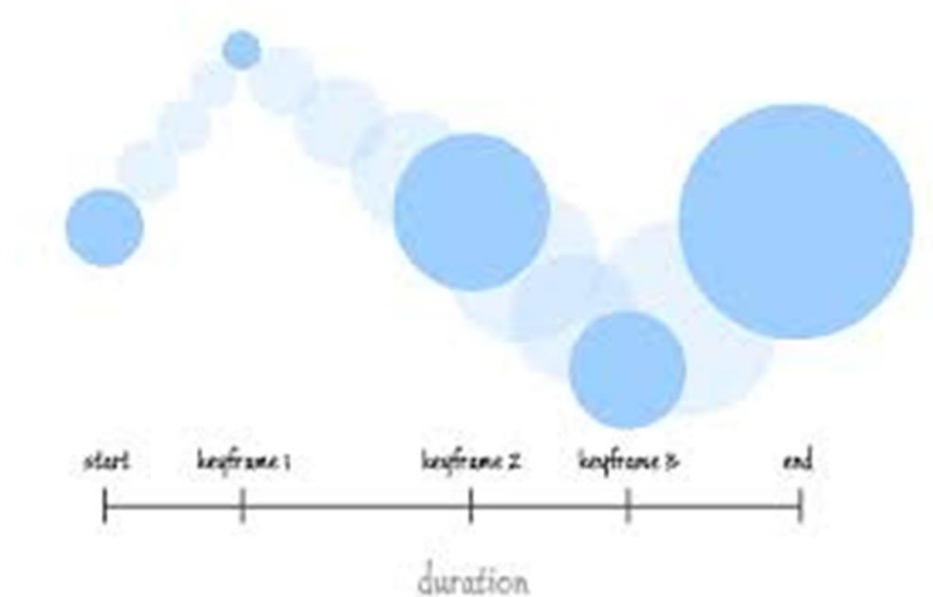
Adorable animated submarine
by Alberto Jerez
codepen.io/ajerez/pen/EaEEOW



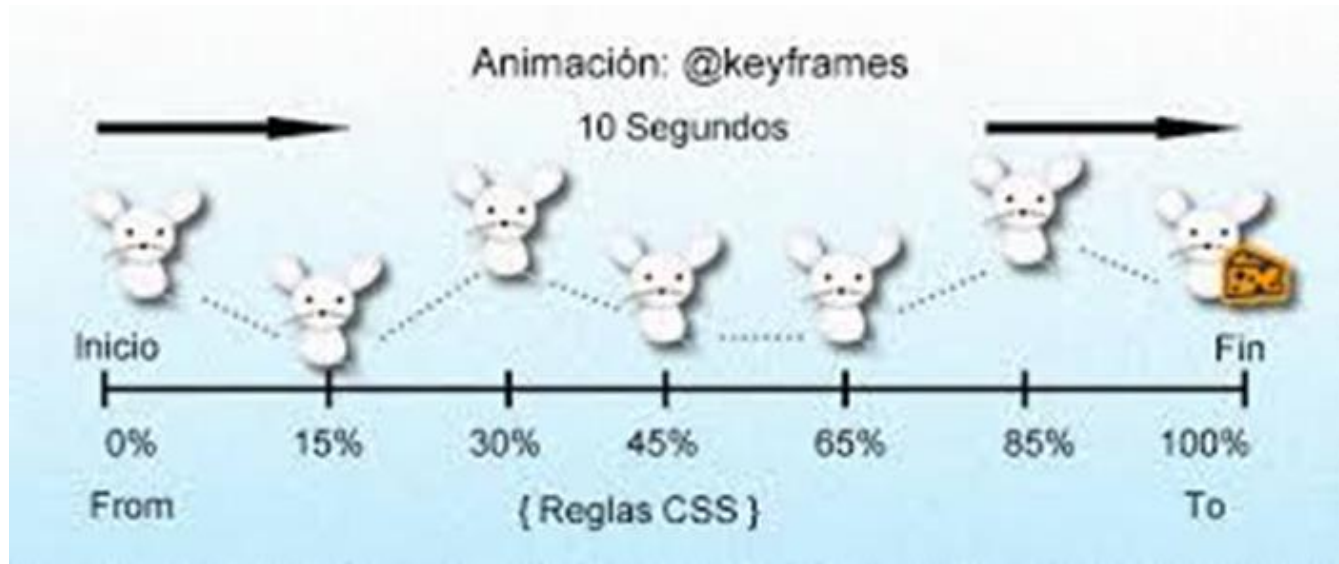
Animated Web Banner
by Caleb Jacob
tympanus.net/codrops/2012/01/10/animated-web-banners-with-css3/

FIGURE 18-17. Examples of animations using only CSS.

- **Key frames** are the frames that used to interpolate the other frames when used in video compression algorithm or animation.
- To be short, Key frames are the frame with real data. Other frames are created by these frame data.



What is
@keyframe



What is
@keyframe

1. [CSS3Gen CSS3 Animation Generator](#)
2. [CSS Animate](#)
3. [Coveloping CSS Animation Generator](#)
4. [Magic Animations](#)
5. [Animate.css](#)
6. [Bounce.js](#)
7. [AniJS](#)
8. [Single Element CSS Spinners](#)
9. [Odometer](#)
10. [Snabbt.js](#)

CSS Animation Tools

<https://www.hongkiat.com/blog/css3-animation-tools/>

Animation Tool:

If you want to use CSS Animations but lack the wherewithal to learn to code it all yourself, there are tools that give you a timeline interface for creating your animations and generate the HTML and CSS for you. Here are a few as of this writing:

- Tumult Hype, <http://www.tumultco.com/hype/> (Mac Only)
- Sencha Animator, <http://www.sencha.com/products/animator/>
- Adobe Edge, <http://labs.adobe.com/technologies/edge/>

CSS Animation Tools

<https://www.hongkiat.com/blog/css3-animation-tools/>

Establishing the Keyframes

The animation process has two parts:

1. Establish the keyframes with a **@keyframes** rule.
2. Add the animation properties to the elements that will be animated.

Here is a very simple set of keyframes that changes the background color of an element over time. It's not a very action-packed animation, but it should give you a basic understanding of what a **@keyframes** rule does.

```
@keyframes colors {  
  0% { background-color: red; }  
  20% { background-color: orange; }  
  40% { background-color: yellow; }  
  60% { background-color: green; }  
  80% { background-color: blue; }  
  100% { background-color: purple; }  
}
```

- The keyframes at-rule identifies the name of the animation, the stages of the animation represented by percentage (%) values, and the CSS properties that are affected for each stage. Here's what a **@keyframes** rule looks like abstracted down to its syntax:

```
@keyframes animation-name {  
    keyframe { property: value; }  
    /* additional keyframes */  
}
```

Keyframe

The sample **@keyframes** rule says: create an animation sequence called “colors.” At the beginning of the animation, the **background-color** of the element should be red; at 20% through the animation runtime, the background color should be orange; and so on, until it reaches the end of the animation. The browser fills in all the shades of color in between each keyframe (or *tweens* it, to use the lingo). This is represented the best I could in [FIGURE 18-18](#).

Each percentage value and the property/value declaration defines a keyframe in the animation sequence.

Keyframe



As an alternative to percentages, you can use the keyword **from** for the start of an animation sequence (equivalent to 0%) and the keyword **to** for denoting the end (100%). The following example makes an element slide in from right to left as the left margin reduces to 0:

```
@keyframe slide {  
  from { margin-left: 100% }  
  to { margin-left: 0%; }  
}
```

FIGURE 18-18. Animating through the colors of the rainbow by using keyframes.

Keyframe

```
#magic {  
  ...  
  animation-name: colors;  
  animation-duration: 5s;  
  animation-timing-function: linear;  
  animation-iteration-count: infinite;  
  animation-direction: alternate;  
}
```

Or,

```
#magic {  
  animation: colors 5s linear infinite alternate;  
}
```

Applies to `<div id="magic">Magic !</div>`

Keyframe

Adding Animation Properties

- I am going to apply the rainbow animation to the **#magic div** in my document:

```
<div id="magic">Magic!</div>
```

- In the CSS rule for **#magic**, I make decisions about the animation I want to apply:
 - Which animation to use (**animation-name**) (*Required*).
 - How long it should take (**animation-duration**) (*Required*).
 - The manner in which it should accelerate (**animation-timing-function**). This property uses the same timing function keywords that we covered for CSS Transitions.

Adding Animation Properties

animation-name

Which animation to use?

animation-duration

How long it should take?

animation-timing-function

The manner in which it should accelerate.

animation-delay

Whether to pause before it starts.

Adding Animation Properties

animation-iteration-count

How many times the animation should repeat. This can be set to a whole number or **infinite**.

animation-direction

Whether the animation plays forward (**normal**), in reverse (**reverse**), or alternates back and forth starting at the beginning (**alternate**), or alternates starting from the end (**alternate-reverse**).

animation-fill-mode

The animation fill mode determines what happens with the animation before it begins and after it ends. By default (**none**), the animation shows whatever property values were not specified via **@keyframes**. If you want the last keyframe to stay visible after the animation plays, use the **forwards** keyword. If there is a delay set on the animation and you want the first keyframe to show during that delay, use **backwards**. To retain the beginning and end states, use **both**.

Adding Animation Properties

animation-play-state

Whether the animation should be **running** or **paused** when it loads. The play-state can be toggled on and off based on user input with JavaScript or on hover.

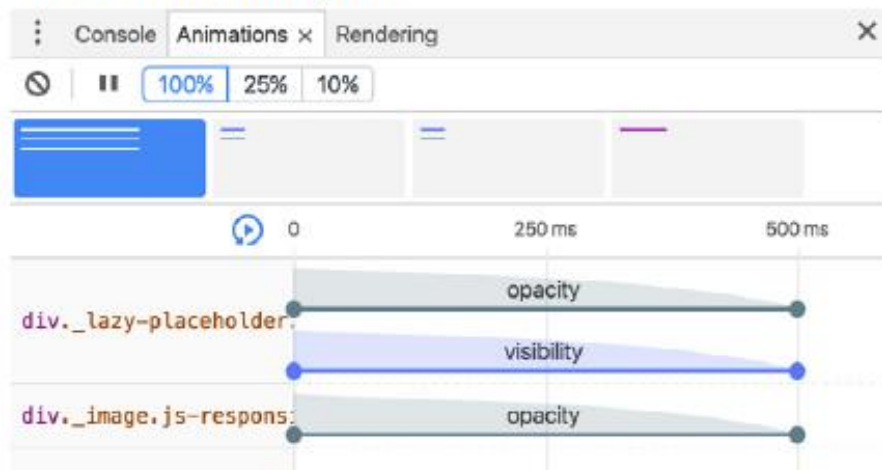
The **animation-name** property tells the browser which keyframe sequence to apply to the **#magic div**. I've also set the duration and timing function, and used **animation-iteration-count** to make it repeat infinitely. I could have provided a specific number value, like 2 to play it twice, but how fun are only two rainbows? And for fun, I've set the **animation-direction** to **alternate**, which makes the animation play in reverse after it has played forward.

- Both Chrome and Firefox offer tools to inspect and modify web animations (FIGURE 18-19). When you inspect an animated element in the Developer Tools, click the Animations tab to see a timeline of all the animations applied to that object.

Firefox Animation Inspector



Chrome Animation Inspector



Animation Inspectors

FIGURE 18-19. Animation inspectors are part of the developer tools offered by Firefox and Chrome browsers.

CSS Techniques

SECTION 1



CSS Techniques

- Styling forms
- Style properties for tables
- Using a CSS reset or normalizer
- Image replacement techniques
- CSS sprites
- CSS feature detection

Styling Forms

SECTION 1

Styling Forms

The following is a quick rundown of the types of things you can do for each form control type.

Text inputs (text, password, email, search, tel, url)

Change the appearance of the box itself with **width**, **height**, **background-color**, **background-image**, **border**, **border-radius**, **margin**, **padding**, and **box-shadow**. You can also style the text inside the entry field with the **color** property and the various font properties.

The textarea element

This can be styled in the same way as text-entry fields. **textarea** elements use a monospace font by default, so you may want to change that to match your other text-entry fields. Because there are multiple lines, you may also specify the line height. Note that some browsers display a handle on the lower-right corner of the **textarea** box that makes it resizable, but you can turn it off by adding the style **resize: none**. Text areas display as **inline-block** by default, but you can change them to **block** with the **display** property.

Styling Forms

Button inputs (submit, reset, button)

Apply any of the box properties to submit and reset buttons (**width**, **height**, **border**, **background**, **margin**, **padding**, and **box-shadow**). It is worth noting that buttons are set to the border-box sizing model by default. Most browsers also add a bit of padding by default, which can be overridden by your own padding value. You can also style the text that appears on the buttons.

Radio and checkbox buttons

The best practice for radio and checkbox buttons is to leave them alone. If you are tenacious, you can use JavaScript to change the buttons altogether.

Drop-down and select menus (select)

You can specify the width and height for a **select** element, but note that it uses the border-box box-sizing model by default. Some browsers allow you to apply **color**, **background-color**, and font properties to **option** elements, but it's probably best to leave them alone to be rendered by the browser and operating system.

Styling Forms

Fieldsets and legends

You can treat a **fieldset** as any other element box, adjusting the border, background, margin, and padding. Turning the border off entirely is one way to keep your form looking tidy while preserving semantics and accessibility. By default, **legend** elements are above the top border of the **fieldset**, and, unfortunately, browsers make them very difficult to change. Some developers use a **span** or **b** element within the **legend** and apply styles to the contained element for more predictable results. Some choose to hide it in a way that it will still be read by screen readers (**legend {width: 1px; height: 1px; overflow: hidden;}**).

Custom Sneaker Order Form

- Name:
- Email:
- Telephone:
- Tell us about yourself:
- Size: Sizes reflect standard men's sizes
- Sneaker Color:
 - Color
 - ☐ Red
 - ☐ Blue
 - ☐ Black
 - ☐ Silver
- Add-on Features:
 - Feature
 - ☐ Sparkley laces
 - ☒ Metallic logo
 - ☐ Light-up heels
 - ☐ MP3-enabled
-

Original Form

FIGURE 19-1. Forms tend to be ugly and difficult to use with HTML alone. Don't worry—this one gets spiffed up in [FIGURE 19-2](#).

Wide viewport

Custom Sneaker Order Form

Name:

Email:

Telephone:

Tell us about yourself:

Size: Sizes reflect standard US men's sizes

Sneaker Color: ☐ Red ☐ Blue ☐ Black ☐ Silver

Add-on Features: ☐ Sparkley laces ☒ Metallic logo ☐ Light-up heels ☐ MP3-enabled

Narrow viewport

Custom Sneaker Order Form

Name:

Email:

Telephone:

Tell us about yourself:

Size: Sizes reflect standard US men's sizes

Sneaker Color: ☐ Red ☐ Blue ☐ Black ☐ Silver

Add-on Features: ☐ Sparkley laces ☒ Metallic logo ☐ Light-up heels ☐ MP3-enabled

Styled Form

FIGURE 19-2. This responsive form uses Flexbox to allow text inputs to resize and to shift the position of the labels on small screens.



Styling Forms

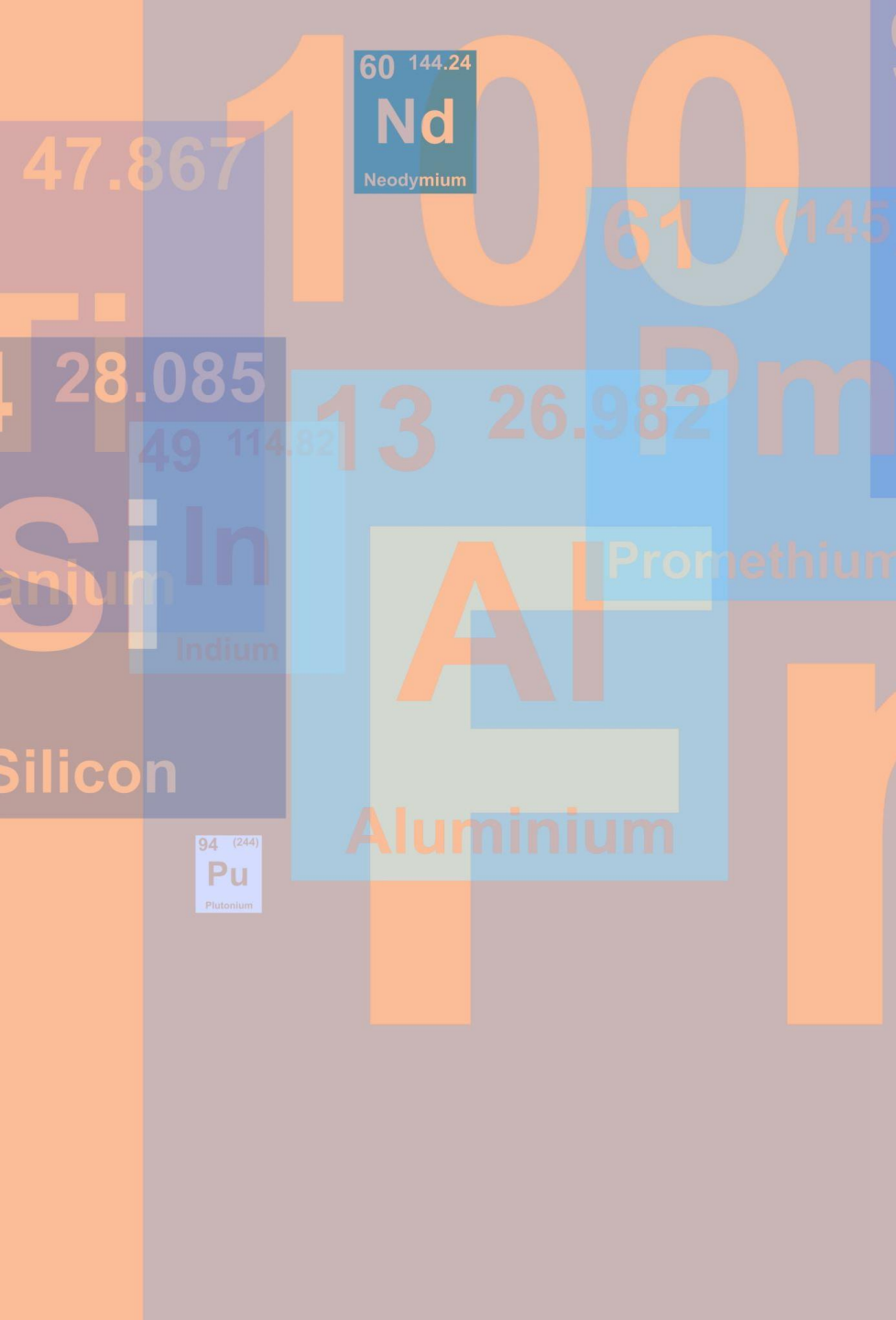
Step 0: Build the basic form

Step 1: Adding basic styles

Step 2: Aligning labels and inputs

Step 3: Fixing fieldsets and minor labels

Step 4: Adjusting the buttons



Styling Forms

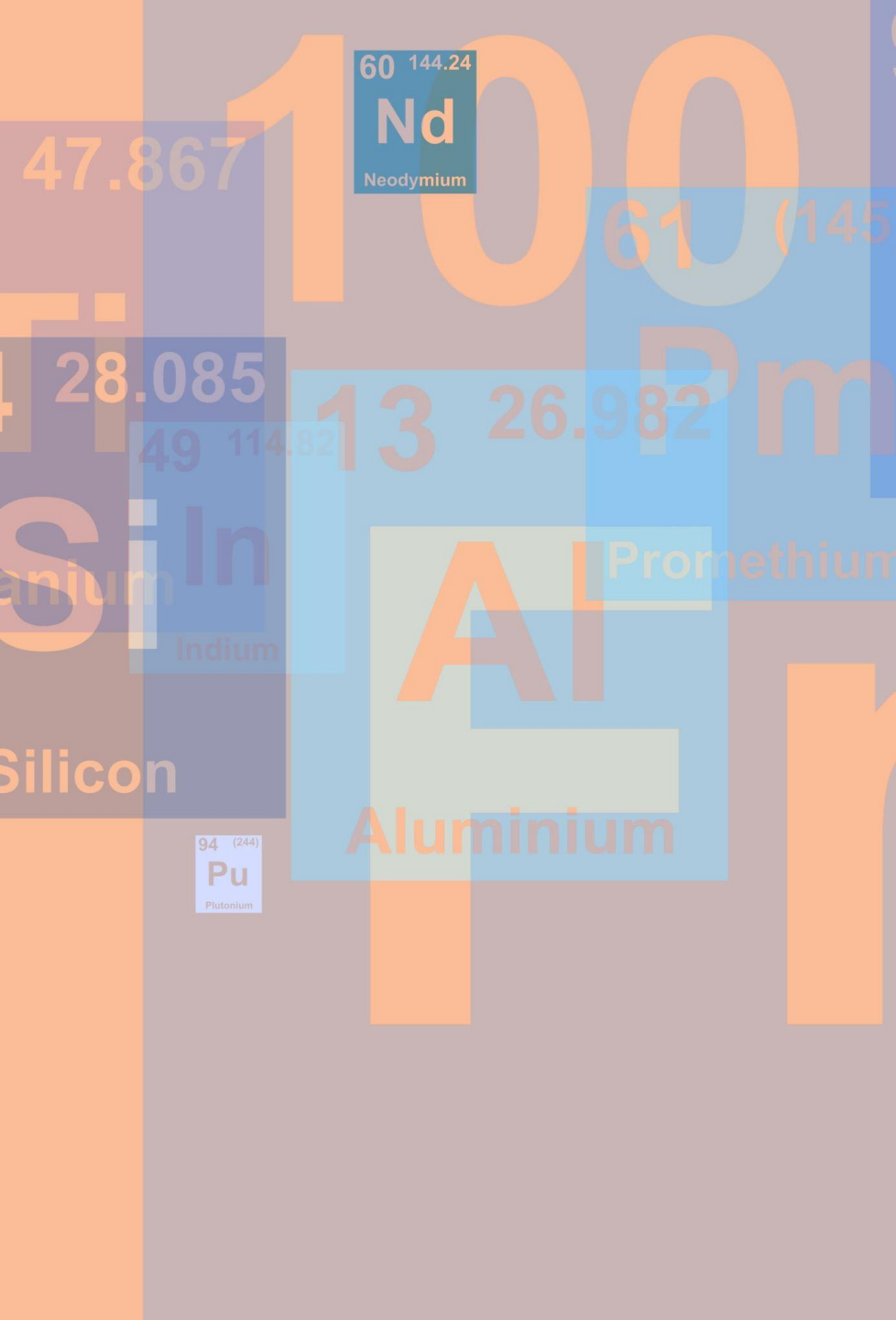
Step 0: Build the basic form

Step 1: Adding basic styles

Step 2: Aligning labels and inputs

Step 3: Fixing fieldsets and minor labels

Step 4: Adjusting the buttons



Step 1: Adding Basic Styles

ul, il: list layout

clear: next Element location

overflow: hidden

color:

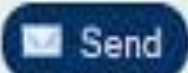
font size:



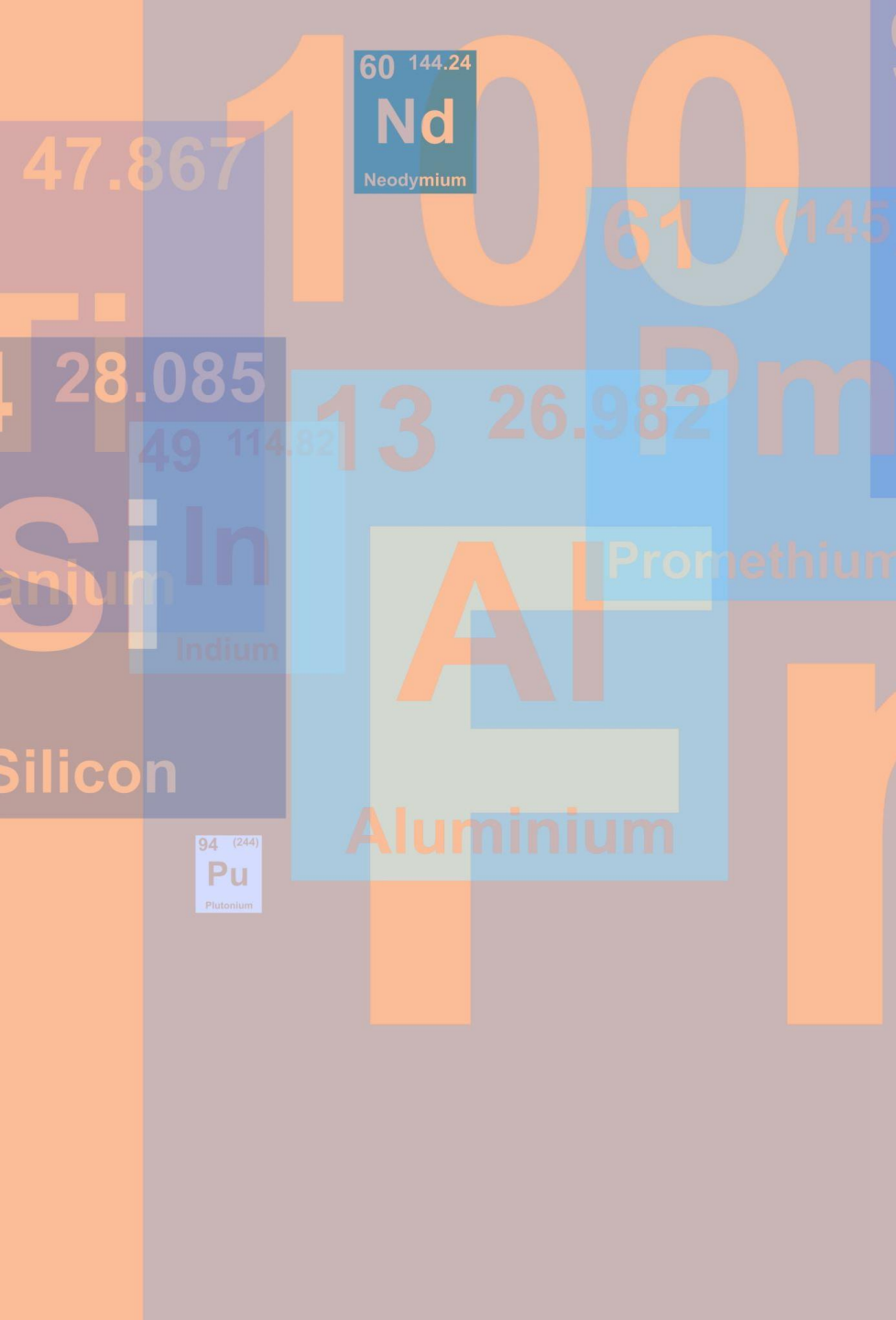
Step 2: Align labels and inputs

Send us your comments and feedback:

Name	<input type="text"/>
Email	<input type="text"/>
Website	<input type="text"/>
Message	<input type="text"/>

 Send

(Aligned Right)



Step 2: Align labels and inputs

Contact Form :

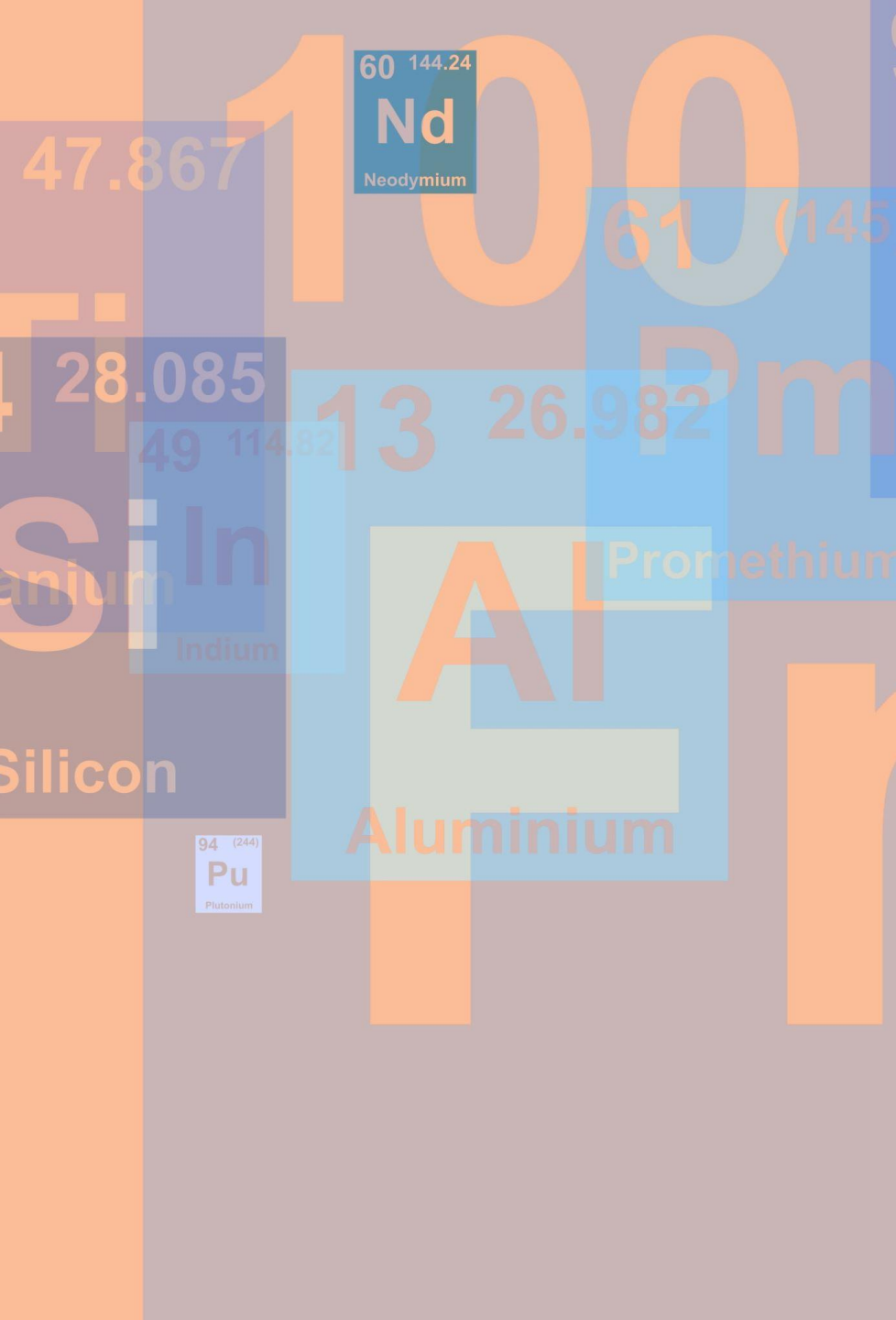
Your name

Email Address

Subject

Message

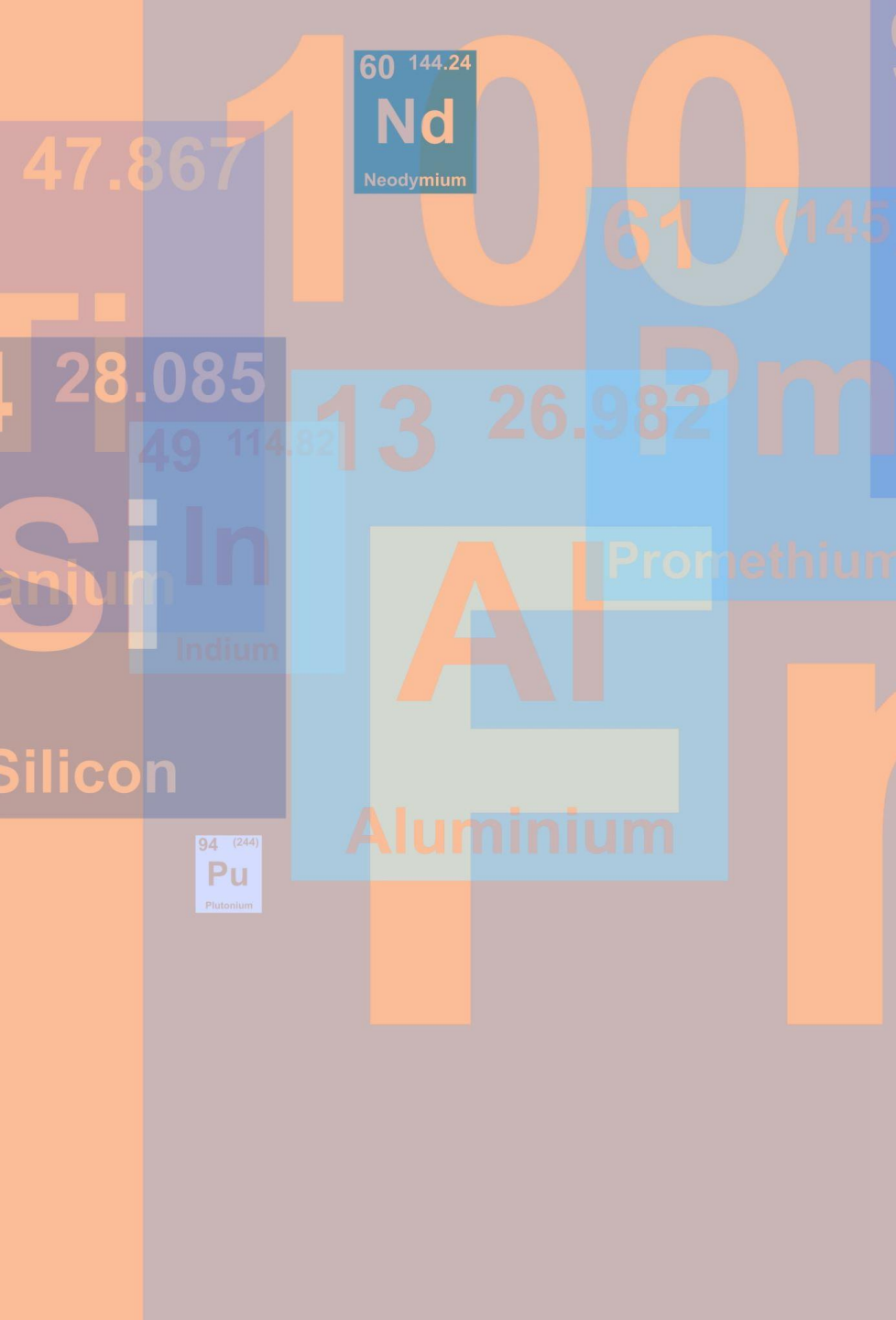
(Stay On Top)



Step 2: Align labels and inputs

A screenshot of a web form titled "INSCRIPTION A C553". The form has a light gray background and rounded corners. It contains three input fields, each with a label to its left: "Nom", "Mot de passe", and "Afficher + d'options". The labels and input fields are aligned to the left. At the bottom right, there is a blue button labeled "ENVOYER" and a link labeled "ou annuler".

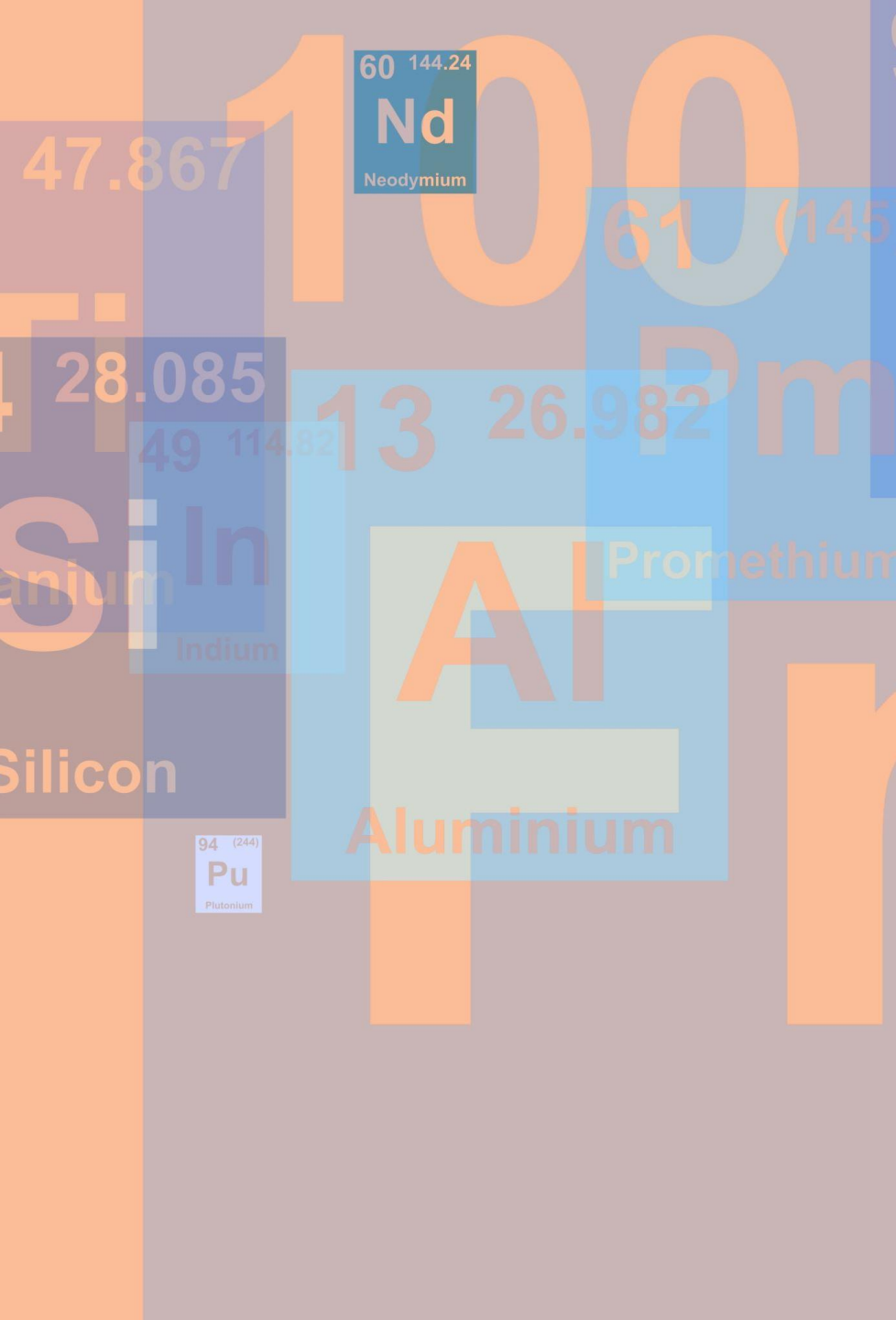
(Aligned Left)



Step 3: Fixing fieldsets and minor labels

Issues:

- **fieldset** border
- **radio display** (inline or block)
- **checkbox** line up or run-down (use clear property)
- **text-align**: text alignment
- Margins, Colors



Step 4: Adjusting the Buttons

Type Selector:

```
input[type="submit"] { ...; }  
  
/* select only a certain type of  
input */
```

Possible Types: submit, reset, radio, checkbox, password, text, file, date, time, datetime-local, date-label, month, week, ...

Styling Tables

SECTION 1

Styling Tables

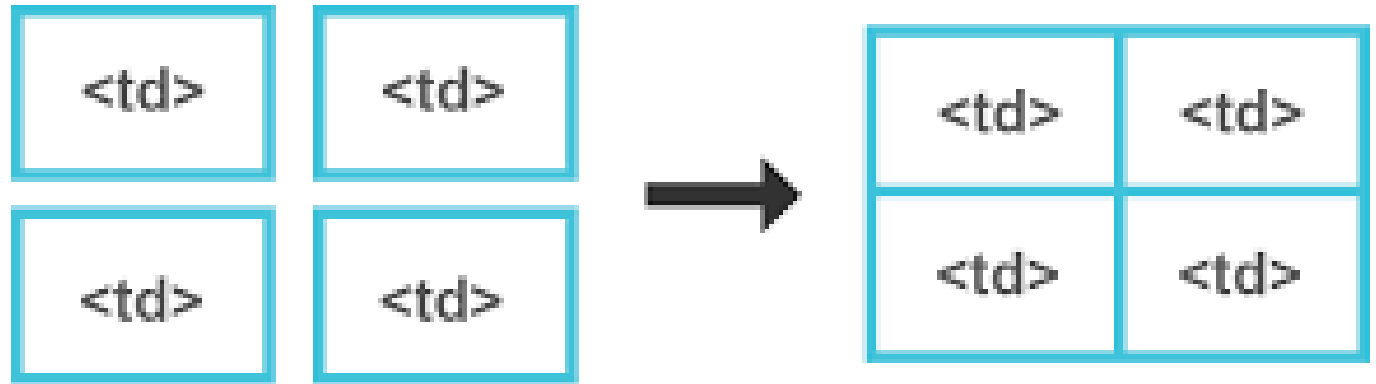
You will probably want to adjust the spacing in and around tables. To adjust the amount of space within a cell ([cell padding](#)), apply the **padding** property to the **td** or **th** element. Spacing between cells ([cell spacing](#)) is a little more complicated and is related to how CSS handles cell borders. CSS provides two methods for displaying borders between table cells: [separated](#) or [collapsed](#). These options are specified with the table-specific **border-collapse** property with **separate** and **collapse** values, respectively.

border-collapse

Values:	separate collapse
Default:	separate
Applies to:	table and inline-table elements
Inherits:	yes.

Styling Tables

BORDER-COLLAPSE: COLLAPSE;



border-spacing

Values: *horizontal-length vertical-length*

Default: 0

Applies to: table and inline-table elements

Inherits: yes

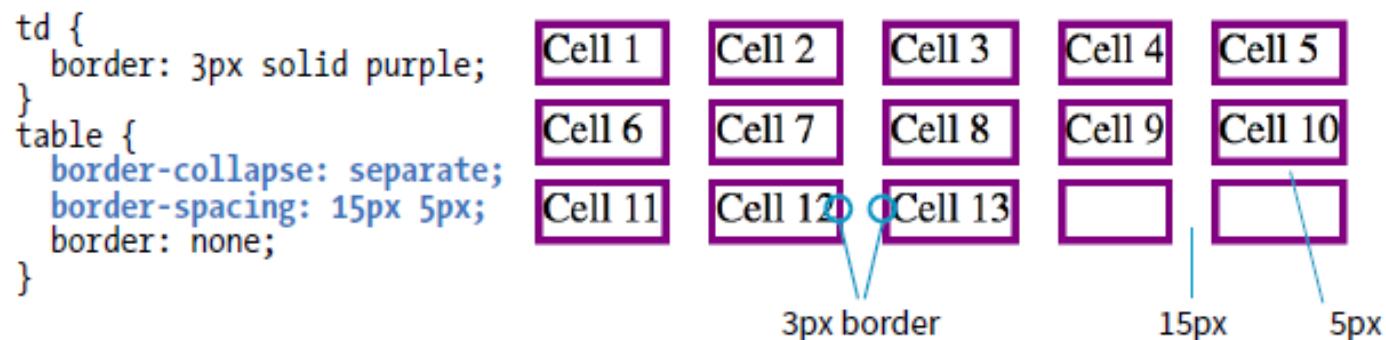


FIGURE 19-3. The separated table border model.

Separated Borders

empty-cells

Values: show | hide
Default: show
Applies to: table cell elements
Inherits: yes

`empty-cells: hide;`

Cell 1	Cell 2	Cell 3	Cell 4	Cell 5
Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 11	Cell 12	Cell 13		

FIGURE 19-4. Hiding empty cells with the `empty-cells` property.

Empty Cells

```
td {  
  border: 3px solid purple;  
}  
table {  
  border-collapse: collapse;  
  border: none;  
}
```

Cell 1	Cell 2	Cell 3	Cell 4	Cell 5
Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 11	Cell 12	Cell 13		

3px border

Collapsed Borders

FIGURE 19-5. The collapsed border model.

table-layout

Values:	auto fixed
Default:	auto
Applies to:	table or inline-table elements
Inherits:	yes

The **table-layout** property allows authors to specify one of two methods of calculating the width of a table. The **fixed** value bases the table width on **width** values provided for the table, columns, or cells. The **auto** value bases the width of the table on the minimum width of the contents of the table. Auto layout may display nominally more slowly because the browser must calculate the default width of every cell before arriving at the width of the table.

Table Layout

table-layout property

table-layout:auto;

Author Name	Age	College
RaviPratap Singh	24	GFG
Rakesh Singh	25	GEEKS

table-layout:fixed;

Author Name	Age	College
RaviPratap Singh	24	GFG
Rakesh Singh	25	GEEKS

Table Layout

When you use the **caption** element in a table, it will appear above the table by default. If you'd prefer it to be below the table, you can use the **caption-side** property to position it there.

caption-side

Values: top | bottom
Default: top
Applies to: table caption element
Inherits: yes

caption-side: top 📌

Populations of cities	
City	Population
Berlin	1
New York City	2

caption-side: bottom 📌

City	Population
Berlin	1
New York City	2
Populations of cities	

Same markup
different caption positions

Pick a Side

A Clean Slate

SECTION 1



A Clean Slate (CSS Reset)

- Browsers have their own built-in style. If you don't like them, you have to override them by your own CSS style sheet. If there are too much of it, it may be a tedious task. If you do not override them with your own design, your elements will inherit them from the built-in styles.
- CSS reset provide a method to reset all CSS setting by putting your own favorite reset condition at the very beginning of your own HTML page. You may get a fresh start and then, you may put your own style sheet later.
- The most popular reset was written by Eric Meyer.
<http://meyer.com/eric/tools/css/reset>
- To use the reset, place these styles at the top of your own style sheet.



CSS Reset

- The older approach is a **CSS reset**, a collection of style rules that overrides *all* user agent styles and creates a starting point that is as neutral as possible.
- With this method, you need to specify all the font and spacing properties for every element you use. It's a truly from-scratch starting point.

<https://meyerweb.com/eric/tools/css/reset/>

CSS Reset

Eric Meyer's "Reset CSS" 2.0	(29,196)	Get The Code
HTML5 Doctor CSS Reset	(23,659)	Get The Code
Yahoo! (YUI 3) Reset CSS	(10,923)	Get The Code
Universal Selector '*' Reset	(7,175)	Get The Code
Normalize.css 1.0	(3,054)	Get The Code

- A more nuanced approach is to use Normalize.css, created by Nicolas Gallagher and Jonathan Neal. They painstakingly combed through the user agent styles of every modern browser (desktop and mobile) and created a style sheet that tweaks their styles for consistency, rather than just turning them all off. Normalize.css gives you a reasonable starting point: paragraphs still have some space above and below, headings are bold in descending sizes, lists have markers and indents as you would expect. It also includes styles that make form widgets consistent, which is a nice service. [FIGURE 19-6](#) shows the difference between CSS reset and Normalize.css starting points.
- You can download Normalize.css at necolas.github.io/normalize.css/ and include it before your own styles. It is too long to print here, but you will find that it is well organized and includes comments with clear explanations for each section. For Nicolas's thoughts on the project, see nicolasgallagher.com/about-normalize-css/.
- Normalize.css is considered a superior successor to the cruder CSS reset, but I think it is important to be aware of both options. Or, if slight differences from browser to browser are just fine with you (as they are for a lot of professional developers), you don't need to use either.

Normalize.css

CSS reset

List Item
List Item
List Item
List Item
List Item
Big letters
Less big of letters
Getting smaller
Hello world
Kid Rock
Ant man
Lorem ipsum dolor sit amet.
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Fugit dignissimos iusto maxime, quibusdam ut harum
consectetur 1 rem consequuntur sunt vero!
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aperiam iste, soluta quibusdam ullam error qui atque 7,
expedita quod dicta vel laudantium tenetur, quos, voluptatibus aspernatur enim magni velit. Distinctio, adipisci.

[Singles in your area](#)
 fancy
loud
loud?
copyright forever

Normalize.css

- List Item
- List Item
- List Item
- List Item
- List Item

Big letters

Less big of letters

Getting smaller

Hello world

Kid Rock

Ant man

Lorem ipsum dolor sit amet.

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Fugit dignissimos iusto maxime, quibusdam ut
harum consectetur 1 rem consequuntur sunt vero!

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aperiam iste, soluta quibusdam ullam error qui
atque 7, expedita quod dicta vel laudantium tenetur, quos, voluptatibus aspernatur enim magni velit.
Distinctio, adipisci.

[Singles in your area](#)

fancy

loud

loud?

copyright forever

CSS reset

FIGURE 19-6. The difference between CSS reset (left) and Normalize.css (right). (Credit: screenshot of a Codepen created by Zach Wolf.)

Image Replacement Techniques

SECTION 1

Image Replacement Techniques

CSS image replacement is a technique of **replacing a text element (usually a header tag) with an image**. An example of this would be including a logo on a page. You may want to use a `<h1>` tag and text for this for the accessibility and SEO benefits, but ideally you'd like to show your logo, not text.

Image Replacement Techniques

In the example in [FIGURE 19-7](#), I use the Phark technique to display the Jenware logo in place of the **h1** “Jenware” text in the HTML source. The markup is simple:

```
<h1 id="logo">Jenware</h1>
```

The style rule is as follows:

```
#logo {  
    width: 450px;  
    height: 80px;  
    background: url(jenware.png) no-repeat;  
    text-indent: -9999px;  
}
```

What users see:



What is actually happening:

`text-indent: -9999px;`

Jenware

The h1 text content is pushed way off to the left, outside the browser window.

Browser window



Image Replacement Techniques

FIGURE 19-7. The Phark image replacement technique hides the HTML text by pushing it out of the visible element box with a large negative text indent so only the background image displays.

Image Replacement Techniques 9 Categories

- The report card consists of five major categories:
- **CSS ON / Images ON**
Represents browsers in their normal states. All techniques should pass this test, since that's the whole point.
- **CSS ON / Images OFF**
Represents browsing with regular stylesheets applied but images turned off. This is rare but a possibility (folks with bandwidth concerns...) This is the most difficult test. Since most of these techniques go to various lengths to hide text, when the images are turned off that means that nothing is displayed which ain't good. Displaying text only here is considered a pass.

Image Replacement Techniques 9 Categories

- **CSS OFF / Images ON**

Represents browsing with no stylesheets being applied. Most techniques default to regular web text here which isn't exactly a fail, but since images may still be turned on, I don't consider it a pass either.

- **CSS OFF / Images OFF**

Represents browsing with both images turned off and no stylesheets applied. Defaulting to text here is considered a pass.

- **Extra Unnecessary Markup**

Having to add markup for the sole purpose of image replacement is not ideal. Does not achieve true separation from content and design.

Technique #1

Background as foreground,
display: none (hide the span)

HTML

```
<h1 id="technique-one">  
  <span>CSS-Tricks</span>  
</h1>
```

CSS

```
h1#technique-one {  
  width: 250px;  
  height: 25px;  
  background-image: url(logo.gif);  
}  
h1#technique-one span {  
  display: none;  
}
```

REPORT CARD			
CSS ON IMAGES ON	CSS ON IMAGES OFF	CSS OFF IMAGES ON	CSS OFF IMAGES OFF
✓ PASS	✗ FAIL	TEXT ONLY	✓ PASS
EXTRA UNNECESSARY MARKUP?		ISSUES	
YES, SPAN		DISPLAY: NONE IS BAD FOR SCREEN READERS	

Image Replacement Techniques

Technique #2

(move content out of
viewport) Giant Box

HTML

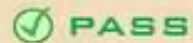
```
<h1 class="technique-two">  
  CSS-Tricks  
</h1>
```

CSS

```
h1.technique-two {  
  width: 2350px; height: 75px;  
  background: url("images/header-image.jpg")  
  margin: 0 0 0 -2000px;  
}
```

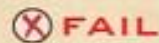
REPORT CARD

CSS ON
IMAGES ON



PASS

CSS ON
IMAGES OFF



FAIL

CSS OFF
IMAGES ON

TEXT ONLY

CSS OFF
IMAGES OFF



PASS

EXTRA UNNECESSARY MARKUP?

NO

ISSUES

REQUIRES GIANT BOX, LESS
EFFICIENT THAN OTHERS

Image Replacement Techniques

Technique #3 (Move text to outer space)

HTML

```
<h1 class="technique-three">  
  CSS-Tricks  
</h1>
```

CSS

```
h1.technique-three {  
  width: 350px;  
  height: 75px;  
  background: url("images/header-image.jpg")  
  text-indent: -9999px;  
}
```

REPORT CARD			
CSS ON IMAGES ON	CSS ON IMAGES OFF	CSS OFF IMAGES ON	CSS OFF IMAGES OFF
✓ PASS	✗ FAIL	TEXT ONLY	✓ PASS
EXTRA UNNECESSARY MARKUP?		ISSUES	
NO		NONE	

Image Replacement Techniques

Technique #4

(Text as alt)

HTML

```
<h1 class="technique-four">
  <a href="#">
    
  </a>
</h1>
```

CSS

```
h1.technique-four {
  width: 350px; height: 75px;
  background: url("images/header-image.jpg");
  text-indent: -9999px;
}
```

REPORT CARD			
CSS ON IMAGES ON	CSS ON IMAGES OFF	CSS OFF IMAGES ON	CSS OFF IMAGES OFF
✓ PASS	✗ FAIL	✓ PASS	✓ PASS
EXTRA UNNECESSARY MARKUP?		ISSUES	
ARGUABLY NO		HEAVYWEIGHT, REQUIRES TWICE THE IMAGES LOADED	

Image Replacement Techniques

Technique #5 (Text as alt and hiding span)

HTML

```
<h1 class="technique-five">
  
  <span>CSS-Tricks</span>
</h1>
```

CSS

```
h1.technique-five {
  width: 350px; height: 75px;
  background: url("images/header-image.jpg");
}
h1.technique-five span {
  display: none;
}
```

REPORT CARD			
CSS ON IMAGES ON	CSS ON IMAGES OFF	CSS OFF IMAGES ON	CSS OFF IMAGES OFF
✓ PASS	✓ PASS	TEXT ONLY	DOUBLE TEXT
EXTRA UNNECESSARY MARKUP?		ISSUES	
YES, TRANSP. GIF		EXTRA PAGE ELEMENT IS WORSE THAN JUST EXTRA MARKUP	

Image Replacement Techniques

Technique #6

HTML

```
<h1 class="technique-six">  
  CSS-Tricks  
</h1>
```

CSS

```
h1.technique-six {  
  width: 350px;  
  padding: 75px 0 0 0;  
  height: 0;  
  background: url("images/header-image.jpg")  
  overflow: hidden;  
}
```

Image Replacement Techniques

(Move Text out of box and set overflow hidden)
Same trick for textbook

Technique #7 (set 0 size and set overflow hidden)

HTML

```
<h1 class="technique-seven">
  <span>CSS-Tricks</span>
</h1>
```

CSS

```
h1.technique-seven {
  width: 350px; height: 75px;
  background: url("images/header-image.jpg")
}
h1.technique-seven span {
  display: block;
  width: 0;
  height: 0;
  overflow: hidden;
}
```

REPORT CARD			
CSS ON IMAGES ON	CSS ON IMAGES OFF	CSS OFF IMAGES ON	CSS OFF IMAGES OFF
✓ PASS	✗ FAIL	TEXT ONLY	✓ PASS
EXTRA UNNECESSARY MARKUP?		ISSUES	
YES, SPAN		NONE	

Image Replacement Techniques

Technique #8

HTML

```
<h1 class="technique-eight">  
  <span></span>CSS-Tricks  
</h1>
```

CSS

```
h1.technique-eight {  
  width: 350px; height: 75px;  
  position: relative;  
}  
h1.technique-eight span {  
  background: url("images/header-image.jpg");  
  position: absolute;  
  width: 100%;  
  height: 100%;  
}
```

Image Replacement Techniques

REPORT CARD

CSS ON IMAGES ON	CSS ON IMAGES OFF	CSS OFF IMAGES ON	CSS OFF IMAGES OFF
✓ PASS	✓ PASS	TEXT ONLY	✓ PASS
EXTRA UNNECESSARY MARKUP?		ISSUES	
YES, SPAN		TRANSPARENT BACKGROUND IMAGES REVEAL TEXT	

Technique #9

Meaningless font size

HTML

```
<h1 class="technique-nine">  
  CSS-Tricks  
</h1>
```

CSS

```
h1.technique-nine {  
  width: 350px; height: 75px;  
  background: url("images/header-image.jpg")  
  font-size: 1px;  
  color: white;  
}
```

Image Replacement Techniques

REPORT CARD

CSS ON
IMAGES ON



PASS

CSS ON
IMAGES OFF



FAIL

CSS OFF
IMAGES ON

TEXT ONLY

CSS OFF
IMAGES OFF



PASS

EXTRA UNNECESSARY MARKUP?

NO

ISSUES

ONLY WORKS ON FLAT-COLOR
BACKGROUNDS

CSS Sprites

SECTION 1

Stay Connected

Join our email list to receive exclusive updates, offers, and content

Enter Email

Join

CSS Sprites



CSS Sprites

- CSS is a technique to reduce the number of image requests in order to improve the site performance. The large image that contains multiple images is known as a sprite, a term coined by the early computer graphic and video game industry. The image gets positioned in the element using the **background-position** property in such a way that only the relevant part of it is visible.
- The strategy:
 - 1) a list of elements used the same background of same sprite image.
 - 2) the elements have a common class and an individual class
 - 3) the common class shares the same image file as background. The individual class assigns different **background-position**.

THE MARKUP

```
<ul>
```

```
  <li><a href="" class="hide twitter">Twitter</a></li>
  <li><a href="" class="hide fb">Facebook</a></li>
  <li><a href="" class="hide gplus">Google+</a></li>
  <li><a href="" class="hide linkedin">LinkedIn</a></li>
  <li><a href="" class="hide blip">Blip TV</a></li>
  <li><a href="" class="hide lanyrd">Lanyrd</a></li>
  <li><a href="" class="hide slides">Slideshare</a></li>
  <li><a href="" class="hide sched">Schedule</a></li>
  <li><a href="" class="hide attendees">Attendee List</a></li>
```

```
</ul>
```

CSS Sprites

THE STYLE

```
.hide { text-indent: 100%; white-space: nowrap; overflow: hidden; }
```

```
li a { display: block; width: 29px; height: 18px; background-image:  
url(social.png); }
```

```
li a.twitter      { background-position: 0 0; }
```

```
li a.fb           { background-position: 0 -20px; }
```

```
li a.gplus        { background-position: 0 -40px; }
```

```
li a.linkedin     { background-position: 0 -60px; }
```

```
li a.blip         { background-position: 0 -80px; }
```

```
li a.lanyrd       { background-position: 0 -100px; }
```

```
li a.slides       { background-position: 0 -120px; }
```

```
li a.sched        { background-position: 0 -140px; }
```

```
li a.attendees    { background-position: 0 -160px; }
```

CSS Sprites



The separate icons in this panel are contained on one sprite image (*social.png*) that is positioned in each `a` element.

social.png



`background-position: 0, -40px;`

`background-position: 0, 0;`

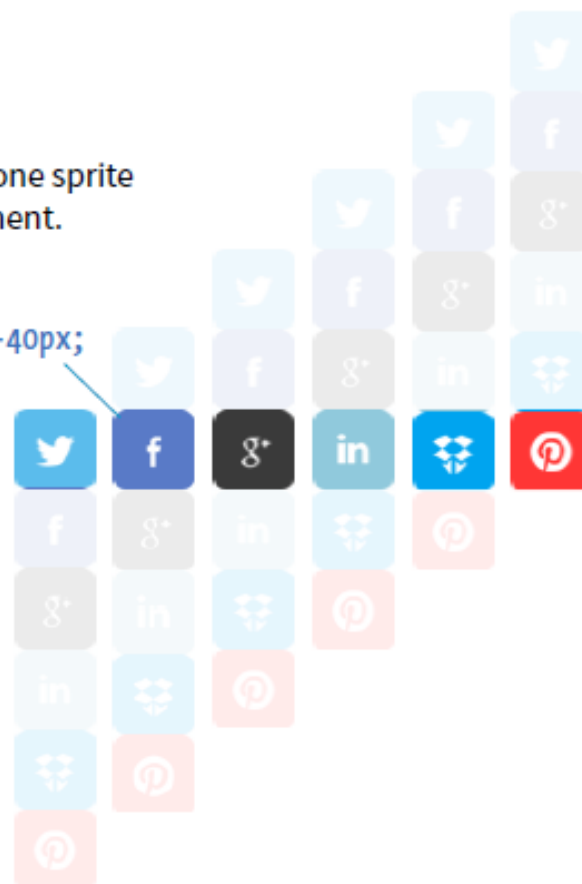


FIGURE 19-8. Replacing separate graphic files with one sprite image cuts down on the number of HTTP requests to the server and improves site performance.

CSS Feature Detection

SECTION 1



CSS Feature Detection

- One of the dominant challenges facing web designers and developers is dealing with **uneven browser support**. Useful new CSS properties emerge regularly, but it takes a while for them find their way into browsers, and it takes much longer for the old non-supporting browsers to fade into extinction.
- Fortunately, we have a few methods for checking to see if a browser supports a particular feature so we can take advantage of cutting-edge CSS while also providing thoughtful fallbacks for non-supporting browsers. Using feature detection with fallbacks sure beats the alternatives of a) not using a property until it is universally supported, or b) using it and letting users with nonsupporting browsers have a broken experience.
- We'll look at two ways to detect whether a feature is supported: feature queries with a new CSS at-rule (`@supports`) and a JavaScript-based tool called **Modernizr**.

CSS Feature Queries (@supports)

- The CSS3 Conditional Rules Module Level 3 (www.w3.org/TR/css3-conditional/) introduces the **@supports** rule for checking browser support of a particular property and value declaration. Commonly referred to as a **feature query**, it works like a media query in that it runs a test, and if the browser passes that test, it applies the styles contained in the brackets of the at-rule. The syntax for **@supports** is as follows:

```
@supports (property: value) {  
  /* Style rules for supporting  
  browsers here */  
}
```

CSS Feature Queries (@supports)

THE MARKUP

```
<div id="container">
  <figure class="blend">
    
  </figure>
</div>
```

THE STYLES

```
#container {
  background-color: #96D4E7;
  padding: 5em;
}
.blend img {
  opacity: .5;
}
@supports (mix-blend-mode: multiply) {
  .blend img {
    mix-blend-mode: multiply;
    opacity: 1;
  }
}
```



Original image (no effect)



As seen on browsers that support
`mix-blend-mode: multiply;`



Fallback for non-supporting browsers
(`opacity: .5`)

CSS Feature Queries (@supports)

FIGURE 19-9. The original image (left), the result using the **mix-blend-mode** property with **multiply** keyword (center), and the fallback style using **opacity** (right).

The **@supports** rule can be used with three operators to refine the feature test: **not**, **and**, and **or**:

not

The **not** operator lets us test for when a specific property/value pair is *not* supported.

```
@supports not (mix-blend-mode: hue) {  
    /* styles for non-supporting browsers */  
}
```

Someday, this will be useful for supplying fallback styles, but with the current browser support, you risk non-supporting browsers skipping everything in the **@supports** rules, including the fallbacks. That's why I used the override method in the previous example.

Operators

and

Applies styles only when all of the conditions in a series of two or more are met.

```
@supports (border-radius: 10em) and (shape-outside:
circle()) {
    /* styles that apply only when the browser supports
    shape-outside AND border-radius */
}
```

or

Use the **or** operator to apply styles when any of a series of conditions are met. This one is particularly useful for vendor-prefixed properties.

```
@supports (-webkit-transform: rotate(10deg)) or
(-ms-transform: rotate(10deg)) or
(transform: rotate(10deg)) {
    /* transform styles */
}
```

Operators

Modernizr

- [Modernizr](#) is a lightweight JavaScript library that runs behind the scenes and tests for a long list of HTML5 and CSS3 features when the page is loaded in the browser. For each feature it tests, it stores the result (supports/doesn't support) in a JavaScript object that can be accessed with scripts and optionally as a class name in the html root element that can be used in CSS selectors. I'm going to focus on the latter CSS method.

Modernizr

How it works

When Modernizr runs, it appends the **html** element with a class name for each feature it detects. For example, if it is configured to test for Flexbox, when it runs on a browser that *does* support Flexbox, it adds the **.flexbox** class name to the **html** element:

```
<html class="js flexbox">
```

If the feature is not supported, it adds the feature name with a **.no-** prefix.

On a non-supporting browser, the Flexbox test would be reported like this:

```
<html class="js no-flexbox">.
```

Modernizr

With the class name in place on the root element, everything on the page becomes part of that class. We can use the class name as part of a selector to provide different sets of styles depending on feature support:

```
.flexbox nav {  
    /* flexbox styles for the nav element  
    here */  
}  
.no-flexbox nav {  
    /* fallback styles for the nav element  
    here */  
}
```

This example is short and sweet for demonstration purposes. Typically, you'll use Modernizr to test for many features, and the **html** tag gets filled with a long list of class names.

Modernizr

How to use it

First, you need to download the *Modernizr.js* script. Go to [Modernizr.com](https://modernizr.com) and find the Download link. From there you can customize the script to contain just the HTML and CSS features you want to test, a nice way to keep the file size of the script down. Click the Build button, and you will be given several options for how it can be saved. A simple click on Download saves the script in a *.js* file on your computer.

Modernizr

Once you have your script, put it in the directory with the rest of the files for your project. Add it to the **head** of your HTML document, before any linked style sheets or other scripts that need to use it:

```
<head>  
  <script src="modernizr-  
    custom.js"></script>  
  <!--other scripts and style sheets -->  
</head>
```

Finally, open your HTML document and assign the **no-js** class name to the **html** element.

```
<html class="no-js">
```

Modernizr will change it to **js** once it detects that the browser supports JavaScript. If JavaScript (and therefore Modernizr) fails to run, you will not know whether or not features are supported.

Modernizr

Pros and cons

Modernizr is one of the most popular tools in web developers' arsenals because it allows us to design for particular features rather than whole browsers. It is easy to use, and the Modernizr site has thorough and clear documentation to help you along. Because it's JavaScript, it works on the vast majority of browsers.

The flip side to that, however, is that because it relies on JavaScript, you can't be 100% certain that it will run, which is its main disadvantage. It will also be slightly slower than using CSS alone for feature detection.

CSS3

TOGGLE

- ☐ @font-face
- ☐ background-size
- ☐ border-image
- ☐ border-radius
- ☐ box-shadow
- ☐ Flexible Box Model (flexbox)
- ☐ Flexbox Legacy
- ☐ hsla()
- ☐ multiple backgrounds
- ☐ opacity
- ☐ rgba()
- ☐ text-shadow
- ☐ CSS Animations
- ☐ CSS Columns
- ☐ CSS Generated Content (:before/:after)
- ☐ CSS Gradients
- ☐ CSS Reflections
- ☐ CSS 2D Transforms
- ☐ CSS 3D Transforms
- ☐ CSS Transitions

HTML5

TOGGLE

- ☐ applicationCache
- ☐ Canvas
- ☐ Canvas Text
- ☐ Drag 'n Drop
- ☐ hashchange
- ☐ History (pushState)
- ☐ HTML5 Audio
- ☐ HTML5 Video
- ☐ IndexedDB
- ☐ Input Attributes
Note: does not add classes
- ☐ Input Types
Note: does not add classes
- ☐ localStorage
- ☐ postMessage
- ☐ sessionStorage
- ☐ Web Sockets
- ☐ Web SQL Database
- ☐ Web Workers

Misc.

TOGGLE

- ☐ Geolocation API
- ☐ Inline SVG
- ☐ SMIL
- ☐ SVG
- ☐ SVG clip paths
- ☐ Touch Events
- ☐ WebGL

Extra

- ☒ html5shiv v3.7
- ☐ html5shiv v3.7.1 w/ printshiv
- ☒ Modernizr.load
([yepnope.js](#))
- ☐ Media Queries
- ☒ Add CSS Classes

className prefix:

Modernizr

Conclusion

SECTION 1



Conclusion

- That concludes our whirlwind tour of Cascading Style Sheets. You've come a long way since styling an **h1** and a **p** back in Chapter 11, Introducing Cascading Style Sheets. By now, you should be comfortable formatting text and even doing basic page layout. While CSS is easy to learn, it takes a lot of time and practice to master. If you get stuck, you will find that there are many resources online to help you find the answers you need. The nice thing about CSS is that you can start with just the basics and then build on that knowledge as you gain proficiency in your web development skills.
- In the next chapter, I'll introduce you to tools that web developers use to improve their workflow, including tools for writing CSS more efficiently and optimizing the results. But if you're feeling overwhelmed with CSS properties, you can breathe a sigh of relief. We're *done*!