Designing The Well-Tempered Web

By Rob Flaherty

January 17th, 2012 | Design, User Experience | 28 Comments

As technology evolves, so does the art and craft of Web design. New technology creates new challenges, which require new solutions. Often we're working in uncharted territory, where the solutions demanded really are new. Other times, we're faced with problems of a more universal nature, problems that have a history.

Given the limited history of Web design, we have to look beyond our immediate domain for answers to the more challenging questions. We do this all the time when we draw on the rich history of graphic design and visual arts. But we're not limited to sibling disciplines. We can identify the abstractions and patterns that constitute our challenges, we can look to any source for guidance. We can look to a seemingly unrelated field, such as psychology or music. We can even look to an episode from the early 18th century about Johann Sebastian Bach.

In this article we'll look at what Bach has to do with modern Web challenges — particularly the challenge of designing for devices with diverse attributes and capabilities.

Bach And “The Well-Tempered Clavier”

In 1722, Bach put together a book of solo keyboard works intended as a collection of educational pieces for young musicians. The book contained 48 pieces — a prelude and fugue in every major and minor key. Now a staple of the Western canon, it's regarded as one of the most important works in the history of Western music. He named the book The Well-Tempered Clavier.
This webinar is about techniques.
This webinar is about techniques.
And about tricky front-end strategies.
This webinar is about **techniques**.
And about tricky **front-end strategies**.
And what works in **real-life projects**.
CHOOSE ADVENTURE

EASY MEDIUM HARDCORE

PRESS ENTER TO CONTINUE
It’s your lucky day. You grow, and your company expands to foreign markets. Your site has to support 23 languages. How do you architect CSS/JS to support it?
The crucial asset of longevity is building “neutral”, configurable components which can be *easily* extended and adjusted.
// english.json
{
  serviceName: 'english';
  language: 'en';
  textDirection: 'ltr';
  socialMediaButtons: ['twitter', 'facebook', 'reddit'];
}

// russian.json
{
  serviceName: 'russian';
  language: 'ru';
  textDirection: 'ltr';
  textLength: 'verbose';
  socialMediaButtons: ['twitter', 'facebook', 'vk'];
}
With a *templating* language, we can then plug data from config files and hence customize HTML output for every language.
// english.scss
$english = true;
$script = 'latin';
$direction = 'left';
@include(mixins/directions);
@include(mainstyles);

// arabic.scss
$arabic = true;
$script = 'arabic';
$direction = 'right';
@include(mixins/directions);
@include(mainstyles);

@if $arabic {
    // apply styling only to Arabic version
}
// directions.scss
margin-left: margin-left;
if $direction == 'right' {
    margin-left: margin-right;
}

padding-left: padding-left;
if $direction == 'right' {
    padding-left: padding-right;
}

left: left;
if $direction == 'right' {
    left: right;
}
// directions.scss
$margin-left: margin-left;
if $direction == 'right' {
  $margin-left: margin-right;
}

$padding-left: padding-left;
if $direction == 'right' {
  $padding-left: padding-right;
}

$left: left;
if $direction == 'right' {
  $left: right;
}

$margin-right: margin-right;
if $direction == 'right' {
  $margin-right: margin-left;
}

$padding-right: padding-right;
if $direction == 'right' {
  $padding-right: padding-left;
}

$right: right;
if $direction == 'right' {
  $right: left;
}
// global.scss
.nav-element {
    ${margin-left}: 10px;
    ${padding-right}: 10px;
    ${left}: 10px;
}

// english.css
.nav-element {
    margin-left: 10px;
    padding-right: 10px;
    left: 10px;
}

// arabic.css
.nav-element {
    margin-right: 10px;
    padding-left: 10px;
    right: 10px;
}
// global.scss
.nav-element {
  float: flip(left, right);
  padding: flip(10px 10px 0 0, 10px 0 0 10px);
  line-height: get-script-value(latin 1.3, arabic 1.6);
}

// english.css
.nav-element {
  float: left;
  padding: 10px 10px 0 0;
  line-height: 1.3em;
}

// arabic.css
.nav-element {
  float: right;
  padding: 10px 0 0 10px;
  line-height: 1.6em;
}
Gahuza

Uhuru asaba Afrika kwikura ku mfashanyo

Umukuru w’igihu w Kenya, Uhuru Kenyatta, yahamagariye bagenzibwe b’abanye Afrika guhagarika kwakira imfashanyo yi’igihu bikiye.

Haolye amasaha 4 Amakuru

Ibigi 30:00

Ibigi 30:00

Imvo n’Imvano 1:00:00

BBC itanga inyigisho kw’itangazamakuru

Ubukerarugendo burabangamiwe muri Kenya

Igigiri c’abakerarugendo bagendera igihu ca Kenya cagabunyiseko 25%.

Haolye amasaha 8 Amakuru

Avuga ko ari umwirabura tari we

Abavyeyi b’umugore aharanira agateka k’abirabura muri Amerika, bavugwa ko amaze imyaka abesha ko ari umwirabura.

Haolye isaha 1 Amakuru

Mu Burundi imyiyerekano "itumwa n’abamenyeshamakuru":leta

Ubutegisi mu Burundi twemweza ko aita myiyerekano ikirangwa i Bujumbura.

11 Ukwa gatandatu 2015 Amakuru
Украина: Порошенко отправил в отставку главу минобороны

Президент Украины Петр Порошенко отправил в отставку министра обороны страны Валерия Гелетя. Гелетя занимал этот пост с начала июля 2014 года.

Британец Хэмптон выиграл первый Гран-при России в Сочи

Британский гонщик Люис Хэмптон из команды "Мерседес" стал победителем первого в истории "Формулы-1" Гран-при России, который прошел в Сочи.

Глава Гонконга: протестующие не изменят позицию Китая

Глава администрации Гонконга Лян Чжэньвин заявил, что у участников уличных акций протеста нулевой шанс изменить позицию Пекина по поводу формата выборов 2017 года.
Лента новостей

39 минут назад
Египет: полиция разогнала студенческие протесты в Каире

В Египте студенты нескольких крупных университетов устроили в Каире антиправительственные протесты.

Они требовали от властей освободить задержанных ранее участников студенческих демонстраций.

Полиция при поддержке нескольких бронемашин разогнала протесты, несколько студентов было арестовано.

Сообщается, что протесты были организованы сторонниками свергнутого президента Мохаммеда Мураи.

1 час назад
Европейское турнирное объединение не смогла обыграть Молдавию

1 час назад
На севере Ливии возобновились бои
Responsive News

Where BBC News developers blog about responsive design.

Opinions expressed on this blog are those of the individual contributors, and are not necessarily those of the BBC as a whole.

13 tips for making responsive web design multi-lingual

Responsive Web Design (RWD) builds on the primary design principle underlying the web’s core usefulness and growth: universality. A content out approach that is device agnostic makes your responsive website future friendly as it will in theory work on any device. The web wins the more viewable your website is. By adapting our responsive websites to work with multiple languages we can further increase the number of users who are able to use our content.

The BBC News responsive codebase is responsible for the rendering of 28 different
You want to *add a background* to inline text for headings, but the text should be *padded* along both the left and right edge of each line. Left/right padding will only apply to the very first and very last line.
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fugiat suscipit ut odit animi consequatur error numquam perspiciatis voluptas quod eum.
Adam Campbell's box-decoration-break Method

During a discussion that popped up over this, Adam pointed out there is a new CSS property that is (as I understand it) specifically for this. This removes the need for three elements. Technically you only need one, the inline element, but it's likely you'll be doing this on a header so you'll probably end up with a block-parent anyway, which is best for spacing.

Here is the original and my stripped down demo:

<table>
<thead>
<tr>
<th>HTML</th>
<th>CSS</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How do I add padding to subsequent lines of an inline text element?

This is working in Chrome and Safari, but not Firefox now working in Firefox 32+, in my quick tests. Chrome and Safari require it to be -webkit-box-decoration-break.
The `box-decoration-break` in CSS specifies element’s appearance if the box for the element is fragmented, i.e. when an inline box wraps onto multiple lines, or when a block spans more than one column inside a column layout container.
**CSS box-decoration-break**

Controls whether the box's margins, borders, padding, and other decorations wrap the broken edges of the box fragments (when the box is split by a break (page/column/region/line)).

<table>
<thead>
<tr>
<th>Current aligned</th>
<th>Usage relative</th>
<th>Show all</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>IE</th>
<th>Edge</th>
<th>Firefox</th>
<th>Chrome</th>
<th>Safari</th>
<th>Opera</th>
<th>iOS Safari</th>
<th>Opera Mini</th>
<th>Android Browser</th>
<th>Chrome for Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>43</td>
<td>46</td>
<td>45</td>
<td>42</td>
<td>8</td>
<td>9</td>
<td>8.4</td>
<td>9.3</td>
<td>4.3</td>
<td>49</td>
</tr>
<tr>
<td>9</td>
<td>44</td>
<td>47</td>
<td>49</td>
<td>47</td>
<td>8.4</td>
<td>9.2</td>
<td>8</td>
<td>9.3</td>
<td>4.4</td>
<td>49</td>
</tr>
<tr>
<td>10</td>
<td>44</td>
<td>47</td>
<td>49</td>
<td>48</td>
<td>8.4</td>
<td>9.2</td>
<td>8</td>
<td>9.3</td>
<td>4.4</td>
<td>49</td>
</tr>
<tr>
<td>11</td>
<td>45</td>
<td>49</td>
<td>49</td>
<td>49</td>
<td>8.4</td>
<td>9.2</td>
<td>8</td>
<td>9.3</td>
<td>4.4</td>
<td>49</td>
</tr>
<tr>
<td>14</td>
<td>46</td>
<td>50</td>
<td>9.1</td>
<td>9.1</td>
<td>9.3</td>
<td>8</td>
<td>47</td>
<td>49</td>
<td>49</td>
<td>49</td>
</tr>
</tbody>
</table>

**Notes**

1 Partial support refers to working for inline elements but not across column or page breaks.
DEFAULT: THIS IS HOW HIGHLIGHTED TEXT USALLY WRAPS. IT GETS TIGHT ON THE LEFT AND RIGHT EDGES.
We can use a little trick: apply a \textit{zero-spread box-shadow} on an inline element by defining the shadow only on the $x$-axis.
Fabien Doiron's box-shadow Method

Turns out you can use zero-spread box-shadow on an inline element on only the x-axis to pad each line. Essentially:

```css
.padded-multi-line {
  display: inline;
  background: orange;
  box-shadow: 10px 0 0 orange, -10px 0 0 orange;
}
```

Here is the original and then my fork to show how it works:

DEFAULT: THIS IS HOW HIGHLIGHTED TEXT USALLY WRAPS. IT GETS TIGHT ON THE LEFT AND RIGHT EDGES.
Multi-Line Padded Text

This is one of those tricky CSS things that I see come up every few months. I guess what better place to address it than CSS-Tricks eh?

The situation involves ragged-right inline text. Like when a paragraph row over the next word won't fit (e.g.
CHECK OUT REACT FOR BEGINNERS

I LOVE TO LEARN!

ARE YOU A SUBLIME TEXT POWER USER?

SURE DO LOVE FLEXBOX!
a {
    font-size: 4rem;
    text-decoration: none;
    color: #1e1f23;
    text-shadow: 1px 1px 0 rgba(255,255,255,0.4);
}
.fun-hover {
    background-image: -webkit-linear-gradient(left, #25b0a9 50%, #fee603 50%);
    background-image: linear-gradient(to right, #25b0a9 50%, #fee603 50%);
    background-position: 0;
    background-size: 200%;
    transition: all 0.4s;
}
.fun-hover:hover {
    background-position: -100%;
}
CSS Highlight on Hover Animation

JAN 20 2018

CHECK OUT REACT FOR BEGINNERS
ARE YOU A SUBLIME TEXT POWER USER?

I LOVE TO LEARN!
I SURE DO LOVE FLEXBOX!

I saw this great little highlight hover animation on Web Designer Depot the other day and I wondered how it worked. Turns out it’s just a few lines of CSS
You have to build in **fluid, flexible type**, and designers want you to implement perfect *modular scale*. The proportions have to stay consistent across screens.
<table>
<thead>
<tr>
<th>Font Size</th>
<th>Line Height</th>
<th>Line Height @ 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>121.5px</td>
<td>7.594em</td>
<td>7.594em @ 16</td>
</tr>
<tr>
<td>81px</td>
<td>5.063em</td>
<td>5.063em @ 16</td>
</tr>
<tr>
<td>54px</td>
<td>3.375em</td>
<td>3.375em @ 16</td>
</tr>
<tr>
<td>36px</td>
<td>2.25em</td>
<td>2.25em @ 16</td>
</tr>
<tr>
<td>24px</td>
<td>1.5em</td>
<td>1.5em @ 16</td>
</tr>
<tr>
<td>16px</td>
<td>1em</td>
<td>1em @ 16</td>
</tr>
</tbody>
</table>

The quick brown fox jumps over the lazy dog
Fluid modular scale headings

These heading scale between a modular scale of 1.067 and 1.333. Resize the window to see the effect.

Fluid modular scale

Fluid modular scale

Fluid modular scale

Fluid modular scale
CSS Architecture

- *Main CSS* contains default type styles:

  /* CSS Reset of your choice */
  body { font-size: 100%; line-height: 1.45em; }

  /* 2:3 Perfect Fifth: 7.111, 10.667, 16 (i), 24, 36, 54 */
  h1 { font-size: 3.375rem }
  h2 { font-size: 2.25rem }
  h3 { font-size: 1.5rem }
  h4 { font-size: 1rem }
  caption { font-size: 0.667rem }
  small { font-size: 0.444rem }
CSS Architecture

/* CSS Reset of your choice */
body { font-size: 100%; line-height: 1.45em; }

/* 2:3 Perfect Fifth: 7.111, 10.667, 16 (i), 24, 36, 54 */
h1 { font-size: 3.375rem }
h2 { font-size: 2.25rem }
h3 { font-size: 1.5rem }
h4 { font-size: 1rem }
caption { font-size: 0.667rem }
small { font-size: 0.444rem }

/* Ideal line length: 66 ch; => max-width: 33em */
article { max-width: 33em; }
:lang(de) article { max-width: 40em; }
p, ul, ol, dl, table { margin-bottom: 1.45rem; }
/* CSS Reset of your choice */
body { font-size: 100%; line-height: 1.45em; }

/* 2:3 Perfect Fifth: 7.111, 10.667, 16 (i), 24, 36, 54 */
h1 { font-size: 54px; font-size: 3.375rem }  
h2 { font-size: 36px; font-size: 2.25rem }  
h3 { font-size: 16px; font-size: 1rem; }  
h4 { font-size: 24px; font-size: 1.5rem }  
caption { font-size: 7px; font-size: 0.667rem }  
small { font-size: 11px; font-size: 0.444rem }

/* Ideal line length: 66 ch; => max-width: 33em */
article { max-width: 33em; }
:lang(de) article { max-width: 40em; }
p, ul, ol, dl, table { margin-bottom: 1.45rem; }
How do you *efficiently* scale up / down any UI component (e.g. a slider or calendar) and keep all the proportions intact—without fiddling with width, height or border-radius manually?

— @simurai
"By sneaking a *Trojan horse* into your components. We use *rem* for components “root” and *em* for sub-parts of the components. Then, by adjusting the *font-size of the root*, we adjust *all* size-related CSS properties of a component at once.

— @simurai
Let me show you in an example: For every CSS property that has a direct impact on the component’s size, you use the **EM** unit.

```css
.Calendar {
    width: 5em;
    height: 2em;
    border-radius: .5em;
    border: 1px solid gold;
}
```

Note that the border is set to **1px** since it should stay always like that, unrelated to size changes.

In some cases you need to override the font-size that comes from the UA style sheet. For example when you use a `<button>` or `<input>` element. You can add a font-size of **100%, 1em** or **inherit** to make it inherit from its parent. Or use something like [normalize.css](http://necolas.github.io/normalize.css) which already takes care of...
Default
(100%/1em)

.Component {
    font-size: 75%;
}

.Component {
    font-size: x-large;
}

Example used: Digit components
rem (root em) units

Type of unit similar to "em", but relative only to the root element, not any parent element. Thus compounding does not occur as it does with "em" units.
With media queries, we can target specific screen *width ranges* and adjust type by just manipulating the font-size *rem* value of the article’s *container*. 
To achieve *fluid typography*, we can combine the `calc()` function in CSS with viewport units (vw/vh/vmin/vmax). But what if you want to apply a *modular scale* to font sizes?
We can get perfectly fluid type with

```html
{ font-size: calc(1em + 1vw); }
```

but it gives us little control over the rate at which viewport units change. Media queries? Well, with them usually there is an annoying “visual” jump between fixed and fluid values.

— Mike Riethmuller
Using `calc` in combination with `vw` and `vh` units for font-size to create text that always fills the viewport. No matter what ratio.
...E.g. if we wanted to choose a font-size of 16px at a screen resolution of 400px and then transition to 24px at a resolution of 800px, we couldn’t do it without a breakpoint.

— Mike Riethmuller
<table>
<thead>
<tr>
<th></th>
<th>1vw</th>
<th>2vw</th>
<th>3vw</th>
<th>4vw</th>
<th>5vw</th>
</tr>
</thead>
<tbody>
<tr>
<td>400px</td>
<td>4px</td>
<td>8px</td>
<td>12px</td>
<td>16px</td>
<td>20px</td>
</tr>
<tr>
<td>500px</td>
<td>5px</td>
<td>10px</td>
<td>15px</td>
<td>20px</td>
<td>25px</td>
</tr>
<tr>
<td>600px</td>
<td>6px</td>
<td>12px</td>
<td>18px</td>
<td>24px</td>
<td>30px</td>
</tr>
<tr>
<td>700px</td>
<td>7px</td>
<td>14px</td>
<td>21px</td>
<td>28px</td>
<td>35px</td>
</tr>
<tr>
<td>800px</td>
<td>8px</td>
<td>16px</td>
<td>24px</td>
<td>32px</td>
<td>40px</td>
</tr>
</tbody>
</table>
font-size: calc(16px + (24 - 16) * (100vw - 400px) / (800 - 400));

min font size

min screen size

max font size - min font size

max screen size - min screen size
You choose the *min* and *max* font-size and the *screen sizes*, over which the font should scale and plug them into the equation. You can use any unit type including ems, rems or px.

— Mike Riethmuller
Fluid modular scale headings

These heading scale between a modular scale of 1.067 and 1.333. Resize the window to see the effect.

Fluid modular scale

Fluid modular scale

Fluid modular scale

Fluid modular scale
# calc() as CSS unit value

Method of allowing calculated values for length units, i.e. width:

calc(100% - 3em)

<table>
<thead>
<tr>
<th>Current aligned</th>
<th>Usage relative</th>
<th>Show all</th>
</tr>
</thead>
<tbody>
<tr>
<td>IE</td>
<td>Edge</td>
<td>Firefox</td>
</tr>
<tr>
<td>8</td>
<td>11</td>
<td>13</td>
</tr>
</tbody>
</table>

Notes

Support can be somewhat emulated in older versions of IE using the non-standard expression() syntax.

Due to the way browsers handle sub-pixel rounding differently, layouts using calc() expressions may have unexpected results.

1 Partial support in Android Browser 4.4 refers to the browser lacking the ability to multiply and divide values.
Truly Fluid Typography With vh And vw Units

By Michael Riethmuller

May 10th, 2016  CSS, Responsive Web Design, Typography

28 Comments  Edit

Embracing fluid typography might be easier than you think. It has wide browser support, is simple to implement and can be achieved without losing control over many important aspects of design.

Unlike responsive typography, which changes only at set breakpoints, fluid typography resizes smoothly to match any device width. It is an intuitive option for a web in which we have a practically infinite number of screen sizes to support. Yet, for some reason, it is still used far less than responsive techniques.

This might be because typography is so deeply rooted in the centuries-old history of typesetting. The concept of having “fluid” anything is often at odds with this tradition. In print, dimensions have always been fixed, but they don’t need to be on the web.
Manually adjusting line-height with media queries for optimum readability across vast number of screen sizes can be very hard. What makes it even harder, is, that instead of the screen width, the line-height should be relative to its container’s width and its font settings in order to achieve proper readability and appropriate spacing.

Thanks to @Wilto, there has been a jQuery plugin called *Molten Leading* around for quite some time already which makes it possible to automate this process and define a minimum width at which the adjustment starts, a maximum element width where it stops, and a minimum and maximum line-height to adjust through.
And here's what a CSS lock looks like in code:

```css
line-height: calc(1.3em + (1.5 - 1.3) * ((100vw - 21em)/(35 - 21)));
```

To understand how the formula works within `calc()`, we're going to work through it backwards.

1. See the very last part? 35-21. That gives us the full range of our paragraph's width. It resolves to 14, because 14em is the difference between our paragraph's width at its most narrow and most wide.

2. To the left of that, we've got 100vw-21em. Because of the way CSS calc works, this resolves to an em-based value—and gives us a numerator to place above the 14em we already figured out. So, for example, let's say the viewport width (100vw) is equivalent to 34em. 34em-21em = 13em. Note that the viewport unit in this step is our secret sauce. The fact that this value can change dynamically with browser window width is what makes a dynamic line-height value possible.

3. So the whole expression to the right of the multiplication sign gets distilled down to this: 13em / 14em, or 0.928571429em. Think of this as how close we are to the "upper gate" of our lock. If it's near zero, we're close to the lower gate. If it's near one, we're close to the upper gate.

4. Moving to the left of the multiplication sign, we compute the difference between our maximum and minimum line heights. 1.5-1.3 = 0.2. This gives us the full range of our fluid line height.

5. Now we multiply the full range of our fluid line height (step 4) by how far along we are toward the upper gate of our lock (step 3): 0.2 * 0.928571429em = 0.185714286em.
And here's what a CSS lock looks like in code:

```css
line-height: calc(1.3em + (1.5 - 1.3) * ((100vw - 21em)/(35 - 21)));
```

To understand how the formula works within `calc()`, we're going to work through it backwards.

1. See the very last part? 35–21. That gives us the **full range of our paragraph's width**. It resolves to 14, because 14em is the difference between our paragraph's width at its most narrow and most wide.

2. To the left of that, we've got 100vw–21em. Because of the way CSS `calc` works, this resolves to an em-based value—and gives us a numerator to place above the 14em we already figured out. So, for example, let's say the viewport width (100vw) is equivalent to 34em. 34em–21em = 13em. **Note that the viewport unit in this step is our secret sauce. The fact that this value can change dynamically with browser window width is what makes a dynamic line-height value possible.**

3. So the whole expression to the right of the multiplication sign gets distilled down to this: 13em / 14em, or 0.928571429em. Think of this as how close we are to the "upper gate" of our lock. If it's near zero, we're close to the lower gate. If it's near one, we're close to the upper gate.
By and by, an old friend of mine, a miner, came down from one of the decayed mining camps of Tuolumne, California, and I went back with him. We lived in a small cabin on a verdant hillside, and there were not five other cabins in view over the wide expanse of hill and forest. Yet a flourishing city of two or three thousand population had occupied this grassy dead solitude during the flush times of twelve or fifteen years before, and where our cabin stood had once been the heart of the teeming hive, the centre of the city. When the mines gave out the town fell into decay, and in a few years wholly disappeared—streets, dwellings, shops, everything—and left no sign. The grassy slopes were as green and smooth and desolate of life as if they had never been disturbed. The mere handful of miners still remaining, had seen the town spring up spread, grow and flourish in its pride; and they had seen it sicken and die, and pass away like a dream. With it their hopes had died, and their rest of life. They had long ago resigned themselves to their exile, and ceased to
The math of CSS locks

September 2016

A CSS lock is a Responsive Web Design technique that lets you transition smoothly between two values, depending on the current viewport size, rather than jump straight from one value to the other.

This concept, and one implementation, were proposed by Tim Brown in Flexible typography with CSS locks.

When I tried wrapping my head around Tim's implementation, and creating variants of it, I had a hard time figuring out what was going on exactly. I did a lot of back-of-the-envelope calculations, and I thought it would be useful to share a mathematical explanation.

I'll describe the technique, its limitations, and the math that make it work. But don't worry about the math: it's basically addition and multiplication, and I've broken down the steps as much as possible—also there will be nice graphs.

Table of contents

1 What's a CSS lock?
   Viewport-relative sizes
n early 2012, I shared a formula for “molten leading”, or fluid line spacing. Experienced typographers know that long lines of text need more line spacing, but line spacing can be tighter for short lines of text. The problem is that on the web, our texts are flexible. So our line spacing needs to flex, too:

Manually adjusting line-height with media queries for optimum readability across vast number of screen sizes can be very hard. What makes it even harder is, that instead of the screen width, the line-height should be relative to its container’s width and its font settings in order to achieve proper readability and appropriate spacing.

Thanks to @WItto, there has been a jQuery plugin called Molten Leading around for quite a while now.

Head of Typography for Adobe Typekit & Adobe Type.
Practicing typography and web design every day. I write, speak, and make tools to share what I learn. I try to be helpful. I love my wife, kids, family, friends, teachers, and dogs. I’m a volunteer firefighter.
We've barely scratched the surface of typography on the web, and the reason why will shock you to your core. (Just kidding about that last bit). The web is so different to print, yet we are stuck in the ways of designing around our knowledge of print media. We need better tools to experiment, design, and better typography that is native to the web medium. Now's the time for us to start to push forwards with the future of typography on the web, and to forge its own language and best practices.
Implement the **baseline rhythm in CSS**. Insert two differently formatted elements next to each other and they’ll seem *out of phase*. How do we fix it?
Quote of the day

Success usually comes to those who are too busy to be looking for it. — J.H. Thoreau

MISAPPLIED CONCEPT

JUST RIGHT
So you want to implement a baseline rhythm in CSS. Insert two differently formatted elements next to each other and they’ll seem out of phase. How do we bring them under control?

— Jan Dudek
Often we define a common *line height* value (or its multiple) that’s used for all elements, including their *paddings* and *margins*, occasionally taking *border widths* into the equation.
What if we *align the baseline* instead? So that all type—regardless of its size—lies on the same *grid line*? We just need to calculate the *offset* and then shift the content by that offset.
Quote of the day

Success usually comes to those who are too busy to be looking for it. —J.H. Thoreau

Last updated: yesterday

MISAPPLIED CONCEPT

JUST RIGHT
By default, browsers center the cap height (the height of a capital letter above the baseline) between grid lines. So we shift it by the half of the difference between line height and cap height.
To determine the *cap height*, we fiddle with *offset values* until the type is properly aligned with the grid.
• vertical-rhythm.scss:

    $line-height: 24px;

    $font-stacks: ( 
        s: $font-stack-text,
        m: $font-stack-text,
        l: $font-stack-display,
        xl: $font-stack-display
    );

    $font-sizes: (s: 13px, m: 15px, l: 19px, xl: 27px);
    $cap-heights: (s: 0.8, m: 0.8, l: 0.68, xl: 0.68);
$\text{line-height: 24px;}$

$\text{font-stacks: (}$
  \text{$s: \text{font-stack-text},}$
  \text{$m: \text{font-stack-text},}$
  \text{$l: \text{font-stack-display},}$
  \text{$xl: \text{font-stack-display}$}$
$\text{) ;}$

$\text{font-sizes: (} s: 13px, m: 15px, l: 19px, xl: 27px\text{);}$

$\text{cap-heights: (} s: 0.8, m: 0.8, l: 0.68, xl: 0.68\text{);}$

@function rhythm-shift($\text{size-name}$) {  
  $\text{font-size: map-get($\text{font-sizes}, \text{size-name});}$  
  $\text{cap-height: map-get($\text{cap-heights}, \text{size-name});}$  
  $\text{offset: ($\text{line-height} - \text{cap-height} * \text{font-size}) / 2;}$  
  \text{return round($\text{offset});$}
}
Now we just need to *apply the offset*, and do so reliably. We can combine *positive* top margin and *negative* bottom margin to make it work.
$offset: rhythm-shift(m);

.rhythm-m {
    margin-top: $offset;
    margin-bottom: -1 * $offset;
}

Collapsing works differently with positive and negative margins:

- **Two positive margins**
  The bigger one wins.

- **Two negative margins**
  The lower (i.e. the more negative) wins.

- **One positive, one negative margin**
  The margins *sum up.*
If an element doesn’t have a `border` nor `padding`, and its first child has a margin, that margin will flow out of the parent. Use `overflow: hidden`.
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque gravida eget massa a tincidunt. Morbi interdum elit at sapien vestibulum sodales.

Morbi efficitur et ipsum sit amet ullamcorper.

Curabitur vel est eget sapien scelerisque efficitur a sit amet risus. Sed scelerisque tortor mi, vel venenatis odio tempor eu. Vestibulum a aliquet nibh.

Phasellus mattis vulputate felis, eget porta lectus maximus ac.

Proin sed mi vitae diam tempor pharetra.
Implementing baseline rhythm in CSS

Written by Jan Dudek

15 June 2016

Vertical rhythm is a typographic concept that’s so often misunderstood by front-end engineers.

By aligning type to a vertical grid, designers can make their work look harmonious and clean. And by having the same visual rhythm implemented correctly, front-end architects can achieve consistent, good-looking results more easily and in shorter time. All that without the need for designers’ input in the process.
Often we want to target a specific child in the DOM, but the parent might have many children. Usually we style all children first and the overwrite the styles.
// instead of putting it on
border-right:1px solid #424242;
&:last-child {
  border-right: 0; // and then taking it off
}

// use CSS :not() to only apply to the elements you want
&:not(:last-child) {
  border-right:1px solid #424242;
}

🔥Protip: Use CSS :not() instead of applying and unapplying borders on navigations. Supported wherever last-child is

7:04 PM - 3 Jun 2015

638
1,056
What if you want all links to have an underline except the ones you specify? Or you want all `<li>`’s in the navigation to have a right border, except the last one. Normally you would use `.last-child` (or extra class) to overwrite a default CSS rule.

— Ire Aderinokun
For example, I may want all links on my site to have an underline, except ones which I specify. Normally, I would write -

```css
a {
  text-decoration: underline;
}

a.no-underline {
  text-decoration: none;
}
```

Doing this means that the links with the class .no-underline have the default styling unnecessarily applied to them. Using the :not selector, I can avoid this extra declaration -

```css
a:not(.no-underline) {
  text-decoration: underline;
}

a.no-underline {
  text-decoration: none;
}
```
On :not and Specificity

The negation pseudo-class, :not, can be incredibly useful. It allows us to target elements based on what attributes they don’t have, rather than what they do. This helps us avoid writing extra, increasingly specific, rules in an attempt to override previous ones.

A common example of this is when we want to apply a style to all list items, except the last one. For example -

```css
/* Without :not */
li { border-right: 1px solid #000; }
li:last-child { border-right: none; }

/* Using :not */
li:not(:last-child) { border-right: 1px solid #000; }
```
Using :not

This effect of :not has made me re-think the way I use it. Even though it feels like we are bypassing the need to write increasingly specific rules to override others, like in the li:last-child example, it seems like :not inadvertently does the same thing.

I will definitely keep using :not because, in many circumstances, it is still the cleaner way to write styles. However, I will use it with a few caveats -

- **Never** use it with IDs, e.g. :not(#bar)
- Restrict using it with generic type selectors, e.g. div:not(.foo)
- Define :not rules earlier in CSS so they can be overridden if necessary
You’ve built an alert message box. To be resilient to failure, how can we make sure that the box will be hidden when there is no content within it?

— Ire Aderinokun
.alert {
    background-color: beige;
    border: 2px solid rgb(150, 150, 150);
    border-radius: 5px;
    padding: 5px 10px;
    display: inline-block;
}

.alert:empty {
    display: none;
}
CSS3 selectors

Advanced element selection using selectors including:
- [foo^="bar"]
- [foo$="bar"]
- [foo="bar"]
- :root
- nth-child()
- nth-last-child()
- nth-of-type
- nth-last-of-type()
- :last-child
- :first-of-type
- last-of-type
- :only-child
- :only-of-type
- empty
- :target
- :enabled
- :disabled
- :checked
- :not()
- ~ (general sibling)
What if you want a tidy grid with fine and consistent line endings? Sometimes you might end up with not enough space to display all content blocks in a row, or not enough items to properly fill a row.

— Patrick Clancey
Using CSS Module Queries with Range Selectors. fig 4

A PEN BY Patrick
A **quantity selector** is a CSS selector that allows styles to be applied to elements based on the number of siblings.
Build a query

Which element will be counted?

ul

Type of query?

At-Least

Amount of items?

# of items to count

Create query

Your Code

Copy and paste the code below into your styles

```
// Build a query on the left
```

Try it out

Your quantity query will be reflected on the items below by a change in colour. Add and remove items to see the styling be applied when the query matches.

Add item | Remove item
li:nth-last-child(6):first-child
• CSS:

```css
li:nth-last-child(6):first-child,
li:nth-last-child(6):first-child ~ li {
  color: green;
}
```
- CSS:
  
  ```css
li:nth-child(n+6) {
    color: green;
  }
  ```
li:nth-last-child(n+6) {
  color: green;
}

7 items

6 items

5 items
 fewer than 6 items

-7
-6
-5
-4

6 or more items

-7
-6

threshold (break point)

li:nth-last-child(n+6):first-child,
li:nth-last-child(n+6):first-child ~ li {
    color: green;
}
Selector support

All of the CSS2.1 and CSS3 selectors used in this article are supported in Internet Explorer 9 and above, including all reasonably recent mobile/handheld stock browsers.

Internet Explorer 8 support is good for most selector types, but technically partial, so you might want to consider a JavaScript polyfill. Alternately, you could pair the selectors for the “safer” of the layout strategies with IE9-specific classes. In the case of the navigation menu, the safer option is the one catering to more items, using `inline-block`. The declaration block would look something like this:

```css
nav li:nth-last-child(n+6),
nav li:nth-last-child(n+6) ~ li,
.lt-ie9 nav li {
    display: inline-block;
/* etc */
}
```
To create a **perfect grid**, we’ll need to define layout for *any* number of items with *specific quantity selectors* within media queries.
• “Mod query selector” in CSS:

```css
li:nth-last-child(3n):first-child,
li:nth-last-child(3n):first-child ~ li {
  /* ... styles for list items in a list divisible by 3 ... */
}
```
— *Select all following siblings (~ li)* which follow after

— *The first child* (first li in the list here), (:first-child) that also is

— *Every third item* starting from the end (:nth-last-child(3n)).

```css
li:nth-last-child(3n):first-child, 
li:nth-last-child(3n):first-child ~ li {
    /* ... styles for list items in a list divisible by 3 ... */
}
```
“Range selector” in CSS:

```css
li:nth-child(n+3):nth-child(-n+5) {
  /* ... styles for list items from 3 to 5 ... */
}
```

— Select all the items up to and including the fifth item, then
— Select all the items from the third item onwards.
<table>
<thead>
<tr>
<th>Product title</th>
<th>Product detail text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product title</td>
<td>Product detail text</td>
</tr>
<tr>
<td>Product title</td>
<td>Product detail text</td>
</tr>
<tr>
<td>Product title</td>
<td>Product detail text</td>
</tr>
<tr>
<td>Product title</td>
<td>Product detail text</td>
</tr>
</tbody>
</table>
We use a **mod query** to check if the number of items is divisible by 3. Then we use a **range selector** to style items differently, e.g. apply one styling to *first three*, another styling to the fourth *through* ninth, and another to 10th *onwards*. Voilà!
“Mod query selector” in CSS:

```css
li:nth-last-child(3n):first-child /* mod query */
~ li:nth-child(n+4):nth-child(-n+6) { /* range selector */
    /* ... styles for 4th to 6th elements, in a list divisible by 3 ... */
}
```
Tidy list using Mod Queries and Range Selectors

This demo is based on my short article about CSS Mod Queries and Range Selectors. The CSS query matches a Mod and styles the remainder accordingly.

The queries will style the items near the top of the list so the last line is always tidy and you’ll never getorphans at the end of the list.

Use the form below to set the mod and add / remove list items. Check the generated CSS to see how the list works.

Select the mod base for your list

Mod 3

Add remove list items

Add Item
Remove Item

The queries generated are as follows:

First two items:
First item is always 100% wide, the second child is 50% if there are only two items
Remainder 0:
As first item is 100%, the second row items have a width of 100%/mod-1
Remainder 1:
No action needed, the default width for all items plus the first item
Remainder 2:
Second row has two 50% width items, third row has width of 100%/mod-1

/* mod 3 */
/* first two, common to all queries */
li:first-child {
  width: 100%;
}
li:nth-child(2):last-child {
  margin-left: 25%;
  width: 50%;
}
/* styles for less than 3 items */
li:nth-child(3):last-child { /* li { width: 5% */
  margin-left: 25%;
  width: 50%;
}
/* mod query 3 */
li { width: 33.333%;
  margin-left: 25%;
  width: 50%;
}
li:nth-last-child(3+2):first-child ~ li:nth-child(n=3) { width: 50%;
  margin-left: 25%;
  width: 50%;
}
li:nth-last-child(3+1):first-child ~ li:nth-child(n=5) { width: 50%;
  margin-left: 25%;
  width: 50%;
}
li:nth-last-child(3+0):first-child ~ li:nth-child(n=2) { width: 50%;
  margin-left: 25%;
  width: 50%;
}
/* first child message */
Using CSS Mod Queries with Range Selectors

by Patrick Clancy • October 11, 2016

Published in CSS

Recently, I was asked to build a simple list that would display in a grid—one that could start with a single element and grow throughout the day, yet always be tidy regardless of the length. So, as you do sometimes when you’re busy with one thing and asked if you can do something completely different, I tried to think of any reason why it couldn’t be done, came up blank, and distractedly said, “Yes.”

At the time, I was working on a London-based news organization’s website. We’d spent the previous year migrating their CMS to the Adobe AEM platform while simultaneously implementing a responsive UI—both big improvements. Since that phase was complete, we were starting to focus on finessing the UI and building new features. The development project was divided into a number of small semi-autonomous teams. My team was focusing on hub pages, and I was leading the UI effort.

Each hub page is essentially a list of lists, simply there to help readers find content that interests them. As you can imagine, a news website is almost exclusively made of content lists! A page full of generic vertical lists would be unhelpful and unappealing; we wanted readers to enjoy browsing the content related to their sphere of interest.
Quantity Queries for CSS

by Heydon Pickering - March 03, 2015
Published in CSS, HTML

Don’t you just hate documentaries that don’t deliver? They have enticing names like In Search of the Giant Squid, and tease you with shots of murky underwater shapes and excited scientists pointing far out to sea. You settle down to watch, eyes narrowed with suspicion, thinking, “I better see some squid or I’m writing an angry letter to the network.”

Sure enough, 90 minutes of interviews with bored-looking fishermen later, the presenter is forced to conclude, “No... no, we didn’t find any big squids. But maybe one day [majestic orchestral flourish].” Great. You wanted Finding Nemo and got Not Finding Nemo instead.

I wouldn’t do that to you, friends. This is your guide to creating style breakpoints for quantities of HTML elements, much as you already do with @media queries for viewport dimensions. I’m not pointing at some blurry specification in the distance or a twinkle in an implementer’s eye. We’re going to do this today, with CSS that’s already
Holy smokes! You are tasked to build a pattern library for your company. You know well where to start, but what’s your strategy to keep it up-to-date long-term?
Pain Points and Bottlenecks

Almost every *mid-size/large company* has similar pain points, issues and concerns:

- *Digital* not properly understood and applied,
- *Subpar workflow* and loose communication,
- *Tech-driven* design decisions made by developers,
- *Legacy* code base, CMS, slow workflow — “watergile”,
- *Inconsistency* reflected in different views,
- *Gap* between screen mock-ups and front-end prototypes.
Pain Points and Bottlenecks

• Almost every *mid-size/large company* has similar pain points, issues and concerns:
  
  • *Design-driven* approach is prioritized by management,
  • *Responsive design* is difficult to estimate and plan for,
  • *Focused* on content first, performance and longevity,
  • *Pattern library* seen as ultimate source of consistency.

• *Goal:* one (responsive) site, serving tailored multi-screen experiences. *Maintainable,* future-proof.
1 Containers

The .container class is used to horizontally constrain content at the highest level. It has an exact width of 1024px. It should be used to wrap all content, ensuring that it stays within the maximum width of the page. Also includes padding on the left and right side to prevent content from resting against the edge of small screens.

In this example, the blue background on .module spans the width of the screen, while the .container inside constrains content to the target width. The .fullwidth class on .module gives the containing element a minimum width equal to our minimum page width. This prevents an issue where the background doesn't extend to the edge of the page when the browser is more narrow than the page and a user scrolls to the right.

```html
<div class="module fullwidth" style="background: blue;">
  <div class="container">
    <p>Content goes here.</p>
  </div>
</div>
```

In this example, .container can be the top level element because there is no background (or border, or other visual treatment) that must span the screen outside of the width of the container.

```html
<div class="container">
  <p>Content without a background goes here.</p>
</div>
```
Success with a pattern library means moving meaningful metrics, such as increasing bookings or decreasing costs.
We collected components in a **master Sketch file**. After a week or two we began to see **huge leaps in productivity** by using the library when iterating on designs...

— Karri Saarinen, AirBnB

http://airbnb.design/co-creating-experiences-with-our-community/
…One day, while putting together a last-minute prototype, our team was able to create nearly **50 screens** within **just a few hours** by using the framework our library provided.

— Karri Saarinen, AirBnB

<table>
<thead>
<tr>
<th>Type</th>
<th>A11y Color</th>
<th>Spacing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title 1</td>
<td>Rauch #FFFB4F</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>3.05/1</td>
<td>tiny</td>
</tr>
<tr>
<td>Title 2</td>
<td>A11y Babu #00A099</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>3.03/1</td>
<td>small</td>
</tr>
<tr>
<td>Title 3</td>
<td>A11y Arches #1D642D</td>
<td>24</td>
</tr>
<tr>
<td>Large</td>
<td></td>
<td>base</td>
</tr>
<tr>
<td>Regular</td>
<td>A11y Hof #40B840</td>
<td>48</td>
</tr>
<tr>
<td>Small</td>
<td>#268E77</td>
<td>large</td>
</tr>
<tr>
<td></td>
<td>4.34/1</td>
<td>x-large</td>
</tr>
</tbody>
</table>
User Marquee
Optional caption


Paragraph two
Description: America’s early beginnings are etched into the earth of Boston, a traditional New England city.
```
var size = window.getComputedStyle(document.body,':after').getPropertyValue('content');
if (size == 'desktop') {
  // Load some more content.
}
```
“Re-usable components can be used in many different but similar ways. It leaves room for interpretation. This opens the door for all kinds of disjointed experiences and makes the system harder to maintain.

— Karri Saarinen, AirBnB

http://airbnb.design/building-a-visual-language/
Fahr nicht nur hin.
Lebe dort.

Video anschauen

Letzte Suchen

San Francisco
2 Gäste

Paris
May 2 - May 16, 1 Gast

Filter
atoms \rightarrow \textit{molecules} \rightarrow \textit{organisms} \rightarrow \textit{templates} \rightarrow \textit{pages}
A design system should not simply be a collection of *UI components* along with some design theory. A library that simply provides a “kit of parts” leaves a lot open to *interpretation*.

— Jeff Crossman, GE

https://medium.com/ge-design/predix-design-system-8236447b0691
Beyond Atomic Design

• Having a shared understanding of *building blocks* helps, but they need *context* to be used effectively.

• *Pattern library isn’t the end game.* It shines when internal teams use it to extend the product.

• *Show examples.* The team should know how to apply patterns in appropriate and meaningful ways.

• The context exists on the most concrete levels of atomic design — *applications and features.*
Asset Performance
Management
Product Applications
Components

Designed to achieve more complex user interactions than are possible with basics alone. Components are formed by using multiple basic elements and principles.
```
JavaScript:
var size = window.getComputedStyle(document.body, ':after').getPropertyValue('content');
if (size == 'desktop') {
// Load some more content.
}
```
GE’s Predix Design System

Our first design system at GE, created in 2012, was similar to many design systems you find on the Internet today. It was designed to appeal to and support a broad audience within GE and contained a large selection of common design patterns (a term I’ll be using to encompass both user
Find the features your team needs. **1:1 code base** mapping. Automated updates. Masterlods don’t scale. Show *context*, interface examples.
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>About</td>
<td>Bootstrap</td>
<td>Foundation</td>
<td>Pattern Primer</td>
<td>Style Guide Boilerplate</td>
<td>Barebones</td>
<td>Typeplate</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>About</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>Layout</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Grid system</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Baseline grid (vertical rhythm)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Containers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Container boxes with headings</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Spacing (common dimensions)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Vertical alignment helpers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Fixed positioning helpers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Float helpers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Arrange/flex-table</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Misc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Colors</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Icons</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Sprites</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Region/culture flags (US, GB, FR, etc.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Visibility classes (based on viewport size)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PAGELAYERS

CONVERTS YOUR WEBSITE INTO PHOTOSHOP LAYERS
Taking Pattern Libraries To The Next Level

By Vitaly Friedman

Design Systems, Pattern Libraries, Style Guides

No thorough conversation about the front end today can end without mention of **pattern libraries**. Sometimes a pattern library appears in the form of a **living style guide**, or as a design system, or as the outcome of an **atomic design process**, or as an all-knowing user interface framework. In all of these cases, designers and developers seek the right strategy to approach the complexity of the web with a modular, components-based approach.

However, finding the right way to architect a **lasting pattern library** and to integrate it into an existing workflow seems to be a challenging task and one that most design and development teams eventually give up on. In this article, I’d love to highlight some practical techniques and strategies to
By default, broken images look pretty unspectacular. Is there any way to improve the experience by changing the styling if images are actually broken?
We're sorry, the image below is broken :(

![Kanye Laughing](http://bitsofco.de/broken.jpg)
The `<img>` element is a replaced element. This is an element “whose appearance and dimensions are defined by an external resource. Pseudo-elements typically shouldn’t work with it.
We're sorry, the image below is broken :(  

![Kanye Laughing](http://bitsotoo.de/broken.jpg)
Kanye Laughing
§§ Broken Image of Kanye Laughing
```css
img {
  /* Same as first example */
  min-height: 50px;
}

img:before {
  content: " ";
  display: block;

  position: absolute;
  top: -10px;
  left: 0;
  height: calc(100% + 10px);
  width: 100%;
  background-color: rgb(230, 230, 230);
  border: 2px dotted rgb(200, 200, 200);
  border-radius: 5px;
}
```
```css
img::after {
  content: "\f127" " Broken Image of " attr(alt);
  display: block;
  font-size: 16px;
  font-style: normal;
  font-family: FontAwesome;
  color: rgb(100, 100, 100);

  position: absolute;
  top: 5px;
  left: 0;
  width: 100%;
  text-align: center;
}
```
<table>
<thead>
<tr>
<th>Browser</th>
<th>Alt Text</th>
<th>:before</th>
<th>:after</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chrome (Desktop and Android)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Firefox (Desktop and Android)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Opera (Desktop)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Opera Mini</td>
<td>✓ **</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Safari (Desktop and iOS)</td>
<td>✓ *</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>iOS Webview (Chrome, Firefox, others)</td>
<td>✓ *</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

* The alt text will only show if the width of the image is large enough to accommodate it. If no width is specified for the img, the alt text may not be displayed at all.

** Font styling not applied
Styling Broken Images

Mar 8th, 2016  •  CSS

Broken images are ugly.

This image is broken! Ugly, isn’t it?

But they don’t always have to be. We can use CSS to apply styles to the `<img>` element to provide a better experience than the default.

Two Facts About The `<img>` Element

To understand how we can style broken images, there are two facts about the way the `<img>` element behaves that we need to understand first.

1. We can apply regular typography-related styling to the `<img>` element. These styles will be applied to the alternative text, if it is displayed, and will not affect the working image.

2. The `<img>` element is a replaced element. This is an element “whose appearance and

What if you wanted the color of the SVG icon to **inherit** the color property of a button in which it resides? Can we use CSS alone (no SASS/LESS) to establish this *relationship*?
This is a fancy link styling.
Klaipėda is a city in Lithuania situated at the mouth of the Danė River where it flows into the Baltic Sea. It is the third largest city in Lithuania and the capital of Klaipėda County. The city has a complex recorded history, partially due to the combined regional importance of the Port of Klaipėda, a usually ice-free port on the Baltic Sea, and the Akmena – Danė River.

It has been controlled by the Teutonic Knights, the Duchy of Prussia, the Kingdom of Prussia, the German Empire, the Entente States immediately after World War I, Lithuania as a result of the 1923 Klaipėda Revolt, and the Third Reich following the 1939 German ultimatum to Lithuania. The city was incorporated into Lithuania during its tenure as a Soviet Socialist Republic and has remained within Lithuania following its re-establishment as an independent state.
This is my favorite. Take a very common example on the Web — a button with SVG icon and a title in it. I have these here on my website too:

Of course, you are a very responsible web designer, and you style :hover, :focus, :active states of the button for a better interaction with a user. This is how your code usually looks like:
Of course, you are a very responsible web designer, and you style `:hover`, `:focus`, and `:active` states of the button for a better interaction with a user. This is how your code usually looks like:

```css
.button {
  color: #000;
  border: 2px solid #000;
}
.button:hover, 
.button:focus {
  color: #333;
  border-color: #333;
}
.button:active {
  color: #666;
  border-color: #666;
}
.button svg {
  fill: #000;
}
.button:hover svg, 
.button:focus svg {
  fill: #333;
}
.button:active svg {
  fill: #666;
}
```
Currently I am writing a front-end code for client's e-commerce website which has a few different button designs. Moreover, there are anchors that have `:visited` state styled in addition. And there are many more similar SVG usage cases (toolbars, etc.) where SVG has to have the color of the text. `currentColor` helps to reduce the code twice:

```css
/* put this in your reset-normalize-defaults.css file */
.svg
{
    fill: currentColor;
}

/* now you don't have to style SVG and border-color at all */
.button
{
    color: $000;
    border: 2px solid currentColor;
}
.button:hover, .button:focus
{
    color: $333;
}
.button:active
{
    color: $666;
}
```
Keeping CSS short with `currentColor`

27 NOV

Turns out `currentColor` has been here for quite some time now, but I heard about it only a few months ago when I read Dudley Storey’s [post](#). He states that it is supported very well across the browsers (IE9+). This was enough for me to start using it in production. I was quite surprised how useful the keyword is: it helps to keep CSS code shorter and smarter.

Before diving into practical usage examples, here is a short theory course. This is how MDN describes `currentColor`:

> The `currentColor` keyword represents the calculated value of the element’s `color` property. It allows to make the color properties inherited by properties or child’s element properties that do not inherit it by default.

**SVG**

This is my favorite. Take a very common example on the Web — a button with SVG icon and a title in it. I have these here on my website too:
# CSS currentColor value

A CSS value that will apply the existing `color` value to other properties like `background-color`, etc.

<table>
<thead>
<tr>
<th></th>
<th>IE</th>
<th>Firefox</th>
<th>Chrome</th>
<th>Safari</th>
<th>Opera</th>
<th>iOS Safari</th>
<th>Opera Mini</th>
<th>Android Browser</th>
<th>Chrome for Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current aligned</td>
<td>8</td>
<td>35</td>
<td>40</td>
<td>7.1</td>
<td>7.1</td>
<td>7.1</td>
<td>8</td>
<td>37</td>
<td>40</td>
</tr>
<tr>
<td>Usage relative</td>
<td>10</td>
<td>40</td>
<td>41</td>
<td>8</td>
<td>27</td>
<td>8.1</td>
<td>8</td>
<td>37</td>
<td>40</td>
</tr>
<tr>
<td>Show all</td>
<td>11</td>
<td>36</td>
<td>41</td>
<td>8</td>
<td>27</td>
<td>8.1</td>
<td>8</td>
<td>37</td>
<td>40</td>
</tr>
</tbody>
</table>

**Notes**

No notes

**Known issues (0)**

**Resources (3)**

**Feedback**
LEVEL 6
OUTBREAK
What if you want to use a full-width element in a **fixed-width container**? E.g. when you want some content to extend *beyond the boundaries* of the container?
**HTML:**

```html
<div class="u-containProse">
  <p>...</p>
  <p>...</p>
</div>
```

**CSS:**

```css
.u-containProse {
  margin: 0 auto;
  max-width: 40em;
}
```
• HTML:

```html
<div class="u-containProse">
  <p>...</p>
</div>

<img src="..." alt="..." />

<div class="u-containProse">
  <p>...</p>
</div>
```

• CSS:

```css
.u-containProse {
  margin: 0 auto;
  max-width: 40em;
}
```
To *release* our child element from its container, we need to know *how much space* there is between the container edge and the viewport edge.
What’s this space exactly? Well, we just need to subtract half the container width from half the viewport width. `calc()` to the rescue!
**HTML:**

```html
<div class="u-containProse">
  <p>...</p>
  <img class="u-release" src="..." />
  <p>...</p>
</div>
```

**CSS:**

```css
.u-release {
  margin-left: calc(-50vw + 50%);
  margin-right: calc(-50vw + 50%);
}
```
When the height or width of the initial containing block is changed, they are *scaled* accordingly. Note that the initial containing block’s size is *affected* by the presence of scrollbars on the viewport.
• HTML:

```html
<div class="u-containProse">
  <p>...</p>
  <img class="u-release" src="..." />
  <p>...</p>
</div>
```

• CSS:

```css
.u-release {
  margin-left: calc(-50vw + 50%);
  margin-right: calc(-50vw + 50%);
}
```
**HTML:**

```html
<div class="u-containProse">
  <p>...</p>
  <img class="u-release" src="..." />
  <p>...</p>
</div>
```

**CSS:**

```css
.u-release {
  margin-left: calc(-50vw + 50%);
  margin-right: calc(-50vw + 50%);
}

html, body {
  overflow-x: hidden;
}
```
We push the container to the exact *middle* of the browser window with *left: 50%*, then pull it back to the left edge with -50vw margin (h/t Sven Wolfermann).

CSS:
```
.u-release {
  width: 100vw;
  position: relative;
  left: 50%;
  right: 50%;
  margin-left: -50vw;
  margin-right: -50vw;
}
```
Breaking Out With Viewport Units and Calc

Written by Tyler Sticka on May 26, 2016

While iterating on a new article layout for the impending Cloud Four redesign, I encountered an old CSS layout problem.

For long-form content, it's usually a good idea to limit line lengths for readability. The most straightforward way to do that is to wrap the post content in a containing element:

```
.u-containProse {
  max-width: 40em;
  margin-left: auto;
  margin-right: auto;
}
```
LEVEL 7
PAYLOAD
Images make up a large portion of bandwidth payload. Is there any way to optimize images beyond good ol’ image optimization? What if a hero image has to render fast, e.g. on landing pages?
• The original photo has 1600px width, **971 Kb**. Quality 60 brings the size down to **213 Kb**.
• Blurring unimportant parts of the photo brings the size down to 147 Kb.
Sequential JPEG  Progressive JPEG

Images taken from http://www.pixeltech.net/article/1374757887-use-progressive-jpeg-to-improve-user-experience
Default Scan Levels
# Initial DC scan for Y,Cb,Cr (lowest bit not sent)
0,1,2: 0-0, 0, 1;

# First AC scan: send first 5 Y AC coefficients, minus 2 lowest bits:
0: 1-5, 0, 2;

# Send all Cr,Cb AC coefficients, minus lowest bit:
# (chroma data is usually too small to be worth subdividing further;
# but note we send Cr first since eye is least sensitive to Cb)
2: 1-63, 0, 0;
1: 1-63, 0, 0;

# Send remaining Y AC coefficients, minus 2 lowest bits:
0: 6-63, 0, 2;

# Send next-to-lowest bit of all Y AC coefficients:
0: 1-63, 2, 1;

# At this point we've sent all but the lowest bit of all coefficients.
# Send lowest bit of DC coefficients
0,1,2: 0-0, 1, 0;

# Send lowest bit of AC coefficients
2: 1-63, 1, 0;
1: 1-63, 1, 0;

# Y AC lowest bit scan is last; it's usually the largest scan
0: 1-63, 1, 0;
# Initial DC scan for Y,Cb,Cr (lowest bit not sent)

```
0,1,2: 0-0, 0, 1;
```

# First AC scan: send first 5 Y AC coefficients, minus 2 lowest bits:

```
0: 1-5, 0, 2;
```

# Send all Cr,Cb AC coefficients, minus lowest bit:

```
# (chroma data is usually too small to be worth subdividing further;
# but note we send Cr first since eye is least sensitive to Cb)
```

```
0b: 1-63, 0, 0;
0b: 1-63, 0, 0;
```

# Send remaining Y AC coefficients, minus 2 lowest bits:

```
0b: 5-63, 0, 2;
```

# Send next-to-lowest bit of all Y AC coefficients:

```
0b: 1-63, 2, 1;
```

# At this point we've sent all but the lowest bit of all coefficients.

# Send lowest bit of DC coefficients

```
0,1,2: 0-0, 1, 0;
```

# Send lowest bit of AC coefficients

```
0b: 1-63, 1, 0;
0b: 1-63, 1, 0;
```

# Y AC lowest bit scan is last; it's usually the largest scan

```
0b: 1-63, 1, 0;
```
1st Scan Layer Has Small Byte Size

Ships Fast
&
Shows Soon
# Interleaved DC scan for Y, Cb, Cr:

0, 1, 2: 0-0, 0, 1;

initial DC for All channels

# AC scans:

0: 1-27, 0, 0;

Half of all brighter values

2: 1-63, 0, 0;

1: 1-63, 0, 0;

All remaining color channel values

# Remaining Y coefficients

5: 28-63, 0, 0;

2nd half of brightness channel
### Progressive JPEGs via HTTP2

<table>
<thead>
<tr>
<th>Load Time</th>
<th>First Byte</th>
<th>Start Render</th>
<th>Visually Complete</th>
<th>Speed Index</th>
<th>DOM Elements</th>
<th>Result (error code)</th>
<th>Time</th>
<th>Requests</th>
<th>Bytes In</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.180s</td>
<td>0.118s</td>
<td>0.436s</td>
<td>3.500s</td>
<td>1537</td>
<td>75</td>
<td>0</td>
<td>3.180s</td>
<td>21</td>
<td>1,741 KB</td>
</tr>
</tbody>
</table>

**RUM First Paint**
- domContentLoaded: 0.822s
- loadEvent: 3.468s - 3.470s (0.002s)

### Optimized Progressive JPEGs via HTTP2

<table>
<thead>
<tr>
<th>Load Time</th>
<th>First Byte</th>
<th>Start Render</th>
<th>Visually Complete</th>
<th>Speed Index</th>
<th>DOM Elements</th>
<th>Result (error code)</th>
<th>Time</th>
<th>Requests</th>
<th>Bytes In</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.486s</td>
<td>0.120s</td>
<td>0.471s</td>
<td>3.800s</td>
<td>1445</td>
<td>75</td>
<td>0</td>
<td>3.486s</td>
<td>21</td>
<td>1,822 KB</td>
</tr>
</tbody>
</table>

**RUM First Paint**
- domContentLoaded: 0.735s
- loadEvent: 3.766s - 3.768s (0.002s)
A Bash script to automate adaptive JPEG compression using common CLI tools

Adept - the adaptive JPG Compressor
Improved JPEG encoder.

- 3,486 commits
- 8 branches
- 7 releases
- 24 contributors

<table>
<thead>
<tr>
<th>Branch</th>
<th>New pull request</th>
</tr>
</thead>
<tbody>
<tr>
<td>master</td>
<td></td>
</tr>
</tbody>
</table>

- pornei Merge pull request #207 from mozilla/jpeg-yuv-classup

<table>
<thead>
<tr>
<th>File</th>
<th>Commit</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmake_scripts</td>
<td>e4e891a</td>
<td>Merge pull request #207 from mozilla/jpeg-yuv-classup</td>
</tr>
<tr>
<td>doohi</td>
<td>5 months ago</td>
<td>Win: Enable testing cross-compiled builds</td>
</tr>
<tr>
<td>java</td>
<td>4 months ago</td>
<td>Bump TurboJPEG C API revision to 1.5</td>
</tr>
<tr>
<td>moS</td>
<td>2 months ago</td>
<td>Merge remote-tracking branch 'libjpeg-turbo/master' into libjpeg-turbo</td>
</tr>
<tr>
<td>release</td>
<td>2 months ago</td>
<td>Merge remote-tracking branch 'libjpeg-turbo/master' into libjpeg-turbo</td>
</tr>
<tr>
<td>sharedlib</td>
<td>2 years ago</td>
<td>Merge libjpeg-turbo r1390</td>
</tr>
<tr>
<td>simd</td>
<td>2 years ago</td>
<td>Merge remote-tracking branch 'libjpeg-turbo/master' into libjpeg-turbo</td>
</tr>
<tr>
<td>testimages</td>
<td>2 years ago</td>
<td>12-bit JPEG support</td>
</tr>
<tr>
<td>win</td>
<td>2 years ago</td>
<td>Merge remote-tracking branch 'libjpeg-turbo/master' into libjpeg-turbo</td>
</tr>
<tr>
<td>.gitauthors</td>
<td>2 years ago</td>
<td>Script for git-svn reinitialization</td>
</tr>
</tbody>
</table>
What if you have a large photo that requires a transparent shadow? PNG is too large in file size, and JPEG isn’t good enough in quality. 

*Trick:* create a regular non-transparent JPG and an 8-bit PNG (alpha mask) and load both images *inside an SVG container.*
• hero-image.svg:

```xml
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     viewBox="0 0 560 1388">
  <defs>
    <mask id="canTopMask">
      <image width="560" height="1388" xlink:href="can-top-alpha.png"/>
    </mask>
  </defs>
  <image mask="url(#canTopMask)" id="canTop"
        width="560" height="1388"
        xlink:href="can-top.jpg"></image>
</svg>
```
• **hero-image.svg:**

```html
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     viewBox="0 0 560 1388">
    <defs>
        <mask id="canTopMask">
            <image width="560" height="1388" xlink:href="can-top-alpha.png"/>
        </mask>
    </defs>
    <image mask="url(#canTopMask)" id="canTop" width="560" height="1388"
           xlink:href="can-top.jpg"/>
</svg>
```

• **HTML/CSS:**

```html
<img src="hero-image.svg" />
```

```css
background: url("hero-image.svg")
```
September 7, 2014

USING SVG TO SHRINK YOUR PNGS

Wouldn’t it be great if you could get the compression of a JPEG and keep the transparency of a PNG? Well, you can, sort of. Here’s a little trick that I discovered while working on the new Sapporo Beer website.

Notice how the beer can on the Sapporo website has a transparent area (it’s hard to notice but there’s a video playing behind it). As a PNG, the beer can would weigh a ton. But if you use SVG, you can shrink it down to something really small.
Applying Alpha Channels to JPGs

In this article I will show how to combine the advantages of JPG's high compression rate with the flexibility of having alpha channels like we use them in PNG. Open the demo to see this technique in action, a semi-transparent JPG in front of HTML text:

🔗 Demo of JPG with alpha channel.

As all modern browsers support images with alpha channels, it has become fairly easy for designers to place semi-transparent images in front of irregular backgrounds.

The weapon of choice will be PNG as it is the only widespread...
JPG+PNG to SVG Mask

Combine the transparency of a PNG with the compression of a JPEG. Based on the idea from Using SVG to Shrink Your PNGs, but adapted to use data URIs instead of external images. Include on your page as inline SVG, using an `<img src="image.svg" />` tag, or as a background image.

Tested in the latest versions of Chrome, Firefox and Safari. This SVG technique's compatibility via an `<img />` tag or as a background image may not be perfect. See this pen to test on your browser. Inline seems to be the best option for compatibility, in which case you should use external assets so that they can be cached. Please fork or comment to improve.

To get started, upload two images:

- One as your primary image, named whatever (Try this one:)
- One as a mask (a black and white PNG is best, just like this:) With `-mask` or `-alpha` in the filename.

Upload:

Images: Choose Files No file chosen

Make sure the mask has `~mask` or `~alpha` in the filename.

Example:
We want nice type, but *performance* matters, too. You either rely on Typekit/Google Fonts or self-host the fonts. What is your strategy for *loading web fonts*?
Declaring `@font-face`

- We can use **bulletproof @font-face syntax** to avoid common traps along the way:

  - **CSS:**
    ```css
    @font-face {
      font-family: 'Elena Regular';
      src: url('elena.eot?#iefix') format('embedded-opentype'),
          url('elena.woff2') format('woff2'),
          url('elena.woff') format('woff'),
          url('elena.otf') format('opentype');
    }
    ```
Declaring @font-face

- If you want only smart browsers (IE9+) to download fonts, declaration can be shorter:

CSS:

```css
@font-face {
  font-family: 'Elena Regular';
  src: url('elena.woff2') format('woff2'),
      url('elena.woff') format('woff'),
      url('elena.otf') format('opentype');
}
```
• CSS:

```css
@font-face {
  font-family: 'Elena Regular';
  src: url('elena.woff2') format('woff2'),
       url('elena.woff') format('woff'),
       url('elena.otf') format('opentype');
}
```

• When a font family name is used in CSS, browsers match it against all `@font-face` rules, download web fonts, display content.
• When a font family name is used in CSS, browsers match it against all \texttt{@font-face} rules, download web fonts, display content.

• CSS:

```css
body {
  font-family: 'Elena Regular', AvenirNext, Avenir, /* iOS */
  'Roboto Slab', 'Droid Serif', /* Android */
  'Segoe UI', /* Microsoft */
  Georgia, 'Times New Roman', serif; /* Fallback */
}
```
• CSS:

```css
body {
    font-family: 'Elena Regular', AvenirNext, Avenir, /* iOS */
    'Roboto Slab', 'Droid Serif', /* Android */
    'Segoe UI', /* Microsoft */
    Georgia, 'Times New Roman', serif; /* Fallback */
}
```

• HTML:

```html
<link href='http://fonts.googleapis.com/css?family=Skolar_Reg' rel='stylesheet' type='text/css'>

<script type="text/javascript" src="//use.typekit.net/tbb3uid.js"></script>
<script type="text/javascript">
    try{Typekit.load();}catch(e){}
</script>
```
• Once DOM and CSSOM are constructed, if @font-face matches, a font will be required.

• If fonts aren’t cached yet, they will be requested, downloaded and applied, deferring rendering.
text-overflow: ellipsis

overflow: hidden
- **FOIT (Flash Of Invisible Text):** no content displayed until the font becomes available.

- **FOUT (Flash Of Unstyled Text):** show content in fallback fonts first, then switch to web fonts.
Async Data URI Stylesheet

• To eliminate FOIT, we display fallback right away, and load web fonts async with `loadCSS`.
  
  • Easy to group requests into a single repaint,
  • Has a noticeable short FOIT during parsing,
  • How to choose a format to load? JS loader needed.

• **Verdict:** bare minimum for the web font loading strategy today. Self-hosting required.
CSS Font Loading API

- Native browser API à la Web Font Loader, with a `FontFace` object representing `@font-face` rules.

- JavaScript:

  ```javascript
  var elena_reg = new FontFace(
    'Elena Regular',
    'url(elena_reg.woff) format("woff"), ' +
    'url(elena_reg.otf) format("otf"),
    { weight: 'regular', unicodeRange: 'U+0-7ff' }
  );
  ```
• **JavaScript:**

```javascript
var elena_reg = new FontFace(
    'Elena Regular',
    'url(elena_reg.woff) format("woff"),'
    +
    'url(elena_reg.otf) format("otf")',
    { weight: 'regular', unicodeRange: 'U+0-7ff' }
);
```

• **JavaScript:**

```javascript
document.fonts.load('1em elena_reg')
  .then(function() {
    var docEl = document.documentElement;
    docEl.className += ' elena_reg-loaded';
  }).catch(function () { 
    var docEl = document.documentElement;
    docEl.className += ' elena_reg-failed';
  });
```
• **JavaScript:**

```javascript
document.fonts.load('1em elena_reg')
  .then(function() {
    var docEl = document.documentElement;
    docEl.className += ' elena_reg-loaded';
  }).catch(function () {
    var docEl = document.documentElement;
    docEl.className += ' elena_reg-failed';
  });
```

• **CSS:**

```css
.elena_reg-loaded h1 {
  font-family: "Elena Regular";
}
```
• JavaScript:

```javascript
document.fonts.load('1em elena_reg').then(function() {
    var docEl = document.documentElement;
    docEl.className += ' elena_reg-loaded';
}).catch(function() {
    var docEl = document.documentElement;
    docEl.className += ' elena_reg-failed';
});
```

• CSS:

```css
.elena_reg-loaded h1 {
    font-family: "Elena Regular";
    font-rendering: "block 0s swap infinite"; // FOUT
    // font-rendering: "block 3s swap infinite"; // FOIT
}
```
• **JavaScript:**

```javascript
document.fonts.load('1em elena_reg').then(function() {
  var docEl = document.documentElement;
  docEl.className += ' elena_reg-loaded';
}).catch(function () {
  var docEl = document.documentElement;
  docEl.className += ' elena_reg-failed';
});
```

• **CSS:**

```css
.elena_reg-loaded h1 {
  font-family: "Elena Regular";
  // font-rendering: "block 0s swap infinite"; // FOUT
  font-rendering: "block 3s swap 3s"; // FOIT, at most 3sec
}
```
CSS Font Loading

This CSS module defines a scripting interface to font faces in CSS, allowing font faces to be easily created and loaded from script. It also provides methods to track the loading status of an individual font, or of all the fonts on an entire page.

<table>
<thead>
<tr>
<th>Current aligned</th>
<th>Usage relative</th>
<th>Show all</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Browser</th>
<th>Support (version)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IE</td>
<td>8</td>
</tr>
<tr>
<td>Edge</td>
<td>13</td>
</tr>
<tr>
<td>Firefox</td>
<td>47</td>
</tr>
<tr>
<td>Chrome</td>
<td>49</td>
</tr>
<tr>
<td>Safari</td>
<td>51</td>
</tr>
<tr>
<td>Opera</td>
<td>9.1</td>
</tr>
<tr>
<td>IOS Safari</td>
<td>39</td>
</tr>
<tr>
<td>Opera Mini</td>
<td>9.3</td>
</tr>
<tr>
<td>Android Browser</td>
<td>4.4</td>
</tr>
<tr>
<td>Chrome for Android</td>
<td>4.4, 4.4, 4.4</td>
</tr>
</tbody>
</table>

Notes:

1. Can be enabled in Firefox using the `layout.css.font-loading-api.enabled` flag. Enabled by default in Firefox 41. See this bug.
Better @font-face with Font Load Events

@font-face is an established staple in the diet of almost half of the web. According to the HTTP Archive, 47% of web sites make a request for at least one custom web font. What does this mean for a casual browser of the web? In this article, I make the argument that current implementations of @font-face are actually harmful to the performance and usability of the web. These problems are exacerbated by the fact that developers have started using @font-face for two completely different use cases: content fonts and icon fonts, which should be handled differently. But there is hope. We can make small changes to how these fonts load to mitigate those drawbacks and make the web work better for everyone.

First—let’s discuss what @font-face gets right.

Initiating a Font Download

What happens when you slap a fancy new @font-face custom web font into your CSS? As it turns out—not much. Just including a @font-face block doesn’t actually initiate a download of the remote font file from the server in almost all browsers (except IE8).

/* Does not download */
@font-face {

**Font Load Events**

- Use the *CSS Font Loading API* with a polyfill to apply web font only after it has loaded successfully.
  - Toggle a class on `<html>`; with Sass/LESS mixins,
  - Optimize for *repeat* views with sessionStorage,
  - Easy to implement with 3rd-party hosts,
  - Requires strict control of CSS; a single use of a web font font-family will trigger a FOIT.

- **Verdict**: good option for web font loading, to integrate with 3rd-party hosting providers.
Flash of Faux Text

- When using multiple weights, we split web fonts into groups: Roman / *Faux* content.
  - *Two-stage render*: Roman first and rest later,
  - Optimize for *repeat* views with `sessionStorage`,
  - Font synthesis is a big drawback.

- **Verdict**: good option for great performance, but font synthesis might produce awkward results.
Critical FOFT

- When using multiple weights, we split web fonts into groups: Roman / *Faux* content.
  - *Two-stage render*: Roman first and rest later,
  - Subset fonts to *minimum* (A–Z, 0–9, punctuation),
  - Optimize for *repeat* views with sessionStorage,
  - Font synthesis is a big drawback.
  - Subset is duplicated in the full Roman font.
  - Licensing issues: requires subsetting.

- *Verdict*: good option for great performance, but font synthesis might produce awkward results.
Critical FOFT With Data URI

• Instead of loading via a JavaScript API, we *inline* the web font directly in the markup.

  • *Two-stage render*: Roman first and rest later,
  • Subset fonts to *minimum* (A–Z, 0–9, punctuation),
  • Load the subsetted font (Roman) *first* inline,
  • Load full fonts with all weights and styles async,
  • Use *sessionStorage* for return visits,
  • Requires self-hosting; data URI blocks rendering.

• *Verdict*: the fastest web font loading strategy as of today. Eliminates FOIT and greatly reduces FOUT.
A COMPREHENSIVE GUIDE TO FONT LOADING STRATEGIES

— 12 July 2016 — Read this in about 20 minutes.

This guide is not intended for use with font icons, which have different loading priorities and use cases. Also, SVG is probably a better long term choice.

JUMP TO:

- Unceremonious @font-face
- font-display
- preload
- Don't use web fonts
- Inline Data URI
- Asynchronous Data URI
- FOUT with a Class
- FOIT, or FOUT with Two Stage Render
Instead of using sessionStorage, we *inline* the web font in the markup *and* use Service Workers cache.

- *Two-stage render*: Roman first and rest later,
- Subset fonts to *minimum* (A–Z, 0–9, *punctuation*),
- Load the subsetted font (Roman) *first* inline,
- Load full fonts with all weights and styles async,
- Use *Service Workers* for return visits,
- Requires self-hosting/HTTPS; data URI blocks rendering.

*Verdict*: the fastest web font loading strategy as of today. Eliminates FOIT and greatly reduces FOUT.
• When a font family name is used in **CSS**, browsers match it against all `@font-face` rules, download web fonts, display content.

• **CSS:**

```css
body {
  font-family: 'Elena Regular', AvenirNext, Avenir, /* iOS */
  'Roboto Slab', 'Droid Serif', /* Android */
  'Segoe UI', /* Microsoft */
  Georgia, 'Times New Roman', serif; /* Fallback */
}
```
CSS:

body {
    font-family: 'Elena Regular',
AvenirNext, Avenir, /* iOS */
'Roboto Slab', 'Droid Serif', /* Android */
'Segoe UI', /* Microsoft */
Georgia, 'Times New Roman', serif; /* Fallback */
}
<table>
<thead>
<tr>
<th>Font</th>
<th>Device Targeted</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-apple-system (San Francisco)</code></td>
<td>iOS Safari, macOS Safari, macOS Firefox</td>
</tr>
<tr>
<td>BlinkMacSystemFont (San Francisco)</td>
<td>macOS Chrome</td>
</tr>
<tr>
<td>Sego UI</td>
<td>Windows</td>
</tr>
<tr>
<td>Roboto</td>
<td>Android, Chrome OS</td>
</tr>
<tr>
<td>Oxygen / Oxygen-Sans</td>
<td>KDE</td>
</tr>
<tr>
<td>Fira Sans</td>
<td>Firefox OS</td>
</tr>
<tr>
<td>Droid Sans</td>
<td>Older versions of Android</td>
</tr>
<tr>
<td>Ubuntu</td>
<td>Ubuntu</td>
</tr>
<tr>
<td>Cantarell</td>
<td>GNOME</td>
</tr>
<tr>
<td>Helvetica Neue</td>
<td>macOS versions &lt; 10.11</td>
</tr>
<tr>
<td>Arial</td>
<td>Any</td>
</tr>
<tr>
<td>sans-serif</td>
<td>Any</td>
</tr>
</tbody>
</table>
CSS:

```css
body {
  font-family: 'Elena Regular', /* Web font */
  AvenirNext, Avenir, /* iOS */
  -apple-system, BlinkMacSystemFont, /* macOS San Francisco */
  Roboto Slab', 'Droid Serif', /* Android */
  'Segoe UI', /* Microsoft */
  Oxygen-Sans, /* KDE */
  Ubuntu, /* Ubuntu */
  Cantarell, /* GNOME */
  Georgia, 'Times New Roman', serif; /* Fallback */
}
```
• CSS:

```css
.lowBattery {
  font-family: /* 'Elena Regular' */ /* Web font */
  AvenirNext, Avenir, /* iOS */
  -apple-system, BlinkMacSystemFont, /* macOS San Francisco */
  Roboto Slab', 'Droid Serif', /* Android */
  'Segoe UI', /* Microsoft */
  Oxygen-Sans, /* KDE */
  Ubuntu, /* Ubuntu */
  Cantarell, /* GNOME */
  Georgia, 'Times New Roman', serif; /* Fallback */
}
```
Battery Status API

In this example, we watch for changes both to the charging status (whether or not we're plugged in and charging) and for changes to the battery level and timing. This is done by listening for the chargingchange, levelchange, chargingtimechange, dischargingtimechange events.

```javascript
navigator.getBattery().then(function(battery) {
    function updateAllBatteryInfo() {
        updateChargeInfo();
        updateLevelInfo();
        updateChargingInfo();
        updateDischargingInfo();
    }
    updateAllBatteryInfo();

    battery.addEventListener('chargingchange', function() {
        updateChargeInfo();
    });

    function updateChargeInfo() {
        console.log("Battery charging? " + (battery.charging ? "Yes" : "No"));
    }

    battery.addEventListener('levelchange', function() {
        updateLevelInfo();
    });

    function updateLevelInfo() {
        console.log("Battery level: ");
    }
});
```
What if you have to translate an interface into many languages? The length of words is unpredictable. How do you manage it across devices?
Rindfleischetikettierungsüberwachungsaufgabenübertragungsgesetz

The word - which refers to the "law for the delegation of monitoring beef labelling", has been repealed by a regional parliament after the EU lifted a recommendation to carry out BSE tests on healthy cattle. German is famous for its compound nouns, which frequently become so cumbersome they have to be reduced to abbreviations. The beef labelling law, introduced in 1999 to protect consumers from BSE, was commonly transcribed as the "RkReUAUG", but even everyday words are shortened to initials so Lastkraftwagen - lorry - becomes Lkw.
Rindfleischetikettierungsüberwachungsaufgabenübertragungsge-setz

The word - which refers to the "law for the delegation of monitoring beef labelling", has been repealed by a regional parliament after the EU lifted a recommendation to carry out BSE tests on healthy cattle. German is famous for its compound nouns, which frequently become so cumbersome they have to be reduced to abbreviations. The beef labelling law, introduced in 1999 to protect consumers from BSE, was commonly transcribed as the "RkReUAÜG", but even everyday words are shortened to initials so Lastkraftwagen - lorry - becomes Lkw.
The word - which refers to the "law for the delegation of monitoring beef labelling", has been repealed by a regional parliament after the EU lifted a recommendation to carry out BSE tests on healthy cattle. German is famous for its compound nouns, which frequently become so cumbersome they have to be reduced to abbreviations. The beef labelling law, introduced in 1999 to protect consumers from BSE, was commonly transcribed as the "RkReÜAÜG", but even everyday words are shortened to initials so Lastkraftwagen - lorry - becomes Lkw.
Rindfleischetikettierung

The word - which refers to the "law for the delegation of monitoring beef labelling", has been repealed by a regional parliament after the EU lifted a recommendation to carry out BSE tests on healthy cattle. German is famous for its compound nouns, which frequently become so cumbersome they have to be reduced to abbreviations. The beef labelling law, introduced in 1999 to protect consumers from BSE, was commonly transcribed as the "RkReÜAUG", but even everyday words are shortened to initials so Lastkraftwagen - lorry - becomes Lkw.
The `overflow-wrap` property is used to specify whether or not the browser may break lines within words in order to prevent overflow when an otherwise *unbreakable* string is too long to fit in its containing box.
Final solution

```
.hyphenate {
    overflow-wrap: break-word;
    word-wrap: break-word;
    -webkit-hyphens: auto;
    -ms-hyphens: auto;
    -moz-hyphens: auto;
    hyphens: auto;
}
```

This solution will show hyphens for every browser supporting it and will break lines in every other browser – perfect. Although I have tested this solution in 26 different browsers I am still not sure this will work 100% – if you find any edge case please let me know.

Update 28.09.2015
Some people asked why I didn’t mention HTML soft hyphens. There are two reasons. First of all, this post is about solutions using CSS and not HTML and second and more important is that I think it’s impractical to define possible hyphens in the text itself and almost impossible for editors to do that by hand.
Conclusion

I tested all the examples above and combinations of them in IE7, IE8, IE9, IE10, IE11, Edge, Firefox 39 (Windows, Linux, Mac), Chrome 44 (Windows, Linux, Mac), Opera 30 (Windows, Mac), Safari 8 (Mac), Safari 5.1 (Windows), Android 5 (Nexus 6), Android 4.4 (Nexus 5), Android 2.3 (Galaxy S2), iOS 8.3 (iPhone 6), iOS 7 (iPhone 5S), iOS 6 (iPhone5), Opera Mini (Android 5), Opera Classic (Android 5), Opera Mobile (Android 5) and Windows Phone 8.1 (Lumia 930) using real devices and BrowserStack – Here is a list of all 26 browsers and their results.

When searching the web you probably will find the following solution:

```css
.hyphenate {
  -ms-word-break: break-all;
  word-break: break-all;
  word-break: break-word;

  -webkit-hyphens: auto;
  -moz-hyphens: auto;
  hyphens: auto;
}
```
CSS3 Overflow-wrap

Allows lines to be broken within words if an otherwise unbreakable string is too long to fit. Currently mostly supported using the word-wrap property.

<table>
<thead>
<tr>
<th>Current aligned</th>
<th>Usage relative</th>
<th>Date relative</th>
<th>Show all</th>
</tr>
</thead>
<tbody>
<tr>
<td>IE</td>
<td>Edge</td>
<td>Firefox</td>
<td>Chrome</td>
</tr>
<tr>
<td>11</td>
<td>14</td>
<td>50</td>
<td>55</td>
</tr>
</tbody>
</table>

Partial support refers to requiring the legacy name "word-wrap" (rather than "overflow-wrap") to work.
Dealing with long words in CSS

July 31, 2015
by Michael Scharnagl

The web consists of content, content consists of words and words can be long, very long. Everyone involved with the web will sooner or later have to deal with long words.

Rindfleischetikettierungsüberwachungsaufgabenübertragungsgesetz

The word - which refers to the “law for the delegation of monitoring beef labelling”, has been repealed by a regional parliament after the EU lifted a recommendation to carry out BSE tests on healthy cattle. German is famous for its compound nouns, which frequently become so cumbersome they have to be reduced to abbreviations. The beef labelling law, introduced in 1999 to protect consumers from BSE, was commonly transcribed as the “RkReÜAUG”, but even everyday words are shortened to initials so Lastkraftwagen - lorry - becomes Lkw.
You’ve built a *perfect grid* with *perfectly sized* thumbnail images (e.g. squared size), but when the client uploads images with incorrect dimensions, they are *squished* into the rectangle, incorrectly resized.
**Base style:**

```css
img { width: 200px; height: 200px; border: 1px solid; background: #eee; }
```

Images squished
For *img src*, we can use `object-fit` property in CSS to "letterbox" the images, preserving the ratio, or crop the images inside the block. For background images, we can apply `background-size` exactly the same way.
Base style + img { object-fit: contain }

Images letterboxed, correct aspect ratio maintained
Base style + img { object-fit: cover; }

Images expand to cover width and height, correct aspect ratio maintained
Base style + img { object-fit: none; }

Images expand to actual size, cropped inside set width and height, correct aspect ratio maintained
Method of specifying how an object (image or video) should fit inside its box.

<table>
<thead>
<tr>
<th>Current aligned</th>
<th>Usage relative</th>
<th>Show all</th>
</tr>
</thead>
<tbody>
<tr>
<td>IE</td>
<td>Firefox</td>
<td>Chrome</td>
</tr>
<tr>
<td>8</td>
<td>33</td>
<td>31</td>
</tr>
<tr>
<td>9</td>
<td>34</td>
<td>38</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>11</td>
<td>35</td>
<td>43</td>
</tr>
</tbody>
</table>

Notes: Known issues (0) | Resources (6) | Feedback
Polyfill for CSS object-fit property to fill-in/fit-in images into containers.

v0.3.7

- dist: Update usage of indexed style properties regarding
- src: Update usage of indexed style properties regarding
- tests: Add new info for local testing.
- .bowerrc: Initial commit
- .editorconfig: add pkg things
- .gitattributes: Initial commit
- .gitignore: add html classes, fix tests
- .jshintrc: add pkg things
- CHANGELOG.md: Update README with latest browser developments
- CONTRIBUTING.md: Initial commit
- Gruntfile.js: Typo, whitespace & updated main array
The Widescreen Web: Using CSS object-fit

Estimated reading time: 4 minutes, 15 seconds

Cet article est également disponible en français

Most everyone is familiar with the fact that movies are letterboxed, rescaled or cropped as they move from the theater to televisions and tablets. As a web developer, you’ll also be aware of the different ways background images can be made responsive on web pages, stretching to cover the viewport, or being cropped as the browser window resizes.

An obvious gap in web design is the lack of the same intelligent and automatic resizing rules for images and videos. That’s the role of CSS object-fit.
The CSS3 object-fit and object-position Properties

Introduction

A common problem in CSS concerns how to control the aspect ratio of replaced elements, such as `<img>` or `<video>`. For example, you might want to have all images occupy the same space on a page, but not distort and lose their aspect ratio when someone uses an image file that isn’t quite the right size. Resizing and letter-boxing the image slightly to conserve the aspect ratio is...
Is there any way to **isolate expensive components**, be it JavaScript, comments or off-canvas navigation, similar to lazy loading, and paint important content **faster** using CSS alone?
The `contain` property is a primitive for isolating style, layout and paint. It allows us to limit a specific DOM sub-tree and the rest of the document with *native boundaries*.
• With the *contain* property, we can define priorities for loading and painting CSS components.

  • *Third-party widgets*
    Delay expensive layout with *contain: strict*.

  • *Off-screen modules*
    Delay expensive paints with *contain: paint*.

  • *Container queries*
    Focus on the local scope with *contain: strict*.
With the `contain` property, we can define priorities for loading and painting CSS components.

- **Third-party widgets**
  Delay expensive layout with `contain: strict`.

- **Off-screen modules**
  Delay expensive paints with `contain: paint`.

- **Container queries**
  Focus on the local scope with `contain: strict`.

**Browser support is coming.**

Enabled by default in Chrome 52.
CSS containment

April 5, 2016
by Michael Scharnagl

I haven't heard about the contain property until some weeks ago when I asked about use cases for container queries on twitter and David Baron mentioned it in a response saying that contain: strict; can avoid many of the theoretical problems of container queries. Since then I read the specification and everything I found about it trying to understand it; Here is what I learned.

The contain property

It's a primitive for isolating style, layout, and paint. The contain property allows developers to limit a specific DOM sub-tree and the rest of the document; You can think of it like an iframe. Much like an iframe, this boundary establishes a new layout root, ensuring that DOM changes in the sub-tree never trigger reflows in the parent document.

```
- The contain property allows an author to indicate that an element and its contents are, as much as possible, independent of the rest of the document tree.
- From the W3C Specification
```

none | strict | layout | style | paint | size | contain

There are seven different values for the contain property.
Use cases

So far we know that by using CSS containment we can isolate elements from the rest of the document to mark them as independent parts. To show you where this can be helpful here are some use cases:

Widgets

When integrating third-party widgets you mostly don’t have much control and they can decrease the site performance dramatically by doing expensive layout, style or paint operations. To make them independent from our site we can set `contain: strict;` for the most outer element of the third-party widget. This way, they won’t affect the performance of all the other parts of the page.

Off-Screen

If you build a off-screen navigation or similar, the browser paints the content completely although it is not visible on load. By setting `contain: paint;` the user agent can skip the paint off the off-screen element and therefore paint all the other content faster.

Container queries

As I already mentioned in the beginning, `contain: strict;` can avoid many of the problems of container queries. One of the “problems” of container queries is that the children and their content can have an effect on the size of the container. This can be avoided by using CSS containment.
What’s the deal with emoji? Can we style them with CSS, or change them with JavaScript? Or even better, can they be an alternative to SVG and icon fonts?
Clockwise Rightwards and Leftwards Open Circle Arrows With Circled One Overlay (also known as U+1F502, &#x1f502; or \u1f502).
Person Raising Both Hands in Celebration (also known as Festivus Miracle Emoji, U+1F64C, &#1f64c; or \u1f64c).
Reimagining single-page applications with *progressive enhancement*

What’s the difference between a *web page* and a *web application*? Though we tend to identify documents with reading and applications with interaction, most *web-based applications* are of the blended variety.

Users can consume information and perform tasks in the same place. Regardless, the way we approach building web applications usually dispenses with some of the simple virtues of the readable web. Single-page applications tend to take the form of runtimes. JavaScript executables deployed like popup shops into vacant
Is the internet killing creativity?

The internet is a wonderful place (mostly). An unprecedented revolution in communication, it continues to empower more people to publish and share their knowledge than any other phenomenon in history. It is a limitless playground of ideas and unbridled creativity. Or is it?
Is the internet killing creativity?

The internet is a wonderful place (mostly). An unprecedented revolution in communication, it continues to empower more people to publish and share their knowledge than any other phenomenon in history. It is...
Is the internet killing creativity?

The internet is a wonderful place (mostly). An unprecedented revolution in communication, it continues to empower more people to publish and share their knowledge than any other phenomenon in history. It is a limitless playground of ideas and unbridled creativity. Or is it?

Posted in Coding
Is the internet killing creativity?

The internet is a wonderful place (mostly). An unprecedented revolution in communication, it continues to empower more people to publish and share their knowledge than any other phenomenon in history. It is a limitless playground of ideas and unbridled creativity. Or is it?

Posted in Coding
My name is

Glen Maddern

and I do rad web stuff

ARTICLES

CSS Modules
Welcome to the Future - 2015-08-19

Interoperable CSS
A CSS standard for the Loader Age - 2015-06-21

JavaScript in 2015
*Emoji* are coloured glyphs added into Unicode 6.0 in 2010. They are depicted in the spec, but the exact appearance *isn’t defined* and varies between fonts, just like normal typefaces display letters differently.
These are all the same emoji!

This is what the “grinning face with smiling eyes” emoji looks like on devices for each of these platforms:
“Blissfully happy” or “ready to fight”: Varying Interpretations of Emoji

Hannah Miller, Jacob Thebault-Spieker, Shuo Chang, Isaac Johnson, Loren Terveen, Brent Hecht
GroupLens Research, University of Minnesota
Minneapolis, MN 55455, USA
{hmiller, thebault, schang, ijohnson, terveen, bhecht}@cs.umn.edu

Abstract
Emoji are commonly used in modern text communication. However, as graphics with nuanced details, emoji may be open to interpretation. Emoji also render differently on different viewing platforms (e.g., Apple’s iPhone vs. Google’s Nexus phone), potentially leading to communication errors. We explore whether emoji renderings or differences across platforms give rise to diverse interpretations of emoji. Through an online survey, we solicit people’s interpretations of a sample of the most popular emoji characters, each rendered for multiple platforms. Both in terms of sentiment and semantics, we analyze the variance in interpretation of the emoji, quantifying which emoji are most (and least) likely to be misinterpreted. In cases in which participants rated

Most commonly-used emoji are encoded in the Unicode standard for indexing characters. There are currently 1,282 emoji in the Unicode standard, and for each of these, the Unicode Consortium provides a code and name (e.g., U+1F600 for “grinning face”) but not the actual graphic. This is the same as is the case for Unicode text characters: for example, the Unicode character U+0041 indexes the Latin capital letter ‘A’, but it does not indicate specifically how the ‘A’ should look. Instead, a font renders the Unicode characters a particular way: the appearance of this text that you are reading is dictated by the Times New Roman font.
Because *emoji* are Unicode code points, we can create a font with the emoji that we need in our interface and *override* platform-specific designs to avoid inconsistencies.
<table>
<thead>
<tr>
<th>Native</th>
<th>Apple</th>
<th>Android</th>
<th>Android</th>
<th>Symbola</th>
<th>Twitter</th>
<th>Phantom</th>
<th>Unicode</th>
<th>Bytes (UTF-8)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>😬</td>
<td>😳</td>
<td>😷</td>
<td>😷</td>
<td>😷</td>
<td>😷</td>
<td>😷</td>
<td>U+1F601</td>
<td>\xF0\x9F\x98\x81</td>
<td>grimacing face with smiling eyes</td>
</tr>
<tr>
<td>😃</td>
<td>😃</td>
<td>😻</td>
<td>😻</td>
<td>😻</td>
<td>😻</td>
<td>😻</td>
<td>U+1F602</td>
<td>\xF0\x9F\x98\x82</td>
<td>face with tears of joy</td>
</tr>
<tr>
<td>😃</td>
<td>😃</td>
<td>😻</td>
<td>😻</td>
<td>😻</td>
<td>😻</td>
<td>😻</td>
<td>U+1F603</td>
<td>\xF0\x9F\x98\x83</td>
<td>smiling face with open mouth</td>
</tr>
<tr>
<td>😃</td>
<td>😃</td>
<td>😻</td>
<td>😻</td>
<td>😻</td>
<td>😻</td>
<td>😻</td>
<td>U+1F604</td>
<td>\xF0\x9F\x98\x84</td>
<td>smiling face with open mouth and smiling eyes</td>
</tr>
<tr>
<td>😃</td>
<td>😃</td>
<td>😻</td>
<td>😻</td>
<td>😻</td>
<td>😻</td>
<td>😻</td>
<td>U+1F605</td>
<td>\xF0\x9F\x98\x85</td>
<td>smiling face with open mouth and cold sweat</td>
</tr>
<tr>
<td>😃</td>
<td>😃</td>
<td>😻</td>
<td>😻</td>
<td>😻</td>
<td>😻</td>
<td>😻</td>
<td>U+1F606</td>
<td>\xF0\x9F\x98\x86</td>
<td>smiling face with open mouth and tightly-closed eyes</td>
</tr>
<tr>
<td>😃</td>
<td>😃</td>
<td>😻</td>
<td>😻</td>
<td>😻</td>
<td>😻</td>
<td>😻</td>
<td>U+1F607</td>
<td>\xF0\x9F\x98\x87</td>
<td>winking face</td>
</tr>
</tbody>
</table>
Internally strings are represented in *UTF-16*, so each code point can be represented by one or more 16-bit *code units*. Some emoji use only 1 code unit, others 2 and more.
Not all *emoji* are created equal. They can be modified with emoji *modifiers*, so some are *surrogates* which is why sometimes icons are rendered incorrectly.
"👍".length // returns 4
"🇳🇴".length // returns 4
"👨‍👩‍👧‍👦".length // returns 11
• HTML:

```html
<p style="font-family: 'Comic Sans', sans-serif;">Apple</p>
```
• **HTML:**
  ```html
  <p style="font-family: 'Comic Sans', sans-serif;">📍</p>
  ```

• **The browser will:**
  — Look up the glyph in the *Comic Sans* font,
  — If it can’t find the glyph, it will fall back to the *fallback* font,
  — In this case, fallback is *sans-serif* (Helvetica/Arial),
  — The fallback doesn’t have the glyph either,
  — Browser will try to figure out the glyph type,
  — Eventually it will look up in a locally installed Emoji font (e.g. *AppleColorEmoji*),
  — The browser will render the icon.
 Emoji: how do you get from U+1F355 to 🍕?

APRIL 4, 2016

You know that scene in The Rock where Nicolas Cage is super dreamy (like he is) and decides his life mission is to look for VX poison gas and save San Francisco (like he would)? That's baasically me, if by “look for VX poison gas” you mean “nerd out on emoji”, and by “save San Francisco” you mean “and tell everyone about it!”. I mean, you clicked on this link, what did you think was going to happen?

How did we get so lucky?

An emoji is a coloured glyph. They appeared around 1999 in Japan, where each mobile carrier implemented their own variants, and people were sending them around in text messages. This was a bit of a mess, as you can imagine proprietary formats interacting with other proprietary formats to be, so in 2000 there was a proposal to standardize them. It wasn’t until 2009, though, that emoji got specced in Unicode 5.2.

Spec trivia: each emoji has a design guideline and name, which is a description/suggestion of what the emoji should look like. This is why 🤖 for example, often gets in trouble for being labelled as Information Desk Person, but is actually just a sassy lady: it’s the implementation of the emoji that doesn’t match its original description, not the other way around. If you take sassy lady away from me though, there will be words.
Styleguides

Git Commit Messages

- Use the present tense ("Add feature" not "Added feature")
- Use the imperative mood ("Move cursor to..." not "Moves cursor to...")
- Limit the first line to 72 characters or less
- Reference issues and pull requests liberally
- When only changing documentation, include [ci skip] in the commit description
- Consider starting the commit message with an applicable emoji:
  - 🍃 :tart: when improving the format/structure of the code
  - 🏅 :racehorse: when improving performance
  - 🚫 :non-potable_water: when plugging memory leaks
  - 📕 :memo: when writing docs
  - 🐧 :penguin: when fixing something on Linux
  - 🍌 :apple: when fixing something on macOS
  - 🚀 :checkered_flag: when fixing something on Windows
  - 🐛 :bug: when fixing a bug
  - 🔥 :fire: when removing code or files
  - ❤️ :green_heart: when fixing the CI build
  - ✅ :white_check_mark: when adding tests
  - 🔒 :lock: when dealing with security
  - ⬆️ :arrow_up: when upgrading dependencies
  - ⬇️ :arrow_down: when downgrading dependencies
  - 🚫 :shirt: when removing linter warnings

JavaScript Styleguide
How do you highlight both a row and a column on hover and on tap in a multi-column table? Highlighting the current row is easy, but what about the column?
<table>
<thead>
<tr>
<th></th>
<th>50kg</th>
<th>55kg</th>
<th>60kg</th>
<th>65kg</th>
<th>70kg</th>
</tr>
</thead>
<tbody>
<tr>
<td>160cm</td>
<td>20</td>
<td>21</td>
<td>23</td>
<td>25</td>
<td>27</td>
</tr>
<tr>
<td>165cm</td>
<td>18</td>
<td>20</td>
<td>22</td>
<td>24</td>
<td>26</td>
</tr>
<tr>
<td>170cm</td>
<td>17</td>
<td>19</td>
<td>21</td>
<td>23</td>
<td>25</td>
</tr>
<tr>
<td>175cm</td>
<td>16</td>
<td>18</td>
<td>20</td>
<td>22</td>
<td>24</td>
</tr>
</tbody>
</table>
We create *tall pseudo elements* on `<td>`’s with a negative *top*-value of half of that value. Then we hide these pseudo elements with *overflow: hidden*, and use *negative z-index* to keep it below the content. Then we make all cells focusable and focus them on *touchstart*.

— @simurai
<table>
<thead>
<tr>
<th></th>
<th>50kg</th>
<th>55kg</th>
<th>60kg</th>
<th>65kg</th>
<th>70kg</th>
</tr>
</thead>
<tbody>
<tr>
<td>160cm</td>
<td>20</td>
<td>21</td>
<td>23</td>
<td>25</td>
<td>27</td>
</tr>
<tr>
<td>165cm</td>
<td>18</td>
<td>20</td>
<td>22</td>
<td>24</td>
<td>26</td>
</tr>
<tr>
<td>170cm</td>
<td>17</td>
<td>19</td>
<td>21</td>
<td>23</td>
<td>25</td>
</tr>
<tr>
<td>175cm</td>
<td>16</td>
<td>18</td>
<td>20</td>
<td>22</td>
<td>24</td>
</tr>
</tbody>
</table>

**CSS:**
```css
table { overflow: hidden; }
td, th { position: relative; }
tr:hover { background-color: #fffa; }
td:hover::after { content: "";
    position: absolute;
    width: 100%;
    height: 10000px;
    left: 0;
    top: -5000px;
    background-color: currentColor;
    z-index: -1;
}
```
<table>
<thead>
<tr>
<th></th>
<th>50kg</th>
<th>55kg</th>
<th>60kg</th>
<th>65kg</th>
<th>70kg</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>160cm</strong></td>
<td>20</td>
<td>21</td>
<td><strong>23</strong></td>
<td>25</td>
<td>27</td>
</tr>
<tr>
<td><strong>165cm</strong></td>
<td>18</td>
<td>20</td>
<td><strong>22</strong></td>
<td>24</td>
<td>26</td>
</tr>
<tr>
<td><strong>170cm</strong></td>
<td>17</td>
<td>19</td>
<td><strong>21</strong></td>
<td>23</td>
<td>25</td>
</tr>
<tr>
<td><strong>175cm</strong></td>
<td>16</td>
<td>18</td>
<td><strong>20</strong></td>
<td>22</td>
<td>24</td>
</tr>
</tbody>
</table>

**CSS:**

```
table { overflow: hidden; }
tr:hover { background-color: #ffa; }

td:hover::after, th:hover::after { content: "";
  position: absolute;
  width: 100%;
  height: 10000px;
  left: 0;
  top: -5000px;
  background-color: currentColor;
  z-index: -1;
}
```
Highlighting rows of a table is pretty darn easy in CSS. `tr:hover { background: yellow; }` does well there. But highlighting columns has always been a little trickier, because there is no single HTML element that is parent to table cells in a column. A dash of JavaScript can handle it easily, but Andrew Howe recently emailed me to share a little trick he found on StackOverflow, posted by Matt Walton.

It was a few years old, so I thought I’d just clean it up and post it here.

The trick is using huge pseudo elements on the `<td>`s, hidden by the table.
“By default, tables are quite unpredictable and spongy, and if you don’t know how lengthy the content inside cells will be, some columns can be *unpredictably wide*, destroying the layout. What to do?

— Louis Lazaris
<table>
<thead>
<tr>
<th></th>
<th>1-1</th>
<th>1-2</th>
<th>1-3</th>
<th>1-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Example text goes here. Example text goes here.</td>
<td>2-2</td>
<td>2-3</td>
<td>2-4</td>
</tr>
<tr>
<td>3-1</td>
<td></td>
<td>3-2</td>
<td>3-3</td>
<td>3-4</td>
</tr>
<tr>
<td>4-1</td>
<td></td>
<td>4-2</td>
<td>4-3</td>
<td>4-4</td>
</tr>
</tbody>
</table>
# Default Tables are Weird

<table>
<thead>
<tr>
<th>I'm a table and stuff</th>
</tr>
</thead>
<tbody>
<tr>
<td>I'm as wide as the cells are</td>
</tr>
<tr>
<td>I can easily become 100% wide. and I'll stop there if I possibly can.</td>
</tr>
<tr>
<td>If you don't specify it, the cells will be kind of arbitrarily wide depending on the content inside. This one is way wider right? <del>(°_o)</del></td>
</tr>
<tr>
<td>If you <strong>do</strong> specify it, that will be respected if it's possible to do so. These should be 50% wide.</td>
</tr>
<tr>
<td>But things can get weird. I told both of these cells to be 1000px wide and it's like NOPE.</td>
</tr>
<tr>
<td>It's kinda like a calculation. I'm told to be 2000px wide, thus I'm 66.66% wide (2/3 of total 3000px). And I'm told to be 1000px wide, thus I'm 33.33% wide (1/3 of total 3000px).</td>
</tr>
<tr>
<td>An image that's too wide though? That'll bust out like nobody's business. You can try hiding the overflow but that doesn't work.</td>
</tr>
<tr>
<td>Or some non-wrapping text? That'll blast out a table in short order like this.</td>
</tr>
</tbody>
</table>
But seriously, as always, a live example will help. In the following demo, the table has `table-layout: fixed` added in the CSS. Click the toggle button to toggle it off, then on, etc.

### Using the `table-layout` property

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>1-2</td>
<td>1-3</td>
<td>1-4</td>
</tr>
<tr>
<td>2-1 Example text goes here...</td>
<td>2-2 Example text</td>
<td>2-3</td>
<td>2-4</td>
</tr>
<tr>
<td>3-1</td>
<td>3-2</td>
<td>3-3</td>
<td>3-4</td>
</tr>
<tr>
<td>4-1</td>
<td>4-2</td>
<td>4-3</td>
<td>4-4</td>
</tr>
</tbody>
</table>

You can see in this example the advantage of using `table-layout: fixed`, as opposed to the default of `auto`. This won't always be the best choice and it won't always be necessary, but it's a nice one to keep in mind when dealing with tables that have cells with variable-width data.

Chris Coyler did a great write-up on this property last year, so if you want a much more comprehensive discussion, that's your best bet.
Fixed Table Layouts

CHRIS COYIER // JULY 2, 2014

There is a CSS property for tables that, it seems to me, is well-supported, little known, and super useful. It changes the way that tables are rendered such that it gives you a sturdier, more predictable layout.

It is this:

```css
.table {
  table-layout: fixed;
}
```

The default property for `table-layout` is `auto`, and that is the table layout I think most of us are familiar with. That style, to me, feels
With `table-layout: fixed;` the layout is fixed based on the widths defined for the first row. Set the width of those, and the rest of the table follows.

— Chris Coyier
<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Job</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>Johnny Five</td>
<td>Robotin'</td>
<td><a href="mailto:need@input.com">need@input.com</a></td>
</tr>
<tr>
<td>0002</td>
<td>Super Superlonglastnamesmith</td>
<td>Doin' stuff</td>
<td><a href="mailto:doing@stuff.com">doing@stuff.com</a></td>
</tr>
<tr>
<td>0003</td>
<td>Roger Wilco</td>
<td>Truckdrivin'</td>
<td><a href="mailto:roger@wilco.com">roger@wilco.com</a></td>
</tr>
<tr>
<td>0004</td>
<td>Mad Hatter</td>
<td>Hat Makin'</td>
<td>loves@mercury....</td>
</tr>
</tbody>
</table>
LEVEL 13 EMAIL
Oh heavens! You’ve been promoted to craft responsive emails. Many of your users use AOL/Outlook clients. How do you make emails bulletproof?
Dreamforce Summer Savings

Dreamforce. It's a smart investment. An astounding 90% of attendees surveyed said their business improvements outweighed the cost of going to Dreamforce, and 98% would recommend it to others. Justify your trip.

NEW to Dreamforce: Get the vision and roadmap for the products you care most about.

Summer Special
Save $500 off the on-sale price of $1,200. Register for $999 until July 31.

Justify your trip: Read our blog about how to convince your boss to let you come to Dreamforce.
- **Content Stacking**

```html
<table>
  <tr>
    <td class="col" width="300">...</td>
    <td class="col" width="300">...</td>
  </tr>
</table>
```
• **Content Stacking**

```css
@media only screen and (max-width: 600px) {
  table, tr, td {
    display: block; /* table-cell -> block */
    width: 100%;
  }
}
```
• *Column Switching*

```html
<table>  /* "Desktop" width = 600px = 300*2 */
<tr>
   <td class="sub-col" width="300">...</td>
   <td class="main-col" width="300">...</td>
</tr>
</table>
```
• **Column Switching**

```css
@media only screen and (max-width: 500px) {
  table, tr, td {
    display: block; /* table-cell -> block */
    width: 100%;
  }

  td[class=main-col] { display: table-header-group; }
  td[class=sub-col] { display: table-footer-group; }
}
```
Order and Re-order

```html
<table class="wrapper">
  <table class="header">Header</table>
  <table class="nav">Navigation</table>
  <table class="content">Content</table>
  <table class="footer">Footer</table>
</table>
```
• Order and Re-order

@media only screen and (max-width: 500px) {
    table[class=wrapper] { display: table; }
    table[class=header] { display: table-caption; }
    table[class=nav] { display: block; }
    table[class=content] { display: table-header-group; }
    table[class=footer] { display: table-footer-group; }
}
calc() & width & min-width & max-width.
• **CSS:**

```css
.box {
  width: 320px;
  min-width: 480px;
  max-width: 160px;
}
```

If the *width* value is greater than the *max-width* value, *max-width* wins.
• **CSS:**

```css
.box {
  width: 320px;
  min-width: 480px;
  max-width: 160px;
}
```

If the *min-width* value is greater than the *width* or *max-width* values, then *min-width* wins.
Let’s build a 2-col-layout that stacks and grows below 480px. No media queries allowed.

• CSS:

```css
.box {
    display: inline-block;
    min-width: 50%; // basically 2-col-desktop version
    max-width: 100%; // basically 1-col-mobile version
    width: calc((480px - 100%) * 480);
    /* 480px = the breakpoint, 100% = width of the parent
    Goal: create a value bigger than our max-width or smaller
    than our min-width, so that either one of those property is
    applied instead. */
}
```
with a parent of 500px

```css
.block {
    display:inline-block;
    min-width:50%;
    max-width:100%;
    width:calc((480px - 100%) * 480); /* 960px */
}
```
with a parent of 400px

.block {
  display:inline-block;
  min-width:50%;200px
  max-width:100%;400px winner
  width:calc((480px - 100%) * 480);38400px
}

Live Twitter Feed
Tweet with #TEDC15 and refresh to see your tweet in the stream!

Amanda Soehnlen @asoehnlen • 3m
this is just about the best thing (minus #tedc15 emails) I have read today
https://t.co/d9dsmrX0x

Jason Tropp @tropp • 4m
hey #TEDC15 the messages you hid in that email code are AMAZING! you really like us you really like us:D #hiddenmessages

Jon Woodrow @jonwoodrow • 5m
This is very cool @connect_agency #TEDC15 http://t.co/qs6RyG3t2

Marc Hedlund @Marc_everything • 6m
Live twitter feed in an email 0_0 #TEDC15

Brittany R @brittus_66 • 6m
Litmus live twitter feed - that 'EVEN works in Outlook - is wizardry! Where's platform 0 2/47 I need that knowledge #TEDC15

Tweet #TEDC15 to join the feed
Save the *dates!*  
We’re going worldwide.  

Tickets—and more details—are coming soon.
Bulletproof email buttons

Design gorgeous buttons using progressively enhanced VML and CSS. You can also create rock-solid background images in emails too.

Button text
Show me the button!

Background image
http://imgur.com/b7tepK9.png

Host your own image or use a free service like Imgur to use "Direct Link" URL.

Background color
#99c739

Font color
#ffffff

Button width
200 px

Button height
38 px

Show me the button!
Bulletproof background images

Use rock-solid background images in your HTML email with some help from VML and CSS. Why not do the same with our bulletproof button generator?

Background Image

http://Iimgur.com/XCnBXwP.png

Host your own image or use a free service like imgur (use "Direct Link" URL).

Fallback color

#7bceeb

Shown if the background image isn’t loaded, and behind images that have transparency.

Apply background image to:

- Full email body
  
  Tile the background image in the full email window.

To: Your next campaign

Subject: Bulletproof background images
Responsive Email Patterns

A collection of patterns & modules for responsive emails

Submit A Pattern

Layout
- Column Drop
- 2 Equal-Width Columns

Navigation
- Top Links
- Stacked Split Links

Lists
- List w/ Thumbnails
- List w/ Thumbnails v2
Responsive Email Resources

A collection of tools & resources for responsive email design

Submit A Resource

Design Tools
- Style Tiles
- Element Culls

Design Patterns & Guides
- Responsive Email Patterns
- Sub-Category: Responsive email

Text Editors & Plugins
- Sublime Text
- Code
The Fab Four technique to create Responsive Emails without Media Queries

I think I found a new way to create responsive emails, without media queries. The solution involves the CSS `calc()` function and the three `width`, `min-width` and `max-width` properties.

Or as I like to call them all together: the Fab Four (in CSS).
Responsive Emails has been a trending topic in Email Marketing since last five years. Today most emails are read on mobile devices, so responsiveness is a must, but many email clients and webmails have a really limited support of media query, the main css way to make responsive html layout.

Previously...

Until Sept 2016 (when Gmail started supporting non-inline styles and media queries), the most problematic clients were Outlook and Gmail (both web and app); to overcome these problems many techniques have been developed - mainly table based layouts with conditional comments to satisfy Outlook a lot of inline css to make Gmail happy.

Someone started to use mobile-first templates, letting media queries manage the desktop version - where it was possible.
Can we replicate interactivity that require *maintaining state*, such as switching tabs or panels or toggle menus, with CSS? JS binds events that manipulate classes and CSS restyles elements based on those classes. What if we used *radio buttons* for the same purpose?..  

— Art Lawry
We use a connected label and checkbox input to control another element (e.g. `<div>`). We hide the checkbox but `<label>` still toggles its value on and off. By using *adjacent sibling combinator*, we can style the `<div>` differently based on the :checked state.

— Chris Coyier
**HTML:**

```html
<label for="toggle-1">
  <input type="checkbox" id="toggle-1" />
  <div>Responsive component without JavaScript</div>
</label>
```

**CSS:**

```css
input[type=checkbox] { position: absolute; 
  top: -9999px; left: -9999px; }
/* Or checkbox on top of clickable area with opacity: 0; */

div { color: grey; } /* default state */
input[type=checkbox]:checked ~ div { color: red; } /* toggled */
```
Stuff you can do with the “Checkbox Hack”

Published December 21, 2011 by Chris Coyier

The "Checkbox Hack" is where you use a connected label and checkbox input and usually some other element you are trying to control, like this:

```
<label for="toggle-1">Do Something</label>
<input type="checkbox" id="toggle-1"/>
<div>Control me!</div>
```

Then with CSS, you hide the checkbox entirely. Probably by kicking it off the page with absolute positioning or setting its opacity to zero. But just because the checkbox is hidden, clicking the label still toggles its value on and off. Then...
Radio-Controlled Web Design

by Art Lawry · July 29, 2014

Published in CSS, HTML, JavaScript · 18 Comments

Interactive user interfaces are a necessity in our responsive world. Smaller screens constrain the amount of content that can be displayed at any given time, so we need techniques to keep navigation and secondary information out of the way until they’re needed.
Reclaim Your Gear + Get 10% Off!

You left these items in your cart. Purchase these items in the next 24 hours to grab your discount.

<table>
<thead>
<tr>
<th>ITEM</th>
<th>QTY</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goodale Norwood Corduroy Pant</td>
<td>1</td>
<td>Sale $249.00</td>
</tr>
<tr>
<td>Retrofit Aztec Raglan Hoodie</td>
<td>-2</td>
<td>$116.00</td>
</tr>
<tr>
<td>Of All Threads Bull Dogs &amp; Birds Socks</td>
<td>-1</td>
<td>$12.00</td>
</tr>
</tbody>
</table>

Subtotal: $377.00  
Tax: $18.85  
10% Discount: $39.59  
Total Price: $395.85
• **HTML:**

```html
<input type="radio" id="itemA-1"/>
<input type="radio" id="itemA-2"/>
<input type="radio" id="itemA-3"/>
...
<label for="itemA-1">Carousel</label>
<label for="itemD-11">Div</label>
...
```

• **CSS:**

```css
#itemA-3:checked ~
#itemB-6:checked ~
#itemC-2:checked ~
#itemD-11:checked ~
#itemE-5:not:checked ~
#itemF-2:checked ~
#itemG-5:checked ~ * .div1 {
  display: block;
}
```
## Reclaim Your Gear + Get 10% Off!

You left these items in your cart. Purchase these items in the next 24 hours to grab your discount.

<table>
<thead>
<tr>
<th>ITEM</th>
<th>QTY</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goodale Norwood Corduroy Pant</strong></td>
<td>1</td>
<td>Sale $249.00</td>
</tr>
<tr>
<td>Original Price: $359.18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Color: Tan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Size: 32</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Retrofit Aztec Raglan Hoodie</strong></td>
<td>2</td>
<td>$116.00</td>
</tr>
<tr>
<td>Original Price: $158.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Color: Dk Char Hsa</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Size: M</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Of All Threads Bull Dogs &amp; Birds</strong></td>
<td>1</td>
<td>$12.00</td>
</tr>
<tr>
<td>Original Price: $12.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Subtotal: $377.00  
Tax: $18.85  
10% Discount: $39.59  
Total Price: $395.85
• Shopping Cart Email Checkout
  — 117 radio buttons,
  — 4 checkboxes,
  — Multi-page layout,
  — Adding/removing products,
  — Edit quantity, color, size,
  — See live calculation,
  — Select payment and delivery,
  — Form validation — all in the email,
  — Fallback: just a regular ol’ email.

• CSS:
  
  ```css
  body {
    counter-reset: amount;
  }

  #itemA-3:checked ~ #itemE-5:not:checked {
    counter-increment: amount;
  }

  .price {
    content: '$' + counter(amount);
  }
  ```
In email clients, we just need support for `:checked` values and siblings selectors. **But:** In email, file size is limited to 102 Kb. In Gmail, CSS is limited to 12,000 characters.
Support and fallbacks

- Static: 20%
- Limited: 19%
- Interactive: 61%
<table>
<thead>
<tr>
<th>Rank</th>
<th>Device</th>
<th>Market Share</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>Apple iPhone</td>
<td>33%</td>
<td>-0.1</td>
</tr>
<tr>
<td>#2</td>
<td>Gmail</td>
<td>16%</td>
<td>+0.3</td>
</tr>
<tr>
<td>#3</td>
<td>Apple iPad</td>
<td>11%</td>
<td>+0.21</td>
</tr>
<tr>
<td>#4</td>
<td>Google Android</td>
<td>10%</td>
<td>+1.06</td>
</tr>
<tr>
<td>#5</td>
<td>Apple Mail</td>
<td>7%</td>
<td>-0.2</td>
</tr>
<tr>
<td>#6</td>
<td>Outlook</td>
<td>7%</td>
<td>-0.45</td>
</tr>
<tr>
<td>#7</td>
<td>Outlook.com</td>
<td>5%</td>
<td>+0.13</td>
</tr>
<tr>
<td>#8</td>
<td>Yahoo! Mail</td>
<td>3%</td>
<td>-0.14</td>
</tr>
<tr>
<td>#9</td>
<td>Windows Mail</td>
<td>1%</td>
<td>-0.11</td>
</tr>
<tr>
<td>#10</td>
<td>Windows Live Mail</td>
<td>1%</td>
<td>+0.01</td>
</tr>
</tbody>
</table>