Responsive Web Design with HTML5 and CSS3
Second Edition

The changing way in which we access the Web means that there has never been a greater range of screen sizes and user experiences to consider. Web pages built to be responsive can adapt their content for not just today’s devices but tomorrow’s too.

Learn how to build websites with a “responsive and mobile first” methodology, allowing it to display effortlessly on every device that accesses it. Packed with examples, and a thorough explanation of modern techniques and syntax, this book provides a comprehensive resource for all things “responsive”.

The second edition covers all the most up-to-date techniques and tools needed to build great responsive designs, ensuring that your projects are future-proof.

Who this book is written for

Are you writing two websites: one for mobile and one for larger displays? Or perhaps you’ve already implemented your first RWD but are struggling to bring it all together? If so, this book gives you everything you need to take your websites to the next level.

Some HTML and CSS knowledge will help; everything else you need is included in the book.

What you will learn from this book

- Understand what responsive design is, and why it’s vital for modern web development
- Learn to write cleaner, faster, and more semantically-rich HTML5 markup and understand all the new key elements
- How to use CSS3 media queries to apply different styles to different device capabilities and understand evolving media queries coming in CSS4
- Learn how to load different images depending upon screen size, device resolution or context
- Conquer forms! Add validation and useful interface elements such as date pickers and range sliders with HTML5 markup alone
- Implement SVGs into your responsive designs to provide resolution-independent images, and learn how to adapt and animate them
- Learn how to use the latest features of CSS including Flexbox, custom fonts, nth-child selectors, CSS custom properties (variables), and CSS calc

Responsive Web Design with HTML5 and CSS3
Second Edition

Build responsive and future-proof websites to meet the demands of modern web users


Ben Frain

Responsive Web Design with HTML5 and CSS3
Second Edition

ISBN 471-1-7845-857-6

$ 39.99 US
£ 25.99 UK

Prices do not include local sales tax or VAT where applicable

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'The Essentials of Responsive Web Design'
- A synopsis of the book’s content
- More information on Responsive Web Design with HTML5 and CSS3 Second Edition
Ben Frain has been a web designer/developer since 1996. He is currently employed as a Senior Front-end Developer at Bet365.

Before the web, he worked as an underrated (and modest) TV actor and technology journalist, having graduated from Salford University with a degree in Media and Performance.

He has written four equally underrated (his opinion) screenplays and still harbors the (fading) belief he might sell one. Outside of work, he enjoys simple pleasures. Playing indoor football while his body and wife still allow it, and wrestling with his two sons.

His other book, Sass and Compass for Designers is available now. Visit Ben online at www.benfrain.com and follow him on Twitter at twitter.com/benfrain.
Preface

A responsive web design provides a single solution that looks great on a phone, desktop, and everything in-between. It will effortlessly respond to the size of the user’s screen, providing the best experience possible for both today’s and tomorrow’s devices.

This book covers every essential aspect of responsive web design. In addition, it extends the responsive design methodology by applying the latest and most useful techniques provided by HTML5 and CSS3, making designs leaner and more maintainable than ever before. It also explains common best practice methods of writing and delivering code, images, and files.

If you can understand HTML and CSS, you can build a responsive web design.

What this book covers

Chapter 1, The Essentials of Responsive Web Design, is a whistle-stop tour of the key ingredients in coding a responsive web design.

Chapter 2, Media Queries – Supporting Differing Viewports, covers everything you need to know about CSS media queries: their capabilities, their syntaxes, and the various ways you can wield them.

Chapter 3, Fluid Layouts and Responsive Images, shows you how to code proportional layouts and responsive images, and provides a thorough exploration of Flexbox layouts.

Chapter 4, HTML5 for Responsive Web Designs, covers all the semantic elements of HTML5, text-level semantics, and considerations of accessibility. We also cover how to insert video and audio into our pages with HTML5.
Preface

Chapter 5, CSS3 – Selectors, Typography, Color Modes, and New Features, gets to grips with the endless possibilities of CSS: selectors, HSLA and RGBA colors, web typography, viewport relative units, and a whole lot more.

Chapter 6, Stunning Aesthetics with CSS3, covers CSS filters, box shadows, linear and radial gradients, multiple backgrounds, and how to target background images to high-resolution devices.

Chapter 7, Using SVGs for Resolution Independence, explains everything we need to use SVGs inside documents and as background images, as well as how to interact with them using JavaScript.

Chapter 8, Transitions, Transformations, and Animations, our CSS gets moving as we explore how to make interactions and animations using CSS.

Chapter 9, Conquer Forms with HTML5 and CSS3, web forms have always been tough but the latest HTML5 and CSS3 features make them easier to deal with than ever before.

Chapter 10, Approaching a Responsive Web Design, explores the essential considerations before embarking on a responsive web design and also provides a few last minute nuggets of wisdom to aid you in your responsive quest.
The Essentials of Responsive Web Design

Only a few years ago, websites could be built at a fixed width, with the expectation that all end users would get a fairly consistent experience. This fixed width (typically 960px wide or thereabouts) wasn’t too wide for laptop screens, and users with large resolution monitors merely had an abundance of margin either side.

But in 2007, Apple’s iPhone ushered in the first truly usable phone browsing experience, and the way people access and interact with the Web changed forever.

In the first edition of this book, it was noted that:

"in the 12 months from July 2010 to July 2011, global mobile browser use had risen from 2.86 to 7.02 percent."

In mid-2015, the same statistics system (gs.statcounter.com) reported that this figure had risen to 33.47%. By way of comparison, North America’s mobile figure is at 25.86%.

By any metric, mobile device usage is rising ever upwards, while at the other end of the scale, 27 and 30 inch displays are now also commonplace. There is now a greater difference between the smallest and the largest screens browsing the Web than ever before.

Thankfully, there is a solution to this ever-expanding browser and device landscape. A responsive web design, built with HTML5 and CSS3, allows a website to ‘just work’ across multiple devices and screens. It enables the layout and capabilities of a website to respond to their environment (screen size, input type, device/browser capabilities).

Furthermore, a responsive web design, built with HTML5 and CSS3, can be implemented without the need for server based/back-end solutions.
Beginning our quest

Whether you're new to responsive web design, HTML5, or CSS3, or already well versed, I'm hoping this first chapter will serve one of two purposes.

If you're already using HTML5 and CSS3 in your responsive web designs, this first chapter should serve as a quick and basic refresher. Alternatively, if you're a newcomer, think of it as a 'boot camp' of sorts, covering the essentials so we're all on the same page.

By the end of this first chapter, we will have covered everything you need to author a fully responsive web page.

You might be wondering why the other nine chapters are here. By the end of this chapter, that should be apparent too.

Here's what we will cover in this first chapter:

- Defining responsive web design
- How to set browser support levels
- A brief discussion on tooling and text editors
- Our first responsive example: a simple HTML5 page
- The importance of the viewport meta tag
- How to make images scale to their container
- Writing CSS3 media queries to create design breakpoints
- The shortfalls in our basic example
- Why our journey has only just begun

Defining responsive web design

The term, "responsive web design" was coined by Ethan Marcotte in 2010. In his seminal A List Apart article (http://www.alistapart.com/articles/responsive-web-design/), he consolidated three existing techniques (flexible grid layout, flexible images/media, and media queries) into one unified approach and named it responsive web design.

Responsive web design in a nutshell

Responsive web design is the presentation of web content in the most relevant format for the viewport and device accessing it.
In its infancy, it was typical for a responsive design to be built starting with the 'desktop', fixed-width design. Content was then refloowed, or removed so that the design worked on smaller screens. However, processes evolved and it became apparent that everything from design, to content and development, worked much better when working in the opposite direction; starting with smaller screens and working up.

Before we get into this, there are a couple of subjects I'd like to address before we continue; browser support and text editors/tooling.

**Setting browser support levels**

The popularity and ubiquity of responsive web design makes it an easier sell to clients and stakeholders than ever before. Most people have some idea what responsive web design is about. The notion of a single codebase that will just work across all devices is a compelling offering.

One question that almost always comes up when starting a responsive design project is that of browser support. With so many browser and device variants, it's not always pragmatic to support every single browser permutation fully. Perhaps time is a limiting factor, perhaps money. Perhaps both.

Typically, the older the browser, the greater the work and code required to gain feature or aesthetic parity with modern browsers. Therefore, it may make more sense to have a leaner, and therefore faster, codebase by tiering the experience and only providing enhanced visuals and capabilities for more capable browsers.

In the previous edition of this book, some time was spent covering how to cater for very old desktop-only browsers. In this edition, we will not.

As I write this in mid-2015, Internet Explorer 6, 7, and 8 are all but gone. Even IE 9 only has a 2.45% worldwide share of the browser market (IE 10 is only 1.94% while IE 11 is rising nicely at 11.68%). If you have no alternative but to develop for Internet Explorer 8 and below, you have my sympathies and I'm afraid I must be upfront and advise you that there won't be a terrific amount you can use in this book.

For everyone else, you owe it to your client/paymaster to explain why developing for ailing browsers might be a mistake and investing development time and resource primarily for modern browsers and platforms makes good fiscal sense in every respect.

Ultimately however, the only statistics that really matter are yours. In all but extreme cases, the sites we build should at least be functional in every common browser. Beyond basic functionality, for any web project it makes sense to decide, in advance, what platforms you want to fully enhance the experience for, and which you are happy to concede visual/functional anomalies to.
You'll also find that practically, starting with the simplest 'base level' experience and enhancing (an approach known as **progressive enhancement**) is easier than coming at the problem from the opposite direction — building the ultimate experience first then attempting to provide fall backs for less capable platforms (an approach known as **graceful degradation**).

To exemplify why knowing this in advance matters, consider that if you were unlucky enough to have 25% of your website visitors using Internet Explorer 9 (for example), you'd need to consider what features that browser supports and tailor your solution accordingly. The same caution would be required if large amounts of your users are visiting with older mobile phone platforms such as Android 2. What you can consider a 'base' experience will vary depending upon the project.

If suitable data isn't available, I apply a simple and crude piece of logic to determine whether I should spend time developing a particular platform/browser version: if the cost of developing and supporting browser X is more than the revenue/benefit created by the users on browser X; don't develop specific solutions for browser X.

It's rarely a question of whether you could 'fix' an older platform/version. It's a question of whether you should.

When considering which platforms and browser versions support which features, if you aren't already, become familiar the [http://caniuse.com](http://caniuse.com) website. It provides a simple interface for establishing what browser support there is for the features we will be looking at throughout.
A brief note on tooling and text editors

It makes no difference what text editor or IDE system you use to build your responsive web designs. If the simplest of text editors allows you to write your HTML, CSS, and JavaScript efficiently, that's absolutely fine. Similarly there are no requisite pieces of tooling that are essential to get a responsive web design out of the door. All you actually need is something that enables you to write HTML, CSS, and JavaScript. Whether your preference is Sublime Text, Vim, Coda, Visual Studio, or Notepad - it matters little. Just use what works best for you.

However, be aware that there are more tools available now (often free) to negate many of the manual and time-intensive tasks of building web sites than ever before. For example, CSS processors (Sass, LESS, Stylus, PostCSS) can help with code organization, variables, color manipulations, and arithmetic. Tools like PostCSS can also automate horrible and thankless jobs like CSS vendor prefixing. Furthermore, ‘Linting’ and validation tools can check your HTML, JavaScript, and CSS code against standards as you work, eliminating many time wasting typos or syntax errors.

New tools come out constantly and they are continually improving. Therefore, whilst some relevant and beneficial tools will be mentioned by name as we go, be aware that something better may be just around the corner. Hence we won’t be relying on anything other than standards based HTML and CSS in our examples. You should however, use whatever tools you can to produce your front-end code as quickly and reliably as possible.

Our first responsive example

In the first paragraph I promised that by the end of this chapter you would know all you needed to build a fully responsive web page. So far I've just been talking around the issue at hand. It's time to walk the walk.

Code samples
You can download all the code samples from this book by visiting rwd.education/download.zip or via GitHub at https://github.com/benfrain/rwd. It's worth knowing that where individual examples are built up throughout a chapter, only the final version of the example is provided in the code download. For example, if you download the code samples for Chapter 2, Media Queries – Supporting Differing Viewports, the examples will be in the state they are at by the end of Chapter 2, Media Queries – Supporting Differing Viewports. No intermediate states are provided other than in the text.
Our basic HTML file

We will start with a simple HTML5 structure. Don't worry at this point what each of the lines do (especially the content of the <head>, we will cover that in detail in Chapter 4, HTML5 for Responsive Web Designs).

For now, simply concentrate on the elements inside the <body> tag. I'm pretty sure nothing there will look too unusual; a few div's, a graphic for a logo, an image (a tasty looking scone), a paragraph or two of text and a list of items.

Here's an abridged version of the code. For brevity I have removed the paragraphs of text in the code below as we only need to concern ourselves with the structure. However, you should know that it's a recipe and description of how to make scones; quintessentially British cakes.

If you want to see the full HTML file, you can download it from the rwd.education website.

```
<!doctype html>
<html class="no-js" lang="en">
<head>
  <meta charset="utf-8">
  <title>Our first responsive web page with HTML5 and CSS3</title>
  <meta name="description" content="A basic responsive web page – an example from Chapter 1">
  <link rel="stylesheet" href="css/styles.css">
</head>
<body>
  <div class="Header">
    <a href="/" class="LogoWrapper"><img src="img/SOC-Logo.png" alt="Scone O'Clock logo" /></a>
    <p class="Strap">Scones: the most resplendent of snacks</p>
  </div>
  <div class="IntroWrapper">
    <p class="IntroText">Occasionally maligned and misunderstood; the scone is a quintessentially British classic.</p>
    <div class="MoneyShot">
      <img class="MoneyShotImg" src="img/scones.jpg" alt="Incredible scones" />
      <p class="ImageCaption">Incredible scones, picture from Wikipedia</p>
    </div>
  </div>
  <p>Recipe and serving suggestions follow.</p>
</body>
```
By default, web pages are flexible. If you were to open the example page, even as it is at this point (with no media queries present), and resize the browser window you'll see the text reflows as needed.

What about on different devices? With no CSS whatsoever, this is how that renders on an iPhone:

![iPhone rendering of the example page without CSS](image-url)
As you can see, it's rendering like a 'normal' web page would on an iPhone. The reason for that is that iOS renders web pages at 980px wide by default and shrinks them down into the viewport.

The viewable area of a browser is known technically as the viewport. The viewport is seldom equivalent to the screen size of a device, especially in instances where a user can resize a browser window.

Therefore, from now on, we will generally use this more accurate term when we are referring to the available space for our web page.

We can fix that prior problem easily by adding this snippet in the <head>:

```html
<meta name="viewport" content="width=device-width">
```

This viewport meta tag is a non-standard (but de facto standard) way of telling the browser how to render the page. In this case, our viewport meta tag is effectively saying "make the content render at the width of the device". In fact, it's probably easier to just show you the effect this line has on applicable devices:
Great! The text is now rendering and flowing at a more 'native' size. Let's move on.

We will cover the meta tag and its various settings and permutations (and the standards based version of the same functionality) in Chapter 2, Media Queries – Supporting Differing Viewports.

**Taming images**

They say a picture is worth a thousand words. All this writing about scones in our sample page and there's no image of the beauties. I'm going to add in an image of a scone near the top of the page; a sort of 'hero' image to entice users to read the page.

![Image of a scone near the top of the page.](image.png)

Oh! That nice big image (2000px wide) is forcing our page to render more than a little wonky. We need to fix that. We could add a fixed width to the image via CSS but the problem there is that we want the image to scale to different screen sizes.

For example, our example iPhone is 320px wide so we could set a width of 320px to that image but then what happens if a user rotates the screen? The 320px wide viewport is now 480px wide. Thankfully it's pretty easy to achieve fluid images that will scale to the available width of their container with a single line of CSS.
I'm going to create the css/styles.css CSS file now that's linked in the head of the HTML page.

Here is the first thing I'm adding. Ordinarily I'd be setting a few other defaults, and we'll discuss those defaults in later chapters, but for our purposes I'm happy to open with just this:

```css
img {
    max-width: 100%;
}
```

Now when the page is refreshed we see something more akin to what we might expect.

All this `max-width` based rule does is stipulate that all images should be a maximum of 100% of their width (in that they should expand to 100% of their size and no more). Where a containing element (such as the `body` or a `div` it sits within) is less than the intrinsic width of the image, it will simply scale up to the maximum available space.
Why not simply width: 100%?

To make images fluid you could also use the more widely used width property. For example, width: 100% but this has a different effect. When a property of width is used then the image will be displayed at that width, regardless of its own inherent size. The result in our example would be the logo (also an image) stretching to fill 100% of its container. With a container far wider than the image (as is the case with our logo) this leads a massively oversized image.

Excellent. Everything is now laid out as expected. No matter the viewport size, nothing is overflowing the page horizontally.

However, if we look at the page in larger viewports, the basic styles start to get both literally and figuratively stretched. Take a look at the example page at a size around 1400px:

Oh dear! In fact, even around 600px wide it's starting to suffer. Around this point it would be handy if we could rearrange a few things. Maybe resize the image and position it off to one side. Perhaps alter some font sizes and background colors of elements.
Thankfully, we can achieve all this functionality quite easily by employing CSS media queries to bend things to our will.

**Enter media queries**

As we have established, somewhere beyond the 600px wide point, our current layout starts to look stretched. Let’s use CSS3 media queries to adjust the layout depending upon the screen width. Media queries allow us to apply certain CSS rules based upon a number of conditions (screen width and height for example).

---

**Don't set breakpoints to popular device widths**

‘Breakpoint’ is the term used to define the point in which a responsive design should change significantly.

When people first started making use of media queries it was common to see breakpoints in designs built specifically around the popular devices of the day. At the time it was typically iPhone (320px x 480px) and iPad (768px x 1024px) that defined these ‘breakpoints’.

That practice was a bad choice then, and it would be an even worse one now. The problem is that by doing that we are catering a design to a specific screen size. We want a responsive design—something that is agnostic of the screen size viewing it; not something that only looks at its best at specific sizes.

Therefore, let the content and the design itself determine where a breakpoint is relevant. Maybe your initial layout starts to look wrong at 500px wide and greater, perhaps 800px. Your own project design should determine when a breakpoint is needed.

---

We will cover the entire gamut of CSS media queries in *Chapter 2, Media Queries – Supporting Differing Viewports*, inventively titled **Media Queries**.

However, for the purpose of whipping our basic example into shape, we will concentrate on just one type of media query; a minimum width media query. CSS rules within this type of media query only get applied if the viewport is a minimum defined width. The exact minimum width can be specified using a raft of different length units including percent, em, rem, and px. In CSS, a minimum width media query is written like this:

```css
@media screen and (min-width: 50em) {
    /* styles */
}
```
The `@media` directive tells the browser we are starting a media query, the `screen` part (declaring 'screen' is technically not needed in this situation but we will deal with that in detail in the next chapter) tells the browser these rules should be applied to all screen types and the `and (min-width: 50em)` tells the browser that the rules should be limited to all viewports above 50em of size.

I believe it was Bryan Rieger (http://www.slideshare.net/bryanrieger/rethinking-the-mobile-web-by-yiibu) who first wrote that:

"The absence of support for media queries is in fact the first media query."

What he meant by that is that the first rules we write, outside of a media query should be our 'base' rules which we then enhance for more capable devices.

For now, simply be aware that this approach re-enforces our smallest screen first mentality and allows us to progressively layer on detail as and when the design necessitates it.

**Amending the example for a larger screen**

We’ve already established that our design is starting to suffer at around 600px/37.5rem width.

Therefore, let’s mix things up a little by way of a simple example of how we can lay things out differently at different viewport sizes.

Almost all browsers have a default text size of 16px so you can easily convert widths to rems by dividing the px value by 16. We will discuss why you might want to do this in Chapter 2, Media Queries – Supporting Differing Viewports.

First off, we will stop that main 'hero' image getting too big and keep it over on the right. Then the intro text can sit to the left.

We will then have the main portion of text, the 'method' that describes how to make the scones, on the left below with a small boxed out section detailing the ingredients over on the right.
All these changes can be achieved relatively simply by encapsulating these specific styles within a media query. Here’s what things look like with the relevant styles added:

It still looks essentially the same as it did before on smaller screens but adjusts to the new layout as soon as the viewport is 50rem or wider.

Here are the layout styles that were added:

```css
@media screen and (min-width: 50rem) {
  .IntroWrapper {
    display: table;
    table-layout: fixed;
    width: 100%;
  }

  .MoneyShot, .IntroText {
    display: table-cell;
    width: 50%;
    vertical-align: middle;
    text-align: center;
  }
}
```
That wasn't too bad was it? With only minimal code we have built a page that responds to the viewport size and offers a preferable layout as needed. By adding just a few more styles things look even easier on the eye. With those in place, our basic responsive page now looks like this on an iPhone:
And like this above 50rem width:

---

Occasionally maligned and misunderstood; the scone is a quintessentially British classic.

Recipe and serving suggestions follow.

**Method**

1. Heat the oven to 220°C (or gas mark 7). Tip the flour into a large bowl along with the salt and baking powder; then mix it all up. Add the butter in, then rub the butter in with your fingers until the mix looks like fine crumbs. When that is done, stir in the sugar.

2. Put the milk into a jug and heat in the microwave for about 20-30 seconds. It should be warm but not hot. Add the vanilla and lemon juice to the milk and then put that to one side and put a baking tray in the oven to warm.

---

These further visual embellishments don't add to the understanding of what's happening responsively, hence I have omitted them here but if you'd like to view the relevant code, download the chapter code at [http://rwd.education](http://rwd.education) or [https://github.com/benfrain/rwd](https://github.com/benfrain/rwd).

This has been a very basic example but it has encapsulated the essential methodology of building out a responsive web design.

To reiterate the essential things we have covered; start with 'base' styles, styles that can work on any device. Then layer enhancements on progressively as the viewport size and/or capabilities increase.

You can find the full specifications for CSS Media Queries (Level 3) here: [http://www.w3.org/TR/css3-mediaqueries/](http://www.w3.org/TR/css3-mediaqueries/)

There is also a working draft for CSS Media Queries (Level 4) here: [http://dev.w3.org/csswg/mediaqueries-4/](http://dev.w3.org/csswg/mediaqueries-4/)
The shortcomings of our example

In this chapter we've covered all the essential component parts of a basic responsive HTML5 and CSS3 powered web page.

But you and I both know that this basic responsive example is rarely the limit of what we're tasked with building. Nor should it reflect the limit of what we are capable of building.

What about if we want our page to respond to different light conditions? What about changing the size of links when people use different pointing devices (a finger rather than a mouse for example)? What about being able to animate and move visual elements simply, using nothing but CSS?

Then there's the markup. How do go about marking up pages with more semantic elements; article, section, menu, and the like, or make forms with built in validation (no JavaScript needed)? And what if we want to change the visual order of elements at different viewports?

Let's not forget images. We have fluid images in this example but if people visit this page on a mobile phone, they will need to download a large graphic (2000px wide no less) that will only be shown on their phone at a fraction of that size. That will make the page considerably slower to load than needed. Surely there's a better way?

And what about logos and icons? We've used a PNG in this example, but we could easily use scalable vector graphics (SVGs) to enjoy graphics with resolution independence. That way they will look pin-sharp, regardless of the resolution of the viewing screen.

Hopefully you have time to stick around, as these are the very questions we will answer in the coming chapters.

Summary

Well done, you now know and understand the essential elements needed to create a fully responsive web page. However, as we have just discovered, there are plenty of places where things could be improved.

But that's fine. We don't just want the ability to make competent responsive web designs, we want to be able to create 'best of breed' experiences. So let's press on.

First up, we will wrap our heads around all that Level 3 and Level 4 CSS Media Queries have to offer. We have already seen how a web page can respond to viewport width but there's so much more we can do right now – and a lot more fun stuff coming to your browser soon. Let's go and take a look.
Where to buy this book

You can buy Responsive Web Design with HTML5 and CSS3 Second Edition from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.