sitepoint

14 ESSENTIAL RESPONSIVE CSS TECHNIQUES

REAL WORLD ADVICE

14 Essential Responsive CSS Techniques

- **Authors:** Guy Routledge & Adrian Sandu
- **Cover Designer:** Alex Walker

Notice of Rights

Notice of Liability

Trademark Notice

sitepoint

Published by SitePoint Pty. Ltd.

48 Cambridge Street Collingwood
VIC Australia 3066
Web: www.sitepoint.com
Email: books@sitepoint.com

About SitePoint

SitePoint specializes in publishing fun, practical, and easy-to-understand content for web professionals. Visit http://www.sitepoint.com/ to access our blogs, books, newsletters, articles, and community forums. You'll find a stack of information on JavaScript, PHP, Ruby, mobile development, design, and more.

Table of Contents

Preface

Conventions Used

You'll notice that we've used certain typographic and layout styles throughout this book to signify different types of information. Look out for the following items.

Code Samples

Code in this book is displayed using a fixed-width font, like so:

```
<h1>A Perfect Summer's Day</h1>
<p>It was a lovely day for a walk in the park.
The birds were singing and the kids were all back at school.</p>
```

Where existing code is required for context, rather than repeat all of it, ⋮ will be displayed:

```
function animate() {
   ⋮
new_variable = "Hello";
}
```

Some lines of code should be entered on one line, but we've had to wrap them because of page constraints. An ➥ indicates a line break that exists for formatting purposes only, and should be ignored:

```
URL.open("http://www.sitepoint.com/responsive-web-
➥design-real-user-testing/?responsive1");
```

Tips, Notes, and Warnings

Hey, You!

Tips provide helpful little pointers.

Ahem, Excuse Me ...

Notes are useful asides that are related—but not critical—to the topic at hand. Think of them as extra tidbits of information.

Make Sure You Always ...

... pay attention to these important points.

Watch Out!

Warnings highlight any gotchas that are likely to trip you up along the way.

Live Code

This example has a Live Codepen.io Demo you can play with.

Github

This example has a code repository available at Github.com.

Chapter

An Introduction to the Benefits of Em Values

1

The first part of this course will cover the two relative units available for use in responsive projects. In today's lesson, we'll cover the first: the `em`.

The Pros And Cons Of Using Em

When working on a responsive project it’s more flexible to use relative units like *em* for sizing text and spacing in and around elements rather than pixels. This is because this unit is relative to the font size of its parent element, allowing an element’s size, spacing and text content to grow proportionally as the `font-size` of parent elements change.

Using these relative units enables you to build a system of proportions where changing values of font-size on one element has a cascading effect on the child elements within. A system of proportions is a good thing, but this behavior of *em* does come with a downside.

Take the following snippet of HTML:

```
<ul>
  <li>lorem ipsum</li>
  <li>dolor sit
   <ol>
    <li>lorem ipsum</li>
    <li>lorem ipsum</li>
    <li>lorem ipsum</li>
    <li>lorem ipsum</li>
   </ol>
 </li>
</ul>
```

This nested list isn’t the most common thing in the world but could likely appear in a page of terms and conditions or some other kind of formal document.

If we wanted to make the list items stand out, we could set their *font-size* to be 1.5 times the size of the base size of *16px*.

```
li {
  font-size: 1.5em; /* 24px/16px */
}
```

But this will cause an issue with the nested `li` as they will be 1.5 times the size of their parent too. The nested items will be 1.5 times `24px` rather than 1.5 times `16px`. The result is that any nested list items will grow exponentially with each level of nesting. This is likely not what the designer intended!

A similar problem occurs with nested elements and `em` values of less than 1. In this case, any nested items would keep getting incrementally smaller with each level of nesting. What can we do instead?

Chapter

2

Understanding and Using Rem for Responsive Projects

Last time, we made ourselves acquainted with the `em` unit, one of the relative units we can use in responsive projects. We also saw one limitation of the `em` unit. Today let's take a look at our other options.

To recap: using a positive *em* value in nested lists means items will grow exponentially with each level of nesting. If a designer uses an *em* value less than 1, the items will shrink exponentially. Either way, this is likely not what the designer intended! So what can we do instead?

We could use pixels but relative units are more flexible in responsive projects as mentioned earlier. Instead, we can use the *rem* unit as this is always calculated based on the `font-size` of the root element which is normally the `html` element in the case of a website or web application. In a .svg or .xml document the root element might be different but those types of documents aren't our concern here.

If we use *rem* for setting *font-size* it doesn't mean the humble *em* should never get a look in. I tend to use *em* for setting padding within elements so that the spacing is always relative to the size of the text.

Use Sass To Help With Rem Browser Support

The *rem* unit is only supported from IE9 and above. If you need to support IE8 (or below) then you can use a JS polyfill or provide a *px* fallback in the following way:

```
li {
  font-size: 24px;
  font-size: 1.5rem;
}
```

If you're using Sass you could create a mixin and a function for calculating the desired size in *rem* and providing the fallback automatically.

```scss
@function rem-calc($font-size, $base-font-size: 16) {
  @return ($size/$base-font-size) *1rem;
}
@mixin rem-with-px-fallback($size, $property:font-size) {
  #{$property}: $size * 1px;
  #{$property}: rem-calc($size);
}
li {
  @include rem-with-px-fallback(24);
}
```

Chapter

Font Sizing with Rem

3

Today, we'll recap our font-sizing learning so far and look at how a few clever devs have handled this problem.

One of the pioneers of using rem units for font sizing is Jonathan Snook with his Font sizing with REM article, back in May, 2011. Like many other CSS developers, he had to face the problems that em units bring in complex layouts.

At that time, older versions of IE still had large market shares and they were unable to zoom text that was sized with pixels. However, as we saw earlier, it is very easy to lose track of nesting and get unexpected results with em units.

The main issue with using rem for font sizing is that the values are somewhat difficult to use. Let's see an example of some common font sizes expressed in rem units, assuming, of course, that the base size is 16px:

- 10px = 0.625rem
- 12px = 0.75rem
- 14px = 0.875rem
- 16px = 1rem (base)
- 18px = 1.125rem
- 20px = 1.25rem
- 24px = 1.5rem
- 30px = 1.875rem
- 32px = 2rem

As we can see, these values are not very convenient for making calculations. For this reason, Snook used a trick called "62.5%". It was not a new discovery, by any means, as it was already used with `em` units:

```
body { font-size:62.5%; }  /* =10px */
h1   { font-size: 2.4em; } /* =24px */
p    { font-size: 1.4em; } /* =14px */
li   { font-size: 1.4em; } /* =14px? */
```

As rem units are relative to the root element, Snook's variant of the solution becomes:

```
html { font-size: 62.5%; }  /* =10px */
body { font-size: 1.4rem; } /* =14px */
h1   { font-size: 2.4rem; } /* =24px */
```

One also had to take into account the other browsers that didn't support rem. Thus the code from above would have actually been written this way:

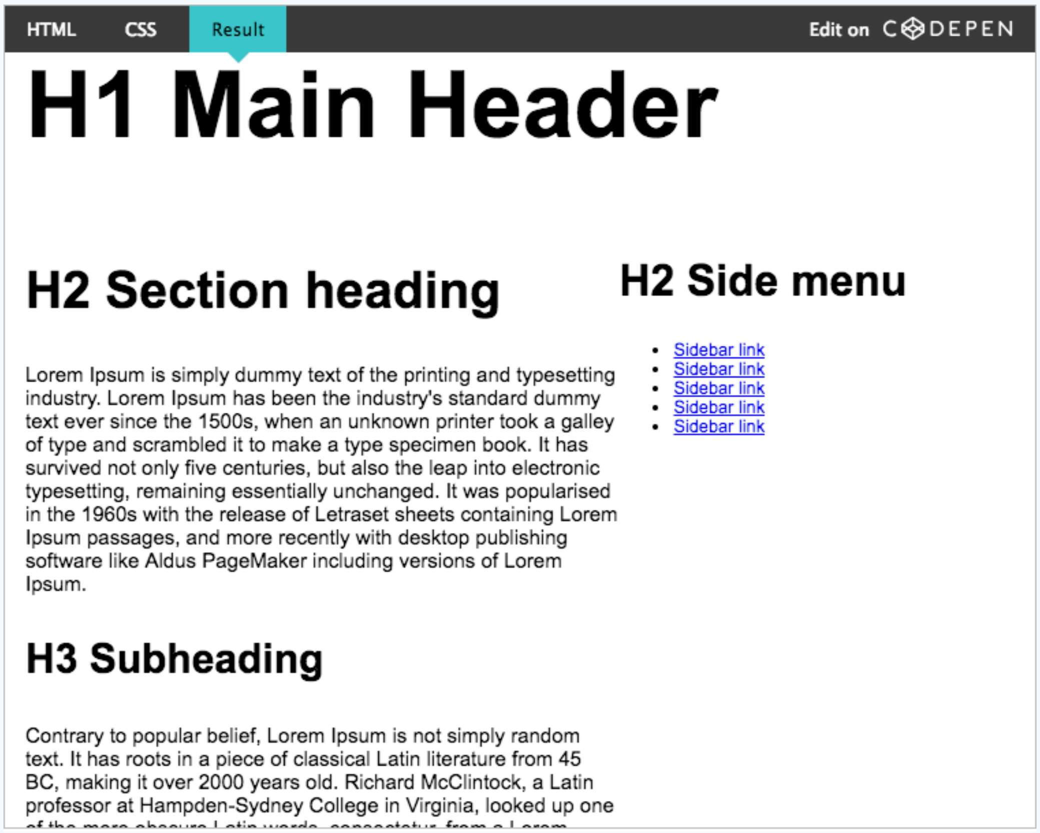
```
html {
    font-size: 62.5%;
}

body {
    font-size: 14px;
    font-size: 1.4rem;
}

h1 {
    font-size: 24px;
    font-size: 2.4rem;
}
```

While this solution seems to be close to the status of a "golden rule", there are people who advise against using it blindingly. Harry Roberts writes his own take on the use of rem units. In his opinion, while the **62.5%** solution makes calculation easier (as the font sizes in *px* are 10 times their rem values), it ends up forcing developers to explicitly rewrite all the font sizes in their website.

A third view comes from Chris Coyier of CSS-Tricks. His solution makes use of all three units we encountered so far. He keeps the root size defined in `px`, modules defined with `rem` units, and elements inside modules sized with `em`. This approach makes easier to manipulate global size, which scales the type in the modules, while the module content is scaled based on the module font size itself. Louis Lazaris discussed that latter concept in The Power of em Units in CSS.

In the example below you can see how Chris's approach would look:

Live Code (click image for link to CodePen)



H1 Main Header

H2 Section heading

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

H3 Subheading

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one

H2 Side menu

- Sidebar link
- Sidebar link
- Sidebar link
- Sidebar link
- Sidebar link

3-1. Chris Coyier's approach to font sizing

As you can see, there is no "silver bullet" solution. The combinations possible are limited only by the imagination of the developers.

Chapter

Using Rems with Media Queries

4

We've covered the font sizing with *em* and *rem*, now we'll look into using *rem* with media query breakpoints.

The use of *em* or *rem* units inside media queries is closely related to the notion of “optimal line length” and how it influences the reading experience. In September 2014, Smashing Magazine published a comprehensive study on web typography called Size Matters: Balancing Line Length And Font Size In Responsive Web Design. Among many other interesting things, the articles gives an estimate for optimal line length: between 45 and 75-85 characters (including spaces and punctuation), with 65 the “ideal” target value.

Using a rough estimate of 1rem = 1character, we can control the flow of text for a single column of content, in a mobile-first approach:

```
.container {
  width: 100%;
}

@media (min-width: 85rem) {
  .container {
    width: 65rem;
  }
}
```

There is, however, one interesting detail about rem and em units when used as units for media queries: they always keep the same value of 1rem = 1em = browser-set font size. The reason for this behavior is explained in the media query spec (emphasis added):

> *Relative units in media queries are based on the initial value, which means that units are never based on results of declarations. For example, in HTML,* ***the em unit is relative***

> ***to the initial value of font-size, defined by the user agent or the user's preferences, not any styling on the page.***

Let's see a quick example of this behavior:

Live Code

View Media Query Demo on CodePen.

First, in our HTML, we have a element where we will write the width of the viewport:

```
Document width: <span></span>px
```

Next we have two media queries, one with rem units and the other with em units (this uses Sass for simplicity):

```
html {
  font-size: 62.5%; /* 62.5% of 16px = 10px */

  @media (min-width: 20rem) {
    /* 20*16px = 320px */
    background-color: lemonchiffon;
    font-size: 200%;
    /* 200% of 16px = 32px */
  }

  @media (min-width: 30em) {
    /* 30*16px = 480px */
    background-color: lightblue;
    font-size: 300%; /* 300% of 16px = 30px */
  }
}
```

Finally, we use a bit of jQuery to display the viewport width on the page, updating the value when the window size changes:

```
$('span').text($(window).width());

$(window).on('resize', function(e) {
  $('span').text($(window).width());
});
```

We begin with the **62.5%** trick to show that the modified root font size does not have any effect on the values used for the media queries. As we change the width of the browser window we can see that the first media query kicks in at 320px (20 × 16px) while the second one becomes active at 480px (30 × 16px). None of the *font-size* changes we declared had any effect on the breakpoints. The only way to change the media query breakpoint values is to modify the default font size in the browser settings.

For this reason it doesn't really matter if we use `em` or `rem` units for media query breakpoints.

Using Rem Units For Scaling Documents

A third use we can find for rem units is to build scalable components. By expressing widths, margins, and padding in rem units, it becomes possible to create an interface that grows or shrinks in tune with the root font size. Let's see how this thing works using a couple of examples.

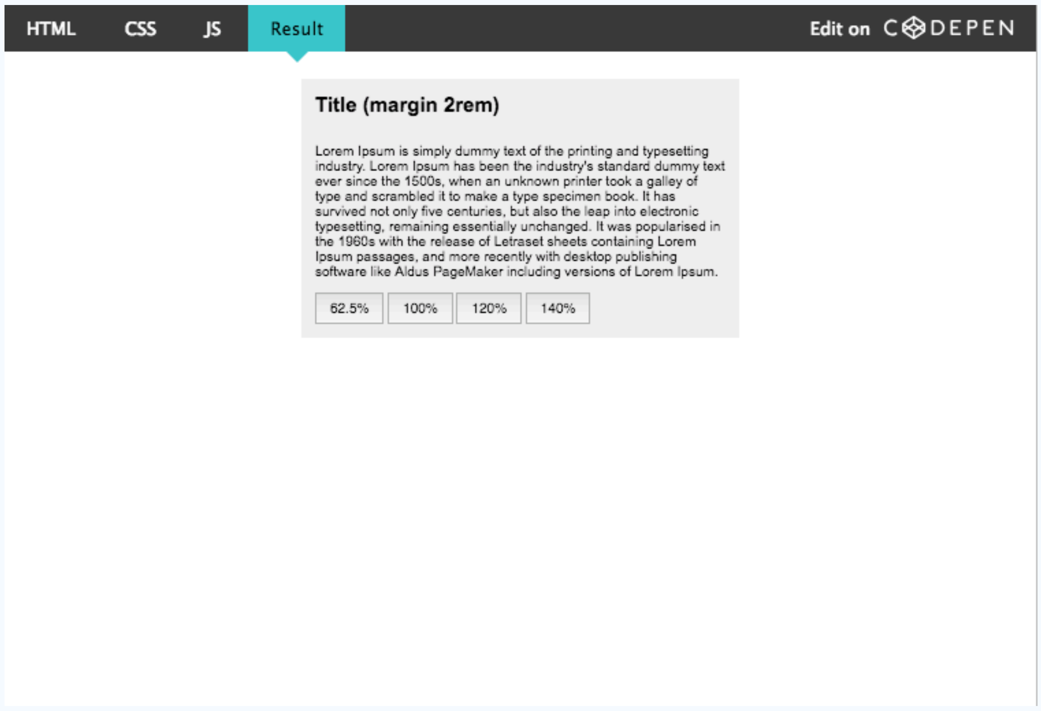
Live Code

Using rem Units for Scaling Documents Demo #1.

In this first example, we change the root font size using media queries. Just like in the previous section, the purpose is to customize the reading experience for the device used. As element padding values and margins are expressed using rem, the entire component scales with the device size.

Let's see another:

Live Code (click image for link to CodePen)



Title (margin 2rem)

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

4-1. Scaling documents with rem units

In the second example we do the same alteration using JavaScript. This time the user has control over the size of the interface, adjusting it to fit his needs. Add a way to store these custom values (using either a database, cookies or local storage) and you have the base of a personalization system based on user preferences.

We end here our encounter with CSS rem units. It is obvious that there are many advantages in using these units in our code, like responsiveness, scalability, improved reading experience, and greater flexibility in defining components. Rem units not a universal silver bullet solution but, with careful deployment, they can solve many problems that have irked developers for years. It's up to each one of us to unlock the full potential of rems. Start your editors, experiment and share your results with the rest of us.

Chapter

Media Queries

5

We popped in a little bit of media query usage last time, but today we'll get properly acquainted with them.

The *@media* rule allows conditional styling of elements.

The conditions can be based on the type of media or known characteristics of the device being used.

Combining media queries with fluid layout and flexible images, allows us to implement responsive web design.

Media Queries

Sometimes we only want certain styles to apply to certain types of devices or when certain characteristics of the device are true.

For example, we might want to remove the header and footer when printing a web page.

Using the query `print` will restrict the styles of the at-rule to the `print` media type.

```
@media print {
  .site-header, .site-footer {display: none;}
}
```

Other media types include

- all
- braille
- embossed
- handheld

- projection
- screen
- speech
- tty
- tv

The only two media types I use are *print* and *screen*, which is a bit of a catch-all for any screen-based device including mobile devices, tv and projection.

Device Queries

We can check more fine-grained details about the device being used by passing a query into the *@media* rule. A common property to query is the `min-width` of the browser window:

```
body {
  font-size: 0.75em;
}
@media (min-width: 600px) {
  body {
    font-size: 1em;
  }
}
```

In this example the initial `font-size` for all devices is `0.75em` but if the device has a minimum width of `600px` (ie. is `600px` or wider) then the `font size` will be increased to `1em`.

There are a number of things we can query about the device:

- `width min-width max-width`
- `height min-height max-height`
- `device-width min-device-width max-device-width`
- `device-height min-device-height max-device-height orientation`

- `aspect-ratio min-aspect-ratio max-aspect-ratio`
- `device-aspect-ratio min-device-aspect-ratio max-device-aspect-ratio`
- `resolution min-resolution max-resolution`
- `color min-color max-color`
- `color-index min-color-index max-color-index`
- `monochrome min-monochrome max-monochrome`
- `scan grid`

I use `min-width` and `max-width` a lot, `orientation` , `aspect-ratio` and `resolution` occasionally and `min-height` and `max height` from time to time. I've never used the others as far as I can remember.

Width is by far the most common thing to query about the device, but as the reported width and device width are often different, it's necessary to add the following *meta* tag to your HTML which will make them equivalent:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

The initial-scale is set to prevent devices zooming out to fit the whole site in the viewport. It's possible to set `maximum-scale=1` but then this removes the ability for a user to zoom the page in which isn't good user experience.

Combined Queries

It's possible to combine queries together using the `and` keyword:

```
@media screen and (min-width: 600px) and (max-width: 800px) { }
@media screen and (orientation: portrait) and (min-width: 800px) { }
```

It's also possible to use negation:

```
@media not screen { } { }
```

And limit applicability using `only`:

```
@media only screen { }
```

These `@media` blocks can contain any CSS you'd write elsewhere in the stylesheet and cascade the same way too. This means you will likely not have to write that much CSS to change the design for multiple devices.

Responsive Design

As `@media` queries allow the conditional styling when certain device characteristics are true, we can use them to control the styling of a page across a range of different devices or device sizes.

We can control fine details or big-picture layout. It's common for websites viewed on a large screen to have multiple columns of text and images, but this would be impossible to read on a screen one fifth of the width.

As building and modifying complex layouts is time-consuming, let's use a simple example of four boxes to represent four sections of a page. Each box contains an image and a few lines of text.

Without any styles applied, the images, text, and boxes stack on top of each other. We can space them out a bit and add some borders and backgrounds to make them stand out a bit more.

As the screen gets wider, the layout looks a bit stretched and the small amount of text starts looking odd compared to the size of the image. Around `500px`, we could add a `@media` query to create a two column layout instead of a one column layout.

```
@media screen and (min-width: 500px) {
  .box {
    float: left;
    width: 50%;
  }
}
```

As the screen gets wider again, we could fit 4 columns in so could change the *width* of each box to 25% instead.

```
@media screen and (min-width:500px) {
  .box {
     width: 25%;
  }
}
```

Because of how CSS styles cascade, we don't need to specify *float:left* again.

This approach of starting with the small screen and adding styles to make a more complex layout is known as Mobile First, as coined in the book of the same name by Luke Wroblewski.

Chapter

Media Queries Tips

6

Tip 1: Don't Use Device Specific Breakpoints

Hopefully this goes without saying, but just in case you need a reminder or you haven't come across this best practice before, I thought it was worth reiterating.

Device specific breakpoints are easily identified in your code with media queries that look like this (comments added for clarity):

```
/* ipad portrait */
@media screen and (min-width: 768px;) {}

/* ipad landscape */
@media screen and (min-width: 1024px;) {}

/* iphone */
@media screen and (min-width: 320px) and (max-width: 480px;) {}
```

These breakpoints have been set up for Apple devices and have "magic number" values such as `768px` or `1024px`.

What if a user's window is `1025px` or `1023px`?

The media queries wouldn't take affect and the styles for that device size would not apply.

Sometimes you may need a very specific value for your breakpoint (more on that in a second) but seeing these device specific breakpoints is a code smell as far as I'm concerned.

So what should you do instead?

Tip 2: Set Major *Breakpoints* And Minor *Tweakpoints*

When working on a responsive project, I tend to set arbitrary whole-number breakpoints that are approximately the dimensions of the majority of phones, tablets, and desktop/laptop devices.

I would tend to use the following major breakpoints (although sometimes this may be altered on a project by project basis):

```
/* large phones and small tablets */
@media screen and (min-width: 500px;) {}

/* tablets and small monitors */
@media screen and (min-width: 800px;) {}

/* laptops and desktops */
@media screen and (min-width: 1200px;) {}}
```

Using these breakpoints doesn't limit the design to only change at these points but gives a good foundation for working with the three major device types.

For content-based tweaking of the design (ie: when the content starts to look broken, unbalanced, or doesn't quite fit) you can then use tweakpoints to nudge elements around and polish the details of your project.

```
/* tweak position of share button */
@media screen and (min-width: 1150px;) {
  margin-right: 1em;
}
```

Tip 3: Use Em Or Rem As Your Breakpoint Units

Instead of `px`, use one of these relative units for better scalability if the user zooms the page or increases the size of the text. As an example, my major breakpoints above would look as follows when sized in ems.

```
/* 500px / 16px = 31.25em */
@media screen and (min-width: 31.25em;) {}

/* 800px / 16px = 50em */
@media screen and (min-width: 50em;) {}

/* 1200px / 16px = 75em */
@media screen and (min-width: 75em;) {}
```

OK, so... now what?

Ideas for what you can tackle next

There you have it, plenty of tips for handling responsive design with CSS. But if you're still keen to learn more, you've come to the right place!

- If you're looking for a guide to Bootstrap, one of the most stable and responsive front-end frameworks on the web, check out our coverage of Bootstrap.
- If you're more keen on a JavaScript framework, we also take a good look at React and Angular.
- For the full guide to responsive web design, check out our recently-updated book, *Jump Start Responsive Web Design, 2nd Edition*, by Chris Ward. In this book you'll learn how to support a myriad of devices with tools like Flexbox, CSS Grid, and various APIs for user context.