

FIFTH ANNIVERSARY EDITION

Hardboiled WEB DESIGN

by Andy Clarke



Hardboiled Web Design

by Andy Clarke

Published in 2015 by
Smashing Magazine GmbH
Werthmannstr. 15
79098 Freiburg
Germany

On the web: hardboiledwebdesign.com
Please send errors to errata@stuffandnonsense.co.uk

Development editor (first edition): Chris Mills
Technical editors: Vitaly Friedman and Stu Robson
Editor: Owen Gregory
Art direction: Andy Clarke
Layout: Markus Seyfferth
Front cover illustration: Natalie Smith
Part opening illustrations: Elliot Jay Stocks
Copyright 2015 Andy Clarke

All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy, recording or any information storage and retrieval system, without prior permission in writing from the publisher.
ISBN: 978-3-945749-37-1

For my wife Sue

“ I looked at Berin and laughed. He turned his head and stared right into the muzzle of his own gun. The killer’s face was a vile mask of hatred. Berin had his mouth open, screaming with all the furies of the gods dethroned, but my laugh was even louder. He was still screaming when I pulled the trigger.”

— *My Gun is Quick*, Mickey Spillane, 1950

Acknowledgements

To Vitaly Friedman and Markus Seyfferth and everyone at Smashing Magazine, for helping to bring this fifth anniversary edition to life and for giving me a platform to share my ideas.

To the best editor in the business, Owen Gregory, for being Lewis to my Morse one more time.

To Natalie Smith and Elliot Jay Stocks, for their cover and interior illustrations.

To Trent Walton and Jeffrey Zeldman, for their forewords.

To Marc Thiele, for my author photo.

To Rachel Andrew, Shane Hudson, Mandy Michael and Sara Soueidan, for reading the first edition again and helping to shape the content in the second.

To my dear friends, Rachel Andrew (for the second time), Paul Boag, Vitaly Friedman (again), Owen Gregory (again), Petra Gregorova, Jon Hicks, Leigh Hicks, Drew McLellan, David Roessli, Jared Spool and Jeffrey Zeldman (again), for being there when it mattered and for keeping me safe.

To our clients at Stuff & Nonsense, for agreeing to delay their projects for a second time and being OK with me ignoring their calls and emails while I concentrated on writing; and to Sue Davies, Steven Grant and Joe Spurling for putting up with my bad temper while I wrote.

Finally, but most importantly, to my family, because all that really matters is them.

About the Author

Andy Clarke's been called many things since he started designing for the web over fifteen years ago. His ego likes words like “ambassador for CSS”, “industry prophet” and “inspiring”, but he's proudest that Jeffrey Zeldman once called him a “triple talented bastard”.

Andy runs Stuff & Nonsense,¹ a small web design company based in North Wales in the United Kingdom, which specialises in creative visual design for websites and web applications. For seventeen years they've worked with clients from all around the world. You'll see examples of some of his designs later in this book. Andy's a popular public speaker and presents at web design conferences worldwide. He teaches web design techniques and technologies to professional web designers and developers at sold-out workshops all over the world.

He wrote the best-selling *Transcending CSS: The Fine Art Of Web Design*² (New Riders, 2006) and the first edition of *Hardboiled Web Design* (Five Simple Steps, 2010).³ He still writes occasionally on his blog And All That Malarkey,⁴ and is the host of popular web design industry podcast Unfinished Business.⁵ He tweets as @malarkey.

¹ stuffandnonsense.co.uk

² stuffandnonsense.co.uk/buy/transcendingcss

³ stuffandnonsense.co.uk/buy/hardboiledwebdesign

⁴ stuffandnonsense.co.uk/blog

⁵ stuffandnonsense.co.uk/podcast

About the editor

Owen Gregory⁶ is a professional editor, copy editor and proofreader based in Birmingham, United Kingdom. Owen has experience of working with small, independent and digital publishers from manuscript to print and digital editions. His particular expertise encompasses all aspects of front-end web design and development and he's worked with several well-known authors from the web industry.

About the reviewer

Vitaly Friedman⁷ loves beautiful content and doesn't like to give in easily. Vitaly is writer, speaker, author and editor-in-chief of Smashing Magazine. He runs responsive web design workshops and loves solving complex UX, front-end and performance problems in large companies.

About the reviewer

Stuart Robson⁸ is a freelance front-end developer based in Wiltshire. He helps companies and organisations with their front-end architecture and development workflows. He also speaks and writes about Sass and front-end development.

Previous contributors

The 2010 edition of this book was edited by Chris Mills⁹ and the technical editor was Tim Van Damme.¹⁰

⁶ twitter.com/fullcreammilk

⁷ twitter.com/smashingmag

⁸ twitter.com/StuRobson

⁹ twitter.com/chrisdavidmills

¹⁰ twitter.com/maxvolar

Foreword

Not every CSS design wizard still lives in his mum's basement and cries himself to sleep each night wearing a soiled Tron T-shirt. For there's also Andy Clarke: dapper, charismatic and perpetually brimming with ideas, insights and enthusiasm for the design of great experiences and the experience of great design.

The man is a walking epiphany, a King Midas of CSS-powered creativity. And the book you're now browsing may be his greatest gold classic yet. For here you'll learn why, when and how to use HTML5 and CSS3 in your daily work. Daily as in every day. Daily as in right now, today.

Every web designer should possess this book, but be warned, it is not for the timid. If you tremble at the thought of your web layout boasting rounded corners in one browser but not another; if the mere notion of even trying a CSS drop-shadow fills you with a sinner's remorse, Hardboiled Web Design is so not for you. Leave now. No judgements. Return to your safe, soft-boiled life. Okay, maybe that was a judgement.

But if you're among the restless, enlightened and daring few who embrace the future of web design, and know that we can't get there by clinging to the past, brother slash sister, has Andy got a book for you.

Jeffrey Zeldman

About Jeffrey Zeldman

What can I say about Jeffrey Zeldman¹¹ that hasn't been written a thousand times or more? Sure, he was the co-founder of the Web Standards Project during its formative years. Sure, he's one of the most recognisable faces in the web industry with his blog, magazine A List Apart,¹² conference An Event Apart¹³ and publisher A Book Apart,¹⁴ "for people who make websites".

Sure, he's the author of *Designing with Web Standards*, the book that popularised standards-based HTML and CSS. He's also my inspiration, my mentor, my critic and my friend. What more can I say?

¹¹ zeldman.com

¹² alistapart.com

¹³ aneventapart.com

¹⁴ abookapart.com

Foreword

If you'd have told me five years ago that the command-line would be a core part of my web design workflow I'd have likely laughed you out of the room. The same goes for compiling CSS or using something other than floats for layout.

As web designers, we stand on ever-eroding foundations of technologies, techniques, and tastes. While core concepts might remain constant, the innumerable orbiting details fluctuate as the web evolves. Rarely is there a single solution to a problem, and answers to most questions begin with, "It depends..." As thrilling as it is to work in an industry developing so rapidly, the rate of change can be exhausting and often terrifying. How do we keep up?

We work with a medium that encompasses the sum of all human knowledge, but the web's boundlessness can make staying up-to-date overwhelming. It can be time consuming to wade through disparate, often contradictory sources. Given the web's nonlinear nature, learning seldom feels comprehensive—more like grasping around in the dark.

It can be tempting to pillage pages for code snippets and quick fixes rather than taking time to digest and fully comprehend the context of the problem and the logic behind the solution. What does it say about me that at one point I knew the ideal search terms to find Chris Coyier's guide to flexbox better than I knew how to actually use flexbox itself? Ahem. Don't answer that.

Using a shortcut may be fine for a busy work day. However, like any discipline, it becomes necessary to carve out time and space to understand the techniques and tools we use in order to maintain proficiency and broaden our perception, ultimately allowing for more nimble problem solving in the future.

As a developing industry, we need books like *Hard Boiled Web Design* to organise and explicate the best practices for so many of the new technologies and techniques used on the web today. The glorious fact that the original *Hard Boiled Web Design* needs an update after just five short years is a testament to our industry's dedication to push the web beyond current capabilities. Thanks to Andy's diligent research, we can continue to equip ourselves for the next phase of the web's evolution.

Trent Walton

About Trent Walton

What can I say about Trent Walton?¹⁵ Yes, he's part of the three amigos web design studio Paravel Inc.¹⁶ based in Austin, Texas. Yes, he's been responsible for some of the finest examples of responsive web design at scale, including his work for Microsoft. Yes, he's written some of the most thoughtful pieces about designing responsively. Yes to all this. He's also one of the humblest people I know. He's my Wild West hero. Let's just say that.

¹⁵ trentwalton.com

¹⁶ paravelinc.com

About this book

If you've been working on the web for any length of time, your bookshelves may already be groaning and your ebook reader may be bulging under the weight of books about HTML and CSS. You may even own the first edition of this very book. Do you really need another one? *Hardboiled Web Design: The Fifth Anniversary Edition* is different.

It's for creative people who want to understand why, when and how to use the latest HTML and CSS technologies in their everyday design and development work. Not tomorrow or next week, but today. It won't teach you the basics of writing markup or CSS but if you're hungry to learn about how the latest techniques and technologies will help the websites and web applications you make be more creative as well as more responsive, *Hardboiled Web Design: The Fifth Anniversary Edition* is the book for you.

If you care about good, clean markup, you're in for a treat as we'll focus on how best to use the semantic elements contained within HTML. We'll cover the latest microformats2 and WAI-ARIA landmark roles, looking at how they'll reduce your reliance on presentational elements and attributes, helping to make your website perform faster.

If you're a designer who wants to learn about the creative opportunities offered by the latest CSS, this book will teach you how to use those properties in browsers that support them, and offers a creative approach on how to handle older, less capable browsers.

Why a new edition?

We first published *Hardboiled Web Design* in 2010. Although that's only five years ago, at the time I'm writing now, so much about the web that we design for has changed. In 2010, weeks before we launched our first edition, Apple launched the iPad and it went on to change many of the ways people interact with websites. Ethan Marcotte published his "Responsive Web Design" article only five months before and the inevitable changes it caused to the web – and the industry that designs and builds it – hadn't begun to sink in when we published *Hardboiled Web Design*. In 2010, designers and developers were in many ways looking to the past while coping with the limitations of older browsers, rather than to a future where more people access websites using a smart-phone than they do a conventional PC.

Five years on and much has changed significantly. The legacy browsers that some believed held back our creativity have faded into obscurity. We no longer need to write hacks for browsers that don't support properties such as border-radius, box-shadow, opacity and even RGBa colour values. Contemporary web browsers from Apple, Firefox, Google, Opera and even Microsoft all have high levels of support for CSS and, better still, they're close to parity on the properties they support.

For many of us, our bosses and clients, the focus today is on designing digital products rather than websites. Whatever we make, the way we design has changed as we've moved from designing static visuals of complete web pages to systems of components. We make prototypes using HTML and CSS earlier in our design process and we iterate by writing code, not by making more visuals. Our clients have become used to – in fact, many have come to expect – that we'll demonstrate our responsive design concepts to them on their own smartphones or tablets.

Yet despite the differences that five years have made, many of our attitudes to design and development have stayed the same. We might not be bothered by `border-radius` support any more but we're equally as frustrated by support for flexbox. In the CSS workshops that I teach I'm regularly surprised by the numbers of designers and developers who hold off from using CSS properties such as `border-image`, `background-blend-mode`, `filter` and multicolumn layouts because of lack of support in even contemporary browsers.

The emerging technologies may have changed but the ways we think about using them has not. That's why the approach I outlined in *Hardboiled Web Design* is still relevant and is perhaps more important in the mobile, multi-device and responsive web we design for today than it was when I first wrote it. Are you ready to get started? Is the engine running in your heap? Then buckle up, let's go.

First, some assumptions

This book is about using the latest HTML and CSS technologies so I'll assume that you're already familiar with writing well-structured, meaningful HTML markup and CSS to implement your designs. Do you need to know everything there is to know about CSS? No, although understanding selectors and current layout techniques will help you follow along. If you're new to CSS, I hope that you'll be inspired to learn more about what it means to be hardboiled.

What you'll need

So that you can see how the 'Get Hardboiled' examples are displayed across screens of all sizes and types, you'll need a Mac or PC with several current web browsers and their developer tools installed, as well as a smartphone, tablet or both. I recommend that you have the most up-to-date versions of the following browsers installed:

Safari

On the Mac, make sure you go to Safari's *Preferences*, click on the *Advanced* tab and check *Show Develop menu in menu bar*. This will give you access to Safari for OS X's developer tools. Safari is also the default browser and only rendering engine on Apple's iOS.



Chrome

Whereas Safari uses the WebKit rendering engine, Google's popular Chrome web browser now uses Blink, its own fork of WebKit. Chrome has extensions that help us design websites using a browser and test them during development.



Firefox

Install the latest version of Firefox and any available beta version. Firefox is still a popular browser because of its extensions.



Edge

While Edge's logo may be reminiscent of Internet Explorer, Microsoft's latest browser carries none of its baggage and, unlike other browsers, Edge only runs on the Windows 10 operating system for PCs, smartphones and tablets, and the Xbox console. Edge can't be used on previous versions of Windows.



Opera

Previous releases of the Opera browser used Opera's own Presto rendering engine, but more recent versions have used the same Blink rendering engine as Chrome.



You won't need any special code writing software, so feel free to use your favourite text editor. Mine is still Espresso.¹⁷

¹⁷ macrabb.it.com/espresso

Introducing our examples, ‘Get Hardboiled’

Throughout this book we’ll be working through examples I designed for the fictitious ‘Get Hardboiled’ site for fans of hardboiled detective fiction. It illustrates the fabulous things we can achieve when we use the most up-to-date technologies and leave behind old-fashioned attitudes and ideas about designing for the web.

You’ll find links to the full repository of code on GitHub.¹⁸

Now, pick up your hat, slip on your raincoat and leave your comfort zone behind, because you, ol’ buddy, are about to get hardboiled.

¹⁸ github.com/malarkey/hardboiledwebdesign



FIFTH ANNIVERSARY EDITION

Hardboiled WEB DESIGN

by **Andy Clarke**

It's time to get hardboiled

In hardboiled detective stories since the 1920s, crime, violence and characters—both good and bad—have been portrayed without a veneer of sentimentality. The term 'hardboiled' means tough, like an overcooked egg. The crimes are tough too, so the heroes have attitude, don't sugar-coat the truth and never play it cute. They—and, by association, we as readers—demand the truth, no matter what it takes and how rotten it might be.



Hardboiled heroes are almost always down at heel, usually broke, often drunk and living on a diet of black coffee and smokes—hey, that sounds like most web designers I know. They have a good woman to help them stay on the straight and narrow but don't

always treat her as well as they should. When a glamorous redhead walks in the room, a hardboiled hero can't help but turn his head. (OK, this is getting weird. I could be describing myself.)

Piled high with examples, 'Get Hardboiled' will inspire and teach you to use the latest HTML and CSS in everything you make for the web.

Must reads

Red Harvest

Dashiell Hammett 1929

The Maltese Falcon

Dashiell Hammett 1930

The Postman Always Rings Twice

James M. Cain 1934

Double Indemnity

James M. Cain 1943

The Big Sleep

Raymond Chandler 1939

Contents

Part 1 Getting Hardboiled

WHAT THE HELL IS HARDBOILED?	24
(GIVE ME THAT) OL' TIME RELIGION	32
THE WAY STANDARDS DEVELOP	41
IT DOESN'T HAVE TO LOOK THE SAME	50
ATOMS AND ELEMENTS	62
DESIGNING ATMOSPHERE	80

Part 2 Hardboiled HTML

DESTINATION HTML5	106
HARDBOILED MICROFORMATS 2	136
WAI-ARIA ROLES	164

Part 3 Hardboiled CSS

HARDBOILED FOUNDATIONS	172
FLEXIBLE BOX LAYOUT	199
RESPONSIVE TYPOGRAPHY	237
BORDERS	274
BACKGROUND IMAGES	294
GRADIENTS	307

Part 4 More Hardboiled CSS

BACKGROUND BLENDS AND FILTERS	330
TRANSFORMS	354
TRANSITIONS	388
MULTI-COLUMN LAYOUT	416
IT'S TIME TO GET HARDBOILED	439





GETTING HARDBOILED

The web we design and develop for has changed beyond all recognition since the introduction of smartphones and other mobile devices. But the way we design and demonstrate those designs to our bosses and clients, and our attitude to HTML and CSS, have changed very little.

In **Getting Hardboiled**, you'll learn what it means to be hardboiled. You'll discover why it's important to constantly re-evaluate concepts such as progressive enhancement and graceful degradation, and you'll find out the cold, hard truth about how standards are really developed. You'll find out how to create the atmosphere of a design independent of responsive layouts and how to demonstrate those designs to our bosses and clients. Above all else, you'll learn that responsive web design is an opportunity to make fabulous creative work, an opportunity that you should grab with both mitts.

No. 1

What the hell is hardboiled?

SINCE I WAS IN MY TEENS I've been fascinated by detective fiction. Not the English country house murders or whodunnit mysteries of Agatha Christie — oh no, the Sunday evening adaptations of those never did it for me — I'm talking about gritty, hard-hitting, anything goes stories from writers like Raymond Chandler, Dashiell Hammett and my own personal favourite, Mickey Spillane.

Turn back a few pages and read the quotation at the beginning of this book. That isn't from the notes I made during a client meeting, nor is it from the minutes of a W3C CSS Working Group meeting — although it could quite easily be. No, it's from one of my favourite books, the hardboiled classic *My Gun Is Quick* by Mickey Spillane.

Even if you're not a fan of detective stories, you might know a little about them or have seen a few hardboiled film noir movies. You might be familiar with Humphrey Bogart's portrayal of private detective Sam Spade in Dashiell Hammett's *The Maltese Falcon* from 1941.¹ *The Maltese Falcon* is the second best detective film ever made, after *Who Framed Roger Rabbit*.²

How about Stacey Keach? His 1980s portrayal of Spillane's Mike Hammer on TV was more poached than hardboiled but, still, better a slow detective than no detective is my motto.

Then, of course, there's this guy.

¹ [en.wikipedia.org/wiki/The_Maltese_Falcon_\(1941_film\)](https://en.wikipedia.org/wiki/The_Maltese_Falcon_(1941_film))

² en.wikipedia.org/wiki/Who_Framed_Roger_Rabbit



Want to read some hardboiled for yourself? I hope so, but not until you've finished this book. Start with a classic — the old ones are the best — perhaps Dashiell Hammett's *The Maltese Falcon* or Raymond Chandler's *The Big Sleep*. In the mood for archetypal hardboiled action with a big mug detective, dames and dirty cops? Mickey Spillane's Mike Hammer novels are my favourite. Start with *My Gun Is Quick* and *Vengeance Is Mine*.

In hardboiled detective stories since the 1920s, crime, violence and characters both good and bad have been portrayed without a veneer of sentimentality. The term 'hardboiled' means tough, like an overcooked egg. The crimes are tough too, so the heroes have attitude, don't sugar-coat the truth and never play it cute. They — and, by association, we as readers — demand the truth, no matter what it takes and how rotten it might be.

It's always been the heroes — Hammett's Sam Spade, Chandler's Philip Marlowe and especially Spillane's Mike Hammer — who have fascinated me most about hardboiled detective fiction.

What's with him?

Hardboiled heroes are almost always down at heel, usually broke, often drunk and living on a diet of black coffee and smokes — hey, that sounds like most web designers I know. They have a good woman to help them stay on the straight and narrow but don't always treat her as well as they should. When a glamorous redhead walks in the room, a hardboiled hero can't help but turn his head. (OK, this is getting weird. I could be describing myself.)

To a hardboiled hero, jamming a pistol into a guy's temple or ramming a fist into his guts is part of a day's work. When you're 'that guy', the one who can get the job done when no one else can, rules are for sissies — and cops.

Hardboiled detectives sometimes work alongside the police, but they're on the outside because they're also not afraid to break the law when they need to. Being hardboiled means they make their own rules to get a case cracked and see a bad guy behind bars — or dead. Laws, rules and conventions matter, of course, but sometimes those same rules can get in the way of justice being served. When we can't do what we know is right, we need heroes who aren't afraid to step outside to get the right thing done.

Hardboiled detectives do what cops and the rest of society can't — because a detective's actions aren't limited by the rules or conventions that society has imposed on itself. We cheer them on, no matter how ugly or how brutal they can be — we root for them because we need them. Web designers can learn a lot from hardboiled detectives. This brings us to the title of this book and an approach I've called *hardboiled web design*.

What's hardboiled web design?

'Hardboiled' web design is about never compromising on creating the best work we can for the web. Hardboiled is about challenging assumptions. Hardboiled is never being afraid to push boundaries, break rules or invent new ones. Hardboiled is stripping our markup to the bone to make it more adaptable to whatever the web might throw at it. Hardboiled is not hesitating to make the most of new technologies.

Being hardboiled won't be easy, but if you're ready to challenge yourself, light a smoke, take a lungful and steel yourself as you're in for a long night.

Here's to the pencil pushers: may they all get lead poisoning

In life, as well as on the web, we need rules, we need conventions, we need standards — but we should always use them to inform what we do, not define it, and certainly never limit it. Although the web's twenty-five years old (at the time I'm writing this), we've already developed standards for it — standards bodies like the W3C act as the guardians of so-called web standards technologies like HTML, CSS and JavaScript.

We've also built up a series of best practices, such as mobile first, progressive enhancement and responsive web design, dictating how to use these technologies to build websites that are usable, cross-browser compatible, accessible, visually appealing, indexable by search engines, and more.

But the world's far from perfect and these standards and best practices are only really recommendations — the W3C even uses this word to describe the specifications they maintain.

There's no legal entity and no other body that can force browser vendors and web professionals to adopt these standards and best practices, other than through peer pressure and common sense. If it wasn't for them, this would be a very different kind of book.

We've also built up a series of best practices, such as mobile first, progressive enhancement and responsive web design, dictating how to use these technologies to build websites that are usable, cross-browser compatible, accessible, visually appealing, indexable by search engines, and more.

When I first wrote this book five years ago it was standard practice to fight hard to create a website that looked and worked the same across all browsers, no matter what their capabilities. To do this meant making compromises such as avoiding using technologies not supported by all browsers.

Was that hardboiled?

Don't kid yourself, sweet cheeks. It wasn't and still isn't the way to evolve our craft or build a better web. This kind of old-fashioned thinking holds us back. It forces us to make excuses for not doing what we know is the right thing. The worst that we, as the current custodians of the web, can do is allow anything to limit what's possible.

“*But we have to do what our bosses and customers want! We have to do what they expect!*”

I've been around the block a few times — I know the score. But I also know it's possible to give our clients what they want while we use the most up-to-date features in HTML and CSS to expand our creative options. This is what hardboiled web design is all about.

Before we look at how to move beyond approaches we take for granted, let's ask ourselves why we're often so reticent to embrace new web technologies.

Nice shirt. Who's your tailor? Quasimodo?

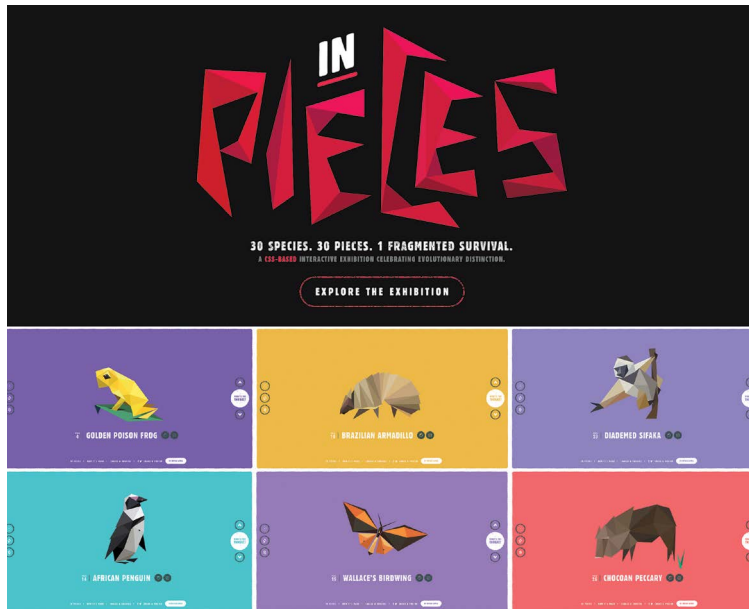
When my first book, *Transcending CSS*, went to print in 2006, there was very little support for CSS3. Only Firefox supported CSS3 multicolumn layouts and only Safari supported multiple background images. Even though *Transcending CSS* was described as an advanced book at the time, those two properties were about as advanced as it got.

By the time I wrote the first edition of *Hardboiled Web Design* five years later, the scene had changed beyond recognition. On the desktop, the dominant browser of the time, Internet Explorer, had dipped to less than sixty per cent market share.³ Rival browsers had gained ground and mobile internet browsing was growing fast.

We had an amazing array of CSS properties and most had been implemented by browsers at the time, including Internet Explorer 9. We had amazing CSS tools at our disposal so you might think we were all doing amazing things with them.

You'd be wrong. Think again, chump. Far from focusing on what we could do, most of us focused on what we couldn't. Far from embracing the possible, most of us complained about limitations. Far from getting excited, most of us whined and moaned.

³ <http://smashed.by/losingshare>



Using web standards-based HTML, CSS and JavaScript, it's possible to make emotionally engaging, powerful pieces of creative work such as Bryan James's⁴ incredible In Pieces⁵, portraits of thirty endangered animal species.

Who are you callin' a chump, chimp?

The pace of change in web design and development has been unprecedented. Technologies such as HTML, CSS and JavaScript have matured.

People's use of websites and web applications on smartphones and tablets has exploded to the point where more people use mobile devices than PCs. Responsive web design has gone from being a concept to a widely accepted approach for designing websites.

⁴ bryanjamesdesign.co.uk

⁵ species-in-pieces.com

The biggest changes have been in the ways we design and develop the web. To help cope with the demands of responsive design, many designers have moved from designing pages to designing systems made up of components. We've reinvented style guides in HTML and CSS and turned them into full-blown pattern libraries that serve as working design tools, not simply documentation.

To help manage style sheets on large scale websites, developers use Sass⁶ to add extends, mixins and variables to CSS. They've also introduced naming methodologies to HTML and CSS, including BEM (block, element, modifier)⁷ to make relationships between HTML elements and CSS styles clearer. Finally, the people we work for don't need to be convinced about the benefits of responsive websites any more — they ask for them.

Breaking it up

I may not be one of the hardboiled heroes I dream about — I never was much good in a brawl — but over the next few chapters I'll challenge many popularly held ideas about how we use new and emerging technologies. Then I'll set out a plan that satisfies everyone's needs while allowing us to push ourselves creatively.

I'm passionate about how we can make the best designs using the best, most up-to-date tools. So I won't be afraid to tell it like I see it. Don't expect me to be soft-spoken.

⁶ sass-lang.com

⁷ en.bem.info

No. 2

(Give me that) ol' time religion

PROGRESSIVE ENHANCEMENT has been one of the foundations of modern web development and my first exposure to it was an entry posted by Dave Shea on his blog,⁸ when he introduced what he called *MOS_e* — Mozilla, Opera and Safari enhancement. You should've heard of Dave, because he's the guy who created the CSS Zen Garden.⁹

Dave explained his *MOS_e* method as follows:

“*[A]fter creating a basic, functioning page in IE, you add extra functionality [for more capable browsers with advanced selectors]. [...] This is the only way we can keep moving forward in the next few years. Let's embrace it.*”¹⁰

Dave suggested we should first create a page that's accessible and usable to low-capability browsers, most notably earlier versions of Internet Explorer. Then — by using CSS child, sibling and attribute selectors — apply styles understood only by more capable browsers. You'll notice that Dave discussed how a page should work, not necessarily how a design should look.

Hold that thought.

⁸ mezzoblue.com

⁹ csszengarden.com

¹⁰ mezzoblue.com/archives/2003/06/25/mose

Scotch on the rocks... and I mean ice

Earlier that same year, Steve Champeon¹¹ wrote and spoke about what he termed *progressive enhancement*. If you haven't heard of Steve, he's the other guy who co-founded the Web Standards Project¹² alongside Jeffrey Zeldman.

“*Rather than hoping for graceful degradation, PE [progressive enhancement] builds documents for the least capable or differently capable devices first, then moves on to enhance those documents with separate logic for presentation, in ways that don't place an undue burden on baseline devices but which allow a richer experience for those users with modern graphical browser software.*”¹³

This notion of progressive enhancement is what many of us still regard as the ideal way to design and develop websites — starting with a design that can be rendered by less capable browsers, then layering on details that will only be seen by more modern and generally more capable browsers.

In practical terms this means starting with widely supported CSS selectors and properties, and only using new and emerging properties sparingly. In theory this approach to progressive enhancement makes sense, but in practice how we choose to apply the principles of progressive enhancement can easily lead to creative work that never reaches its full potential. Even though Steve used the term ‘inclusive web design’, I'm sure he never intended that we should limit our creativity to the capabilities of a lowest common denominator browser. Even if he did, can you guess when his and Dave's articles were written? 2003!*

¹¹ twitter.com/schampeon

¹² webstandards.org

¹³ hesketh.com/publications/inclusive_web_design_for_the_future

* The same year that former US president George W. Bush declared “mission accomplished” in Iraq.

Work's been kinda slow since cartoons went to colour

If you were carrying a bleeding edge MP3 player in 2003, you'd have had a massive 30Gb iPod in your pocket or purse. If you were designing, developing or just browsing the web in 2003, here's what software you were using:

- Apple Mac OS X 10.2 (Jaguar)
- Windows XP (SP2)
- Adobe Photoshop CS
- Macromedia Dreamweaver 7
- Microsoft FrontPage 2003
- Internet Explorer 6
- Apple Safari 1
- Mozilla Phoenix/Firebird
- Opera 7

In terms of software, we accept that time marches on and upgrades are both necessary and desirable. But in other ways — particularly the way we practically apply the principles of progressive enhancement — we still doggedly stick by received wisdom.

I'm as good as dipped

That isn't to say that the aims of progressive enhancement aren't laudable, quite the opposite:

- Basic content and functionality should always be accessible.
- Lean, clean, semantic markup should describe content.
- Style sheets should accomplish all aspects of visual design.
- Behaviour should be enabled using unobtrusive scripting.

When we develop following these principles, our content never relies on CSS or JavaScript to be available or accessible. When we use meaningful HTML, it will be lighter and more adaptable. CSS makes pages easier to format for screens of every size and type.

Progressive enhancement still has much to offer, but we must be careful not to allow adherence to its principles to limit creative potential. Instead of rigidly applying its ideas — especially with regard to visual design — we must continually re-evaluate how we use them to avoid our work becoming stale and ordinary.

I'm not bad, I'm just drawn that way

In her presentation “Enhancing Responsiveness with Flexbox”¹⁵ about the applications of CSS flexible box layout (flexbox), designer and author of *Flexible Web Design* Zoe Gillenwater advocates using flexbox to progressively enhance the implementation of a design.

The trouble is, enhancing still treats CSS properties, even powerful tools like flexbox, as visual rewards for people who use the most up-to-date browsers and devices.

Enhancement so often means starting at the bottom, with a lowest common denominator design for less capable browsers — and that's never good enough. When we use new and emerging CSS simply as a means — as Dan Cederholm wrote in his book *Handcrafted CSS*¹⁶ — to “enhance documents [to] allow a richer experience for those users with modern graphical browser software”, it's no wonder that so much design for the web today appears ordinary.

¹⁵ vimeo.com/124796320

¹⁶ handcraftedcss.com

S'MORES BUILDER

This is your chance to get creative. As long as you have a roasted marshmallow sandwiched between something with some chocolate added, I say you got a s'mores. So pick your frame, marshmallow, and candy, add an optional accessory or two, and build a crazy s'mores concoction.



FRAME

- ☐ graham crackers
- ☐ cinnamon grahams
- ☐ chocolate grahams
- ☐ Fudge Stripe cookies
- ☐ Oreos
- ☐ chocolate chip cookies
- ☐ Ritz crackers
- ☐ other:



MARSHMALLOW

- ☐ plain marshmallow
- ☐ chocolate 'mallow
- ☐ strawberry 'mallow
- ☐ giant 'mallow
- ☐ other:



CANDY

- ☐ Hershey's chocolate
- ☐ dark chocolate
- ☐ chocolate with almonds
- ☐ Nutella
- ☐ Reese's P.B. Cup
- ☐ white chocolate
- ☐ other:



ACCESSORIES

- ☐ peanut butter
- ☐ banana
- ☐ strawberries
- ☐ caramel sauce
- ☐ cream cheese
- ☐ bacon
- ☐ other:

NAME YOUR S'MORES:
[BUILD IT](#)

NEED SOME INSPIRATION?

CLASSIC plain graham crackers plain marshmallow Hershey's chocolate	ELVIS chocolate graham crackers plain marshmallow Hershey's chocolate peanut butter banana	PB&J plain graham crackers plain marshmallow Hershey's chocolate peanut butter strawberries	CHOCOLATE-COVERED STRAWBERRY chocolate graham crackers plain marshmallow dark chocolate strawberries
THANKSGIVING CASSEROLE cinnamon graham crackers plain marshmallow Hershey's chocolate sweet potato puree	NUTTY almond thins cookies plain marshmallow Nutella	BANANA PUDDING Nilla Wafers plain marshmallow white chocolate banana	THE MONSTER chocolate graham crackers chocolate marshmallow Reese's peanut butter cup caramel sauce
SALTY-SWEET Ritz crackers plain marshmallow Hershey's chocolate peanut butter	TRIPLE CHOCOLATE chocolate graham crackers chocolate marshmallow dark chocolate	FRUIT SALAD plain graham crackers plain marshmallow white chocolate banana strawberries	COOKIES 'N' CREAM Oreos plain marshmallow Hershey's Cookies 'n' Creme bar

Over the past few years, Zoe has produced some of the most educational and illuminating examples of practical applications for flexbox. This edition of *Hardboiled Web Design* wouldn't have been possible without her teaching: zomigi.com/publications/#pub-fwd

That's because when we start at the bottom and design for the capabilities of the lowest-performing browsers first, there's only so far up we can reach.

I had to shake the weasels

The hardboiled approach to web design doesn't accept that our creativity must be limited by the capabilities of older, less capable browsers and devices. Instead, we should take full advantage of new technologies, and design every user's experience so that it's appropriate to the capabilities of the browser they are using. This way we can make the most of everything that more capable browsers and emerging technologies have to offer, enabling us to reach higher and design better.

I can guess what you're thinking. Isn't this just graceful degradation?

You've been hanging around rabbits too long

The flip side to progressive enhancement — graceful degradation — ensures that when styles and scripts are not available or understood, the content of a document will remain accessible. Taking a graceful degradation approach means that a website's functionality will always be usable, albeit to a lesser extent and perhaps with a lower fidelity design, and its content will remain accessible.

This is how we handle things down in Toontown

Considering accessibility and how websites function in older or less capable browsers is a fundamentally important part of what we do. But the term graceful degradation, as traditionally applied to web design, implies that we should compromise.

To hell with being graceful!

The hardboiled approach pushes graceful degradation further and demands that we use our creative talents to make designs that are not only responsive to a device's screen size but also tailored to its browser's capabilities. Hardboiled web design aims to redefine graceful degradation for the challenges we face today.

If we're going to create the inspiring websites that our customers expect, we must look beyond how we've approached progressive enhancement and graceful degradation in the past. Simply rewarding people who use more capable browsers with enhancements or enrichments isn't enough.

Instead, we should take full advantage of new technologies, and craft every user's experience so that it's appropriate to the capabilities of the browser they are using. That will probably mean that designs will look different — sometimes very different — across browsers and across devices.

For some people this approach might seem radical — hardboiled even — but it makes better use of today's technologies and it is creatively liberating. It allows us to reach higher and design better, more inspiring and imaginative websites and applications.

I'll bake you a carrot cake

When progressive enhancement and graceful degradation were first described, the web was an altogether different place. There were relatively few differences between competing browsers in terms of absolute support for new features. Today, that's all changed. The gap between the most capable and the least capable browsers is wider than ever. In contemporary browsers there's solid support for even the newest CSS selectors and properties:

- Selectors to bind styles to any element without using `id` and `class` attributes.
- More ways to work with transparency, including RGBA, opacity and CSS filters.
- More ways to work with backgrounds and borders.
- Transforms to translate (move), rotate, scale and skew elements.
- Transitions to add subtle interactive effects.
- Keyframe animations that were previously only possible using JavaScript or Flash.

Support for CSS properties in current desktop browsers

	Safari 9	Chrome 47	Firefox 43	Opera 32	Edge
<code>background-blend</code>	●	●	●	●	●
<code>border-image</code>	●	●	●	●	●
Columns	●	●	●	●	●
Filter Effects	●	●	●	●	●
Flexbox	●	●	●	●	●
Gradients	●	●	●	●	●
Keyframe Animations	●	●	●	●	●
SVG	●	●	●	●	●
Transforms	●	●	●	●	●
Transitions	●	●	●	●	●

● Full support ● Partial support ● Prefix ● No support

CSS has given us the tools and creative freedom to make amazing things happen. To dismiss the creative possibilities of its newest properties as bells and whistles would be short-sighted and foolish. There are no technical reasons why we can't use every single one of these new properties today. There really is no need to wait.

So what's stopping us?

Nothing more than a few old-fashioned ideas.

Breaking it up

Neither progressive enhancement nor graceful degradation should be treated as doctrine or applied without thinking to everything we design for the web. Instead, they provide the starting points and now it's up to us to keep redefining how we adapt and apply their principles to suit the changing landscape of the web.

The way standards develop

No. 3

PEOPLE OFTEN MISTAKENLY BELIEVE that the W3C innovates new technologies, but its role is primarily as a standards body, not an innovation body. Its job is to standardise patterns of existing technologies. CSS Working Group specification writer Erika Etemad sums up the role of that group well.

“The Working Group exists for the purpose of standardization. If nobody's interested in implementing something, we're wasting our time writing a spec on it. Also, if only one implementor is interested in implementing something, we can't really make a cross-platform standard out of it.”¹⁷

For a long time I thought that the W3C's CSS Working Group first innovated, then released working drafts and recommendations. I imagined that when W3C recommendations were complete, browser makers would implement them (or not). The reality is that a standard is formed only when there is consensus on what's already been implemented.

“The CSS Working Group's work is considered a success if there are multiple independent complete and interoperable implementations of its deliverables that are widely used.”¹⁸

¹⁷ fantasai.inkedblade.net/weblog/2009/css-wg-charter

¹⁸ w3.org/Style/2008/css-charter

If we care about standards and want to ensure that our work conforms to them, what does this mean? How can we use new technologies when a standard for them hasn't yet been finalised? If we did, we'd miss out on years of creative opportunities.

So we needn't wait for HTML or CSS modules to become recommendations at the W3C: we can make the most of emerging standards today.

There's no such thing as one CSS3 specification

Unlike previous versions of CSS, CSS3 isn't a single, monolithic specification but is divided into modules. The CSS Working Group develops each module separately and according to the group's priorities:

“*CSS beyond Level 2 is being developed as a set of modules each of which may advance on the W3C Recommendation Track independently. Among them are modules for syntax, cascading and inheritance, and, of course, many aspects of typography, page layout and presentation.*”¹⁹

Breaking CSS3 into modules is good news for browser makers because it enables them to gradually implement new features to fit with their release schedules. It's also great news for us because it allows us to work with modules' properties as they're implemented — rather than waiting for a single, large specification to be complete.

Standards in development

The CSS Working Group's charter sets out the modules that are currently in active development. This isn't an exhaustive list. I've chosen ten modules that are most relevant to the work we do.

¹⁹ w3.org/Style/2008/css-charter#scope

CSS Animationsw3.org/TR/css3-animations

Animate the values of CSS properties over time, using keyframes. The behaviour of these keyframe animations can be controlled by specifying their duration, number of repeats, and repeating behaviour.

CSS Backgrounds and Bordersw3.org/TR/css3-background

Enables us to control the size, repetition and fit of a background image, use images within borders and round the corners of a box.

Compositing and Blendingw3.org/TR/compositing

Allows us to mix the backgrounds of several elements using blending modes similar to those you'll find in tools such as Adobe Photoshop.

Filter Effectsw3.org/TR/filter-effects

Apply filter effects similar to those you'll find in tools such as Adobe Photoshop using CSS.

CSS Flexible Box Layoutw3.org/TR/css3-flexbox

An important new tool for making layouts in CSS, flexbox enables us to easily arrange elements horizontally and vertically along two axes.

CSS Grid Layoutw3.org/TR/css3-grid-layout

An emerging standard for dividing available space within a layout into columns and rows. We won't cover CSS Grid Layout in this book. However, I recommend reading Rachel Andrew's book *CSS3 Layout Modules 2nd Edition* available from rachelandrew.co.uk/books/css3-layout-modules

CSS Multi-column Layoutw3.org/TR/css3-multicol

Generate pseudo-columns without additional markup, and control their quantity and width as well as gutters and dividers.

CSS Shapesw3.org/TR/css-shapes-1

Enables us to flow text around shapes set in CSS. Shapes can be geometric circles, polygons or rectangles and also created by images with alpha channels.

CSS Transformsw3.org/TR/css3-transforms

Matching many of the controls available in SVG, this module adds controls in CSS to translate (move), rotate, scale and skew an element.

CSS Transitionsw3.org/TR/css3-transitions

Different to animations, CSS transitions enable a property to transition smoothly between two states using CSS instead of scripting; for example, the colour of a hyperlink as it changes between normal and hover states.

Vendor-specific prefixes

When I demonstrate CSS in later chapters, you'll soon notice a recurring theme — not all browsers support the same properties in the same ways. For example, Edge and Safari on both Mac OS X and iOS support multicolumn layout in its native form:

```
.content {  
  columns : 10rem; }
```

But multicolumn layout needs vendor-specific prefixes to work in other browsers. For example, Chrome, Opera and the Android browser require the `-webkit-` prefix before the `columns` property, and Firefox requires the `-moz-` prefix. Cross-browser multicolumn layout therefore means writing rules several times — vendor-prefixed properties, followed by the W3C's unprefixed syntax.

```
.content {  
  -moz-columns : 10rem;  
  -webkit-columns : 10rem;  
  columns : 10rem; }
```

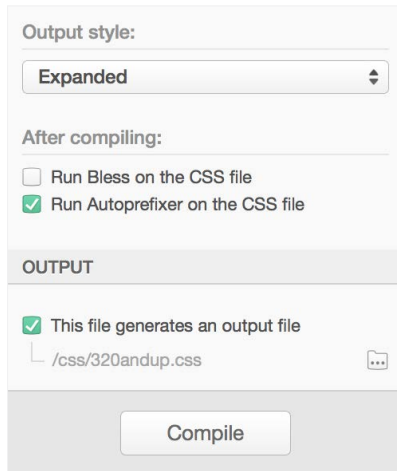
If you find writing multiple vendor-prefixed properties tedious, Autoprefixer²⁰ is a handy tool which parses your CSS and adds vendor prefixes to your rules using browser popularity and property support data from Can I Use.²¹

You might also use Lea Verou's `-prefix-free`.²² Simply include the `-prefix-free` script anywhere on your page and it will postprocess every linked or embedded style sheet to add those vendor-specific prefixes where needed.

²⁰ github.com/postcss/autoprefixer

²¹ caniuse.com

²² leaverou.github.io/prefixfree



You can use Autoprefixer in several ways depending on your development environment. Me? I'm over the moon that Autoprefixer is built into CodeKit,²³ the tool I use to process my Sass into CSS every day.

While standards emerge, writing long lists of vendor-prefixed properties is a hassle, so in 2010 Peter Paul Koch (PPK) called for browser makers to stop using them altogether.²⁴

“Vendor prefixes force web developers to make their style sheets much more verbose than is necessary. Why do we need to use several declarations for obtaining one single effect? Guys, let's stop the vendor prefix nonsense. Enough's enough.”²⁵

²³ incident57.com/codekit

²⁴ *Should I Prefix* is a useful site that tells you which CSS properties need vendor-specific prefixes. Press each property title to see code examples for that property: shouldiprefix.com

²⁵ quirksmode.org/blog/archives/2010/03/css_vendor_pref.html

I respectfully disagreed. PPK would have found plenty more to complain about if emerging properties had been implemented without vendor prefixes and each browser had rendered them differently.

Does writing multiple vendor-prefixed properties take more time? What, you expected being a web professional would be easy? There's still an upside though: we don't need to write the box model hack²⁶ any more. In fact, does anyone but me remember what they were?

Vendor-specific prefixes were originally intended for use only by browser makers, and the CSS2 specification warned us (authors) not to use them.²⁷

Out here in the real world, vendor-specific prefixes are often still a necessity so that we can use new and emerging properties today. Site-Point also failed to take the rapidly changing landscape of the web into account and suggested we played it safe on vendor-specific prefixes:

“*We don't recommend that you use these extensions in a real application. It's fine to use them for testing purposes, and for trying out CSS properties that haven't been implemented yet.*”²⁸

But safe isn't what the web needs now — it needs us to make the most of emerging standards and technologies so that we can create amazing things.

²⁶ tantek.com/CSS/Examples/boxmodelhack.html

²⁷ w3.org/TR/CSS2/syndata.html#vendor-keywords

²⁸ reference.sitepoint.com/css/vendorspecific

Are vendor-prefixed properties valid?

In the standards development process, properties prefixed with a `-` (dash) or an `_` (underscore) are reserved for vendor-specific extensions. Using these will render a style sheet technically invalid, but an invalid style sheet has been a small price to pay for all we've achieved using CSS standards as they've emerged.

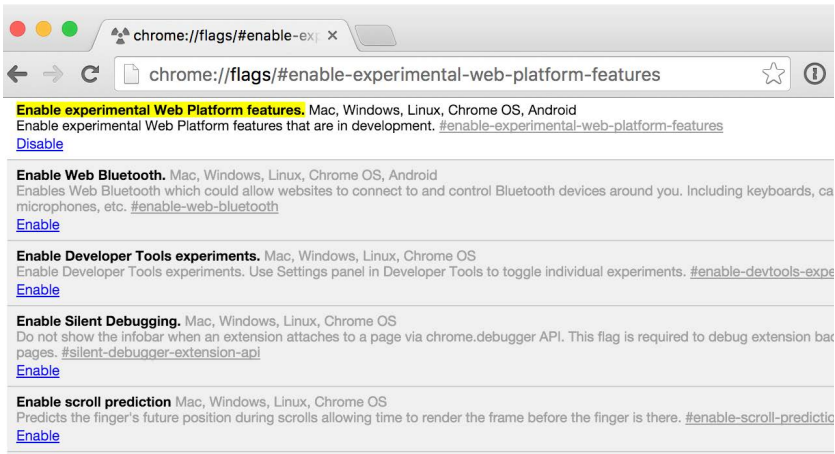
Browser flags

There's little doubt in my mind that, in general, we've benefited from vendor-specific prefixes because they've allowed us to not only experiment with emerging CSS properties but also use them in production code long before they've been developed into a standard by the W3C.

As with any experimental technologies, there are problems associated with using vendor-specific prefixes. When a browser vendor considers that a CSS property is unlikely to change further, they can choose to support it without their vendor-specific prefix.

However, it's common for us to keep outdated prefixes in our style sheets, often for years after they become unnecessary. This is especially true for prefixes of `border-radius`, still found in many sites, authoring tools and frameworks that have chosen not to remove them.

While still supporting properties prefixed with `-webkit-`, in Chrome, Google has implemented a system of flags that users must turn on in their browser to enable experimental properties. These flags allow you to try out new CSS properties while they're still being developed. One example of this is CSS shapes support. While Chrome supports shapes without a vendor prefix, users must have the experimental web platform features flag enabled in order to see them.



To see the effect of using CSS shapes and other experimental properties in Chrome, navigate to <chrome://flags/> in the address bar, search on the page for *Enable experimental web platform features* and press *Enable*.

In my experience, it's uncommon for users to know about the existence of flags, making it impossible for us to use the experimental properties they enable in our production code. Although this is sometimes inconvenient in the short term, in the long term we and our users will benefit from this sandboxing of experimental features.

Breaking it up

When we understand that CSS3 comprises a series of independent modules, we can leave behind the idea that we should wait until a single specification is finished before we use its properties. Instead, by carefully using vendor-specific prefixes where necessary, we can use these properties now; there's no reason to wait any longer. Yet even with the ever increasing rate of adoption of new CSS properties in browsers, there will always be differences between browser and device capabilities. Rather than hacking around these differences, we should learn to embrace them.

No. 4

It doesn't have to look the same

ONE PRECONCEPTION THAT'S LONG PREVENTED US from making the most of emerging technologies is that websites should look and be experienced exactly the same in every browser and on every device.

Dan Cederholm answers the question “Do websites need to look exactly the same in every browser?”²⁹ with an emphatic “No!” He’s right, too.



When we use a highly capable browser such as Safari, we see a design that is appropriate for a browser that supports **@font-face** web fonts.

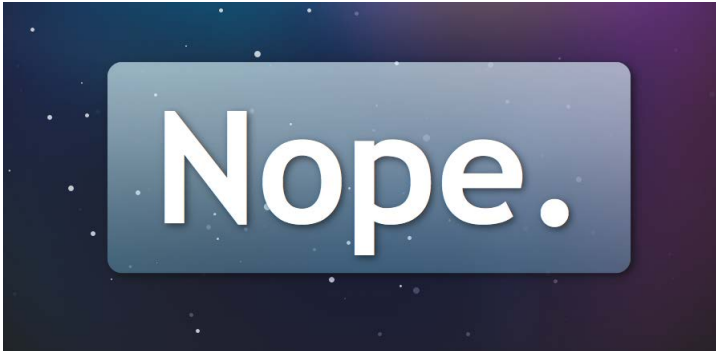


If a user has a less capable browser, such as Opera Mini, they won't see web fonts. That's OK, because Dan's design doesn't look broken and they won't know they're missing something.

What about experience? Dan's penchant for very long domain names answers that question too — “Do websites need to be experienced exactly the same in every browser?”

Of course they don't.

²⁹ dowebbsitesneedtobeexperiencedexactlythesameineverybrowser.com



Move your mouse over “Do websites need to be experienced exactly the same in every browser?” and the experience you have will depend entirely on the capabilities of the browser you’re using. This is the cornerstone of the hardboiled approach.

Dan may have a talent for choosing catchy domains but he wasn’t the first person to raise this issue.

“*[W]e need to step back from our endless battle to make it look the same across all platforms. We can’t make our site look the same on a PDA as a 21” monitor, we can’t make our site ‘the same’ for someone on a speaking browser, and although things are improving there are still differences in support and implementation of various W3C standards. Let go, its [sic] not going to look the same.*”³⁰

Rachel Andrew wrote that in 2002. Why are we still having that same conversation today? Especially when we now use an even wider variety of screen sizes and types — from watches to smartphones, tablets and high-resolution PCs — to access the web? Of course, some people still think that websites need to look and be experienced exactly the same in every browser, but those people probably still put two spaces after a period.

³⁰ edgeofmyseat.com/blog/2002-04-01-it-doesnt-have-to-look-the-same

Responsive design

The idea that websites need not look the same in every browser isn't new. John Allsopp explained this in his seminal A List Apart article, 'The Dao of Web Design', way back in 2000.

“*The control which designers know in the print medium, and often desire in the web medium, is simply a function of the limitation of the printed page. We should embrace the fact that the web doesn't have the same constraints, and design for this flexibility. But first, we must 'accept the ebb and flow of things.'*”³¹

A few months before we published the first edition of this book, Ethan Marcotte cleverly brought together several existing techniques and technologies including fluid grids, flexible media and media queries, into a practice that he called “responsive web design” and he wrote about it for A List Apart magazine.³²

In the five years since that article was published, responsive web design has become the de facto standard approach to designing for different sizes and types of screens, and led to some of the most significant changes in the ways that we design for the web. As well as helping designers and developers cope with a constantly changing landscape, responsive web design has also helped our bosses and clients accept that website designs should respond to the shape, size and capabilities of browsers and devices.

³¹ alistapart.com/article/dao

³² alistapart.com/article/responsive-web-design

What does browser support mean?

Many organisations maintain matrices that determine which browsers their sites support. With so many different browser and device capabilities to contend with today, some of the largest have redefined support to mean (as the UK's Government Digital Services (GDS) describes) “displays [content] correctly and key functions work.”³⁵

For GDS, it's critical that people have access to content and functionality on GOV.UK, so they currently test any browser that has over 2% usage. But unlike organisations that have quality assurance teams or marketing department staff dedicated to ensuring that their websites render near pixel-perfect across every browser in their matrix, GDS understands that:

“Not all browsers will render web pages in the same way, often there is a marked difference between browsers in the way that they handle technologies like cascading style sheets (CSS), HTML and javascript.”³³

Accepting that not all browsers should render websites in the same way will enable you to leave pixel-perfect rendering behind and instead focus on providing the most appropriate experience for the capabilities of a browser or device without anyone being left unable to access content or features because support has been dropped.

The BBC also accepts that pixel-perfect rendering shouldn't be made a priority over providing readable content and usable functionality to all users. Its Browser Support Standards describe sorting browsers into levels:

³³ gds.blog.gov.uk/2012/01/25/support-for-browsers

- Level 1: Supported
- Level 2: Partially supported
- Level 3: Unsupported

They accept small variations in experience within these levels of support, and even allow for using new and emerging technologies as long as they don't compromise a user's ability to access basic content or functionality.

“*There's nothing wrong with using all the latest bells and whistles to support funky features of newer browsers, but try to do it in a way that still allows users not supporting (or intentionally disabling) these features to access your basic content.*”³⁴

The BBC doesn't define what features in CSS it considers to be “bells and whistles” and neither does the Guardian in its browser support principles.³⁵ Instead of deciding on a list of browsers to support or not, the developers at the Guardian think about how their design will specifically impact their users:

- What is the core experience for all users?
- What do we enhance when we detect the user has a modern web browser with extra capabilities (e.g. web fonts, geolocation, etc.)

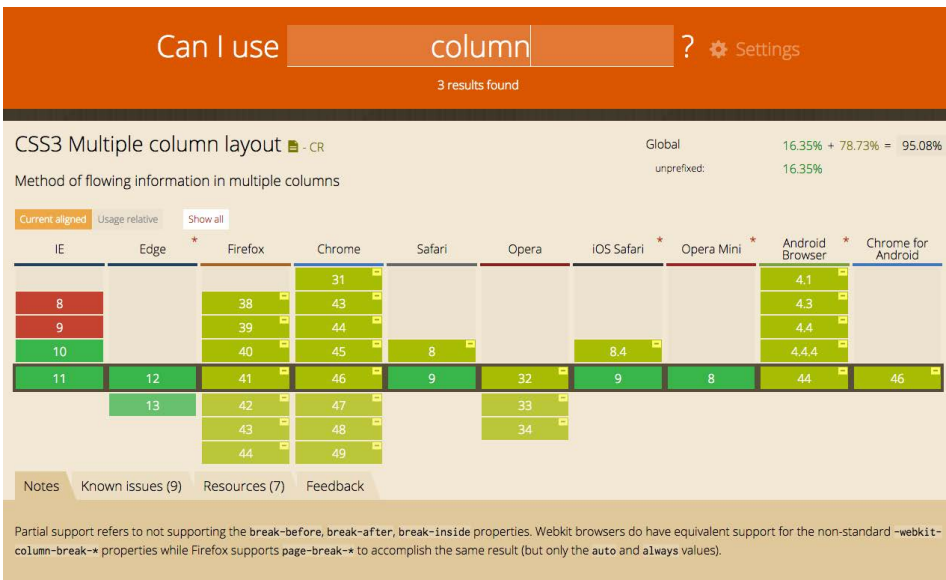
Considering the impact that specific properties may have on a feature-by-feature basis makes better sense to me than grading browsers as a whole. At Stuff & Nonsense, when we're asked by clients which browsers we'll support, we like to rephrase the question because we don't either support browsers or not support them.

³⁴ bbc.co.uk/guidelines/futuremedia/technical/browser_support.shtml#support_table

³⁵ github.com/guardian/frontend/blob/master/docs/browser-support-principles.md

Design fidelity

When a browser hasn't implemented an emerging property, don't grade it or exclude it completely, because that same browser may have excellent support for a different new property. For example, Safari has fully implemented flexible box layout yet only has partial support for multicolumn layout.³⁶ Instead of grading browsers, we decide on the importance of individual features within the context of the design that we're making.



Can I Use has not only become an indispensable reference for checking current levels of feature support in browsers, its data is behind many popular development tools including Autoprefixer.

³⁶ caniuse.com/#search=column

Sometimes a design element is important enough that we need everyone to see and experience it in the same way. A good example of this is a company's branding where it's probable that a client will want everyone to see their visual identity, corporate colours and typefaces. There are also plenty of instances where design elements are less important and need different considerations.

Considering the impact of specific properties means asking questions about how important they are to the fidelity of the design. In practice this means that when we add an element to a design, we consider how important it is for as many people — across a spectrum of browser capabilities — to see and experience it the same way.

For example, how important are web fonts? In some circumstances, choosing a specific typeface is as important to a brand as it is to the readability of its content.

What about multiple columns of text, rendered in CSS? Are they so important that everyone should see them or can we allow them to degrade gracefully; or, even better, get hardboiled and design an alternative? Now consider rounded corners, generated gradients or transparencies. How important are they to the fidelity of your design?

Circles of confusion

Long before I worked on the web, I trained professional photographers on how to use large-format view cameras. These film cameras featured swing and tilt movements to create a *plane of sharpness* — areas that appear sharp and areas that don't — within an image.

In photography, even the best camera lenses can't focus light onto a point. Instead, lenses focus light onto spots, or circles, in the film/image plane. These circles have dimensions, despite being microscopically small, and these are known as *circles of confusion*.

As the circles of light become larger, the more unsharp parts of a photograph appear; and when the circles are smaller, an image looks sharper, more in focus. This is the basis for photographic depth of field and with it comes the knowledge that no photograph can be perfectly focused, never truly sharp. Instead, photographs can only be *acceptably unsharp*.

Acceptable unsharpness

Although a modular, atomic process for design has been adopted by many of us, occasionally our clients still expect us to demonstrate how their completed website or application will look, in the form of a static visual made in Adobe Photoshop or Bohemian Coding's Sketch.

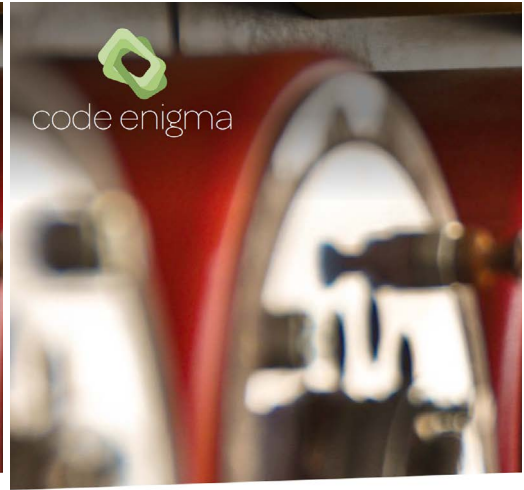
These visuals will contain every element of a design: branding, colour, typography in all its various forms, as well as backgrounds, borders, columns and gradients. We can think of these visuals as the highest-fidelity, sharpest interpretation of a design.

But how critical are these elements to the integrity of a design? Is a particular typeface vital to a user's experience of a brand? How important are blended backgrounds? Are columns really necessary?

These decisions determine how sharp an element should look across browsers and devices that have different capabilities and, therefore, how much time and resources we devote to achieving consistency between them. To help make those decisions, we can learn from the principle of circles of confusion.

The Filament Group takes a similar approach when they "Grade components, not browsers". Like Stuff & Non-sense, they build websites out of modular components and grade each of those components, adding layers of fidelity based on the features that a browser supports.³⁷

³⁷ filamentgroup.com/lab/grade-the-components.html



Come for the Drupal, stay for the relationship

At Code Enigma we don't build things and leave. We want to grow relationships with customers over time, we want to help businesses who use Drupal do so better, that's why we exist.

Come for the Drupal, stay for the relationship

At Code Enigma we don't build things and leave. We want to grow relationships with customers over time, we want to help businesses who use Drupal do so better, that's why we exist.

How critical is the choice of Omnes and Freight Text Pro to the integrity of Stuff & Non-sense's design for Code Enigma?³⁸ *Left:* The design featuring the Omnes and Freight Text Pro web fonts. *Right:* The same design using a system font, Times New Roman.

An environment for meaningful discussions

It may sometimes help to make a visual representation using three concentric rings:

- The innermost ring should contain the design elements that we decide will be sharpest. For example, if layout and typography must remain consistent, place them in this central circle.

³⁸ codeenigma.com

- In the middle ring, place elements that are important, but not vital, to a person's experience of a design. An example might be text that's set into columns created entirely by CSS. Browsers that have implemented multicolumn layout will split that text automatically into columns. For browsers that haven't, we might adjust our typography to take into account the lack of columns.
- The outer ring is for elements that we allow to degrade gracefully without us designing alternatives for them. If achieving blended backgrounds or filters in all browsers isn't important, save time by not creating images or employing JavaScript workarounds.

I've found plotting aspects of a visual design into circles of confusion to be a useful technique when explaining the natural differences between browsers to clients. It sets more realistic expectations and creates an environment for more meaningful discussions about progressive and emerging technologies. Best of all, it enables everyone to make better and more informed decisions about design implementation priorities.

Enhancing a brand

Some organisations have quality assurance teams or marketing department staff who are dedicated to ensuring that their websites remain pixel-perfect across every browser in their matrix. For them, experience and pixel-perfection are indistinguishable from a brand and so differences between browsers are seen as imperfections.

The efforts of these teams should now be directed away from cross-browser perfection and towards ensuring that brand values and a great experience are maintained and tailored for every capability of device. This change might not come easily — but it will come.

We should reassure bosses and clients that when we adopt a hard-boiled approach, differences between browsers will enhance a brand because we can precisely tailor experiences. We can tell them that differences are opportunities for us to be creative and therefore should be embraced.

You should be so lucky

I'm lucky that at Stuff & Nonsense I work with clients who are switched on technically and who appreciate that our time and their money are better spent on creating tailored, responsive designs rather than on workarounds to attempt cross-browser pixel-perfect rendering. But not all clients are the same.

Some care or know little about the changing capabilities of browsers and devices. How can we help them understand that websites needn't and can't look or be experienced the same in every browser? I've often heard that it's a web professional's job to educate clients. I couldn't disagree more. Our job isn't to educate, it's to design and build amazing websites.

If clients raise the thorny issue that a design looks different in an alternative browser or on a different size device, never be defensive. Explain that making a design that's tailored, depending on a browser's support for a property, will be better for everyone. This helps clients understand the positive impact of differences, rather than seeing them as imperfections.

If you work within a traditional institution, perhaps a large business, government department or in education, how can you sell the idea that your organisation's website should be accessible, responsive — even hardboiled?

Explaining these issues is less difficult today than it was when I first wrote *Hardboiled Web Design*. Back then, most people still experienced the web through a PC on their desk so the differences between browsers were harder to grasp. Today, it's a whole lot easier because more people access the web using a myriad of mobile devices than just PCs.⁴⁰

Chrome	32.45%
Safari	19.84%
Android	17.2%
UC browser	13.77%
Opera	9.98%
IE Mobile	2.13%
Nokia	1.6%
Blackberry	0.96%
NetFront	0.49%
Other	1.59%

Worldwide mobile browser usage,³⁹
February–July 2015

Breaking it up

The reality is that the web changed, and our work and our clients' expectations must move beyond the one-size-fits-all approach we have laboured over for so long if we are to make the most of what it has to offer. No two browsers and devices are the same, so to make the most of emerging technologies, we need to banish the notion that websites should look and be experienced exactly the same in every browser.

Perpetuating this idea will continue to cost us and our bosses and clients time and money on expensive hacks and workarounds instead of tailored experiences. It also prevents us from moving forward and embracing change. To help make change possible, we should explain that differences are not imperfections but are instead opportunities to enhance a brand experience by making websites more responsive.

³⁹ gs.statcounter.com/#mobile_browser-ww-monthly-201502-201507-bar

⁴⁰ smashed.by/popular-smartphones

No. 5

Atoms and elements

AT STUFF & NONSENSE,⁴¹ THE PAST FIVE YEARS have seen the biggest changes to the websites we make and our process for making them. Our design workflow has changed fundamentally as we've come to terms with responsive web design. While our approach to design won't suit everyone, it has proved successful, so in this chapter I'll explain our process.

Loving style guides

I've been fascinated by style guides and the ways that companies present their branding through these guidelines since before I worked on the web. I love seeing how varied similar components look and I especially enjoy seeing typography decisions documented. Sometimes, though, when I receive a company's branding guidelines as part of a project I'm working on, I still look for ways to add my own flourishes to a design while staying within the guidelines.

Struggling with branding guidelines

Traditional style guides that cover all manner of media, including packaging, print and web, aren't always helpful to web designers. For example, I worked on a project for HSBC. Its comprehensive guidelines stipulated that product names — Advance Account, Premier Account and Private Banking — should never be larger than the name of the bank. This rule made their product names unreadable when squeezed into a website banner only 80px tall.

⁴¹ stuffandnonsense.co.uk

SOLID COLOURS

Should you need to print using a solid colour

WHITE LOGO ON A DARK BACKGROUND
Middle section has an 80% opacity.

BRAND COLOURS



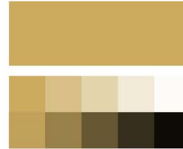
DARK GRAY

HEX: #474745
 RGB: 71, 71, 69
 CMYK: 66, 59, 60, 4
 PANTONE: 446



LIGHT GRAY

HEX: #767681



YELLOW

HEX: #d8ac39
 RGB: 216, 172, 89
 CMYK: 16, 32, 77, 1
 PANTONE: 124



NEUTRAL

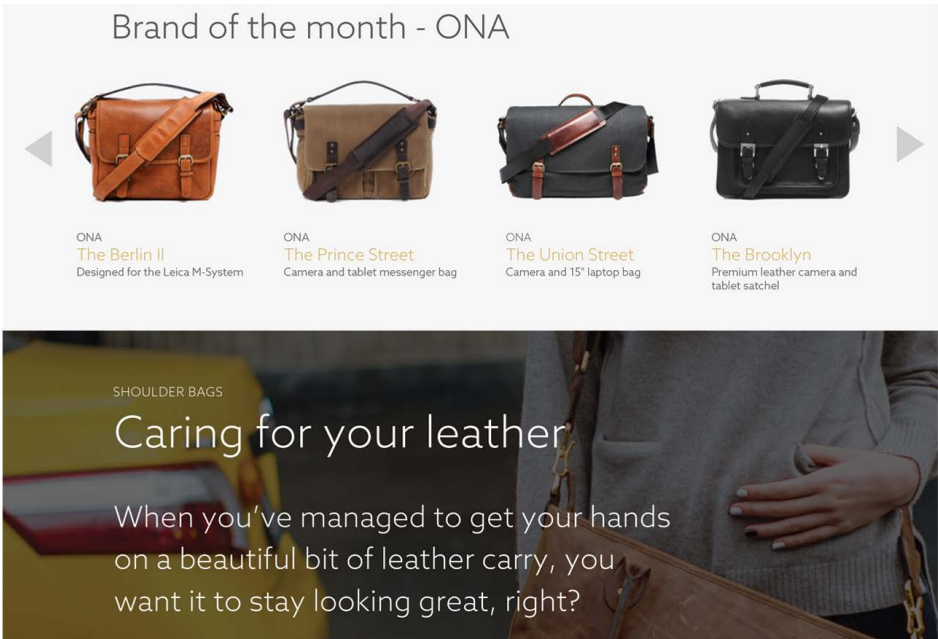
HEX: #bcb3a2

An example of the branding guidelines Stuff & Nonsense designed for Houden Bags

Communicating not documenting

For years, I'd not been satisfied using static visuals of whole web pages to present designs to our clients. These static visuals, made using Photoshop or Sketch, are incapable of demonstrating important aspects of a website's design. They set expectations that the finished website will be a facsimile of a frozen image and this clearly won't be the case. Most importantly, they're poor vehicles for conversations about design and they rarely lead to focused and productive discussions about specific issues.

Clients are often easily distracted when looking at static visuals. I might need a conversation about typographic hierarchy but a client wants their logo bigger. (Yes, I know. Old joke.) I may want a conversation about how search will function, but they comment on an out-of-date product photo. These conversations are unfocused because the static visuals we're basing it on are also unfocused.



Stuff & Nonsense also designed Houden Bags' key website pages.⁴²

When I used to show a client a static visual, they'd sometimes say, "I'm not sure about that design." I found this frustrating, especially after we'd spent hours working on a detailed rendition of their design. When I dug a little deeper I discovered that it often wasn't the important details they were commenting on. It wasn't the typefaces or type treatments we'd chosen. It wasn't the way we'd used colour, line work, borders or shading.

When we quizzed the client further they might say, "The sidebar should be on the left, not the right." In other words, they were talking about layout but expressing their criticism in terms of the design as a whole. Why was I surprised? What did I expect? After all, I'd shown them a visual that mixed all aspects of design into one image.

⁴² houdenbags.com

I knew there had to be a more effective way to present designs and keep clients focused on the aspects I needed to discuss, and over time my studio has developed ways to do that. Most importantly we start by presenting what I call the atmosphere of a design and then designing components separate from layout.

Describing atmosphere

Look up the word ‘atmosphere’ and the dictionary will tell you that it’s “the pervading tone or mood of a place, situation, or creative work”.⁴³

In the context of responsive web design, I think of atmosphere as comprising the combination of colour, typography and texture, distinct from layout. Let’s break that down.

Colour

We use colour to create mood and evoke an emotional reaction in someone using a website or application. We also highlight actions as part of an interaction vocabulary; for example, “What can I click on?”, “What have I clicked on already?”, “What’s potentially dangerous to click on?”

Typography

A good deal of a design’s personality comes from the typefaces we choose, how we use them in combination, treat them — line height, size and weight — and the white space around them.

Texture

Texture might include skeuomorphic textures like paper, stone or wood, but it doesn’t have to. In design atmosphere, texture refers to details including border styles, shading and the shapes of boxes and other elements.⁴⁴

I wrote about the concept of design atmosphere in Smashing Book 3, “Designing Atoms and Elements” in March 2012.⁴⁴

⁴³ oxforddictionaries.com/definition/english/atmosphere

⁴⁴ shop.smashingmagazine.com/products/smashing-book-3

Designing atmosphere

Let's look at a website design that's more than just a little atmospheric: the 2015 dConstruct conference.⁴⁵ What do you think makes its visual style so distinctive?

Of course, your eye is probably going to be drawn first to Paddy Donnelly's⁴⁶ characterful illustrations of this conference's theme, 'Designing The Future'. But there are other aspects that give this design its personality:

- Its combination of typefaces: Futura (naturally) – a humanist sans – and Lamplighter Script.
- The skewed content boxes.
- The playful way the speakers are presented.

Why am I highlighting dConstruct 2015? Visit the site on a device with any size screen and those aspects will look the same. Of course, their arrangement within the layout changes across responsive break-points, but the atmosphere of the website is preserved whether you're viewing it on a smartphone, tablet or PC.



The website for dConstruct 2015 was designed by Paddy Donnelly and developed by Graham Smith for conference organiser Clearleft.

Part of what gives dConstruct 2015 its personality is the use of web fonts Andes, Futura and Lamplighter Script. You'll learn about implementing web fonts later in this book.

⁴⁵ 2015.dconstruct.org

⁴⁶ left.com

In 2014⁴⁷, dConstruct’s atmosphere was altogether different, defined by its use of a flat, simple colour palette. The typeface chosen was PT Sans, a humanistic sans serif, and its utility carefully matches the stark lines around boxes that tie in with that year’s theme of ‘Living With The Network’.

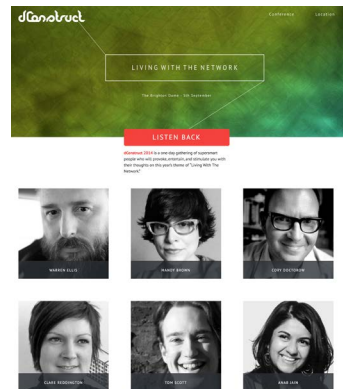
2013’s dConstruct website⁴⁸ had a different atmosphere altogether and so did the years before that, but every year the designs have something in common.

When you look closely at their colour, typography and texture — the aspects of a design that give each its unique personality — that atmosphere is present across responsive breakpoints.

If responsive web design has taught us anything, it’s that we should accept that colour palettes, textural backgrounds, borders and typographic designs maintain their characteristics across responsive screen sizes. In other words, the atmosphere of a design transcends layout.

Using front-end style guides and component libraries

At Stuff & Nonsense, we noticed an immediate improvement in conversations with clients when we started designing atmosphere and presented our designs in ways similar to the style guides that always fascinated me. We’ve since been through several iterations of our style guide format and we’ve followed with interest while a new form of style guide for the web has become popular.



dConstruct’s organisers have done an excellent job of preserving the conference’s websites since 2005, and looking back across a decade of design work illustrates very well how to determine the atmosphere of any website’s design.

⁴⁷ 2014.dconstruct.org

⁴⁸ 2013.dconstruct.org


Guiding visual identity

People looking after an organisation's brand across media need a style guide or branding guidelines to help them maintain consistency, placement and treatment of branding assets. These guidelines commonly begin by describing a brand's personality and values. Here's how chef Jamie Oliver's Fresh Retail Ventures describes his branding's personality:⁴⁹

“honest & challenging — being direct, open-minded and genuine in all we do; passionate & inspiring — true excitement and love for food and healthy living; approachable & fun — unpretentious, accessible and playful, encouraging everyone to have a go”

Most guidelines provide visual examples of how to use and not to use a logo. They also give rules for the typefaces, and should leave you in no doubt about colours and how to treat images. Jamie Oliver's branding guidelines even explain that “Jamie will expect all [photo] shoots to use natural light and freshly cooked food.”

REFLECTING JAMIE



Jamie Oliver the man is fundamental to our brand. Our identity is therefore a true reflection of Jamie, creating an iconic brand that captures the essence of who he is and what he stands for now and in the future. The things that are important to Jamie are in turn important to us:

Jamie the person	Jamie the brand
making it easy anyone can inspiring	simplicity, ease, practicality inclusive, informal, universal inspiration, energy, creativity

MASS: TONE OF VOICE*

Our tone of voice is purposeful passion. It is about bringing meaning and credibility to all that we say. It helps us make sure that our words are not only filled with Jamie's easy warmth, but also have a resounding reason to be there. Copy should always be a balance between purpose and passion, sharing knowledge and opinions in a truly engaging way that resonates with all people. We achieve this by using straight-talking and energetic language.

Straight-talking
Direct, honest, natural, informal, trusted language. Impart knowledge and expertise with others in a generous and easy, yet confident way. Use genuine and accessible words to explain product benefits.

Energetic
Punchy, inspiring descriptions that are full of Jamie's trademark passion and fun. Use fresh language peppered with playful words to instil enthusiasm and love for good food and good life.

The tone shifts slightly between Mass and Premium: in Mass, there is more freedom to be playful and fun because the products are everyday and simple, so play up the energetic tone. Whereas in the Premium Ranges the playful tone should be much subtler and the emphasis should be on straight-talking language to reflect the quality.

* This is the brand tone of voice and not instructions for writing copy in Jamie Oliver's own voice. Any requests for copy from Jamie should be directed to your Licensing Manager.

The guidelines from Jamie Oliver's Fresh Retail Ventures are a terrific example of a strict set of branding guidelines for his retail product business that covers broad branding principles, examples of packaging design and even tone of voice.

⁴⁹ issuu.com/bellfrog/docs/jamie-oliver-frv-brand-guidelines — You'll need Flash to view these brand guidelines. Find this quotation on page 7.

Making web design style guides

Sadly, I've often found that identity guidelines rarely translate well to the web. Some have forced me to work with colours that — while looking fabulous in a Pantone book — make uncomfortable viewing on screen. In other cases, specific typefaces aren't available as web fonts, and if they are, they don't make for comfortable reading. Experience has taught me that identity guidelines should inform designs we make for the web, not instruct them. What we need is a form of style guide that's specific to the web.

Logo Design Love curates a list of branding guidelines examples from companies including Code For America, GOV.UK and even a mid-1970s manual by NASA.⁵⁰

I don't think that anything can illustrate this more clearly than a project Stuff & Nonsense worked on with King's College Hospital NHS Foundation Trust⁵¹ in south London. Being part of the UK's National Health Service means the publications the Trust produces must take the NHS brand guidelines⁵² into account.

As King's College Hospital is an independent organisation, its website needn't adhere strictly to NHS brand guidelines. We made a design that represents its values rather than those of the NHS, and while the NHS brand guidelines gave us a starting point, we needed a style guide that worked for King's College Hospital on the web.

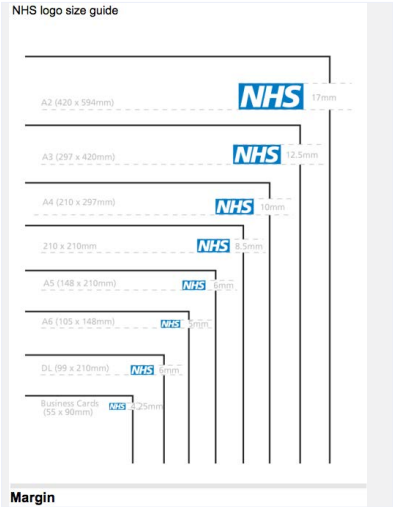
Visual identity guidelines are often delivered in PDF format, but responsive HTML, CSS and JavaScript are a far better medium for delivering a web design style guide. Instead of acting purely as a place to document styles, our web design style guide became a working tool during the design and development of the website.⁵³

⁵⁰ logodesignlove.com

⁵¹ www.kch.nhs.uk

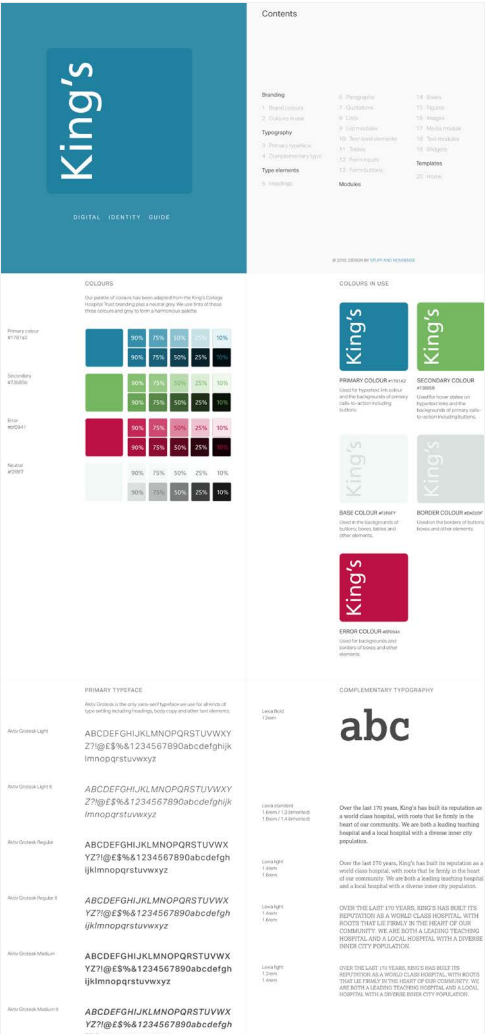
⁵² nhsidentity.nhs.uk/all-guidelines

⁵³ Anna Debenham has written *Front-End Style Guides*, a short but thorough guide to style guides for the web: maban.co.uk/projects/front-end-style-guides



Colour palette					
NHS Dark Green	Pantone® 342	C 100% M 0% Y 69% K 43%	R 0 G 107		
NHS Green	Pantone® 355	C 100% M 0% Y 91% K 6%	R 0 G 158		
NHS Light Green	Pantone® 368	C 65% M 0% Y 100% K 0%	R 91 G 191		
NHS Aqua Green	Pantone® 3272	C 100% M 0% Y 47% K 0%	R 0 G 170		
NHS Aqua Blue	Pantone® 312	C 100% M 0% Y 15% K 0%	R 0 G 173		
NHS Light Blue	Pantone® Process Blue	C 100% M 8.5% Y 0% K 6%	R 0 G 145		
NHS Dark Blue	Pantone® 287	C 100% M 69% Y 0% K 11.5%	R 0 G 56 B		

The NHS brand guidelines are a comprehensive set of rules that cover use of the NHS logo, typefaces (including several weights of Frutiger) and how to use photography and illustrations.



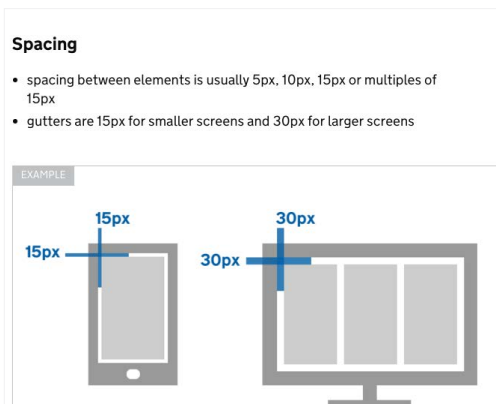
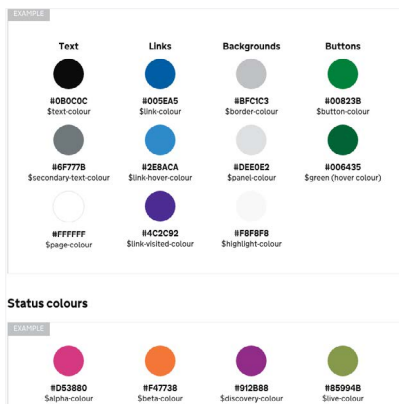
During our project with King's College Hospital, Stuff & Nonsense designed a comprehensive website design guide.

Developing a web design style guide

Components are the blocks from which we develop templates, and by adapting these templates we make pages. Our process of designing atmosphere goes hand in hand with designing individual components and building them into a library of these patterns.

Over the past several years there's been an flurry of people developing component libraries. Several examples of component libraries have been widely cited, including those made by the BBC,⁵⁴ GOV.UK,⁵⁵ MailChimp⁵⁶ and Starbucks.⁵⁷

styleguides.io is a growing collection of articles, books, discussions and examples of pattern libraries, code standards documents and content style guides.



The UK's Government Service Design Manual helps their designers and developers to achieve a consistent look across every part of the GOV.UK website.

⁵⁴ bbc.co.uk/gel

⁵⁵ govuk-elements.herokuapp.com

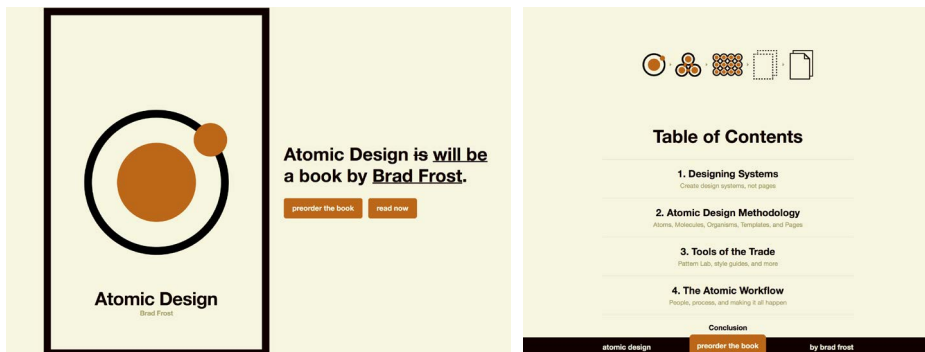
⁵⁶ ux.mailchimp.com/patterns

⁵⁷ starbucks.com/static/reference/styleguide

Atomic design

One design system that's become synonymous with responsive web design is Brad Frost's atomic design.⁵⁸ Brad first discussed atomic design in 2013 when he wrote:

“Lately I’ve been more interested in what our interfaces are comprised of and how we can construct design systems in a more methodical way.”⁵⁹



Brad describes atomic design as “a methodology used to construct web design systems.” That methodology even has a tool, Pattern Lab, to create atomic design systems.⁶⁰

Brad goes on to describe how his atomic design system consists of atoms, molecules, organisms, templates and pages:

- **Atoms:** The building blocks of HTML, elements including buttons, form inputs and labels.
- **Molecules:** Groups of elements that function together. For example, a label, input and button that combine to make a search form.

⁵⁸ atomicdesign.bradfrost.com

⁵⁹ bradfrost.com/blog/post/atomic-web-design

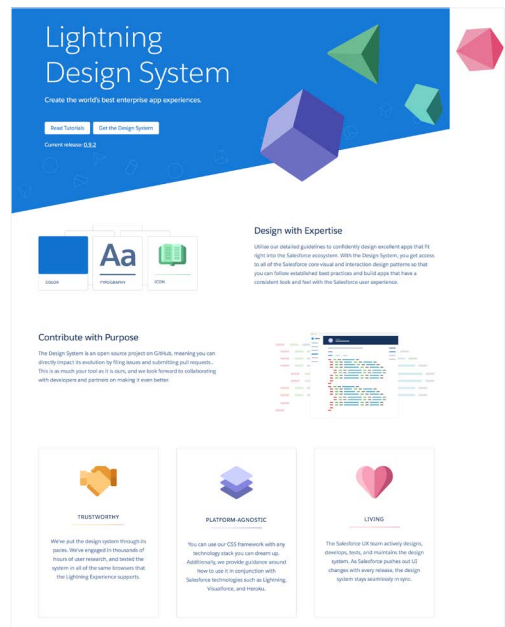
⁶⁰ patternlab.io

- **Organisms:** Groups of molecules joined together to form part of an interface.
- **Templates:** Mostly organisms combined to form page-level objects.
- **Pages:** Essential for testing the effectiveness of the design system.

Not everyone has been convinced about abstracting design the atomic design way, though. Mark Boulton wrote about his concerns and I'm forced to agree with him:

“Conformity and efficiency have a price. And that price is design. That price is a feeling of humanity. Of something that's been created from scratch. What I described is not a design process. It's manufacturing. It's a cupcake machine churning out identical cakes with different icing. But they all taste the same.”⁶²

When you work on aspects of a design separately from one another, it's often hard to know if they'll come together to form a consistent whole.



Done well, component libraries can be enormously useful resources. Salesforce's Lightning Design System is one of best available.⁶⁰

⁶¹ lightningdesignsystem.com

⁶² markboulton.co.uk/journal/design-abstraction-escalation

Without seeing everything together in one place, designs can easily feel disjointed and lack connectedness. It's also much harder to break free from any default styling we may have given our component library. You need only look at the thousands of similar looking sites built on frameworks such as Bootstrap and Foundation to appreciate that.

Reducing wasted time


Our component-based approach has made initial design stages more efficient. We're able to prototype our components faster and develop them into responsive templates in far less time than it took to design a set of complete pages using Photoshop or Sketch. Our projects run more smoothly and we have more focused conversations and fewer misunderstandings with our clients.

We've realised that developing components is only one part of the story and we need a suite of tools — including, maybe surprisingly to some, static visuals — as well as a flexible working environment in which to design.


Setting up a web design style guide

At Stuff & Nonsense we've found that to keep our designs looking original when we're using a component-based approach, we need to keep our process light and simple. Over-engineering a pattern library or web design style guide can too easily get in the way of designing, so we've built our own web design style guide toolkit using the simplest HTML, CSS and JavaScript.

To avoid complexity we pull atmosphere and components onto a single page and this allows us to experiment with ideas as freely as we can when using Photoshop or Sketch.




Over the last 170 years, King's has built its reputation as a world class hospital, with roots that lie firmly in the heart of our community. We are both a leading teaching hospital and a local hospital with a diverse inner city population.



Over the last 170 years, King's has built its reputation as a world class hospital, with roots that lie firmly in the heart of our community. We are both a leading teaching hospital and a local hospital with a diverse inner city population.

Media




King's College Hospital

Over the last 170 years, King's has built its reputation as a world class hospital, with roots that lie firmly in the heart of our community. We are both a leading teaching hospital and a local hospital with a diverse inner city population.


Media (reversed)


King's College Hospital



Over the last 170 years, King's has built its reputation as a world class hospital, with roots that lie firmly in the heart of our community. We are both a leading teaching hospital and a local hospital with a diverse inner city population.

Media list

 This ward has a 'Hospicom' entertainment system. (Pay as you go access to TV, radio, web and phone services.)

 You're allowed to bring your own laptops to the ward.

We've made the Stuff & Nonsense style guide available on GitHub. We hope that people will use it and contribute ways to improve it.⁶³

To keep our web design style guide easy to use we've separated our styles into:

- **Branding:** Logo marks and types in several configurations across responsive screen breakpoints.
- **Colours:** Tints of main, secondary, neutral and accent colours for buttons, backgrounds, borders and hyperlinks.
- **Typography:** Primary and secondary typefaces in several sizes and weights.

⁶³ github.com/malarkey/hardboiled-style-guide

- **Type elements:** Heading levels from 1–6. Paragraph styles such as lead (big), secondary (small), tertiary (smaller), and milli (smallest). Styles for block quotations and every type of list.
- **Other HTML elements:** Basic table styles, every type of form input and buttons in different sizes, styles and states.
- **Common component types:** Boxes, blog, event and news summaries and media components.

We include only what we need to start a project but can easily add styles for new components as we need them.

Including only what we need

If we're building a style guide toolkit to kick-start more than one project, it's often tempting to include many components (such as accordions, groups of buttons or tabs) that may come in handy one day. Be wary of doing that — when we bulk up our toolkits with components we don't need for the immediate job at hand, they can quickly become bloated.

There are literally dozens of pattern libraries and frameworks available that offer ready to use components. For example, Sass library Bourbon⁶⁴ includes Bitters⁶⁵ and Refills,⁶⁶ attractive and useful patterns of HTML, CSS and JavaScript. These components are useful for rapid prototyping, but think twice before you grab everything with both hands and pile it all into your web design style guide. Every extra component you add means more complexity and code to maintain, so include only what you need and be ruthless when disposing of components you don't need throughout a project.

⁶⁴ bourbon.io

⁶⁵ bitters.bourbon.io

⁶⁶ refills.bourbon.io

Staying away from complex frameworks

Front-end frameworks like Bootstrap and Foundation might be perfectly suited for developers who want off-the-shelf grids and theming systems, but a developer's needs are different to those of a designer working with HTML and CSS. Bootstrap and Foundation are powerful but designers don't need even a fraction of their features.

Instead of relying on a framework, we'll start our own web design style guide using just what we need, knowing that we can build on it later if we need to. We'll begin with colour, add typography and then common components that include type. We'll include common HTML elements before bringing everything together to create components.

We must remember to keep things simple and bear in mind that we're making a creative tool, not a development environment. That doesn't mean we can't make the most of developer tools where they're appropriate, so if we're used to dividing up style sheets into Sass partials, by all means carry on. Just remember, it's the design we're working towards that matters, not the sophistication of our tools.

Making collaborative tools

Not everyone who designs writes code, and despite me thinking for a long time that knowing how to write HTML and CSS is an essential part of being a web designer, I've realised that some designers are better when focusing on their graphic design, typography and colour skills. Designers needn't know how to write even a line of HTML or CSS.

That said, designers should understand modern web design tools and these are more frequently being made with HTML and CSS. I'd argue that it's even more important that designers are involved in designing these tools. That way they'll be more likely to know how to use them without much technical knowledge.

Making tools is a fabulous opportunity to get designers and developers working together. After all, these tools are intended to make collaborating easier. Made well, they can also reduce friction and eliminate the misunderstandings which can so easily occur when people use the wrong tools to communicate.



PATTERNS

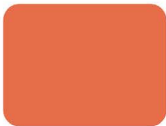
TEMPLATES

© 2015. Design by [Stuff and Nonsense](#).

Base colours for each of our fourteen species, derived from the WWF colour palette.



COLOURS IN USE



Primary colour #f37043

Used for hypertext link colour and the backgrounds of primary calls-to-action including buttons.



Secondary colour #007476

Used for hover states on hypertext links and the backgrounds of primary calls-to-action including buttons.

COLOUR TINTS

Our palette of colours has been derived from the WWF colour palette. We use tints of these colours to form a harmonious palette.

Primary colour #f37043



Secondary colour #007476



We've not only invested time in making our web design style guide a tool for use inside our studio, we've made it an accessible demonstration tool for our clients.

At Stuff & Nonsense, we've found that our web design style guide isn't just useful in getting designers and developers working together more efficiently. It's also helped us to work more closely with our clients. We realised that they were using our toolkit to demonstrate work to others, so we made it more approachable by including their visual identity and a contents pane, and we swapped our faux Latin text for snippets of their own content.

We found that simply making our web design style guide more accessible turned it into something our clients wanted to share around their organisations. People view our work more widely and on a larger spectrum of devices and this means we receive better feedback. I'm convinced our work has improved because of it.

Breaking it up

At Stuff & Nonsense our work and the processes and tools we use to make it have changed almost beyond recognition in the past five years. As we've learned to cope with the demands of responsive web design, we've learned to design atmosphere and elements using a web design guide. In the next chapter, we'll work on building a new web design style guide, starting with typography.

No. 6

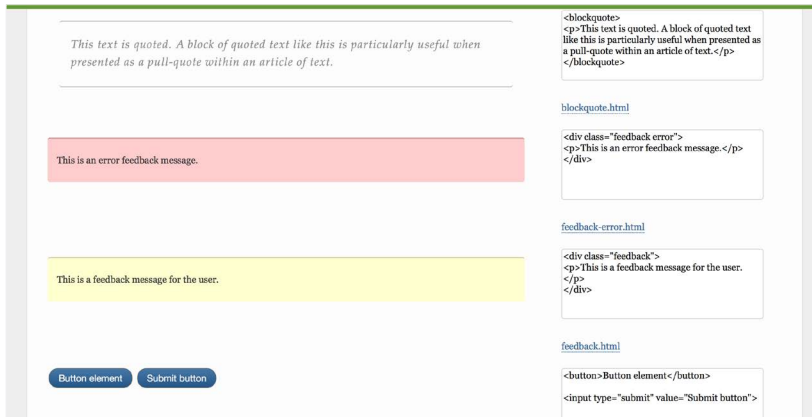
Designing atmosphere

WHEN TWO PEOPLE ARE HAVING AN ARGUMENT, you know it the minute you walk into a room. There's an atmosphere, and not a good one. A fabulous concert or maybe a football game can have incredible atmospheres too. It's often hard to describe, difficult to pin down. It's the way that something makes you feel. In a design, atmosphere comprises colour, typography and texture, separate from layout, and in this chapter we'll explore this notion of atmosphere and learn about designing it.


Starting with type

While people often focus on colourful graphics and images when they look at a design, a good deal of its personality can come from the typefaces you choose. That's why, when we made the web design style guide that we use for clients at Stuff & Nonsense, we focused first on typographic elements: headings, paragraphs, lists of all kinds, quotes, form and table text, and other miscellaneous elements.

When we're designing typography, we often look to balance personality with readability. Occasionally, we'll find those two attributes combined within one typeface; other times we need the combination of two typefaces: one that acts as workhorse, the other to compliment it with character and personality. In our studio, we spend a large amount of time selecting the right typefaces for the job.



Ours isn't an uncommon collection of HTML type elements. Both Jeremy Keith's pattern primer⁶⁷ and Bourbon Bitters⁶⁸ start from a similar place. You'll find plenty of frameworks that take this approach.


PRIMARY TYPEFACE		COMPLEMENTARY TYPOGRAPHY	
Bliss Light	Bliss is the only sans-serif typeface we use for all kinds of type setting including headings, body copy and other text elements.	Bliss Bold It 12rem	 <p>The National STEM Centre houses the UK's largest collection of STEM teaching and learning resources, in order to provide teachers of STEM subjects with the ability to access a wide range of high-quality support materials.</p>
	ABCDEFGHIJKLMNOPQRSTUVWXYZ ?!@£\$%&1234567890abcdefghijklmnopqrstuvwxyz	Bliss It 1.6rem / 1.3 (inherited from body) 1.8rem / 1.4 (inherited)	
	ABCDEFGHIJKLMNOPQRSTUVWXYZ ?!@£\$%&1234567890abcdefghijklmnopqrstuvwxyz	Bliss Light It 1.4rem 1.6rem	
	ABCDEFGHIJKLMNOPQRSTUVWXYZ ?!@£\$%&1234567890abcdefghijklmnopqrstuvwxyz	Bliss Light 1.4rem 1.6rem	
Bliss Regular	ABCDEFGHIJKLMNOPQRSTUVWXYZ ?!@£\$%&1234567890abcdefghijklmnopqrstuvwxyz	THE NATIONAL STEM CENTRE HOUSES THE UK'S LARGEST COLLECTION OF STEM TEACHING AND LEARNING RESOURCES, IN ORDER TO PROVIDE TEACHERS OF STEM SUBJECTS WITH THE ABILITY TO ACCESS A WIDE RANGE OF HIGH-QUALITY SUPPORT MATERIALS.	
Bliss Regular It	ABCDEFGHIJKLMNOPQRSTUVWXYZ ?!@£\$%&1234567890abcdefghijklmnopqrstuvwxyz	THE NATIONAL STEM CENTRE HOUSES THE UK'S LARGEST COLLECTION OF STEM TEACHING AND LEARNING RESOURCES, IN ORDER TO PROVIDE TEACHERS OF STEM SUBJECTS WITH THE ABILITY TO ACCESS A WIDE RANGE OF HIGH-QUALITY SUPPORT MATERIALS.	

When we designed for STEM Learning, we chose one typeface, the highly versatile Bliss⁶⁹ and used it in several styles and weights throughout the website.

⁶⁷ patternprimer.adactio.com

⁶⁸ bitters.bourbon.io

⁶⁹ typography.net/fontfamilies/view/27

PRIMARY TYPEFACE		COMPLEMENTARY TYPOGRAPHY	
Aktiv Grotesk Light	Aktiv Grotesk is the only sans-serif typeface we use for all kinds of type setting including headings, body copy and other text elements.	Lexia Bold 12em	
		Lexia standard 1.6em / 1.3 (inherited) 1.8em / 1.4 (inherited)	
		Lexia light 1.4em 1.6em	
Aktiv Grotesk Light	ABCDEF GHIJKLMNOPQRSTUVWXYZ XYZ?!@E\$%&1234567890abcdef ghijklmnopqrstuvwxyz		Over the last 170 years, King's has built its reputation as a world class hospital, with roots that lie firmly in the heart of our community. We are both a leading teaching hospital and a local hospital with a diverse inner city population.
Aktiv Grotesk Light It	ABCDEF GHIJKLMNOPQRSTUVWXYZ XYZ?!@E\$%&1234567890abcdef ghijklmnopqrstuvwxyz		
Aktiv Grotesk Regular	ABCDEF GHIJKLMNOPQRSTUVWXYZ XYZ?!@E\$%&1234567890abcde fghijklmnopqrstuvwxyz		Over the last 170 years, King's has built its reputation as a world class hospital, with roots that lie firmly in the heart of our community. We are both a leading teaching hospital and a local hospital with a diverse inner city population.

When Stuff & Nonsense designed for King's College Hospital, we paired two typefaces, Aktiv Grotesk and Lexia, both from type foundry Dalton Maag.⁷⁰

Making type proofs

Choosing typefaces is only part of the process when working with type. In the context of responsive web design we need our type to be legible and readable on many different types of screen. That doesn't just mean considering how type looks across screen sizes; it also means ensuring type displays well on low as well as high-resolution screens.

Whereas in the past we might have spent hours sweating the details of our type design using Photoshop or Sketch, today we can't rely on a graphics tool to give us an accurate rendering of responsive typography. For that, we need to design our type with CSS and test it using HTML in browsers on a variety of devices.

In our studio, we use what we call type proofs. These are HTML and CSS documents that contain only typographic headings and paragraph elements:

⁷⁰ daltonmaag.com/library

- Paragraphs in sizes 12px–21px
- Headings in sizes 12px–38px
- Small text in sizes 9px–12px

Testing readability

Type proofs aren't just simple to make, they're incredibly easy to use. They help us work better together and with our clients to design the best typography. In our process, we make deciding appropriate type sizes a collaborative affair. We ask our clients to join us by testing on whatever devices they're carrying and we encourage them to invite other people from elsewhere in their organisations.

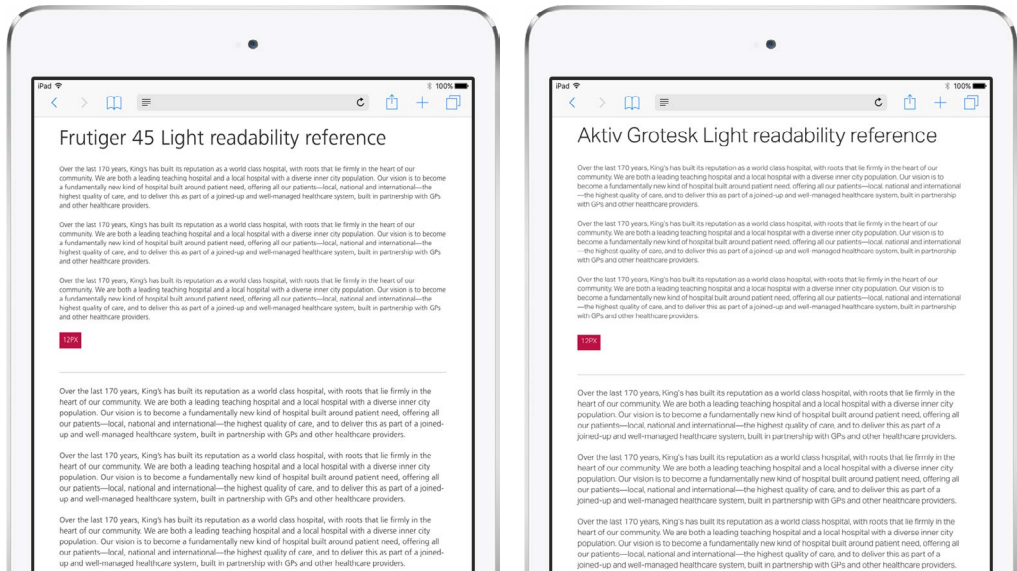
One example of this was when we designed for King's College Hospital. We were concerned that the standard National Health Service typeface, Frutiger 45, might not read comfortably when set in small sizes on small screens. We searched for a more open typeface, one with similar visual characteristics to Frutiger, and we tested several alternatives before settling on Aktiv Grotesk by Dalton Maag.

To help us make our decision between typefaces, we made two type proofs, the first containing Frutiger 45, the other Aktiv Grotesk. We tested the readability of both side by side with our client using several devices and this helped us arrive at a decision quickly and more accurately than we might have had we looked at type on static visuals.⁷²

Modular Scale⁷¹ is a useful online tool for developing typographic hierarchies based on ratios and musical intervals. Type Scale⁷² by Jeremy Church is another helpful tool for developing typographic scales. It links to Google Fonts to make designing with and testing font families and weights in a browser easier.

⁷¹ modularscale.com

⁷² type-scale.com



Within minutes, the team at King's College Hospital were able to decide on whether to choose Frutiger (left) and Aktiv Grotesk (right) for their new design.

Deciding minimum and maximum sizes

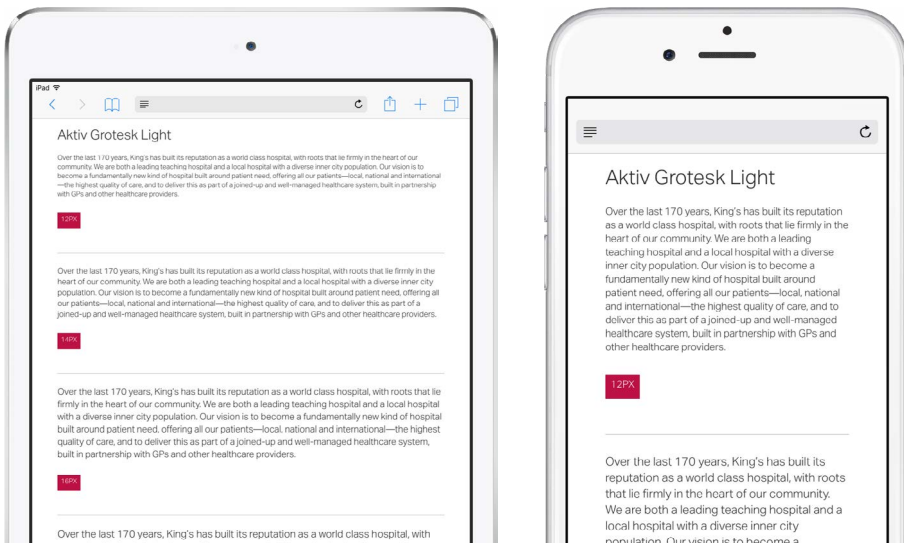
Deciding on the appropriate type size for a particular class of devices is one of the first challenges in any design project. We use type proofs to help us decide the minimum and maximum type sizes across a selection of screen sizes. Which specific devices we use doesn't matter. The exact sizes, makes, models or operating systems are irrelevant. What matters are their general proportions and characteristics. At Stuff & Nonsense, we base type size decisions on these classes of devices:

- Smaller smartphone: iPhone 5s
- Medium smartphone: iPhone 6s
- Smaller tablet: iPad mini (with and without retina display)
- Larger tablet: iPad Air
- Smaller PC: MacBook
- Large PC: iMac

We obviously love our Apple products, but if you have other makes of smartphones, tablets and PCs in similar classes, they'll work just fine for working with type proofs.

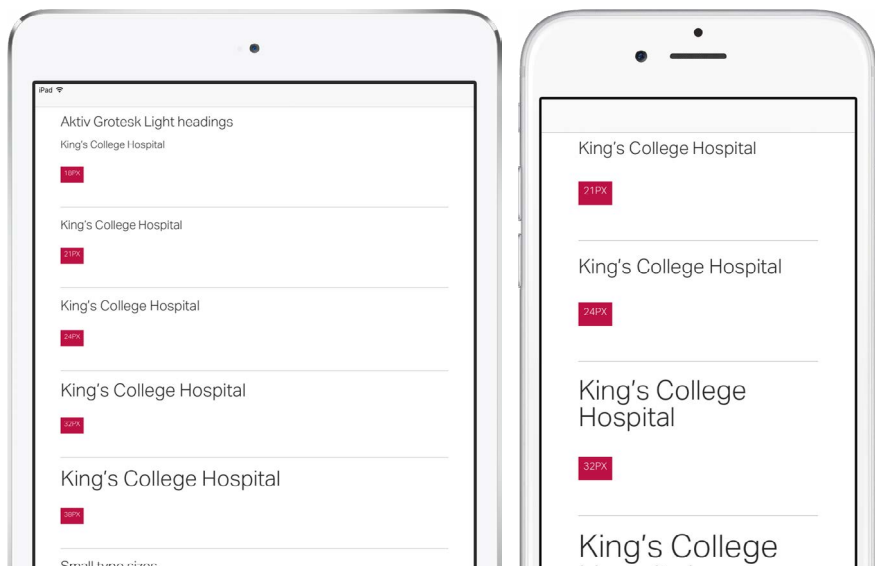
Testing paragraphs, headings and small text

We start by looking at paragraphs on smaller screens and decide on the minimum size when it stops being comfortable to read. Is 12px too small without bringing the device closer to our faces? Would 13px or 14px be more comfortable?



On our type proofs, we size text using pixels as we've found that our clients understand them better than they do em or rem units. Of course, we convert type to flexible units later in our process.

Next, we check those same small paragraphs on a device with a larger screen. Viewing distance matters and we generally hold larger devices further from our eyes, so decide if the size we just chose is still appropriate, and if it isn't we'll need to increase it to suit those longer viewing distances. Working this way, we can quickly determine the most appropriate text size for paragraphs for each class of device.



Type proofs are a simple tool to help us determine the maximum sizes for headings for each class of device.

We can follow the same process to determine appropriate sizes for headings and other elements too. Large headings make a design look dramatic, but while they suit larger screens, unless we're careful they can look clumsy when viewed on smaller ones where there's room for only one or two words per line. Start by deciding the appropriate heading sizes on smaller screens, then work up through device classes, making a note of any changes in sizes until we reach the largest screens.

Follow the same process for small text on buttons, in navigation and in footers, always starting with the smallest screens and working up.

Adding CSS media queries

Type proofs enable us to quickly determine the most appropriate sizes for our typography long before we might need to open Photoshop or Sketch. We'll add these styles to our web design style guide, but before we do that it often helps to test our findings one more time by adding CSS media queries that match our device class sizes.

At Stuff & Nonsense, we've made testing type using media queries the next step in the collaborative process with our clients. We host informal workshops where we first load our updated type proofs onto several classes of devices and then ask people to again judge the typography. We've found that involving clients in our decision-making process has several benefits. They're more confident to sign off type sizes and, because we encourage them to bring their own devices, we get to test our typography on plenty of devices that we don't own.

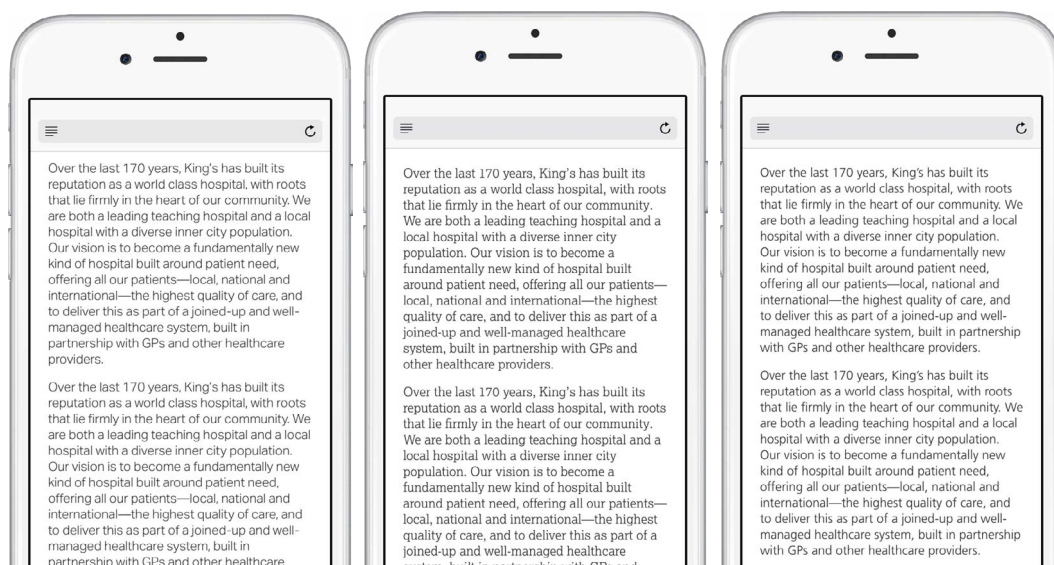
Judging typographic colour

Typographic colour doesn't refer to the colour chosen to style text, but instead the density of blocks of text on a page. Getting that density right isn't only important for the look of a design, it's vital for good readability — especially in responsive web design — and that helps everyone.

Several factors influence typographic colour: the typefaces we choose, spacing between letters (we call that **letter-spacing** in CSS but it's called *tracking* in other fields of design), and the space between lines of text (in our style sheets we call this vertical space **line-height**, but other fields refer to something very similar as *leading*).

Did you know that the term *leading* came from the strips of lead that typographers used between lines of hot metal type?

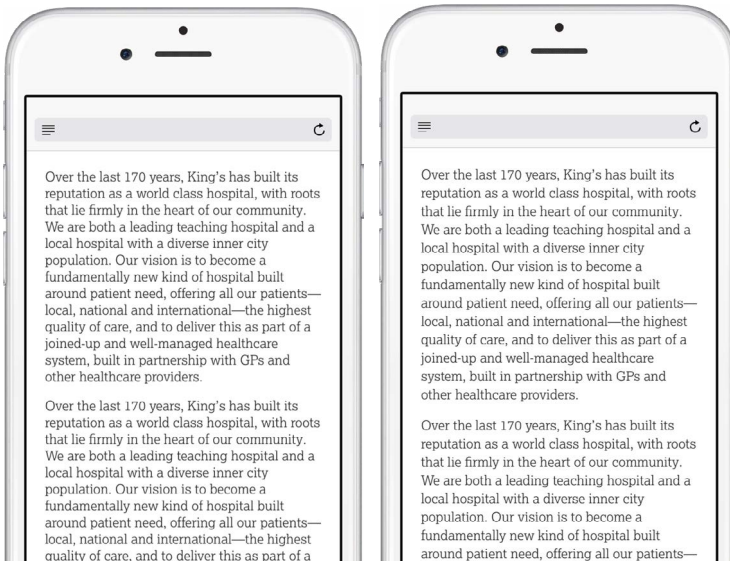
Look at these three screenshots of type on a small screen.



If you squint a little at these you should notice that the example in the centre appears darker, even though its type is the same size and pixel colour as the others. That's because we've chosen a typeface that has darker characteristics to its design.

Using that example again, I've increased our **line-height** to lighten the typographic colour. This has an immediate effect on the look of the type.

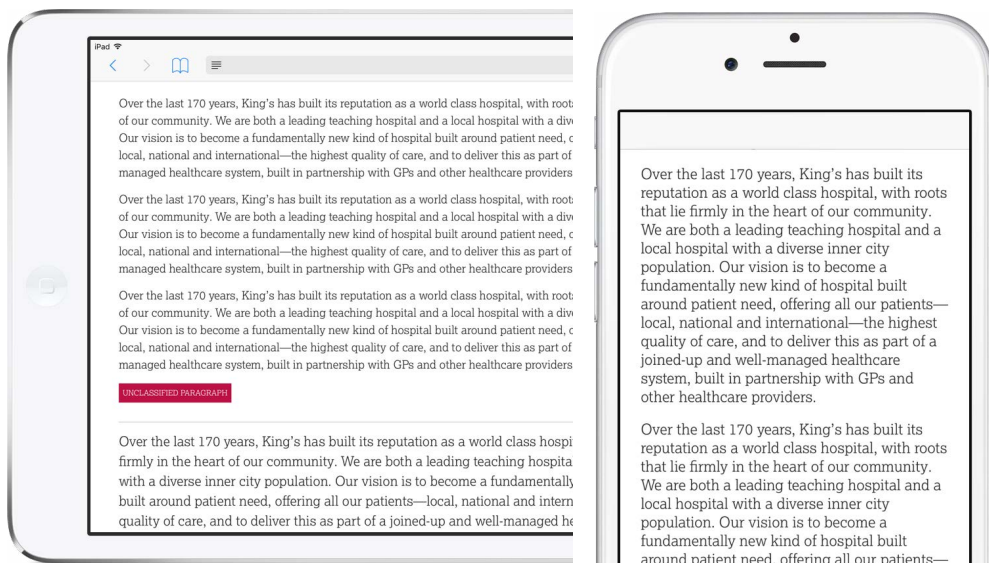
This should tell us that when we're optimising type across responsive breakpoints, we need to pay close attention to how **line-height** affects typographic colour and overall readability as much as we pay attention to the size of our type.



Adjusting line-height

As a rule of thumb, when the width of lines of text gets longer — for example, when displayed on devices of different sizes, or portrait and landscape orientations of the same device — we should increase the height of the space between the lines. However, it's common to see designers set **line-height** just once on the **body** element and then neglect to adjust it as screen widths and line lengths increase. We should always adjust **line-height** across responsive breakpoints and one place to start doing just that is in a type proof. Let's increase **line-height** on text in wider columns and on larger screens.

In typography, the length of a line of text is referred to as the measure. This term comes from a device used by hot metal typographers to measure the width of a column and then set the correct number of characters (including spaces) into it.



I've called the process of adjusting **line-height** across responsive breakpoints *proportional leading* and I first wrote about it in July 2010.⁷³

Checking font weights

Apple's high-resolution Retina screens have got bigger over time, starting with iPhone 4, before coming to iPad, MacBook and finally to the iMac. However, not everyone's fortunate enough to own a high-resolution display, so designers and developers must take into account how type renders on both low- and high-resolution screens.

Type proofs are again an ideal way to check font rendering across screen resolutions. After choosing a fashionably thin typeface for a design, we should test to ensure that it renders well on a low-resolution screen as well as a high-resolution one.

⁷³ stuffandnonsense.co.uk/blog/about/proportional_leading_with_css3_media_queries

partnership with GPs and other healthcare pr

Over the last 170 years, King's has built its re
roots that lie firmly in the heart of our commu
hospital and a local hospital with a diverse in
become a fundamentally new kind of hospita
our patients—local, national and internationa
deliver this as part of a joined-up and well-ma
partnership with GPs and other healthcare pr

Over the last 170 years, King's has built its re
roots that lie firmly in the heart of our commu
hospital and a local hospital with a diverse in
become a fundamentally new kind of hospita
our patients—local, national and internationa
deliver this as part of a joined-up and well-ma
partnership with GPs and other healthcare pr

UNCLASSIFIED PARAGRAPH

built in partnership with GPs and other h

Over the last 170 years, King's has built
with roots that lie firmly in the heart of o
teaching hospital and a local hospital wi
vision is to become a fundamentally new
offering all our patients—local, national a
care, and to deliver this as part of a joine
built in partnership with GPs and other h

Over the last 170 years, King's has built
with roots that lie firmly in the heart of o
teaching hospital and a local hospital wi
vision is to become a fundamentally new
offering all our patients—local, national a
care, and to deliver this as part of a joine
built in partnership with GPs and other h

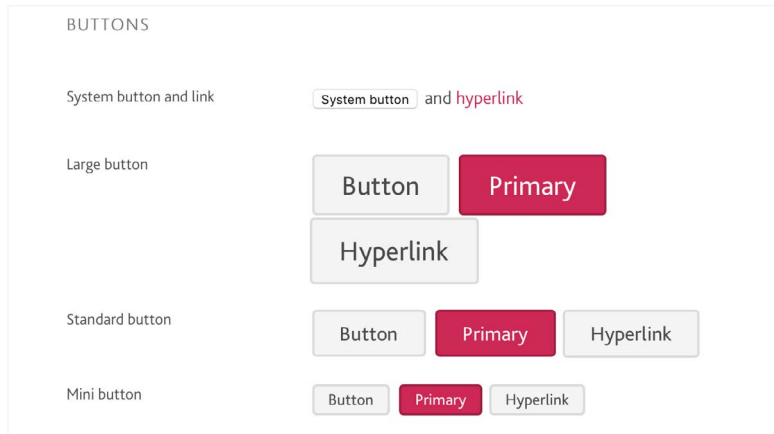
UNCLASSIFIED PARAGRAPH

When low-resolution screens struggle to render thin type (left), we can easily compensate by serving them a heavier weight from the same family (right).

Working with colour

Colour is the next aspect of atmosphere in a web design style guide. Colour creates mood and evokes an emotional reaction in someone using a website or application. We can also use colour to communicate what a user can, can't and sometimes shouldn't do as part of what I refer to as an interaction vocabulary. Think about the interactive elements of any website or application:

- **Hyperlinks** in **active**, **hover** and **visited** states.
- **Form buttons** of various types, including **disabled** as well as **active** and **hover** states.
- **Form inputs** of various types.



We can use colour to emphasise specific types of content, such as when we apply backgrounds and borders to them.

We use colour to help us communicate throughout our design. For example: “What can I click on? What can’t I click on? What have I already clicked on? What should I be cautious about clicking on?”

A web design style guide is the perfect place to document these colour choices. Working inside a guide instead of a static visual makes us think more systematically about how the colours we choose will be used. This helps the people who use our websites and applications because they’ll find our designs simpler to use.

Choosing colours

Some clients bring their own set of colours to a project, often in the form of branding guidelines. Other times they might have a single colour in their logo and it’s our job to create a palette of colours to accompany that. I expect that every designer has their own magical way to create colour palettes. In my studio, we’re always looking for colour inspiration and we take it from wherever we find it. We’ve developed several processes that help us choose colours that have been inspired by the things that our clients tell us about their organisations and their brands.

Deciding on a set of colours

We don't need a complex array of colours to make a design effective. I've found that the simpler the colour palette, the more connected a design will feel. At Stuff & Nonsense, the majority of our designs include four colours or fewer. We categorise the colours that we choose into:

Main: Most commonly used in branding, but also the colour for hyperlinks and backgrounds of primary calls to action, including buttons.

Secondary: Often used to indicate that an element is being interacted with; for example, the hover colour on hyperlinks and the background colour on active buttons.

Neutral: For the background of buttons, boxes, striped table rows and other elements. We often use darker or lighter shades of this neutral colour for borders and horizontal rules.













































Accent: Used sparingly, an accent often contrasts with the main colour. Often used for the background or borders of error and warning boxes.

For our design for King's College Hospital, we based our main colour on its existing branding guidelines which were in turn influenced by the NHS brand guidelines.

If you haven't already realised, a web design style guide is the perfect place to document our colour choices. It provides a reference for the designers and developers who might follow us on the project, and it also helps our clients to explain the choices we made to other people in their organisation.

COLOURS

Our palette of colours has been adapted from the King's College Hospital Trust branding plus a neutral grey. We use tints of these three colours and grey to form a harmonious palette.

Primary colour #1781a2		90% 	75% 	50% 	25% 	10% 
		90% 	75% 	50% 	25% 	10% 
Secondary #73b85b		90% 	75% 	50% 	25% 	10% 
		90% 	75% 	50% 	25% 	10% 
Error #bf0941		90% 	75% 	50% 	25% 	10% 
		90% 	75% 	50% 	25% 	10% 
Neutral #f2f8f7		90% 	75% 	50% 	25% 	10% 
		90% 	75% 	50% 	25% 	10% 

COLOURS IN USE



PRIMARY COLOUR
#1781A2

Used for hypertext link colour and the backgrounds of primary calls-to-action including buttons.



SECONDARY COLOUR
#73B85B

Used for hover states on hypertext links and the backgrounds of primary calls-to-action including buttons.



BASE COLOUR #F2F8F7
Used in the backgrounds of buttons, boxes, tables and other elements.



BORDER COLOUR
#DAE0DF
Used on the borders of buttons, boxes and other

Adobe Color CC (color.adobe.com) is an incredibly useful tool for creating colour palettes. For King's College Hospital, we used it to help us define secondary and accent colours. We then added not one, but two neutral colours to our palette.

A brand personality interview

At the start of almost every project, we ask our clients to answer questions about what they think about their brand. We call this a brand personality interview⁷⁴ and in it we ask our clients to:

⁷⁴ A brand personality interview certainly isn't unique to Stuff & Nonsense. Aaron Walter has published his design persona template which has influenced our process: aaronwalter.com/design-personas

“ Think of your brand as a person. That person can be real, alive today or a historical figure. You might choose a fictional character from a book or a movie. In fact, that person doesn't have to be a human being at all.”

I'm not joking when I say that Morgan Freeman comes up in discussion at every interview we conduct. Next we ask:

“ Describe the parts of that personality that appeal to you and how those traits relate to the personality you want to convey through your brand.”

People often say that Morgan Freeman's traits include him being reassuring and trustworthy. Finally, we ask people to list six personality traits that best describe how they'd like to think about their brand. At the same time, we ask about traits they'd like to avoid. Here are some personality traits we use as examples in our interviews:

- Fun but not silly
- Sensible but not boring
- Serious but not stuffy
- Professional but not corporate
- Friendly but not overfamiliar
- Contemporary but not trendy

We use the answers to these questions when we're looking to find inspiring colours in our photography research. When we're working with someone for the first time, it can be invaluable to help people work through these brand personality questions by facilitating a structured workshop. When a team is small — up to six people — we work through the questions with everyone together. For larger teams it's better for people to work in smaller groups of three or four. At the end of the session we bring everyone together to compare ideas.

Making tones

One of the first entries on my blog — from May 2004 — described a technique I used for creating a wider colour palette from a limited set of core colours. These source colours were like the main, secondary, neutral and accent colours that I described in the previous section.

A lot has changed since 2004, but incredibly the technique I described has stayed exactly the same and I still use it on almost every project I work on. The technique couldn't be simpler:

- Create five squares and fill them with one of the colours from your set.
- Adjust the opacity of the squares to dilute their strength. I use 90%, 75%, 50%, 25% and 10%.
- Place all five squares over a solid black base to create dark tones of your colour.
- Now do the same again, but this time place the squares over a white base to create lighter tones.
- Repeat for every colour in your set to create all the tones you'll need for your design.



In STEM Learning's branding guidelines, one colour represents science, another technology, and so too for engineering and maths. When we designed the new website we followed these branding guidelines and introduced a cool neutral to counter the bright colours in the palette.



We needed more than just STEM Learning's four lively colours, so we used our trusty technique to add more subtle tones to the palette. This technique may be old, but STEM Learning demonstrates that it still creates colour palettes that are right up to date.



Testing colour accessibility

The fact that many designers leave colour contrast and accessibility testing until late in a project has always baffled me. For me, colour contrast and accessibility aren't things we should test, they're things we should design, so we need to find ways to pay attention to them much earlier in our design processes. One of the biggest benefits of designing atmosphere is that we can pay attention to accessibility far earlier and then devote more time to correcting potential problems.

When we're testing colour accessibility, we need to ensure there's enough contrast between an element's background and any text inside it. We can alter contrast by either controlling lightness or by choosing complementary colours for backgrounds and foregrounds.

In this next example from Stuff & Nonsense's work with King's College Hospital, our earliest design provided insufficient contrast between backgrounds and foregrounds.

The Web Content Accessibility Guidelines (WCAG) 2.0 defines three levels of accessibility conformance: (lowest), AA and AAA (highest). Each level has its own minimum contrast ratio.



King's Healthy Passport



Feedback complaints form



King's Health Patnrns



100 Years of King's in Camberwell

Fortunately, we tested our colour contrast early and adjusted the design before we'd sold the colour palette to our client.

-  King's Healthy Passport
-  Feedback complaints form
-  King's Health Patrnrs
-  100 Years of King's in Camberwell

I find designing in greyscale to be one of the most effective ways to test colour accessibility. Not only does removing colour help to focus our attention on contrast, it removes distractions when we're designing layout and typography.

There are several fabulous tools to help check that our colour combinations offer sufficient contrast. My personal favourite is Lea Verou's contrast ratio checker⁷⁵ as it's quick and simple for everyone to use.

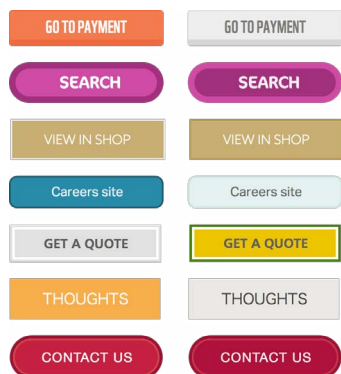


⁷⁵ leaverou.github.io/contrast-ratio

Adding texture

When we're creating the atmosphere of a design, texture refers to the decorative aspects that help give a design its personality. Texture includes border styles, shading and the shapes of boxes. Of course, texture can, and sometimes does, include skeuomorphic textures, so there's nothing to stop you indulging in faux leatherette and torn paper edges when you need to.

Decisions about borders and dividers are decisions about texture. I doubt there's a site design out there that doesn't include a border somewhere on something, but there's more to using a border than first comes to mind. Sure, we can make them **solid**, **double** or **dashed**. We can even use long-forgotten CSS values like **groove** and **ridge**, **inset** and **outset** if we're aiming for a retro look.

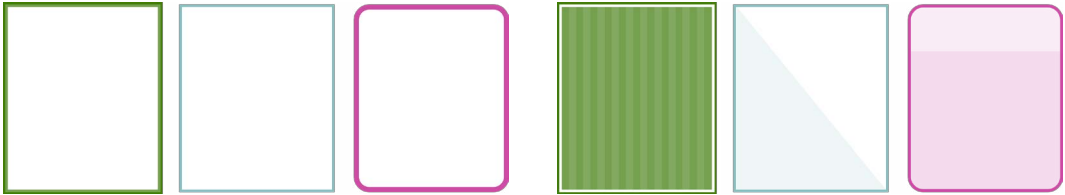


There are still bigger decisions to make, like how wide our borders should be and whether those widths will be uniform. For one design, we may want all borders the same width; for another, we might vary the width to create the impression of three dimensions.

Another question might be how to use various dividing line styles to create hierarchy. We could choose dashed borders between elements inside an **article**, then a single solid border between the articles themselves. A thicker, double border can illustrate the top of a hierarchy.

Decisions about background treatments are also about texture. How will we use backgrounds to shade content areas? Will we succumb to the fashion for flat colours or will our backgrounds be graduated or even patterned?

Box designs are texture too. Will we make box edges rounded or square? If they're rounded, will the four corners have equal radii, or will some be more rounded than others to create unusual shapes?



The look of buttons is also texture. Will we make on-screen buttons look like physical buttons? If we aim to make them appear physical, how will light fall on them to make them skeuomorphic? Will gradients make buttons look glossy, and will shadows lift them from the background? Will we give them backgrounds or make them transparent? Will we follow fashion and add thin outlines and uppercase button text?

How will we treat inline images? Will we give them thin borders? Will we add thick, white borders to simulate the look of Polaroid prints? What style of icons will we choose? Will they be graphic or hand-drawn?

All these design decisions involving texture help to give a design its individual personality. These questions make the difference between a fun design and one that looks sensible; one that's serious and professional but still manages to be friendly.

Finding value in static visuals

Designing atmosphere has tremendous advantages for responsive web design but there are disadvantages, too. When we're working on atmosphere and components separately, it's sometimes difficult to bring them together to form a consistent whole.

Because of this, designs can easily feel disjointed and lack connectedness. We need to look at every aspect of design together in one place to prevent this from happening.

Industry discussions about designing in a browser versus making static visuals can very easily descend into arguments. In truth, there's a need for both code and visuals, HTML and CSS editors and graphic tools, and we should use them both where they're most effective. At Stuff & Nonsense, we start almost every project by sketching ideas on paper because pencils and paper are absolutely the best tools for working through early ideas.

When we've settled on a direction, we move to code to experiment with layout, prototype interactive elements and test early designs in browsers. There's no doubt in our minds that HTML and CSS are the best tools for this part of our process.

Along the way we design typography, colour and texture and we do this almost exclusively in a browser using HTML and CSS because we know that it gives us the most realistic results. We use code to iterate quickly and sometimes to swap between design directions. We've found that code and browsers are by far the best tools when we need to be flexible.

We love designing using HTML and CSS in a browser, but there are some parts of our process where there's no better tool than Photoshop or Sketch. It might come as a surprise, but at Stuff & Nonsense we often create static visuals — sometimes of complete web pages — because we want to see where our design as a whole might be heading.

We make visuals of strategic pages, perhaps a list of articles or a range of products, a blog or news entry — sometimes even a home page — because we find that designing strategic visuals helps keep our designs connected.

But unlike in the past, we don't make visuals of every page. There's little or nothing to gain from that. We don't design small or medium-sized screen versions either — our visuals are strictly desktop only — because we know that a static visual isn't the tool to solve responsive design issues.

Working with a static version of a design can still be useful when we need to add an extra level of design fidelity. Somehow — no matter how proficient we've become designing using a browser — there's often no substitute for taking a design back to Photoshop or Sketch to experiment with a subtle outline, a shadow or some shading.

These tiny details can make all the difference and can easily turn an ordinary design into an interesting one. You might be asking now, "What aspect of that can't be done in a browser?" In truth, the answer's nothing. Yet somehow the environment of a graphics tool — working away from the practical concerns of implementation in HTML and CSS — helps us focus more clearly on those visual details.

Breaking it up

The past five years have seen the biggest changes to the websites and applications that we make and the process we follow when making them. As we've come to terms with designing responsive websites, designing atmosphere and then developing components has become a more commonplace design practice. Designing colour, typography and texture, and developing using web design style guides and pattern libraries as working tools and not just documentation, have also helped us focus more clearly on the aspects of design that transcend responsive breakpoints.



HARDBOILED HTML

Fit and lean HTML makes the web a better place for everyone. The trouble is, our pages are often built on less than optimal markup. Divisions, classes and identifiers are perfectly valid HTML, so what's the problem? Unless you're obsessive about keeping your markup in shape, it can all too easily become flabby, but never fear, there is a better way.

In **Hardboiled HTML**, you'll learn about the latest semantic elements. You'll also discover microformats2 — an evolution of those simple patterns for giving your markup added structure — and investigate WAI-ARIA roles. All of these will reduce your reliance on presentational elements and attributes. Get ready, it's time to make your HTML hardboiled.

No. 7

Destination HTML

COFFEE, EMAIL AND TWITTER are part of my morning ritual. I look at photos on Instagram, then check screenshots uploaded to Dribbble. Then I still read RSS feeds using Digg Reader and see where friends checked into on Swarm. These sites don't have pages in the traditional sense. They're web applications that behave more like desktop software.

Web applications have become extremely powerful and complex, but the markup languages we traditionally used to build them have stayed pretty much the same as they were in the early days of the web. HTML, then later the stricter and XML-inspired XHTML, are tools designed to make pages, not applications. That's where HTML5 came in — but first a little history.¹

After publishing HTML 4.0, the W3C shut down its HTML Working Group. HTML was done. The future — or so they thought — wasn't HTML, it was XML. Then, in 2004, the W3C held a workshop attended by several of the big browser makers. On their minds was how a document language could be used for making web applications.

Mozilla and Opera responded with their recommendations² but the W3C ignored them.

“At present, W3C does not intend to put any resources into [...] extensions to HTML and CSS for Web Applications.”³

¹ Mark Pilgrim has written a thoroughly readable history of HTML5: diveintohtml5.org

² w3.org/2004/04/webapps-cdf-ws/papers/opera.html

³ w3.org/2004/04/webapps-cdf-ws/summary

Like it or not, in the real world it's browser makers, not the W3C, who are the big cheeses. When the W3C refused outright to take up their suggestions, several of these companies took their ideas outside the W3C. They formed the Web Hypertext Applications Technology Working Group⁴ (WHATWG), a “loose, unofficial, and open collaboration of Web browser manufacturers and interested parties”⁵ that includes Apple, Google, Mozilla and Opera. Only Microsoft was originally absent from the group. WHATWG called its specification Web Applications 1.0.

Meanwhile, back at the W3C, work continued on what it saw as a future document language, XHTML 2. Its goals were ambitious and revolutionary, but were also ignored by the big gun browser makers and, without their support, XHTML 2 was doomed. As Mark Pilgrim astutely observed, “The ones that win are the ones that ship.”⁶

In the web standards business, browser makers hold the cards. Those players at WHATWG threw their weight behind HTML5, quickly developing the specification and implementing many parts of it in their browsers. The result? HTML is not only ready to use today, it's already become a de facto standard even though the W3C may not give its stamp of approval until 2022.

Are we going to wait until then before we start using HTML5? Good luck with that. If we do wait, we'll miss out on making a generation of exciting and innovative websites and applications.

I can hear you asking, “What happens when the specification changes? Will I need to rework my HTML?” The short answer? Yes. But we do that anyway.

⁴ whatwg.org

⁵ whatwg.org/news/start

⁶ diveintohtml5.org/past.html

No HTML stays frozen forever and rewriting and relearning markup as HTML5 evolves will just become part of our normal development process.

HTML5 is built on how we already worked with markup — it isn't a new markup language: it's the same markup we're already used to, but with powerful features built on top. From now on we'll simply refer to HTML5 as HTML. Learning it won't be difficult, so let's get started.

Get Shorty

Getting started with HTML couldn't be simpler. Simply declare that the document you're writing is HTML with a short and simple document type:

```
<!DOCTYPE html>
```

That's it. No version number, no language, no URI. Nothing. Just plain HTML.

The doctype is case-insensitive too, so we can write it as

```
<!doctype html>, <!DOCTYPE html> or even <!Doctype HTML>.
```

But you know what? The latest version of HTML doesn't even require a doctype, so you could leave it out altogether and it would still be valid HTML5 — although you probably shouldn't.

The doctype isn't the only thing that got shorter: character encoding did too. Here's a meta element for a document written in HTML.

```
<meta charset="UTF-8">
```

We needn't specify a `type` value of `text/css` on every link to every style sheet. We can simply write:


```
<link rel="stylesheet" href="Hardboiled.css">
```

Because browsers don't need to know, we needn't include `text/javascript` on script links either. We simply write:

```
<script src="modernizr.js"></script>
```

HTML isn't fussy about how we write our markup. Whether we like lowercase, uppercase or mixed case HTML elements, we can use our preferred style. Whether we self-close images or not, or use quote marks around our attributes or not, HTML won't mind. Neither will browsers, so we can carry on writing HTML in whatever style we prefer.

Semantic elements in HTML

HTML5 introduced several new elements to improve the structure of our pages. Your documents may still be full of divisions — what the HTML 4.01 specification described as a “mechanism for adding structure”⁷ — to group together related content.

```
<div class="branding"> [...] </div>
```

```
<div class="nav"> [...] </div>
```

```
<div class="content">
  <div class="content__main"> [...] </div>
  <div class="content__sub"> [...] </div>
</div>
```

```
<div class="footer"> [...] </div>
```

Any semantic meaning in these attributes is largely implicit and they aren't machine-readable so, in practice, user agents will treat `content__main` no differently than they would `you-dumb-mug`.

⁷ w3.org/TR/html401/struct/global.html#h-7.5.4

Adding presentational `id` and `class` attributes simply to build a visual layout dilutes any tenuous meaning even further.

We can replace some of our divisions with more semantically precise structural elements to help reduce our reliance on divisions and presentational `id` and `class` attributes. As a result, our markup will be fitter, leaner and less tied to a single visual layout or design.

In 2005, Google surveyed over three billion web pages⁸ to find out what `id` and `class` attributes web designers most commonly use to name HTML elements. The findings became the names of HTML5 structural elements and many are already widely supported in contemporary browsers. They include:

- `section`
- `article`
- `aside`
- `header`
- `footer`
- `nav`

This list isn't exhaustive because this book isn't intended to be a HTML reference. For that I'd recommend *HTML5 For Web Designers* by Jeremy Keith.⁹

⁸ developers.google.com/webmasters/state-of-the-web/2005/classes?csw=1

⁹ abookapart.com/products/html5-for-web-designers

Section

Pick apart the structure of a typical web page and we'll find divisions. These elements group related areas of content and help us build a visual layout using CSS. Take this example from the 'Get Hardboiled' archives page:

```
<div class="banner"> [...] </div>

<div class="navigation"> [...] </div>

<div class="content">
  <div class="content__uk"> [...] </div>
  <div class="content__usa"> [...] </div>
  <div class="content__world"> [...] </div>
</div>

<div class="footer"> [...] </div>
```

This markup pattern is perfectly valid, but even though we understand that the divisions represent sections of a page, browsers make no distinction between them and render them as anonymous block-level containers.

Instead, the `section` element groups content not into generic containers but into explicit, semantic sections. Think of them as distinct and possibly self-contained parts of a document. In the next example, sections contain news stories from different geographical regions, and all stories from each region are directly related. Notice that because each section should be able to stand alone, we'll include a descriptive heading in each one.

If necessary, we could add `id` attributes to each section to make them individually addressable via a fragment identifier such as

http://hardboiledwebdesign.com#content__uk:

Converting to BEM

If you've followed my writings over the years, you might remember that for a long time I've been fascinated by naming conventions in HTML and I'm a keen proponent of reducing reliance on `class` attributes in our markup. Writing the cleanest HTML was almost my religion.

In the past I've gone to extreme lengths to eliminate as many `class` attributes from my markup as possible. I'd use attribute selectors to bind styles to anchors using their `href` or even `title`, like this:

```
a[title="Get Hardboiled"] {  
  border-bottom : 5px solid #ebf4f6; }
```

I'd use child selectors (with a `>` combinator) to style elements that are the direct descendants of a specified element, as in this example where an unordered list is a child of the header:

```
header > ul {  
  list-style-type : none;  
  display : flex; }
```

Of course, I used so many adjacent sibling selectors it could've been considered unhealthy. These selectors style elements that immediately follow a previously specified element, as in this next example, where the header that follows a top-level heading is styled with a blue bottom-border:

```
h1 + header {  
  border-bottom : 5px solid #ebf4f6; }
```

Once, I even delivered a prototype that contained no classes at all, not a single one. I pity the poor developer who had to work from that.

Over the past few years, as Stuff & Nonsense has worked on larger projects, delivering our designs as HTML and CSS, and collaborating more closely with developers, I've realised there is a real need to deliver code that is not only well-structured and meaningful, but displays clear relationships between elements, not only in HTML but also in CSS. This is one problem that the BEM syntax or naming convention aims to help solve.

Block, element, modifier

When you look closely at the 'Get Hardboiled' examples that accompany this book, you should notice that on many of the elements, the `class` attribute values I've chosen contain either two underscores or two hyphens. These hyphens and underscores are part of the BEM system where BEM means block, element and modifier. They're used like this:

`.block` is used for a higher-level element that contains others that we'll also style. For example, in 'Get Hardboiled', a division given a `class` attribute value of `container` holds several child elements, including main and complementary content. This `container` is a typical BEM block.

`.block__element` represents an element that's a descendent of our container. Our main and complementary content divisions are good examples of these and we can describe their relationship to their container by using two underscores between them and their block in their attribute value: `.container__main` and `.container__complementary`. Descendent elements could also include headings (for example, `.container__heading`) or specific paragraphs such as `.container__lead`.

`.block--modifier` describes a modification to a block element. On the 'Get Hardboiled' home page, the majority of containers have a light-coloured background. Some, however, have certain design attributes that have been modified, in this case a darker background. We can describe this modification by using two hyphens between the value that describes the modification and the original block. For example `.container--dark`.

Using this convention has helped me and the developers I work with define the precise relationships between elements in my design. The developers can do this not just by examining the HTML structure but by reading my style sheets. `.container__main` is clearly a descendent of an element called `container`, as is a heading called `.container__heading`. Developers need not think about the purpose of `.container--dark` since the BEM naming convention tells them that it's an alternative design of the standard `.container`.¹⁰

Using BEM has transformed my work and although part of me still yearns for HTML that is clean and free from `class` attribute values for many elements, I'm prepared to sacrifice my punctilious coding tendencies for the clarity and ease of working that the BEM syntax brings.¹¹

```
<section id="content__uk">
  <h1>Stories from the UK</h1>
</section>

<section id="content__usa">
  <h1>Stories from the USA</h1>
</section>

<section id="content__world">
  <h1>Stories from around the world</h1>
</section>
```

Let's continue building our document by adding articles.

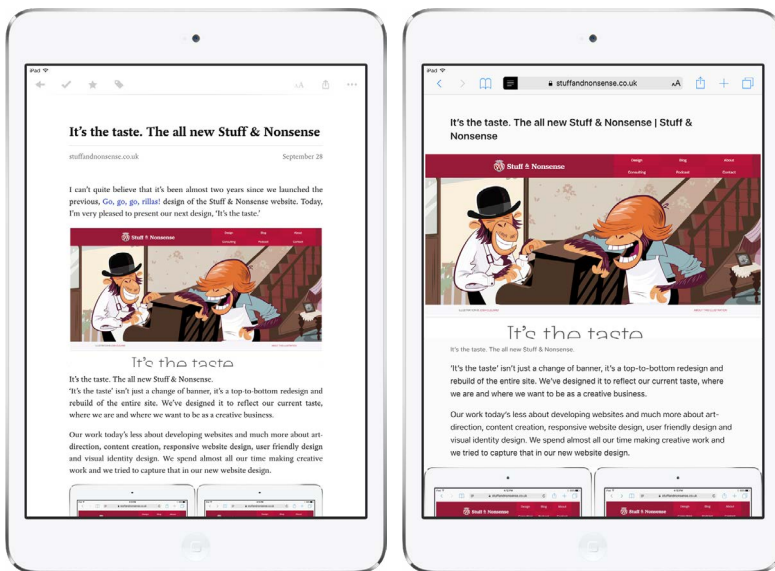
¹⁰ Harry Roberts has written plenty about BEM and namespaces inside HTML. His article 'MindBEMding — getting your head 'round BEM syntax' is an excellent place to start learning about BEM. csswizardry.com/2013/01/mindbemding-getting-your-head-round-bem-syntax

¹¹ The BEM website is a community-driven resource that explains the key concepts, and links to useful articles and tutorials about the naming convention: smashed.by/bem-naming

Article

When we write for a blog or online magazine or news site, we publish articles. In HTML, an article is just like an individual story, in that it should be understandable outside the context of a page. That might sound similar to a **section**, but there's a big difference. Whereas an **article** represents a story that can stand apart, a **section** is a self-contained part of the page and could contain several related articles.

One way to check if **article** is the most appropriate element is to see if an article's content makes sense on its own; for example, would it make sense on its own when viewed in an article reader like Pocket for iPad?



If you own an iPad, Pocket¹² is one of the nicest ways to read articles. In Pocket, content is isolated and shown without advertising, branding, navigation — or other articles that might give it context. Safari on both iOS and Mac OS X offers similar functionality.

¹² getpocket.com

If you're still confused about the difference between sections and articles, Doctor Bruce Lawson explained more in his 'HTML5 articles and sections: what's the difference?'¹³

Let's carry on building up the outline of the 'Get Hardboiled' archives page by adding three articles to each of the sections:

```
<section id="content__uk">
  <h1>Stories from the UK</h1>
  <article> [...] </article>
  <article> [...] </article>
  <article> [...] </article>
</section>

<section id="content__usa">
  <h1>Stories from the UK</h1>
  <article> [...] </article>
  <article> [...] </article>
  <article> [...] </article>
</section>

<section id="content__world">
  <h1>Stories from around the world</h1>
  <article> [...] </article>
  <article> [...] </article>
  <article> [...] </article>
</section>
```

Sections can contain articles, and guess what? An `article` element can contain sections too! What, you thought learning new HTML elements was going to come without a little head-scratching?¹⁴

¹³ brucelawson.co.uk/2010/html5-articles-and-sections-whats-the-difference

¹⁴ To help us understand when to use each HTML5 element, the HTML5 Doctors have prepared their handy 'HTML5 Sectioning Element Flowchart'.
html5doctor.com/downloads/h5d-sectioning-flowchart.pdf

To help clear that up — the `article` / `section` confusion, not your dandruff — this next `article` has three sections, each about a famous fiction writer:

```
<article>
  <section id="chandler"> [...] </section>
  <section id="hammett"> [...] </section>
  <section id="spillane"> [...] </section>
</article>
```

Header

A page's branding area or masthead can be described with the `header` element. These headers traditionally appear at the top of a page, although we could position a `header` on the side, at the bottom or anywhere we choose. We'll replace that classified `banner` division with a much more appropriate `header` element:

```
<header>
  <h1>It's Hardboiled</h1>
</header>
```

We can also add a `header` to any `section` or `article`: we're not limited to using just one `header` on a page either. This means we can use a `header` in several different ways: as the masthead or branding for a entire page, to introduce sections and articles, or a combination of all of the above.

In this next example, we'll add one 'Hardboiled authors' `header` to describe the `article`, followed by another in each of the sections:

```
<article>
  <header>
    <h1>Hardboiled authors</h1>
  </header>

  <section id="chandler">
    <header>
      <h1>Raymond Chandler</h1>
    </header>
  </section>

  <section id="dashiell-hammett">
    <header>
      <h1>Dashiell Hammett</h1>
    </header>
  </section>

  <section id="mickey-spillane">
    <header>
      <h1>Mickey Spillane</h1>
    </header>
  </section>
</article>
```

The specification describes the `header` element as a container for “a group of introductory or navigational aids,”¹⁵ leaving us free to include a search form and the time and date that a page, section or article was written or updated.

Footer

In an amazing turn of events (or was it simply coincidence?), some of the results of my 2004 non-scientific survey of element names¹⁶ were the same as Google’s. We both found that the most common name web designers gave to the foot of a page — the one that typically contains contact and copyright information — was ‘footer’.

¹⁵ w3.org/TR/html5/sections.html#the-header-element

¹⁶ stuffandnonsense.co.uk/blog/about/whats_in_a_name

In a typical HTML 4.01 or XHTML 1.0 document, this footer would be marked up using a division, perhaps with a `class` attribute value of 'footer' applied to it. You too might choose to mark up this footer using a division, perhaps with a `class` of 'footer':

```
<div class="footer"> [...] </div>
```

Even better, though, we should replace that anonymous division with a more appropriate `footer` element:

```
<footer>
  <h3>It's Hardboiled</h3>
  <small>Creative Commons Attribution-ShareAlike 4.0
International License.</small>
</footer>
```

Contrary to its name, we needn't position a `footer` at the bottom of a page, `section` or `article`. In fact, we can place one anywhere inside its containing element. Like `header`, we can use `footer` to define meta information for any `section` or `article`. Inside an `article`, a `footer` might contain information about the author, or the date and time it was published. A `section` footer could include when it was updated or new articles added:

```
<section id="spillane">
  <header>
    <h1>Mickey Spillane</h1>
  </header>

  <footer>
    <small>Published by Andy Clarke on 20th Nov. 2015</small>
  </footer>
</section>
```

Aside

Mickey Spillane was a prolific fiction writer and if we were writing an article about his life and work our biography might include related information about one of my favourite books, *My Gun Is Quick*. Conversely, if we were writing a review of that book, we might want to include a biography of its author. HTML5 defines the relationship using the `aside` element and despite its name, `aside` needn't be visually positioned in a sidebar.

We can use `aside` to describe content that's related to — but not essential for understanding — an `article`. Let's start writing that biographical article of Mickey Spillane. We'll include a `header` containing the title, and a `footer` containing the author's name and the `article`'s publication date for good measure.

```
<article>
  <header>
    <h1>Mickey Spillane</h1>
  </header>

  <footer>
    <small>Published by Andy Clarke on 20th Nov. 2015</small>
  </footer>

  <p>Frank Morrison Spillane, better known as Mickey Spillane,
  was an author of crime novels...</p>
</article>
```

Sweet, just like I take my coffee. Now let's add an `aside` that contains related information about *My Gun Is Quick*.

```
<article>
  <header>
    <h1>Mickey Spillane</h1>
  </header>

  <footer>
    <small>Published by Andy Clarke on 20th Nov. 2015</small>
  </footer>

  <p>Frank Morrison Spillane, better known as Mickey Spillane,
was an author of crime novels...</p>
  <aside>
    <h2>My Gun Is Quick</h2>
    <p>Mickey Spillane's second novel featuring private
investigator Mike Hammer.</p>
  </aside>
</article>
```

Perhaps our page contains information about other fiction writers. This content would be less strictly related to our biography, so we'll place that `aside` outside the article. In this case, we should also wrap both the `article` and `aside` inside a `section` element to make it explicit that the two are related.

```
<section>
  <article>
    <header>
      <h1>Mickey Spillane</h1>
    </header>

    <footer>
      <small>Published by Andy Clarke on 20th Nov. 2015</small>
    </footer>

    <p>Frank Morrison Spillane, better known as Mickey Spillane,
was an author of crime novels...</p>
    <aside>
      <h2>My Gun Is Quick</h2>
      <p>Mickey Spillane's second novel featuring private
investigator Mike Hammer.</p>
```

```
        </aside>
    </article>
    <aside>
        <h2>Other crime fiction writers</h2>
        <ul>
            <li>Raymond Chandler</li>
            <li>Dashiell Hammett</li>
            <li>Jonathan Latimer</li>
        </ul>
    </aside>
</section>
```

Nav

Readers shouldn't need to hire a detective to help them find something on a website — that's what navigation is for. Often when we're building pages our navigation looks something like this.

```
<div class="nav--main">
    <ul>
        <li><a href="about.html">What's Hardboiled?</a></li>
        <li><a href="archives.html">Archives</a></li>
        <li><a href="authors.html">Hardboiled Authors</a></li>
        <li><a href="store.html">Classic Hardboiled</a></li>
    </ul>
</div>
```

We've become accustomed to marking up navigation using lists, but the trouble is, we mark up other things using lists too. How the hell is a browser supposed to know the difference between a list of links and a list of people we owe money to?

Thankfully, we now have the `nav` element for one or more “major navigation blocks”¹⁷ on a page. Not all links, or even groups of links, are major navigation blocks, so we should reserve `nav` for the primary ways that people navigate.

¹⁷ dev.w3.org/html5/spec/Overview.html#the-nav-element

Navigation will probably include lists of links to your most important pages in the page's **header**, a sidebar, or possibly in the page **footer**. Next, we'll replace the previous anonymous division with a meaningful **nav** element.

```
<nav>
  <ul>
    <li><a href="about.html">What's Hardboiled?</a></li>
    <li><a href="archives.html">Archives</a></li>
    <li><a href="authors.html">Hardboiled Authors</a></li>
    <li><a href="store.html">Classic Hardboiled</a></li>
  </ul>
</nav>
```

When our visitors use search to find content, add a search form inside your **nav**. If we've included skip links, these could also be considered as major navigation blocks for people who use assistive technologies.

Figure

Like all private dicks, I appreciate a good figure. This has got me into hot water on more than one occasion. In printed media, images, charts and diagrams are often paired with written captions. Instead of struggling to decide on the right element for captioning, use the **figure** and **figcaption** elements to associate captions with images, charts, diagrams and even code examples.

```
<figure>
  
  <figcaption>I, The Jury by Mickey Spillane</figcaption>
</figure>
```

When we need to caption a group of elements we can nest multiple images, charts or diagrams and label them with a single **figcaption**.

```
<figure>
  
  
  
  <figcaption>Books by Mickey Spillane</figcaption>
</figure>
```

HTML5 dates and times

You might imagine that writing a date in HTML would be as simple as.

```
<footer>
  <small>Published by Andy Clarke on 06/05/2015</small>
</footer>
```

But the problem is, software finds it hard to know that this string of numbers is a date. People, on the other hand, are far more intuitive, but even we sometimes have different interpretations of the same numbers depending on where we live. Coming from the UK, I read those numbers as the sixth (day) of May (month) in the year 2015, but if you live in the United States, you might read the date as June 5th, 2015.

To solve that problem, the `time` element is readable by people — 6 May 2015; May 6th, 2015; next Thursday — and formatted for parsers.

```
<time>May 6th 2015</time>
```

The `time` element is made from two versions of a date or date/time. The first is a human readable, natural language date and the second is a `datetime` attribute with a machine readable, ISO-formatted date: `YYYY-MM-DDThh:mm:ss`. That's year-month-day followed by time in hours, minutes and seconds (if we need to be that precise):

```
<time datetime="2015-05-06">May 6th 2015</time>
```


`time` has had a chequered history. First introduced by HTML5, it was dropped from the specification in 2011 and replaced with a more generic – and in my mind, less semantic – `data` element. Thankfully, `time` returned later that year and while away gained some useful extra functionality. Whereas previously its `datetime` format required precise, ISO-formatted dates, the improved format allows for fuzzy dates:

```
<time datetime="2015"> means the year 2015  
<time datetime="2015-05"> means May 2015  
<time datetime="05-06"> means 6th May (in any year)  
<time datetime="2015-W1"> means week 1 of 2015
```

When you need to describe how long an event lasts, you can use the `datetime` attribute and prefix the duration with a “P” (standing for period). Add suffixes for days “D”, hours “H”, minutes “M” and even seconds “S” if you want to markup how long I’d stay standing in a fist fight. This describes an event that lasts one day:

```
<time datetime="P 1 D">
```

And the next event lasts one day, six hours, ten minutes and thirty seconds. Can you spot the extra “T” (for time) prefix to denote a more precise duration?

```
<time datetime="PT 1D 6H 10M 30S">
```

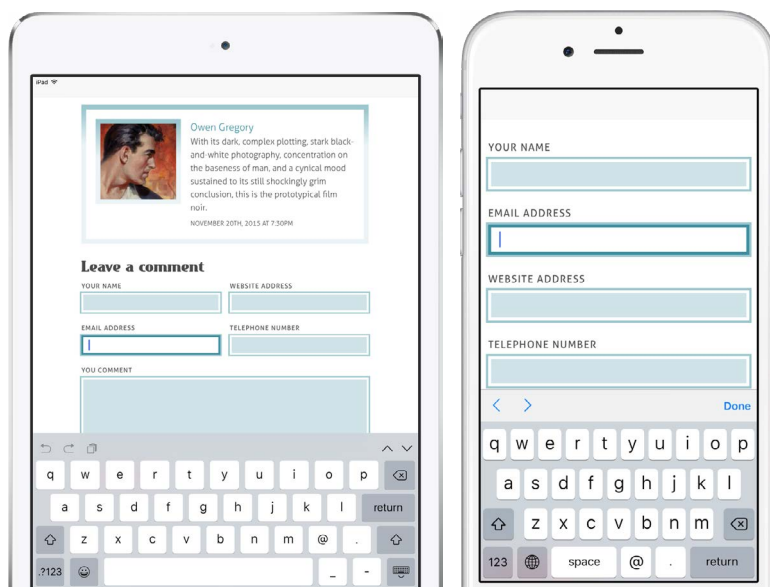
By bringing together precise, structured date formats with dates and times set out in natural language, we’ve implemented a format that is readable by both people and machines.

Form elements

What website or application would be complete with a form or two? Love building them or hate styling them, the web without forms would be like a private eye's night out without a voluptuous redhead. HTML5 introduced more than a dozen `input` types and attributes that make implementing complicated controls and functions — like sliders, date pickers and client-side validation — a breeze. These elements include `email`, `url`, `tel` and `search`. Don't worry about legacy browsers, even ancient ones: these `input` types degrade to simple text fields when a browser doesn't understand them.

Email

Lift the floorboards covering most web forms and we'll probably uncover a field asking for an email address. Contact forms, comment forms, registration forms and sign-ups all demand an email address for their grubby little databases. Choose an `email` input type to describe a field that's intended to grab an email address.



Software keyboards adapt to the job at hand. Here, the iPad's keyboard includes a set of keys to make entering an email address easier.

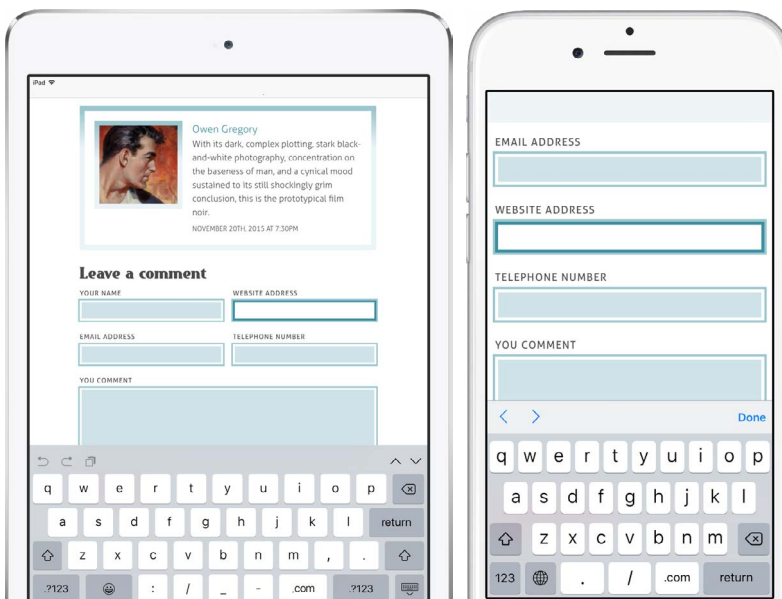
```
<input type="email">
```

All kinds of interesting functionality is now possible, including checking that a form submission includes a valid email address that contains an @ symbol and that it's properly constructed. Tap into an `email` input and Safari on iOS brings up a software keyboard that places an @ symbol and a dot prominently at the foot of the keyboard.

URLs

When we use a URL input — dedicated to entering a website address — Apple's iOS keyboard adapts by adding a prominent slash, a dot and a '.com' key.

```
<input type="url">
```

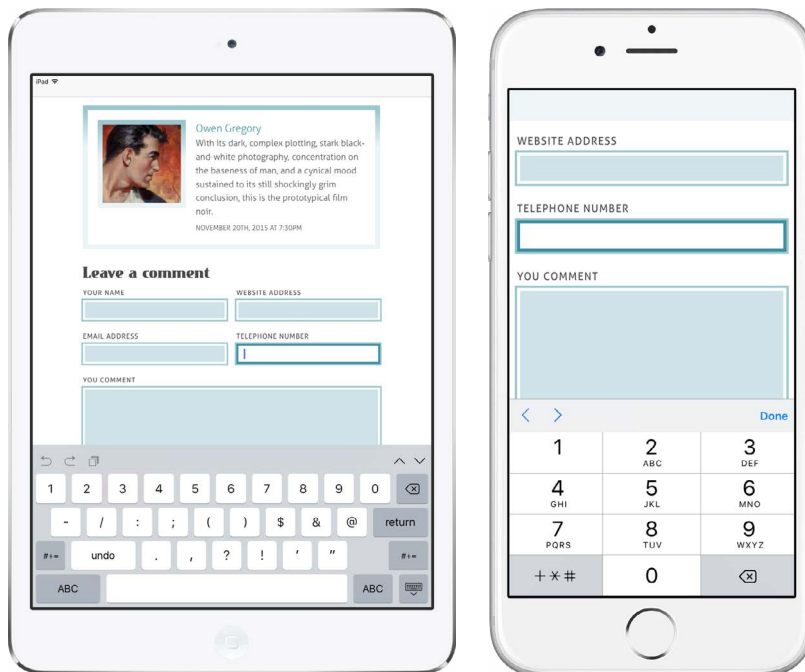


Our thoughtful use of form elements makes life easier for people on the go. When we use `url`, the iPhone's keyboard automatically includes a '.com' button.

Telephone

If we use the `tel` input type, iOS automatically pulls up a numbers-only keypad.

```
<input type="tel">
```



iOS does a great job of adapting its interface to the type of inputs we're using. Now, if only it could call my bookmaker...

Search

If there's plenty of content on a site we'll probably include a search form. Luckily, we have an input type that's dedicated to search:

```
<input type="search">
```

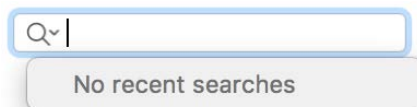
Whereas in Safari on iOS, a `search` input appears more rounded than a normal text input, in Safari on Mac OS X (as well as most other desktop browsers) `search` looks identical to a text input. That is until we interact with it. Start typing a query into a search input in Chrome, Opera or Safari and a handy clear icon appears to make clearing your search easier.



There's more to `search` than meets the eye, as adding some distinctly hardboiled attributes will make your search forms pack more of a punch. Add the `autosave` attribute, along with a value that's unique to your website — in our case 'gethardboiled' — and Safari will not only add a small magnifying glass icon in our search input but will also display a dropdown list of previous searches. We can control how many searches are remembered using the `results` attribute. Our form will remember ten searches:

```
<input type="search" results="10" autosave="getHardboiled">
```

`search` inputs are notoriously difficult to style and results are often unpredictable, so my advice is to leave them as the good browser makers intended.



Number

If we use the `number` input type, iOS again automatically pulls up a numbers-only keypad, but in most desktop browsers something even more interesting happens. Chrome, Firefox, Opera and Safari all add input arrows or spinners to the right of the `number` input. Pressing these arrows or using a keyboard's arrow keys or your mouse's scroll wheel will adjust the inputted number up or down in the steps you specify:

```
<input type="number" steps="10">
```

Should you ever need to, you can remove a `number` input's arrow for Chrome, Firefox, Opera and Safari using a little non-standard, but nonetheless useful CSS:

```
input[type=number]::-webkit-inner-spin-button,  
input[type=number]::-webkit-outer-spin-button {  
  -webkit-appearance: none;  
  -moz-appearance : textfield;  
  margin: 0; }
```

`number`'s functionality remains unchanged and people can still increment numbers using their keyboard's arrow keys or their mouse's scroll wheel.

Native date pickers

As a designer and not a developer, implementing the sort of date pickers we find on airline, car rental and hotel sites always puts me behind the eight-ball. The native date pickers in HTML have made adding them less painful.



CHOOSE A TIME

time for hours, minutes and seconds
(10:10:00)

CHOOSE A WEEK

October 2015 ▾
 ◀
●
▶

Week	Mon	Tue	Wed	Thu	Fri	Sat	Sun
40	28	29	30	1	2	3	4
41	5	6	7	8	9	10	11
42	12	13	14	15	16	17	18
43	19	20	21	22	23	24	25
44	26	27	28	29	30	31	1

week from 1–53

CHOOSE A MONTH

October 2015 ▾
 ◀
●
▶

Mon	Tue	Wed	Thu	Fri	Sat	Sun
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1

month for a year and a month
but no day (2015-11)

CHOOSE A DATE

October 2015 ▾
 ◀
●
▶

Mon	Tue	Wed	Thu	Fri	Sat	Sun
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1

date for a year, month and day
(2015-11-20)

Placeholder text

Damn those troublesome form labels. I bet I'm not the only one who sometimes wants to provide a visual cue for how to use a form element without showing label text. A **placeholder** attribute adds text to any **input** that's either empty or not in focus. Browsers that don't support **placeholder** will safely ignore it by leaving the **input** blank.

An **input** containing a **placeholder** can be used instead of label text and, when used correctly, can remove clutter from an interface design. Focusing on the **search** input removes the placeholder text faster than a bartender takes bills from my billfold.

```
<input type="search" title="Search this site"
placeholder="Search this site">
```

`label` text isn't required for accessibility when the purpose of a form is simple and the text can be substituted with a title or an explicitly titled button.

Autofocus

Like millions of others, when I do a Google search my cursor automatically focuses on its search field. Unlike millions of others, you and I notice these small enhancements. In the past, we were forced to use a script to do this, but we now have the `autofocus` attribute to tell a browser to do the hard work for us. Browsers that don't support `autofocus` will ignore it:

```
<input type="search" autofocus>
```

Autocomplete

We can dictate which form inputs will be prefilled with information from a visitor's previous form submissions:

```
<input type="text" name="name" autocomplete="on">
```

Use `autocomplete` wisely, though, as some fields are best left alone, in particular anything that relates to credit cards or other financial information:

```
<input type="text" name="credit-card" autocomplete="off">
```


List and datalist

Often, one of the best ways to help a visitor complete a form is to suggest an answer to a question or to give them options. The `list` attribute, including a `datalist`, combines the convenience of a `select` with a visitor's own ability to enter text.

Imagine we wanted to ask someone who their favourite detective is. To help them along, we could suggest a few hardboiled heroes in a `datalist`, then associate that list with a text `input` using the `list` attribute as an `id`:

```
<input type="text" list="detectives">
<datalist id="detectives">
  <option value="Mike Hammer">Mike Hammer</option>
  <option value="Sam Spade">Sam Spade</option>
  <option value="Philip Marlowe">Philip Marlowe</option>
</datalist>
```

CHOOSE A DETECTIVE



m

- Mike Hammer
- Sam Spade
- Philip Marlowe

A visitor can type in their own entry, or they can choose from the options available in the `datalist`. Any browser that doesn't support `list` and `datalist` will ignore them and display a normal text `input` field instead.

Min and max

Perhaps you sell books online and your site has a minimum order quantity, or, like me, you run training courses and limit the maximum number of places available to be booked in one go. The `min` and `max` attributes allow us to specify upper and lower limits for the data.

```
<input type="number" id="book" min="1">  
<input type="number" id="course" max="6">
```

It's worth knowing that currently `min` and `max` attributes don't work with the `required` attribute in any browser. In fact, we can't use `required` with a number input at all.

Client-side validation

Writing form validation scripts is one of my least favourite jobs. Maybe that's the reason I pay someone else to do it for me. JavaScript libraries make the job easier but I bet that even the most hardcore JavaScript nerds don't actually enjoy implementing these scripts. Wouldn't it be better if a browser handled form validation for us? This would not only make life simpler, it would also make it harder for unscrupulous types to bypass JavaScript validation (by simply turning JavaScript off). The good news is that HTML includes simple features that make client-side form validation a breeze.

Required

HTML5 added the `required` attribute which will prevent a browser from submitting any form data until all required fields (`text`, `email`, `url`, etc.) have been completed correctly:

```
<input type="email" required>
```

novalidate

If you'd prefer a browser not to validate for you, simply add the `novalidate` attribute to your form to prevent native validation:

```
<form action="search" method="get" novalidate>
```

Breaking it up

HTML5 brought markup into the web application age and there's far more to the specification than just the elements I've covered in this book. There are ways to embed video and audio files, and play them without needing browser plugins, plus ways to enable interaction with web pages and applications offline. How far you go will depend entirely on your specific work and the needs of the people you work for. One thing's for sure, HTML is here to stay and if we're to stay ahead of the curve we should be working with as much of it as possible.

No. 8

Hardboiled microformats2

IF YOU CARE ABOUT MAKING EVERY HTML ELEMENT and attribute that you use count and you want to make their semantics go further, I hope that you'll soon be getting excited about microformats2. These HTML-based patterns are part of a continuing effort to improve how we mark up specific types of information — such as contact details, events, reviews and content like blog entries — so that they become parsable by machines as well as human-readable. I think that Brian Suda explained the purpose of what are now classic microformats best:

“Microformats are all about representing semantic information encoded within a web page, allowing that information to be leveraged in ways that were possibly never conceived by the original publisher.”¹⁸

Microformats were developed around existing standards (attributes added to the elements describing our content) so the only thing we needed to know to start using them was how to write HTML. The same is true of microformats2. In fact, they're simpler and require fewer extra HTML elements than classic microformats did, owing to the way they imply particular content. But what made classic microformats hardboiled and why are microformats2 more so?

View source on any website and we'll see a mass of `id` and `class` attributes used to bind CSS styles to those elements. I'd hope the values you choose describe the content; for example, using a semantic value like 'tagline' instead of a presentational value like 'bold-heading'.

¹⁸ <http://www.sitepoint.com/microformats-meaning-markup/>

Even so, the values we commonly choose add little real value to our content and are of no use to our visitors. Coming up with these names and building our own set of conventions for them can take time, and over the long term they can be tricky to maintain.

When we follow microformatted patterns, we can largely give presentational `id` and `class` attributes a concrete overcoat, because microformats2 brings with it more `class` attributes than you can hang a raincoat on. Using microformats2 will help make your HTML leaner, fitter and less tied to just one design or layout — in short, more hard-boiled. We'll start by investigating some of the most commonly used classic and microformats2 patterns.

Link-based microformats

If you're familiar with links to external files in the `head` of an HTML document, you'll recognise this as a link to an external CSS style sheet:

```
<link rel="stylesheet" href="screen.css" media="screen">
```

This attribute defines the relationship between an HTML document and the link's destination. This relationship is one of a style sheet. We're used to taking a similar approach when linking to RSS feeds:

```
<link rel="alternate" type="application/rss+xml" href="articles.rss">
```

We can also define a relationship to a favicon or an apple-touch-icon, one that's been designed for the home screens of Apple's iOS devices:

```
<link rel="shortcut icon" href="favicon.jpg" type="image/gif">
<link rel="shortcut icon" href="favicon.png" type="image/png">
<link rel="apple-touch-icon" href="ios.png">
```

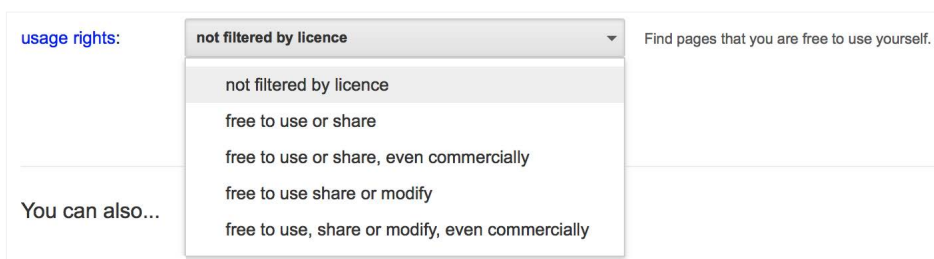
All link-based microformats apply this idea to describe other relationships between documents.

Rel-license

Linking to a licence is one of the most common uses of link-based microformats. When we use rel-license, we mean explicitly that this link points to a licence for this content:

```
<a href="http://creativecommons.org/licenses/by-sa/4.0/" rel="license">
Creative Commons Attribution-ShareAlike 4.0 International License</a>
```

You're probably thinking that any dumb mug can link to a licence using a microformat — and you'd be right on the money. After all, one of the principles of microformats is that they should have a low barrier to entry — in this case, really low. But the benefits can be huge, especially as Google added a usage rights option to its advanced search.



Want to search for something that's free to use, share and modify, even commercially? Hidden in Google's advanced search are options to filter by usage rights.

What's rel-license got to do with being hardboiled? Well, perhaps we want to style a licence link differently to other links on a page, perhaps by adding a small icon. We could added a `class` attribute to that link:

```
<a href="http://creativecommons.org/licenses/by-sa/4.0/"  
rel="license" class="license">Creative Commons Attribution-  
ShareAlike 4.0 International License</a>
```

But we've no need for that extra `class` because we can style the link in exactly the same way by using a CSS attribute selector:

```
a[rel="license"] {  
padding-left : 20px;  
background : transparent url(cc.png) no-repeat 0 0; }
```

By using a microformat, we've immediately made our HTML more hardboiled by eliminating a presentational attribute that offered nothing of value to us or our users.

HTML link relationships

HTML5 introduced more link relationships that enable us to define the meaning of links to other pages. Here are some of the most useful to me:

<code>author</code>	The author of the content; for example, their biography or contact information on the same site or another.
<code>bookmark</code>	A complete page or <code>section</code> element.
<code>next</code>	When the document is a part of a series, indicates a link to the next document in the series.
<code>nofollow</code>	First adopted by Google in an attempt to combat spam comments, this attribute makes it explicit that a link is not an endorsement of the destination.
<code>previous</code>	When the document is a part of a series, indicates a link to the previous document in the series.
<code>search</code>	A dedicated search page or a search interface.

Although there's some overlap between HTML link relationships and both microformats and WAI-ARIA roles, don't let that put you off using them. They all add deeper semantics and provide hooks for us to style links using CSS.

h-card: People, places and organisations

Wouldn't it be cool if, when we find a person's contact information online, we could add them to an address book in just a couple of clicks? Guess what: if they publish contact information using the h-card format, we can. Take a close look at the following paragraph:

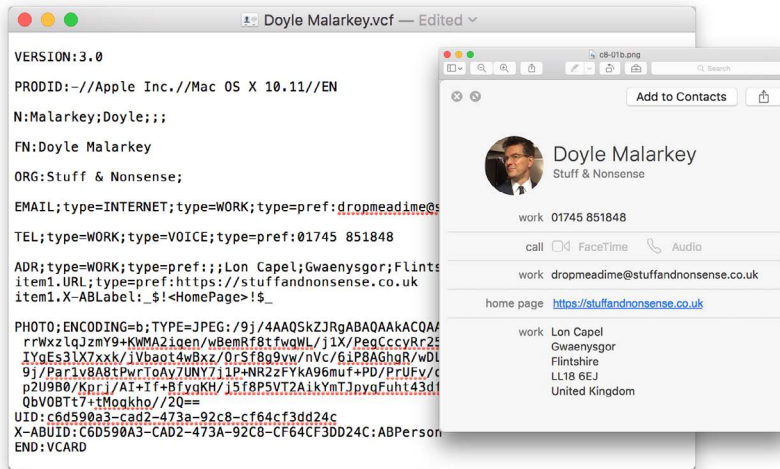
Andy Clarke (Malarkey) is a web designer and wannabe detective based in the United Kingdom. He runs a small agency called Stuff & Nonsense, writes books and speaks at conferences. If you'd like to hire Andy, you can email him at dropadime@hardboiledwebdesign.com or call him on 01745 851 848.

That paragraph is chock-full of information and you shouldn't need to be a detective to spot my full name, nickname, place of work, email address and work telephone number.

Our brains have the smarts to notice these pieces of information and realise that they stand apart from the words around them but, for now at least, machines can't easily make the same distinctions.

Markup languages have a limited vocabulary and can't adequately convey all the meaning in this content. Headings and paragraphs, sections and articles: they're no trouble; but what about elements that describe a person's contact information? Where are they? Nowhere. But h-card¹⁹ is a format that adds that missing meaning to a person's or an organisation's contact information.

¹⁹ microformats.org/wiki/h-card



vCard information inside Apple's Contacts application has a structured data format.

Structured data in markup

The contact management application on Macs and iOS devices uses an open standard called vCard so you'll already be familiar with seeing this type of structured information. You can open a vCard in a text editor to find the details, including: a name (FN for formatted name); organisation (ORG); address (ADR); telephone (TEL); email address (EMAIL), and more.

The best way to learn h-card is to make a card. You could head over to the microformats website, but if you do, you'll wonder what is so damned hardboiled about these formats when you see a mass of `div` and `span` elements. What's so revolutionary about this?

```
<div class="h-card">
  <span class="p-name">Andy Clarke</span>
  <a class="u-email" href="mailto:dropadime@Hardboiledwebdesign.com">
    dropadime@Hardboiledwebdesign.com</a>
  <span class="p-country-name">United Kingdom</span>
  <span class="p-tel">01745 851848</span>
</div>
```

It's true that on first impression the sheer quantity of attribute values looks excessive, but those values are important because they enable our content to be more easily extracted and used by other applications. Keeping that in mind, let's go back and review that earlier block of text. We'll add `class` attribute values from h-card to make my details more meaningful, starting with applying a class of `h-card` to the paragraph as it's the root element of this format:

```
<p class="h-card"><span class="p-name">Andy Clarke</span> (<span class="p-nickname">Malarkey</span>) is a <span class="p-job-title">web designer</a> and wannabe detective based in the <span class="p-country-name">United Kingdom</span>. He runs a small agency called <span class="p-org">Stuff & Nonsense</span>, writes books and speaks at conferences. If you'd like to hire Andy, you can e-mail him at <a class="u-email" href="mailto:dropadime@Hardboiledwebdesign.com">dropadime@Hardboiledwebdesign.com</a> or call him on <span class="p-tel">01745 851848</span>.</p>
```

Remember when you learned that `html` is the root element of an HTML document? Microformats need their own root element to tell an application that a microformat is present. For h-card, it really is as simple as adding the class attribute `h-card`.

Names

Now let's dig deeper by looking at the values that make up an h-card and how to structure them. We'll start by describing a person's name — there are two ways we can choose to do this:

1. A full, formatted name of a person or organisation.
2. A structured name containing separate prefixes, given, middle, family names and suffixes.

Class attributes with prefixes

One significant change between classic microformats and microformats2 `class` attribute values is that newer versions are prefixed to help differentiate microformats from styling hooks. There are five prefixes:

- `h-*` root classnames mark that any element is a microformat; for example, `h-card`.
- `dt-*` parses an element as a date or time.
- `e-*` includes all the HTML within an element.
- `p-*` denotes that an element contains plain text, such as `Andy Clarke`.
- `u-*` signifies an element as a URL, including email and website.

Confusion with namespaces

Attribute naming systems such as BEM²⁰ help to communicate the relationships between HTML elements by using their `class` attribute values. Developer Harry Roberts has been looking to extend these principles to include namespaces²¹ that describe the role of an attribute; for example, components, objects, utilities and themes. Sadly his `u-utility` value conflicts with microformats2's URL value.

Formatted name

When we present a person's name as it would appear on a business card or nameplate on their office door, we can roll up several values including a prefix, their given name, middle name, family name and suffix into a single string. To do this, we need only apply a single value of `p-name` to an element and we're done:

```
<span class="p-name">Nick Jefferies</span>
```

²⁰ en.bem.info/method

²¹ csswizardry.com/2015/03/more-transparent-ui-code-with-namespaces

Structured names

Now we'll structure a name, separating the parts of that name into individual elements, the first for a person's given name, Nick, and the second for his family name, Jefferies.

```
<span class="p-given-name">Nick</span>  
<span class="p-family-name">Jefferies</span>
```

If Nick Jefferies' business h-card includes an honorific prefix (Mr, Mrs, Sir, Professor, etc.) or, more likely, his nickname (Sawbuck), we can include those too:

```
<span class="p-honorific-prefix">Mr.</span>  
<span class="p-Nickname">Sawbuck</span>
```

Mr Nick (Sawbuck) Jefferies' structured name, with all of its separate components, now looks like this.

```
<span class="p-honorific-prefix">Mr.</span>  
<span class="p-given-name">Nick</span>  
<span class="p-Nickname">Sawbuck</span>  
<span class="p-family-name">Jefferies</span>
```

All that remains for us to do is to enclose Nick's structured name in the most appropriate root HTML element — in this case a division, although it could just as easily be a list, paragraph, **section**, **article** or **footer** if one of those is more appropriate — and apply the **class** attribute value **h-card** to that:

```
<div class="h-card">  
  <span class="p-honorific-prefix">Mr.</span>  
  <span class="p-given-name">Nick</span>  
  <span class="p-Nickname">Sawbuck</span>  
  <span class="p-family-name">Jefferies</span>  
</div>
```

URLs

Today's contacts wouldn't be much use without at least one website address, so it shouldn't come as a surprise to find out that h-card includes a value for one. The most obvious way to add a URL to an h-card is like this:

```
<a href="http://Hardboiledwebdesign.com" class="u-url">
http://Hardboiledwebdesign.com</a>
```

Organisations

Describing the company or organisation that a person works for helps me to demonstrate the nesting capabilities of h-card. First, we'll create an h-card for the individual, Cole Henley. Cole's card will contain just his name and URL:

```
<div class="h-card">
  <a href="http://Hardboiledwebdesign.com" class="p-name
u-url">Cole Henley</a>
</div>
```

Now we'll include the name of the organisation that Cole works for and describe it using the `p-org` attribute value. Because the organisation's a separate entity, we give it its own h-card, nested inside Cole's:

```
<div class="h-card">
  <a href="http://Hardboiledwebdesign.com" class="p-name
u-url">Cole Henley</a>
  <span class="p-org h-card">The No. 1 Detective Agency</span>
</div>
```

When we need to display an employee's organisation's logo, we can embed an image inside their h-card and apply the `class` attribute value of `u-logo` to that.

Implied properties in microformats2

Microformats2 are simpler formats than their predecessors because our use of some properties imply the existence of others. For example, we could mark up someone's full, formatted name using `p-name` and their website address using `u-url`:

```
<div class="h-card">
  <a href="http://stuffandnonsense.co.uk" class="p-name u-url">
    Andy Clarke</a>
</div>
```

However, microformats2 make that pattern simpler and more hard-boiled by removing the need for the extra parent element. Now we can simply apply the value of `h-card` to the anchor and remove both `p-name` and `u-url` because both are implied with `h-card`:

```
<a href="http://stuffandnonsense.co.uk" class="h-card">Andy Clarke</a>
```

Addresses

Tracked the guy down? Know where he lives or works? Now add that information to his h-card. We can add values including `p-street-address`, `p-locality`, `p-region` and `p-postal-code`:

```
<span class="p-street-address">221b Baker Street</span>,
<span class="p-locality">London</span>,
<span class="p-postal-code">NW1 6XE</span>,
<span class="p-country-name">United Kingdom</span>
```

At this point you could be wondering, “Why not use an HTML `address` element?” Despite what you may infer from its name, `address` should only be used for marking up contact information for the author of a specific page or block of content. `address` was, rather confusingly, never designed solely to describe physical addresses. Don't take my word for it — here it's from the horse's mouth at WHATWG:

“ The `address` element represents the contact information for its nearest `article` or `body` element ancestor. [...] The `address` element must not be used to represent arbitrary addresses (e.g. postal addresses), unless those addresses are in fact the relevant contact information.”²²

Need more than one address?

What if someone has more than one address? Unless you run an agency from home like I do, you'll have both a home and a work address. Not to worry, we can include both addresses in an h-card; and to make sure it's clear which `p-street-address`, `p-locality`, `p-region` and `p-postal-code` values belong to which h-card, we'll group them together inside individual `h-adr` elements:

```
<div class="h-adr">
  <span class="p-street-address">221b Baker Street</span>,
  <span class="p-locality">London</span>,
  <span class="p-postal-code">NW1 6XE</span>,
  <span class="p-country-name">United Kingdom</span>
</div>
<div class="h-adr">
  <span class="p-street-address">8-10 Broadway</span>,
  <span class="p-locality">London</span>,
  <span class="p-postal-code">SW1H 0BG</span>,
  <span class="p-country-name">United Kingdom</span>
</div>
```

Phone numbers

I still prefer to talk on the phone. I guess that I'm old-fashioned like that. Lucky for me, then, that including a phone number (or two, or three) in an h-card is a piece of cake. We simply use the `p-tel` value:

```
<div class="p-tel">01745 851848</div>
```

²² whatwg.org/specs/web-apps/current-work/multipage/sections.html#the-address-element

Most people have more than one number: at home, in their office and, of course, a mobile. We can add telephone numbers for all of those using the same format.

Email

You shouldn't be surprised to learn that to add email addresses to an h-card we use the `u-email` value:

```
<a href="mailto:dropadime@Hardboiledwebdesign.com"
class="u-email">Drop me a dime</a>
```

Other h-card properties

h-card is an ideal format that helps structure information about a person or an organisation. Of course, there's far more to a person or company than just their name, addresses and how to contact them, so h-card includes plenty more optional values. Here are some of the values that I find most useful:

<code>u-photo</code>	Indicates a specific photo, possibly an avatar, that's associated with a person's h-card.
<code>p-note</code>	Additional notes included in a person's h-card.
<code>dt-bday</code>	Their birthday. Mine's November 20th, in case you were wondering.
<code>p-job-title</code>	A person's job title.
<code>p-role</code>	Their role. It might be different to their job title.
<code>p-sex</code>	A person's biological sex.
<code>p-gender-identity</code>	Their gender identity.

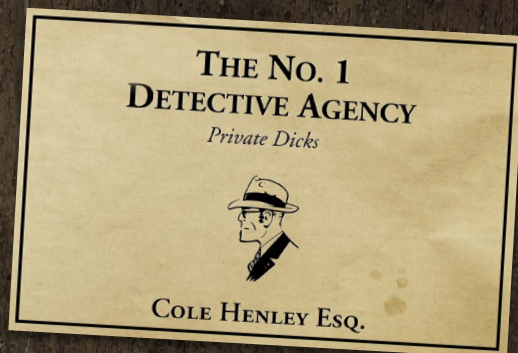
Marking up the 'Get Hardboiled' authors page

There's no better way to understand the nuances of h-card than to work with them. We'll build a series of h-cards for the 'Get Hardboiled' authors page. Each detective's business card uses slightly different values.


```

<div class="h-card">
<h3 class="p-org">
The No. 1 Detective Agency</h3>
<span class="p-given-name">Cole</span>
<span class="p-family-name">Henley
</span>
<span class="p-honorific-suffixes">Esq.
</span>
</div>

```



```

<div class="h-card">
<h3 class="p-name">
Shades & Staches Detective Agency</h3>
<p class="p-role">Private investigator
extraordinaire</p>
</div>

```



```

<h3 class="h-card">
Command F Detective Services</h3>

```





```
<div class="h-card">
<h3 class="p-name">The Fat Man</h3>
<p class="p-role">Private Investigator</p>
<p>$50 a day plus expenses.
By appointment only</p>
<p>Dial: M for Murder</p>
</div>
```



```
<div class="h-card">
<h3 class="p-name">Nick Jefferies</h3>
<p><span class="p-job-title">
Private Eye</span>,
<span class="p-postal-code">WA6-0089
</span></p>
</div>
```



```
<div class="h-card">
<h3 class="p-name">
Elementary, My Dear Watson</h3>
<p class="p-role">Private Investigator</p>
<p>Don't call us, we'll find you</p>
</div>
```

h-event for publishing events

Think for a moment about the event information that you see around the web every day. You'll find details about conferences, sporting events, concerts and movie screenings. Sometimes event information appears structured, other times it might appear in natural language. For example, on my blog I might write, "This November, I'm seeing Kacey Musgraves in concert at the Albert Hall in Manchester."

You don't have to look hard to find event information online, but the HTML used to mark up this information varies enormously from site to site. This is how that Kacey Musgraves concert is marked up on Ticketmaster:

```
<tr>
  <td class="event">
    <div class="summary">  </div>
    <div class="ratingContainer" title="4.8 out of 5 stars"></div>
  </td>
  <td class="location">Albert Hall Manchester, GB</td>
  <td class="date">Mon 16/11/15 19:00</td>
</tr>
```

Seetickets published the same event with different HTML:

```
<div class="browse-width result-text">
  <h3>Kacey Musgraves</a></h3>
  <p>Albert Hall, Manchester</p>
  <p>at 7:00 PM</p>
</div>
```

People can easily understand this contains an event, but there's nothing in that HTML that can help a machine. This makes events a perfect problem for a microformat to solve.

A calendar event will most likely contain:

- Name or summary
- Description
- Location
- Start and end dates and times
- URL pointing to the event page or site
- Venue contact information

We'll build a single event, starting by specifying the root element for it using the `h-event` attribute value. Just as with `h-card`, we could keep our events simple and add that attribute to a single element as `p-name` and even `u-url` are implied:

```
<div class="h-event">The Maltese Falcon</div>
```

We'll need a little more information about this showing of *The Maltese Falcon* before we have a really useful event, though, so we'll turn that division into an `article` and add the `p-name` value to its main heading:

```
<article class="h-event">
  <h1 class="p-name">The Maltese Falcon</h1>
</article>
```

Summary

Next, we'll add a short summary by applying the `p-summary` value to an appropriate HTML element; in our case, it's a paragraph:

```
<article class="h-event">
  <h1 class="p-name">The Maltese Falcon</h1>
  <p class="p-summary">A special showing of the remastered
  mystery that kicked off the film noir genre of the 1940s...</p>
</article>
```

If our summary contains more than just a single paragraph, we can group headings, paragraphs, lists or any other elements inside a `div` or even a `section` and apply the description to that. We mustn't include more than one summary per h-event, though, as that would be an invalid event.

```
<section class="p-summary">
  <p>A special showing of the remastered mystery that kicked off
the film noir genre of the 1940s...</p>
  <p>Private detectives Sam Spade and Miles Archer are hired by a
woman to follow a man called Thursby...</p>
</section>
```

Location

Letting people know where an event will take place involves nothing more than applying the `p-location` value to an element, in this case a text-level `span` wrapped around the venue's name:

```
<p>Showing at <span class="p-location">
The Scala Cinema and Art Centre</span></p>
```

Should we need to provide more information about the venue, perhaps by including its address, we should create an `h-adr` element for the venue and embed it inside our h-event. That `h-adr` will contain the same address values we used when made our h-cards.

```
<div class="p-location h-adr">
  <span class="p-street-address">47 High Street</span>
  <span class="p-locality">Prestatyn</span>
  <span class="p-region">Denbighshire</span>
  <span class="p-country-name">Wales</span>
</div>
```

URL

If the event has a website, we'll use the same `u-url` value as we did when building an h-card:

```
<a href="http://scalaPrestatyn.co.uk" class="u-url">The Scala  
Cinema website</a>
```

Start date and duration

Our h-event microformat is almost complete, but it's still missing a start date to tell people when to show up. We'll first mark up the event's start time using a `time` element and a `datetime` attribute:

```
<time datetime="2015-11-20 T19:30">Nov. 20th, 2015 at 7:30pm</time>
```

To make it explicit that this is a start date, we'll also need to add a `dt-start` class attribute value to the `time` element:

```
<time datetime="2015-11-20 T19:30" class="dt-start">  
November 20th, 2015 at 7:30pm</time>
```

As our event runs for just one evening and finishes at 10pm, we can also add that period to our `datetime` attribute:

```
<time datetime="2015-11-20 T19:30 P 150M" class="dt-start">  
November 20th, 2015 at 7:30pm</time>
```

Mixing events and contacts

Microformats are designed to be modular and embeddable, so we can easily include a contact's h-card in an h-event.

Do you remember earlier when we specified the location of this showing of *The Maltese Falcon*? We added the `p-location` value to the name of the venue:

```
<p>Showing at <span class="p-location">  
The Scala Cinema and Art Centre</span></p>
```

Now would be a good time to add more precise information about the venue, so we'll create an **h-card** and embed it inside our h-event.

```
<div class="p-location h-card">  
  <span class="p-name">The Scala Cinema and Art Centre</span>  
  <span class="p-street-address">47 High Street</span>  
  <span class="p-locality">Prestatyn</span>  
  <span class="p-region">Denbighshire</span>  
  <span class="p-country-name">Wales</span>  
</div>
```

h-review for publishing reviews

I hope you've enjoyed what you've read so far and that you'll write a glowing review because h-review is what's coming next.

If we listen to almost any conversation or read almost any article, we'll find people expressing opinions about almost everything. Our brains are adept at recognising all kinds of reviews wherever we see them:

"Last week I rented the DVD of the 1941 movie The Maltese Falcon starring Humphrey Bogart as Sam Spade. It remains one of my favourite movies and I give it a big thumbs up."

"Movie: The Maltese Falcon. Rating: 10/10"

"I had pretty low expectations of Who Framed Roger Rabbit but I'm giving it five stars."

Computers are less able to recognise the delicate nuances of language. To them, the information in each of these reviews is nothing more than a string of characters. h-review addresses this by providing a rich semantic schema for review content – one that’s built on the lessons learned from established microformats such as h-card and h-event.

As with the other microformats we’ve learned, an h-review comprises elements contained within a root element. This time it’s **h-review** and we can apply that value to any appropriate HTML element, in this case an **article**.

```
<article class="h-review">
  <h1>The best detective film ever made</h1>
</article>
```

Writing an h-review will take less time than you might think, because h-review reuses values you should recognise from both h-card and h-event. First, we’ll name our review using the **p-name** value that we’ve seen in other microformats. It’s worth noting that the name of our h-review can be different to the name of the item that we’re reviewing.

```
<article class="h-review">
  <h1 class="p-name">The best detective film ever made</h1>
</article>
```

Now let’s define the item we’re reviewing. What we include in the **p-item** element needn’t only be the name of a business, person, place or product; we can be creative and include other information related to the item.

```
<p class="p-item">Who Framed Roger Rabbit, starring the late Bob
Hoskins as private investigator Eddie Valiant.</p>
```


Should we need to include more detailed information about the item – rather than about the review – we can embed `h-card` when reviewing a person, `h-adr` for describing a business's or venue's location, `h-product` for a product review, and `h-item` for any other type of item. These microformats structure information about a specific type of item. As we're reviewing a film and not a business or product, we'll choose `h-item` and add that to our `p-item` element,

```
<p class="p-item h-item">Who Framed Roger Rabbit, starring the  
late Bob Hoskins as private investigator Eddie Valiant.</p>
```

A review wouldn't be much use without an opinion and for this we'll use the `e-description` value. If a review is short, apply this to one element, perhaps a list or, in this case, a paragraph:

```
<p class="e-description">How much do I know about show business?  
Only that there is no business like it, no business I know.</p>
```

If the description covers more than one paragraph or includes other HTML elements, group them all into a containing element and apply the `e-description` value to that. As the review is something I'm saying, using a `blockquote` seems appropriate:

```
<blockquote class="e-description">  
<p>How much do I know about show business? Only that there is no  
business like it, no business I know.</p>  
<p>A Classic film has to work on several different levels and ani-  
mated action movie Who Framed Roger Rabbit scores on all of them.  
It's a fantastic children's film with characters like Roger, the  
Weasels and Benny the Cab for them to enjoy. It also plays per-  
fectly as a detective story for adults. And who will ever forget  
Jessica Rabbit?</p>  
</blockquote>
```

URL

You should be able to guess by now how we'll include a permanent URL for the review that we're writing as we've seen it already in both h-card and h-event. Yes, it's the `u-url` value:

```
<a href="Hardboiledwebdesign.com" class="u-url">Canonical Permalink</a>
```

Adding a rating

Star ratings are a popular way to indicate a positive or negative review. They help people see at a glance whether an item's considered either good or bad and we'll find them on thousands of review and shopping websites. We'll stick with convention and use stars to create a scale from zero, the worst rating, to five, the best.

To describe the five stars I'm rating *Who Framed Roger Rabbit* we'll use HTML5's `data` element. If you haven't used `data` before, it's simple: `data` handles the human readable, visible part of the element — in our case the rating's stars — and a `value` attribute provides that same information in a machine readable form:

```
<data class="p-rating" value="5">★★★★★</data>
```

If we need to be more specific and show the best and the worst ratings instead of just an average, we can use both `p-best` and `p-worst` respectively:

```
<data class="p-best" value="5">★★★★★</data>
<data class="p-worst" value="0"></data>
```

Review date

Of course, we should also include a date. This will help other people judge the relevance of our review and it can be especially important for hotel and restaurant reviews. We'll simply reuse the same `time` element and `datetime` attribute that we've already seen in h-event and then add the `dt-reviewed` value:

```
<time datetime="2015-11-20 T19:30" class="dt-reviewed">
November 20th, 2015 at 7:30pm</time>
```

Mixing reviews and contacts

As it's not essential to know who wrote a review, h-review doesn't require us to include a name, but we may choose to add one because a person's identity can greatly enhance a review's credibility. We should always use h-card to describe a reviewer and we can include as much contact information as we want, but here we'll only add the reviewer's name using the p-reviewer value together with their h-card:

```
<a class="p-reviewer h-card"
href="http://stuffandnonsense.co.uk">Andy Clarke</a>
```

h-entry for news articles, blog posts and podcasts

Next, you'll learn about h-entry, a microformat that's been designed for publishing syndicated content such as news articles, blog posts and podcasts. h-entry describes a single entry which we can group with other entries into a collected h-feed. Let's write an h-entry blog post.

As the microformats community suggests, we should start by using “the most accurately precise semantic XHTML building block for each object etc.”²³ Here's our initial HTML, which starts with a heading followed by a paragraph:

```
<h1>The Maltese Falcon</h1>
<p>The film stars Humphrey Bogart as private investigator Sam
Spade and Mary Astor as his femme fatale client.</p>
```

Now that we know how to use an article element for standalone entries like this, we'll group those elements inside that article:

²³ <http://microformats.org/wiki/semantic-xhtml-design-principles>

```
<article>
  <h1>The Maltese Falcon</h1>
  <p>The film stars Humphrey Bogart as private investigator Sam
  Spade and Mary Astor as his femme fatale client.</p>
</article>
```

To turn an article into an h-entry, we'll add a `class` attribute value of `h-entry` as this is the root element for each individual entry:

```
<article class="h-entry">
  <h1>The Maltese Falcon</h1>
  <p>The film stars Humphrey Bogart as private investigator Sam
  Spade and Mary Astor as his femme fatale client.</p>
</article>
```

Some properties of `h-entry` are implied — including `p-name` for the title of the article, post or podcast — and while every property is optional, it's best to include certain properties such as the publication date and the name of the author in every entry. The values for those properties should be very familiar to you by now. Let's first apply an explicit `p-name` value to our main heading:

```
<h1 class="p-name">The Maltese Falcon</h1>
```

When we need to say explicitly that a date or time refer to an article publication date, we use the h-entry `dt-published` value:

```
<time datetime="2015-11-20 T19:30" class="dt-published">
November 20th, 2015 at 7:30pm</time>
```

If an `h-entry` is updated after the published date, we should change the `dt-published` value to one of `dt-updated`:

```
<time datetime="2015-11-20 T21:30" class="dt-updated">
November 20th, 2015 at 9:30pm</time>
```

Our final value to add to an h-entry entry is its author. You should be familiar with h-cards by now, so here we'll combine the **h-card** value with that of **p-author**:

```
<address class="h-card p-author">  
<a href="http://stuffandnonsense.co.uk">Andy Clarke</a>  
</address>
```

There's no need for either **p-name** or **u-url** values here as these are implied by **h-card**.

But wait. W... wh... what's with that **address** element?

As mentioned earlier, the **address** element wasn't intended to describe physical addresses but it is absolutely the right element to use for an author's contact information. Because we're adding a link to this author's website we should define the relationship to that page as one of author using an HTML link relationship:

```
<address class="h-card p-author">  
<a href="http://stuffandnonsense.co.uk" rel="author">Andy Clarke</a>  
</address>
```

Some authors often like to split blog entries across more than one page; for example, we might have summaries on our home page or in our archives, then present the full entry on its own page. h-entry can define a short segment of an article as a summary using **p-summary**.

For our example, here we'll use the first paragraph:

```
<p class="p-summary">The film stars Humphrey Bogart as private in-  
vestigator Sam Spade and Mary Astor as his femme fatale client.</p>
```

For a longer summary — one that consists of several elements — group them together in a `section` and then apply `p-summary` to that:

```
<section class="p-summary">
  <p>The film stars Humphrey Bogart as private investigator Sam
  Spade and Mary Astor as his femme fatale client.</p>
  <p>The story follows a San Francisco private detective and his
  dealings with three unscrupulous adventurers, all of whom are
  competing to obtain a jewel-encrusted falcon statuette.</p>
</section>
```

When a `p-summary` appears on a different page, it's important to include a permanent link to the full article. To make it clear that the link destination is related to the `p-summary`, add a `rel` attribute with a value of `bookmark`:

```
<a href="http://Hardboiledwebdesign.com" rel="bookmark">
Permalink</a>
```

Today, it's common for people to republish their content in several places. For example, you might publish an entry on your blog, but also on Medium to possibly gain a wider audience. When you link to a copy of your entry elsewhere, it's important to mark that link as syndicated content using the `u-syndication` value:

```
<a href="http://medium.com" class="u-syndication">
Also published on Medium</a>
```

Managing multiple h-entry instances

So far, we've been working with a single **h-entry**, but many websites have lists of related articles on their home and archives pages. These entries combined are known as an **h-feed**. To assemble a feed, all we need is an appropriate parent element and for this we'll use a **section**. As a section should make sense when taken out of the context of a page, we'll also give it a descriptive heading:

```
<section class="h-feed">
  <h1>Hardboiled archives</h1>
  <article class="h-entry"> [...] </article>
  <article class="h-entry"> [...] </article>
  <article class="h-entry"> [...] </article>
</section>
```

Breaking it up

Microformats supercharge the meaning of your HTML and give it some often desperately needed structure. For web designers and developers, microformats offer a way to break away from the presentational ways we have written HTML in the past, and to liberate our documents by making them more flexible, adaptable and hardboiled.

No. 9

WAI-ARIA roles

YOU'VE ALREADY LEARNED HOW HTML ELEMENTS and microformats have brought markup into the web application age. You might not have heard that there's another specification that has different but complementary goals. That specification is WAI-ARIA, the Accessible Rich Internet Applications suite.²⁴

WAI-ARIA aims to make web content easier to use by people who use assistive technologies and it includes:

- Roles for widgets such as a navigation menus, sliders and progress meters.
- Properties that define dynamically updated sections of a page.
- Ways to enable keyboard navigation.
- Roles to describe the structure of a page, including headings, regions, and tables (grids).

All this sounds great, but what makes WAI-ARIA hardboiled? As well as providing valuable help to people who rely on assistive technologies, web designers and developers can use WAI-ARIA roles to help us reduce our reliance on presentational `id` and `class` attributes. After all, why would we add a class of `banner` to an HTML element just for the purpose of styling, when we can do away with the `class` altogether and use a CSS attribute selector to bind styles to a WAI-ARIA role?

²⁴ w3.org/TR/wai-aria

WAI-ARIA landmark roles

WAI-ARIA includes a set of navigation landmark roles, which help people with disabilities identify common sections of a page or web application and navigate around them using assistive technologies. These roles can be used in combination with HTML elements to maximise their semantics.

We'll be covering several specific WAI-ARIA roles that offer us the opportunity to hard boil our HTML and CSS by making it less necessary to pack our markup with `class` and `id` attributes. These WAI-ARIA roles include `banner`, `complementary`, `contentinfo`, `main`, `navigation` and `search`.

To add a WAI-ARIA role, we simply apply the `role` attribute to any appropriate element. For example, when we're marking up a branding area or masthead, apply a role of `banner`.

Banner role

In HTML, the `header` element can be used for a branding or masthead area, and it often appears at the top of a page. The WAI-ARIA `banner` role helps people who use assistive technologies recognise this particular `header` and distinguish it from others on the page:

```
<header role="banner">  
<h1>It's Hardboiled</h1>  
</header>
```

But unlike a plain HTML `header` element — which can be used as many times as we need inside multiple `section` and `article` elements — we must use a `header` with the role of `banner` only once.

Complementary role

The WAI-ARIA **complementary** role is similar in function to the HTML **aside** element. It delineates content that is somehow related to and supports other content, although it doesn't have to be either contained by or visually linked to that content. If we're writing an article about the hardboiled author Mickey Spillane, we could apply the **complementary** role to an **aside** about his famous book *My Gun Is Quick*.

```
<aside role="complementary">
  <h2>My Gun Is Quick</h2>
  <p>Mickey Spillane's second novel featuring private
    investigator Mike Hammer.</p>
</aside>
```

Contentinfo role

WAI-ARIA defines the **contentinfo** role as a “perceivable region that contains information about the parent document.”²⁵ Does that sound like a HTML **footer** to you? Me too. Let's continue developing our 'Get Hardboiled' archives page by adding the **contentinfo** role to the main page **footer**.

```
<footer role="contentinfo">
  <h3>It's Hardboiled</h3>
  <p>Hardboiled Web Design, designed by Andy Clarke.</p>
</footer>
```

Just like the **banner** role and unlike HTML's plain **footer** element, we must only use a **footer** with the role of **contentinfo** once on each page.

²⁵ w3.org/TR/wai-aria/roles#contentinfo

Main role

Skip-to-content links form one of the most commonly used web accessibility techniques, intended to help people who rely on assistive technologies to skip past repetitive blocks of navigation. WAI-ARIA's **main** role aims to eliminate the need for skip links because it helps assistive technology users navigate straight to a page's main content.

Where we apply the **main** role depends entirely on our content, and on the 'Get Hardboiled' archives page we're developing, we might choose to add it to a **section** that contains the latest, most important news.

```
<section id="content__uk"> [...] </section>
<section id="content__usa" role="main"> [...] </section>
<section id="content__world"> [...] </section>
```

If we're developing a page that contains just one story, we should add the **main** role to an **article** element.

```
<article role="main">
  <header>
    <h1>Mickey Spillane</h1>
  </header>
  <p>Frank Morrison Spillane, better known as Mickey Spillane,
  was an author of crime novels, many featuring his detective char-
  acter Mike Hammer. More than 225 million copies of his books have
  been sold internationally, including my personal favourite, 'My
  Gun Is Quick'.</p>
</article>
```

Navigation role

WAI-ARIA's **navigation** role is similar in function to HTML's **nav** element as it's intended to describe the major navigation blocks in a page or web application. We'll apply the role even though **navigation** and **nav** serve the same purpose to give the widest possible support.

```
<nav role="navigation">
  <ul>
    <li><a href="about.html">What's Hardboiled?</a></li>
    <li><a href="archives.html">Archives</a></li>
    <li><a href="authors.html">Authors</a></li>
    <li><a href="Classics.html">Classics</a></li>
  </ul>
</nav>
```

Search role

On many sites, searching is the primary way people navigate to content. In HTML, it's therefore perfectly acceptable to embed a search form inside a **nav** element, but the same isn't true of an element given WAI-ARIA's **navigation** role. Why? Because WAI-ARIA includes its own dedicated role for search.

WAI-ARIA's **search** role describes a complete search interface — including labels, inputs, buttons and other HTML elements. In the past, when we wanted to style a search form, we gave it a unique **id** or perhaps a **class** attribute. Now we can stop adding presentational attributes and use WAI-ARIA's **search** role and a CSS attribute selector instead:

```
<form method="post" action="search.html" role="search">
  <fieldset>
    <input type="search" title="Search this site">
    <button type="submit">Go</button>
  </fieldset>
</form>
```

Breaking it up

Accessibility matters, not only to those people who rely on our work being readily available through screen readers and other assistive technologies, but for the integrity of the work that we make. WAI-ARIA roles are just one of the ways that we can help improve accessibility.

But there's more to them than that. Like microformats, WAI-ARIA roles in HTML allow us to reduce our reliance on presentational elements and attributes, setting our markup free and untying it from a single design. By binding CSS styles to WAI-ARIA roles instead of attributes which serve only a visual design, our documents become more flexible and better equipped to render in the many different browsers and devices that people now use to access our websites and applications.



HARDBOILED CSS

In **Hardboiled CSS**, you'll learn about flexible box layout — a new foundation for responsive layouts — web fonts for better type and typography, how to layer colour with RGBA and how to use opacity. You'll discover how use multiple background images and how to make borders rounded and full of images. You'll wind up knowing how to replace many images with CSS gradients to make your designs lighter and more responsive.

All the while you'll be making your design look fabulous across responsive breakpoints and that's where we'll start, with CSS media queries.



No. 10

Hardboiled foundations

IT SEEMS INCREDIBLE NOW, looking back, but I chose to end the first edition of *Hardboiled Web Design* with a chapter that included CSS media queries and I didn't mention them anywhere else in that book. The example sites that I chose to illustrate media queries were the personal sites of well-known web designers because at that time designers and developers were still coming to terms with the challenges of responsive web design and there were few, if any, high-profile commercial examples.

It seems fitting that what was the final chapter in the the last edition is the first technical chapter in this one, because today our industry, what we make and how we make it are very different indeed. So much so, that there's very little we can do without working with the tools and technologies that we'll cover in this chapter.

CSS media queries

CSS2 introduced media types that gave us the ability to specify different styles — and even entirely different style sheets — based on a type of device. In the example below, the styles in *screen.css* will only be applied on screen, while a printer will use styles from the *print.css* style sheet.

Serving alternative style sheets to screens and printers is as simple as adding a **media** attribute:

```
<link rel="stylesheet" media="screen" href="screen.css">  
<link rel="stylesheet" media="print" href="print.css">
```


CSS3 media queries then made it possible for us to define precisely which styles get applied under which circumstances. They work by querying a device's features including its:

aspect-ratio	color	device-aspect-ratio	device-height
device-width	height	monochrome	max-width
max-height	orientation	resolution	width

A media query combines a media type, such as a screen or printer, with a device or screen attribute, such as its size or shape or characteristics. This combination of two or more queries enable us to serve style declarations to devices or screens which match those queries.

Linking media queries

We can use media queries in two ways, the first by linking to an external style sheet that contains styles that suit that particular query; for example, smaller, medium and larger screens. We'll include our query in the `link` element and serve styles to screens with a minimum width of `48rem`.

Using this method we might choose to serve styles for all browsers in our first style sheet, then follow that with other media queries and style sheets:

```
<link rel="stylesheet" media="screen and (min-width: 48rem)"
href="medium.css">
<link rel="stylesheet" media="screen and (min-width: 64rem)"
href="large.css">
```

This might seem at first like a sensible method for keeping styles for different media queries separate — but beware. A browser will download every style sheet linked to every media query — even when it doesn't apply its styles — slowing down the performance of a website or application.

Embedding queries

We might also embed media queries and styles into an external style sheet using the `@media` at-rule. This increases the size of an individual style sheet but it also reduces the number of requests that a browser makes to a server, which also has a positive impact on performance. In this next example we'll move a `figure` element's caption to the top using `display: flex` and `flex-direction: row-reverse` only when a browser is more than `48rem` wide:

```
@media (min-width: 48rem) {  
  .figure--horizontal-reverse {  
    display : flex;  
    flex-direction : row-reverse; }  
}
```

Drinking at a seedy bar on a rainy night, Hammer notices a man come in with an infant. The man, named Decker, cries as he kisses the infant, then walks out in the rain and is shot dead. Hammer shoots the assailant as he searches Decker's body.

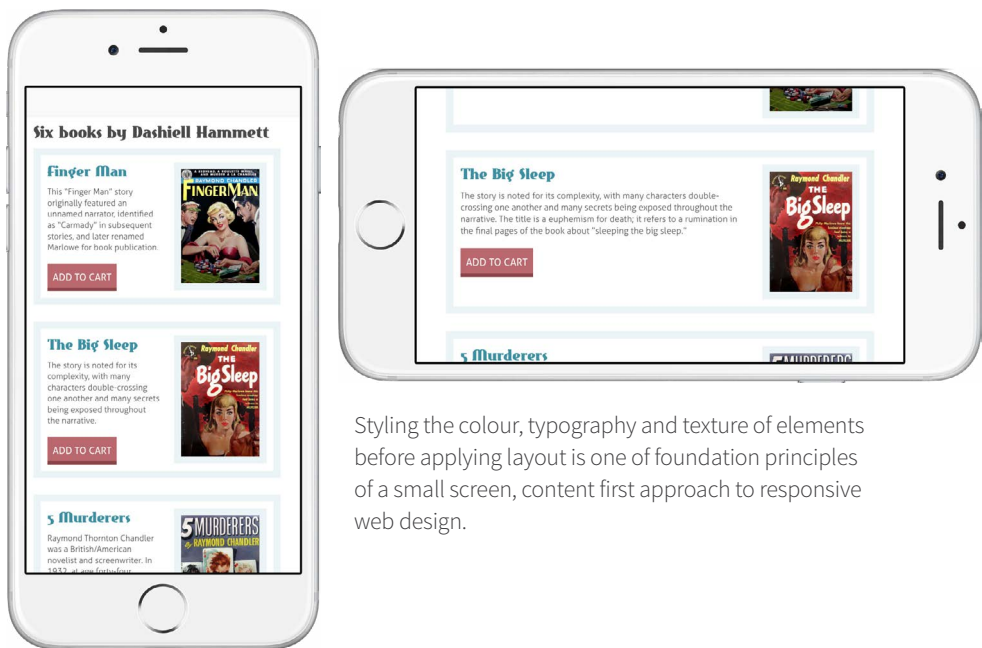


In addition to changing page layouts, we can use media queries to enhance the look of individual page elements across responsive breakpoints.

It's possible to choose any minimum or maximum width, or even height, for our media queries, and we can use units from other parts of CSS in those queries too, including `px`, `em` and `rem`. But how should we choose at which point to break to a new set of styles?

Starting with common styles

It's important for the performance of our websites and applications — as well as for our own sanity — that we don't write styles more often than we need to. There's little point in writing the same style declaration several times across several style sheets or media queries, so instead we should layer styles progressively, starting from the smallest breakpoint, which is in fact no breakpoint at all.



Styling the colour, typography and texture of elements before applying layout is one of foundation principles of a small screen, content first approach to responsive web design.

The styles we create when we design a website's atmosphere will stay pretty much the same across all responsive breakpoints. Sure, we'll no doubt change our font sizes and leading, but the typefaces we chose for the smallest screens will almost certainly be the same for the largest. The padding within our buttons and form elements may vary, but their basic styles will stay exactly the same.

Before we add our first media query we should organise the styles that form the atmosphere of our design. How we organise our style sheets is often a very personal choice. I organise mine into six groups of elements whose styles transcend breakpoints:

1. Reset or normalise
2. Site-wide page styles
3. Typography
4. Form elements
5. Tables
6. Images

Organising styles this way makes it simpler to identify and style elements across breakpoints as a design demands. While we will probably later make a myriad of changes throughout our breakpoints, this small screen first approach means we're building on top of core styles only when needed, resulting in simpler and more maintainable style sheets.

Choosing breakpoints

When designers and developers first started to come to terms with responsive web design, it was commonplace to find us defining breakpoints at the precise widths of specific devices, most commonly iPhone, then iPad and finally every other screen wider than that.

We shouldn't blame ourselves for thinking that way, as often our bosses or clients asked us to design an "iPhone or iPad version" of our websites or applications. As the number of devices and screen sizes grew, this approach not only became impractical, it was undesirable.

As our knowledge of how to design responsively has developed, defining breakpoints more generally to suit the look of our content rather than device sizes has become more common. Choosing breakpoints like this isn't always easy and it demands that we think differently about the ways we approach implementing our designs. Ultimately, though, it will make our designs more adaptable.¹

Let's use this example of a layout that's divided into columns using CSS multicolumn layout. You'll learn about implementing these columns in a short while.

If we're following a device-specific approach, we might base our decision on the number of columns on the width of a device; for example, an iPhone 6s in landscape orientation:

```
@media only screen
and (max-device-width : 41.6875rem) {
  section {
    column-count : 2; }
}
```

Rather than tie our design to a specific screen size, we should use our knowledge of typography to decide the number of columns based on the readability of the measure. Too few words per line and the reading experience will be awkward; too many and it will be difficult.

¹ BreakpointTester is a Chrome plugin to test a sites responsiveness at various screen sizes, helping us to choose breakpoints based on content rather than devices:
smashed.by/breakpointtester



There are no hard and fast rules for when to add breakpoints and we should create them at widths that make sense for the content that we're styling.

Knowing our font size, we can count the number of characters per line and then add columns to produce a layout that's optimised for reading:

```
@media only screen
and (min-width: 48rem) {
  section {
    column-count : 2; }
}

@media only screen
and (min-width: 76.250rem) {
  section {
    column-count : 3; }
}
```

I know from my own experience that shifting my approach to choosing breakpoints away from device types and towards content-based media queries has been more difficult than I imagined. I'd been thinking about my designs in terms of a canvas, as well as its contents, for too many years for my old habits to die easily. It's helped me to use a transitional approach, one that includes general, major breakpoints as well as content-based, adjustment breakpoints.

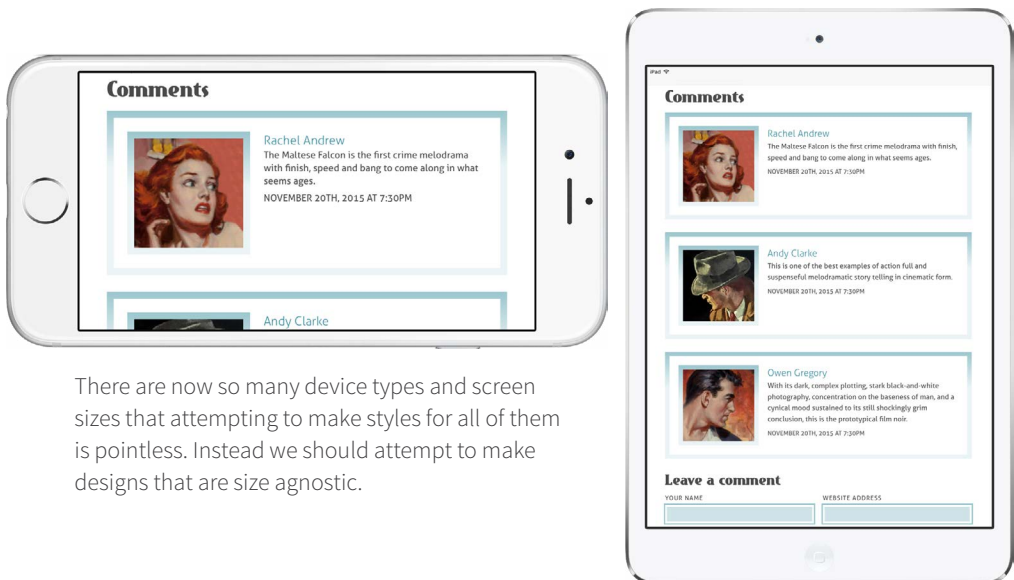
Transitioning breakpoints

Brad Frost wrote that:

“ Every time you see 320px, 480px, 768px, 1024px used as breakpoint values, a kitten gets its head bitten off by an angel...or something like that.²

I'm no fan of kittens and I'm certainly no angel, but while I agree with Brad that we shouldn't base our breakpoints on particular pixel dimensions, a more general range of widths in combination with content-based queries is not a bad approach. In fact, it's one I now use every day.

Distinctions between screen sizes on larger smartphones and smaller tablets have blurred, as have those between larger tablets and smaller PCs. There's very little difference today between designing a layout for an iPhone 6s in its landscape orientation or an iPad mini.



There are now so many device types and screen sizes that attempting to make styles for all of them is pointless. Instead we should attempt to make designs that are size agnostic.

² bradfrost.com/blog/post/7-habits-of-highly-effective-media-queries/

Instead of using pixel units based on the size of screens, we can substitute them for em and rem units for more flexible media queries. As these units are based on the size of our text, our layouts will change at breakpoints that are related to the size of our content. When someone uses their browser controls to zoom our content, our layout will adapt accordingly and they will see a layout that's appropriate to their zoom factor.

We can group classes of screen sizes together to form major breakpoints at which significant aspects of our design, in particular its layout, could change. These major breakpoint sizes are from the toolkit we've developed for our projects at Stuff & Nonsense:

```
/* 768px/16px (base font size) = 48rem */
@media (min-width: 48rem) {
  [...]
}

/* 1024px/16px (base font size) = 64em */
@media (min-width: 64em) {
  [...]
}

/* 1220px/16px (base font size) = 76.25em */
@media (min-width: 76.25em) {
  [...]
}

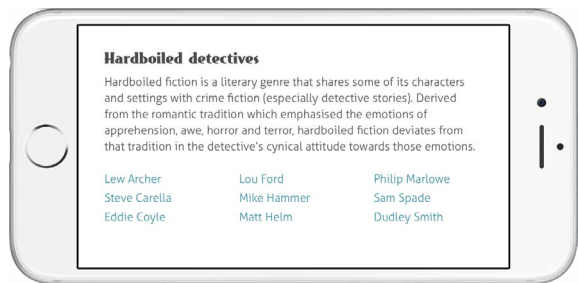
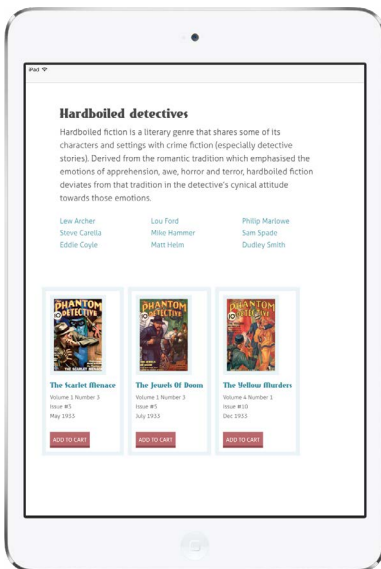
/* 1400px/16px (base font size) = 87.5em */
@media (min-width: 87.5em) {
  [...]
}
```

Styles within these breakpoints build on each other as screen width increases.

Using major and adjustment breakpoints

Even if we adopt a content-based approach to choosing breakpoints, many of the biggest changes to our design will occur at major breakpoints. We might move navigation from the bottom of a page to the top using flexbox. We may need to implement a sidebar when screen width allows, or change the number of columns in our content from two to three. But not every change will happen at these major breakpoints.

Perhaps we'd like to divide an unordered list of items into two columns to maximise its use of space. Perhaps we need to change the padding on a group of buttons, to prevent them from wrapping onto two lines instead of one. Sometimes we need to change the look of elements at breakpoints that are different from the major ones we chose earlier. That shouldn't be a problem — in fact, that attention to detail is really what responsive web design is all about.



The arrangement of elements into different layouts happens at major breakpoints, but it's thoughtful attention to detail at minor breakpoints that can turn an average design into a special one.

Even though styles at our first major breakpoint won't be applied until a screen is **48rem** wide, we should still ensure that our group of buttons looks its best.

```
/* Minor breakpoint */
@media (min-width: 30rem) {
  .btn {
    padding : 1rem 1rem .75rem; }
}

/* Major breakpoint */
@media (min-width: 48rem) {
  .btn {
    padding : 1.25rem 1.25rem 1rem; }
}
```

Jeremy Keith — who has a knack for coming up with useful and witty ways to describe what are often dry, technical subjects — calls these minor breakpoints tweakpoints:

“*It feels a bit odd to call them breakpoints, as though the layout would “break” without them. Those media queries are there to tweak the layout. They’re not breakpoints; they’re tweakpoints.*”³

Dealing with specific issues

With the possible exception of BlackBerry’s square-shaped Passport smartphones, every smartphone and tablet I’ve ever seen has two orientations: portrait, where its height is greater than its width; and landscape, where the reverse is true.

³ adactio.com/journal/6044

Orientation queries

While we should always strive for screen independence, there may be occasions where we might need to style elements differently based on a device's orientation and not just its viewport width.

Mike Hammer wakes up being questioned by the police in the same hotel room as the body of an old friend from friend, Chester Wheeler, has apparently committed suicide with Hammer's own gun after they had been drinking.



Who said that a figure's caption must always go below an image? A change that's unexpected can surprise and delight someone using our websites and applications, so dare to be different.

In that example, we'll place a **figure** element's caption above its image when a device is held in **portrait** orientation:

```
@media (orientation:portrait) {  
  .figure {  
    display : flex;  
    flex-direction : row-reverse; }  
}
```

Now, to make the best use of a device's **landscape** screen, we'll change the layout of that **figure**'s elements by placing the image on the left and the smaller **figcaption** on the right.



Mike Hammer wakes up being questioned by the police in the same hotel room as the body of an old friend from World War II. His friend, Chester Wheeler, has apparently committed suicide with Hammer's own gun after they had been drinking all night.

Making the most of available space is important on smaller screens and media queries are a useful tool for adapting designs to suit landscape orientation screens.

To make the design of this **figure** even more interesting, we'll align both image and caption to the bottom of the **figure**:

```
@media (orientation:landscape) {  
  .figure {  
    display : flex;  
    align-items : flex-end; }  
  img {  
    flex : 2 0 360px; }  
  figcaption {  
    flex : 1; }  
}
```

Regardless of whether the device we're holding is a smaller smart-phone or a medium-sized tablet, this figure's layout will change between those two orientations.

Aspect ratio queries

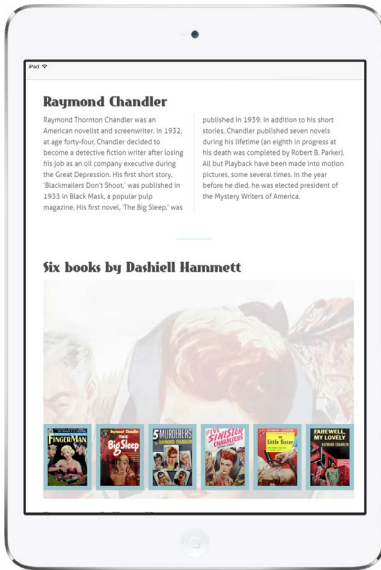
Should a project make it necessary to style elements differently depending on a screen's aspect ratio, there are two queries that can help us, `aspect-ratio` and `device-aspect-ratio`.

The `aspect-ratio` relates to the ratio of height to width of a browser's window and styles within a query will only apply at that precise size. Should someone resize their browser's window and the aspect ratio change, those styles will no longer apply. The `device-aspect-ratio` refers to the fixed aspect ratio of a device itself.

Aspect ratios are represented by two numbers separated by a colon. Two of the most commonly used ratios are 4:3 and 16:9, where the first number represents the horizontal and the second the vertical. In CSS these numbers are separated not by a colon but by a slash.

In our next example we'll change this list of books by detective fiction writer Dashiell Hammett, so that their layout suits first 4:3:

```
@media (device-aspect-ratio: 4/3) {  
  [...]  
}
```



The iPad's 4:3 ratio suits its larger form factor, whereas the iPhone's 16:9 suites its smaller size better.

Then a wider format 16:9 **device-aspect-ratio**:

```
@media (device-aspect-ratio: 16/9) {
  [...]
}
```

While it's unlikely that you'll use these **aspect-ratio** queries in the majority of your projects, they're incredibly useful for fine-tuning the layout of your designs for different sizes and types of screens. After all, that is what responsive web design is really about.

Height-based queries

So far, most of the media queries we've looked at have used width of some kind as their most important factor, but a device's or screen's height can also play an important factor in our decisions about layout. We can't always assume that people reading our content or using the functionality on our sites will have a screen that's tall enough to make doing that comfortable.

Apple's 11" MacBook Air laptop is a wonderfully portable computer, but its screen height is short enough to make our designs look awkward. Fortunately, there are several queries that can help us: `height`, `min-height` and `max-height`; and `device-height`, `min-device-height` and `max-device-height`.

For this device and those like it with vertically challenged screens, we could reduce the vertical spacing between and within elements; for example, a design's overall line height, and top and bottom padding within banners and navigation:

```
p {  
  line-height : 1.5; }  
[role="banner"],  
[role="navigation"] {  
  margin-top : 1.5rem; }  
@media (device-height: 56.25rem) {  
  
  p {  
    line-height : 1.4; }  
    [role="banner"],  
    [role="navigation"] {  
      margin-top : 1.3rem; }  
  }
```

Small adjustments like this can make an enormous difference to a person's experience of using a site or application when their device has a shorter screen.

Combining queries

We might need to target those people who use Apple's 11" MacBook Air even more precisely by combining two or more media queries. We do this by including `and`, `only` and `not` operators between our queries, like this:

```
@media screen
and (min-width: 48rem) {
  [...]
}
```

Styles within these declarations will only apply when a device has a screen that's wider than 48rem. Printers that have that same width won't apply those styles.

In this next example, styles will be applied when a device is only a screen with a `device-height` above 900px:

```
@media only screen
and (min-device-height: 56.25rem) {
  [...]
}
```

This way we can be extremely precise to combine type, `min-device-height` and `device-aspect-ratio` media queries that will match that 11" MacBook Air laptop:

```
@media only screen
and (min-device-height: 56.25rem)
and (device-aspect-ratio: 16/10) {
  [...]
}
```

So far, these combinations have all resulted in a single query, and styles will only be applied when all of the queries return true. If we should ever need to make two queries, perhaps to match the different possible screen resolutions of that 11" MacBook Air, we can make a comma-separated list of media queries which will return true if any of the media queries returns true:


```
@media only screen
and (min-device-height: 56.25rem)
and (device-aspect-ratio: 16/10),

screen
and (min-device-height: 37.5rem)
and (device-aspect-ratio: 4/3) {
  [...]
}
```

You spin me right round, baby, right round

When we first began to make responsive web designs, it was commonplace to target specific devices — in particular iPhone and iPad — now that people use an ever increasing number of device types and screen sizes to look at our work, it makes little sense today.

However, there can be situations when we may need to serve styles to a specific device type, perhaps if we're styling an application that's used by people who use an iPad Pro in landscape orientation. Targeting specific devices involves combining two or more queries to form one long conditional query, so to target that landscape iPad Pro screen we'll group both `min-device-width` and `max-device-width` queries, an orientation query and one that filters out low-resolution screens:

```
@media only screen
and (min-device-width: 96rem)
and (max-device-width: 128rem)
and (orientation: landscape)
and (-webkit-min-device-pixel-ratio: 2) {
}
```

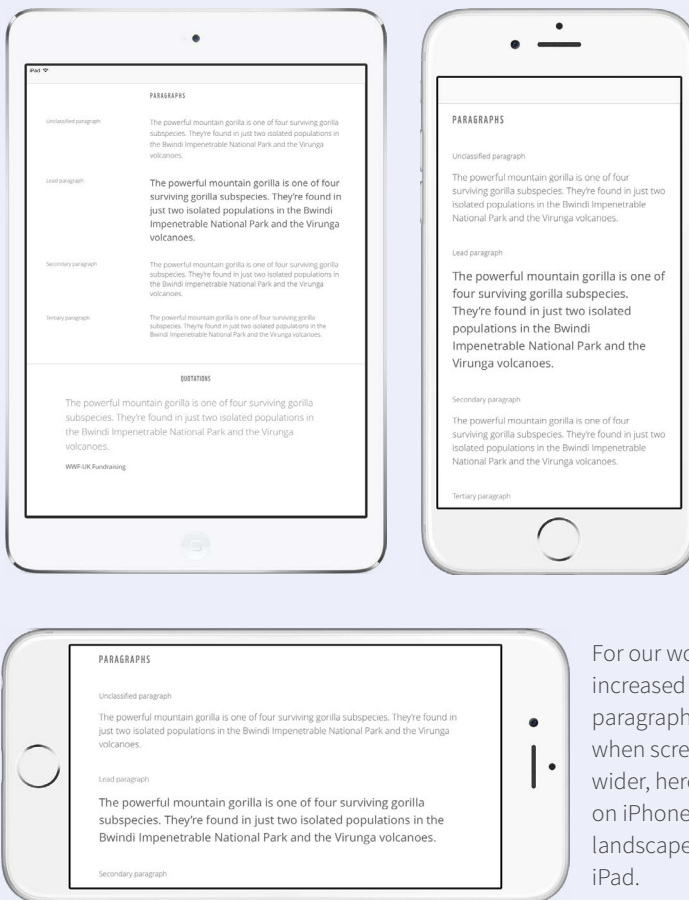
Justin Avery, host of the Responsive Design Podcast⁴, has written about why we should neither need nor use device-specific media queries:⁵

⁴ responsivedesign.is/podcasts

⁵ responsivedesign.is/articles/why-you-dont-need-device-specific-breakpoints

Proportional leading with media queries

Responsive website designs often cry out for fine control over typography. One of the most effective ways to improve readability is to adjust text size and leading (**line-height**) in relation to the width of the measure (text-column width).



For our work for WWF, we increased the leading of paragraph and other text when screen sizes become wider, here demonstrated on iPhone in portrait and landscape orientation and iPad.

As a general rule of thumb, leading should increase as the measure increases. This helps our eyes follow where one line ends and the next begins. Media queries allow us to precisely control leading, using a query of the window's or device's width and descendant CSS selectors. In this example we'll set `line-height` to `1.4` for the smallest of screens and the narrowest of columns:

```
p {  
  line-height : 1.4; }
```

As the screen width and columns become wider, leading should be increased. By how much will depend on the font size we've chosen and the typeface itself. We'll increase the line height in increments, starting with a minimum width of `48rem` and a line height of `1.5`:

```
@media (min-width: 48rem) {  
  p {  
    line-height : 1.5; }  
}
```

Finally, we'll set line height to `1.6` when the minimum window width is `64rem`:

```
@media (min-width: 64rem) {  
  p {  
    line-height : 1.6; }  
}
```

As the measure becomes wider, the more open the leading will become. The narrower the measure, the tighter the leading, improving readability and a person's overall experience of our design.

Feature queries

Media queries use the `@media` at-rule, but they're not the only conditional rules in CSS that use an at-rule. Whereas media queries test for media characteristics and then apply media-specific styles, feature queries use their own `@supports` at-rule to apply styles when a browser supports certain CSS declarations.

In this first example, we'll reduce the `font-size` of a `figcaption` when a browser supports `display:flex`. This will help make the caption more readable in the smaller width that we might apply to it.

```
@supports (display:flex) {  
  .figure--horizontal figcaption {  
    display : flex; }  
}
```

Look closely at that example and you might notice that the query isn't simply testing for support of the `display` property, but for a property/value pair that includes both `display` and `flex`. In practice this means we're able to be precise about the support we're testing for. For example, we might want to apply CSS multicolumn layout styles when we know a browser supports the `column-span:all` declaration. This declaration isn't yet supported in Firefox, making multicolumn layout less useful.

```
@supports (column-span:all) {  
  section {  
    column-count : 2; }  
}
```

Unlike when we build media queries, it's best practice when using feature queries to provide two alternative sets of styles, depending on whether a browser supports a CSS declaration or not.

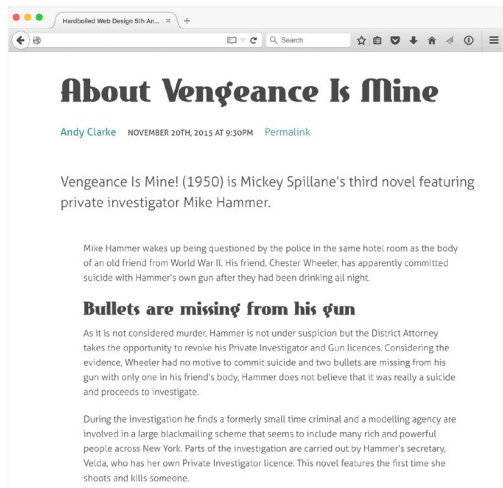
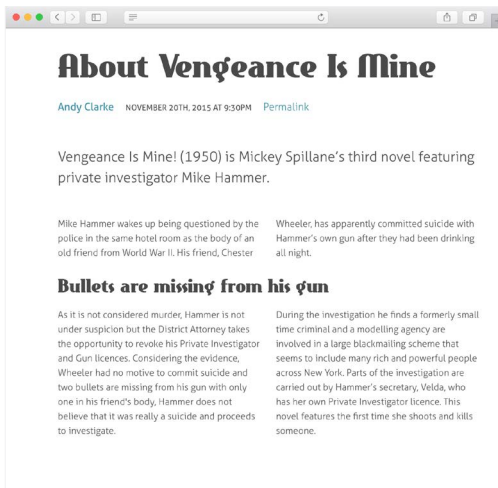
If you want to dive deep into `@supports` feature queries, David Walsh's article is a terrific place to jump to.⁶

⁶ davidwalsh.name/css-supports

We do this using the `not` operator. Let's apply that best practice to our CSS multicolumn layout example and add `padding` to reduce the measure when a browser like Firefox doesn't yet support `column-span:all`:

```
@supports (column-span:all) {
  section {
    column-count : 2; }
}

@supports not (column-span:all) {
  section {
    padding : 0 4rem; }
}
```



All browsers that support `column-span:all` will render those two columns, but browsers like Firefox that don't will add padding to the section instead.

As well as simple feature queries like this, we're able to apply styles only when a browser passes two or more `@supports` tests. This can be particularly useful for targeting browsers that support both the native and vendor-specific prefixed version of a property. We can do this using the `or` operator:

```
@supports (column-count:2)
or (-webkit-column-count:2) {
  section {
    column-count : 2; }
}
```

We might also use the `and` operator to ensure that a browser supports two or more declarations. Using Firefox's lack of support for `column-span:all` as an example again, we can write a feature query that applies style to a section when a browser supports both `column-count:2` and `column-span:all` declarations:

```
@supports (column-count:2)
and (column-span:all) {
  section {
    column-count : 2; }
}
```

Supporting browsers

Microsoft's Edge browser is its first to implement CSS feature queries and Edge completes the line-up of all contemporary desktop and mobile browsers that have implemented the `@supports` at-rule. Unless you demand that styles contained within feature queries be visible to older versions of Microsoft's Internet Explorer, there are no reasons to avoid using `@supports`.

Modernizr

As I was planning the first edition of *Hardboiled Web Design*, an email landed in my inbox asking if I'd like to try an as yet unreleased JavaScript feature detection library. I couldn't have imagined that Modernizr would be so important that it would become the foundation that underpinned the entire concept of hardboiled web design in that first book.

While the technologies in Modernizr have changed almost beyond recognition as the library has been developed, its principles and how we use it have stayed very much the same. Modernizr is a light-weight JavaScript library and it detects HTML and CSS features that are supported by a browser. When a page loads, Modernizr runs its feature tests and adds class attribute values to the `html` element based on its results. These are the feature tests that I commonly configure Modernizr to perform.

Background Blend Mode	Flexbox
Border Image	Gradients
Calc	Shapes
Columns	Supports
Filters	Vw and vh units

We can take advantage of those classes and apply different styles to browsers that either do or don't support the property tested.

Is Modernizr still relevant?

That's a good question. When Modernizr was released and the first edition of *Hardboiled Web Design* was published a few months later, browser support for many new CSS properties was patchy at best. There was still a significant gap between the best and worst performing browsers, and Modernizr was an essential tool for either dividing style sheets into blocks of support or no support, or for progressively enhancing elements using more specific selectors, like this:

```
section {  
  padding : 0 4rem; }  
  
.csscolumns section {  
  padding : 0;  
  column-count : 2; }
```

Today that gap has closed so much that almost all current desktop and mobile browsers offer similar levels of support for most CSS properties. So much so that whereas five years ago I used Modernizr on every website I made, my use of it is now much more targeted. Today I use Modernizr to test for very specific CSS technologies, in particular columns and flexbox, plus its detection of SVG. For those purposes, Modernizr is still incredibly relevant and remains a very powerful tool when put to the right jobs.

Using Modernizr

We don't need to carry a gun to be a hardboiled web designer. We won't be bumping anyone off, unless our clients start being bunnies. What we do need is a pocketful of tools to make hardboiled web design practical and Modernizr is exactly the right type of tool.

On the Modernizr website, choose between a larger development version of the script, or a customised version that includes only the features we're testing for. With performance so critical today, we shouldn't use the development version on a live website. When we've configured the appropriate build, download the script and link to it in a document.

```
<script src="js/modernizr.js"></script>
```

Keeping one principle of progressive enhancement in mind — that when using any script it's important to consider instances when JavaScript might not be available — add the class `no-js` to the `html` element to ensure basic styling for non-JavaScript enabled browsers and to enable Modernizr's functionality:

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


When Modernizr runs, it replaces `no-js` with `js` and allows us to know when JavaScript is enabled. Modernizr helps us to grade browsers, not by sniffing their user-agent strings but by adding class attribute values to the `html` element based on its tests.

```
<html class="js backgroundblendmode borderimage csscalc csscolumns cssfilters flexbox flexboxlegacy flexboxtweener cssgradients shapes cssvhunit cssvmaxunit cssvminunit cssvwunit">
```

When a browser lacks support for a property or feature, Modernizr adds a `no-` prefix to each class.

```
<html class="js no-backgroundblendmode no-borderimage no-csscalc no-csscolumns no-cssfilters no-flexbox no-flexboxlegacy no-flexboxtweener no-cssgradients no-shapes no-cssvhunit no-cssvmaxunit no-cssvminunit no-cssvwunit">
```

With these attribute values, we can take advantage of support for CSS properties in browsers that support them and decide how to tailor a design for browsers that don't.

Take this example of multiple background images. We might start by declaring basic styles that are understood by even the least capable browsers, in this case a single background image applied to a `section` element:

```
section {  
  background : url(section.png) no-repeat 50% 0; }
```

When Modernizr detects that a browser is capable of rendering more than one background image on a single element, we can serve multiple background images via a more specific descendant selector:

```
.multiplebgs section {  
background-image : url(section-left.png), url(section-right.png);  
background-repeat : no-repeat, no-repeat;  
background-position : 0 0, 100% 0; }
```

The aim of Modernizr isn't to bolt on property support to browsers that don't natively support them. It doesn't attempt to make designs look the same in all browsers. Instead, Modernizr's feature detection makes it possible to serve different designs based on the results of its tests. This means Modernizr is still an essential part of a web professional's toolkit.

Breaking it up

With new device types and screen sizes becoming available all the time, the one-size-fits-all approach to web design we clung to for so long now seems like a distant memory. It's essential that the designs we create are responsive to the different ways users access our content. CSS3 media queries have been implemented in every contemporary browser, so our decision is no longer whether to use them, but how best to use them so that they meet the needs of our designs and the people who use them.

Flexible box layout

No. 11

WHEN I LEARNED HOW TO USE CSS to create page layouts, the most influential tutorials of the day — Rob Chandanaï's Blue Robot Layout Reservoir,* Eric Costello's CSS Layout Techniques⁷ and others — taught CSS positioning.

It's hard to imagine today how revolutionary the techniques they taught were, and although fully positioned layouts were often fraught with problems, they gave us a glimpse of what could be possible using CSS. It wasn't long before positioning techniques gave way to the float as our preferred property and for over fifteen years countless websites have used floats to underpin their layouts.

The Layout Reservoir

About The Layout Reservoir

Please feel free to borrow, steal, abduct, and/or torture the documents contained in the Layout Reservoir. Though you need not give credit to BlueRobot.com, a comment in your source code would help other developers to find this resource. Enjoy.

Two Column Layouts

2 columns - left menu

A simple two column layout with the standard left-side menu.

2 columns - right menu

Practically the same HTML as 2 columns - left menu, but with a different stylesheet.

Three Column Layouts

3 columns - flanking menus

Three columns, no tables, intelligent order of elements. What more is there to say?

Related Info at BlueRobot

Many a talented web designer has struggled with CSS-based centering. Though CSS vertical centering eludes us, two techniques for horizontal centering are BlueRobot approved. Take your pick: [Auto-width Margins](#) or [Negative Margin](#).

*The Blue Robot Layout Reservoir was one of the most important sites in web layout history and I can't tell you how sad I am that it's no longer online except via the Internet Archive Wayback Machine.

⁷ <http://glush.com/css>

There can be bugs in any new technology and flexbox is no different. Philip Walton curates a list of flexbox issues⁸ and work-arounds for them and he hosts the list on GitHub for anyone to contribute to.

Anyone who has struggled with a float-based layout will know that floats have never been the ideal layout tool. Often at the mercy of browser bugs, box sizing issues and, let's not forget, clearing, floats have been an imperfect standard whose suitability as a layout tool has only worsened as the complexity of responsive layouts has increased.

While the emerging flexible box layout or flexbox standard wasn't stable enough to write about in the first edition of *Hardboiled Web Design*, boy oh boy, girl oh girl, have things changed since then.

Not only is browser support now solid across all contemporary desktop and mobile browsers, flexbox has captured the imagination of designers and developers.

It seems that flexbox has captured designers' and developers' imaginations like no other CSS property that I can remember. Hundreds of articles have been written about it and HeyDesigner maintains a list of the best of them.⁹

Flexbox is a next generation layout tool that gives enormous benefits over old-fashioned methods. It enables more responsive layouts. It makes it possible to visually reorder content without laying a hand on our markup, and it lays to rest common frustrations such as equal height backgrounds on unequal height columns.

It's not that you could use flexbox today, you should use it. Unless you have to provide very similar layouts for Microsoft Internet Explorers 9 and 10, there really are no reasons why you can't use flexbox on your websites and applications today.

⁸ github.com/philipwalton/flexbugs

⁹ heydesigner.com/flexbox

Coming to terms with flexbox

The most difficult part of learning flexbox is understanding its visual model. You see, with floats the model is easy as they arrange the elements they affect along a one-dimensional, horizontal axis. Flexbox operates in two dimensions and has both horizontal and vertical axes.

When we make an element flex, we arrange its descendants along a main axis, another axis that crosses it — and sometimes both. This gives us the ability to create layouts that are impossible to make when using floats.

You might find it useful to think of a flex as an imaginary line of string that's been stretched and taped inside two sides of a box or container. In flexbox, this first flex axis is called the main axis and we arrange what we'll call flex-items along this line. These flex-items can be almost any HTML elements used to form a layout.

Just like we justify text to the left, right or centre, we can do the same to any flex-items along a main flexbox axis. In practice, this means flex-items can stick to one side of a flex-container or another, be centred or even stretched between the start of a flex and the end.

We can change the direction of a flex so that its flex-items appear to run in the opposite direction to what's specified in our markup. This offers us a major advantage over traditional layout techniques because what people see on screen can be different from our source order.

Zoe Mickley Gillenwater's presentations on flexbox are always worth watching and while her talks are best when you can hear her explain examples, the slide deck from her 'Enhancing Responsiveness with Flexbox' talk is packed with practical information.¹⁰

Ben Gremillion's 'Laying Out A Flexible Future For Web Design With Flexbox' for Smashing Magazine is also a great starting point when learning about the concepts of flexbox.¹¹

¹⁰ slideshare.net/zomigi/enhancing-responsiveness-with-flexbox-css-conf-eu-2015

¹¹ smashingmagazine.com/2015/08/flexible-future-for-web-design-with-flexbox

We can even switch easily from horizontal rows to vertical columns simply by changing the direction of a flex.

Finally — and perhaps most interestingly — we can change the order in which elements are displayed to create layouts that are more appropriate to specific viewport sizes. There's no doubt that flexbox is a powerful tool, so let's get started using it to create the simple list of pulp magazines that's the foundation of many of our 'Get Hardboiled' examples.

Creating a flex container

For 'Get Hardboiled' we're going to create many different designs, but when you look later at our results, you might be surprised to remember that they're based on largely identical markup.



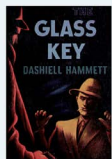
Blood Money

Originally appeared as two short stories in Black Mask in 1927 under the titles "The Big Knockover" and "\$106,000 Blood Money."

[ADD TO CART](#)

The Continental Op

The Continental Op made his debut in an October 1923 issue of Black Mask, making him one of the earliest hard-boiled private detective characters to appear in the pulp magazines of the early twentieth century. He appeared in 36 short stories, all but two of which appeared in "Black Mask."

[ADD TO CART](#)

Glass Key

It was Hammett's favourite of his five novels, and is also the most stylised. The characters are defined only by their outward actions and their inner motivations are often unclear.

[ADD TO CART](#)

A list of hardboiled detective novels where we've styled the images and descriptions using flexbox.

We'll start with a division and classify it as an `item`. Inside, we'll add two further divisions, one for our item's image, the other for a description:

```
<div class="item">
  <div class="item__img">
    </a>
  </div>
  <div class="item__description">
    <h3>The Scarlet Menace</h3>
  </div>
</div>
```

Because of their order in the source, the image division and description will naturally be displayed vertically, one on top of the other, but we're going to change all that by turning the `item` into a flex-container. `flex` is an a new display value that joins `block`, `inline`, `inline-block`, `none` and `table` with its various subproperties.

```
.item {
display : flex; }
```

`flex` turns our item into a flex-container and doesn't remove its `block` attributes, so the division continues to fill all available space inside its own parent element.



Like all block-level elements, this item fills the entire width of its parent element unless we specify otherwise.

When we'd prefer our items not to fill the available horizontal space, or we'd like to turn an inline element into flex-container, we can also create an inline flex:

```
.item {  
  display : inline-flex; }
```

Look at the first example using `display:flex` and you'll notice that our image division and description are no longer displayed vertically. That's because by turning their parent division into a flex-container, we've also turned their descendants into flex-items that are automatically arranged horizontally along the default main axis. The authors of flexible box layout included some very smart default behaviours; for some designs, this could very well be all you'll need.

Flex direction

We're able to arrange flex-items horizontally or vertically and changing these directions is another useful way to create layouts that respond to different screen sizes and orientations. When we specify an element's `flex-direction`, we also specify the direction of its main axis. When we don't specify any direction at all, the initial value is `flex-direction:row`. Let's return to our novel list example and apply a `flex-direction`:

```
.item {  
  flex-direction : row; }
```

Unless we've set the `dir` attribute to `rtl`, for right-to-left languages, this flex starts on the left and ends on the right. As `row` is the initial direction, you shouldn't need to type that last declaration at all, unless you're using it to overwrite a previously applied `flex-direction`.

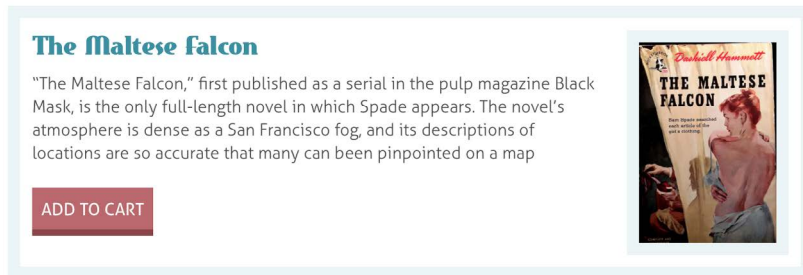
Sara Soueidan wrote a fabulous overview of `flex-direction` as well as other flexbox properties.¹²

¹² tympanus.net/codrops/css_reference/flexbox/#section_flex-direction

Reversing rows

You might remember that the image division came first in our markup and the description second. When we need to display the image on the right instead of its normal position on the left, we don't need to change our HTML. Instead we can simply change the direction of the flex:

```
.item {  
  flex-direction : row-reverse; }
```



Reversing the **flex-direction** changes this horizontal layout without altering the markup.

With the **dir** attribute set to **rtl**, the flex starts on the right instead of the left and arranges flex-items in the opposite direction. This tiny change can have an enormous impact on our layouts.

Reversed columns

Unlike floats which are one-dimensional, flex-containers are two-dimensional as they can be either horizontal rows or vertical columns. Although block-level elements stack vertically, as you'll soon see there are occasions when we need to specify a **flex-direction** as a column:

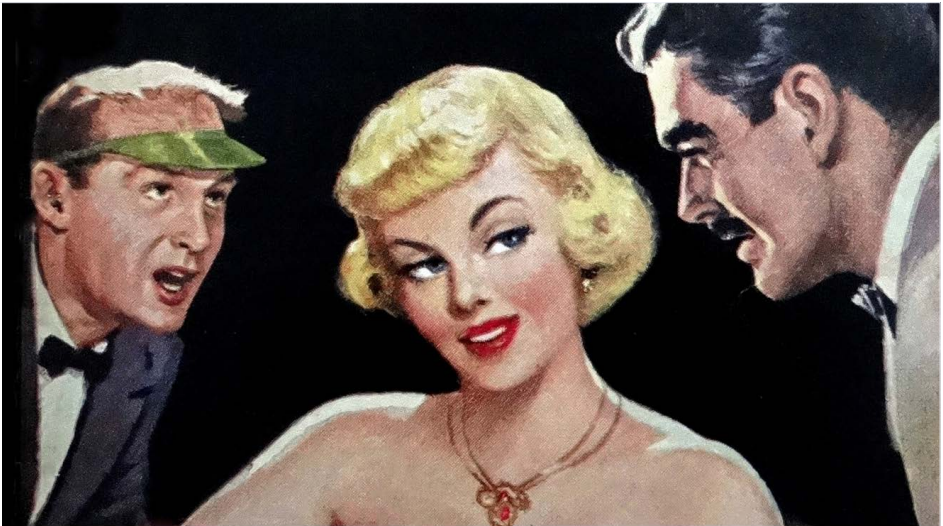
Bennett Feely's Flexplorer is a fabulous visual tool to help you understand the behaviour of the various flexbox properties. It's particular good at demonstrating tricky concepts such as **flex-basis** and **flex-shrink**¹³

¹³ bennettfeely.com/flexplorer

```
.figure--classic {  
  flex-direction : column; }
```

For our next example, we'll use `flex-direction` to make a `figure` element and its caption more interesting. Our markup includes a `figure`, image and its associated `figcaption`:

```
<figure class="figure--classic">  
    
  <figcaption>Pulp magazines were inexpensive fiction magazines  
published until the '50s.</figcaption>  
</figure>
```



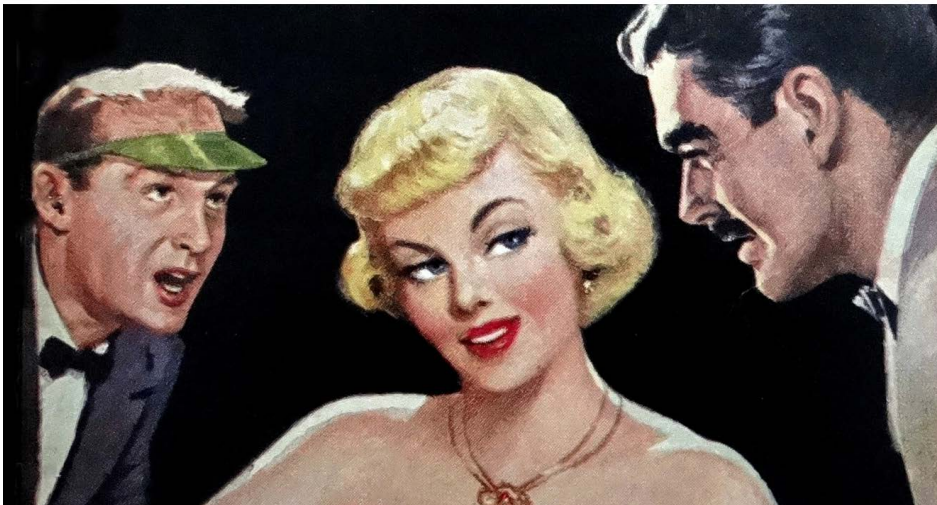
Drinking at a seedy bar on a rainy night, Hammer notices a man come in with an infant. The man, named Decker, cries as he kisses the infant, then walks out in the rain and is shot dead. Hammer shoots the assailant as he searches Decker's body.

In this conventional figure design, the layout matches the source order.

Our `figcaption` is normally displayed below the image, matching its position in the source, but some figures deserve a more interesting layout, so we'll use `flex-direction` to display our caption above the image:

```
.figure--reverse {  
  flex-direction : column-reverse; }
```

Drinking at a seedy bar on a rainy night, Hammer notices a man come in with an infant. The man, named Decker, cries as he kisses the infant, then walks out in the rain and is shot dead. Hammer shoots the assailant as he searches Decker's body.



Reversing the `flex-direction` makes this column design all the more interesting.

So simple, but already so very effective in making this `figure` look more interesting. Now let's finish off that distinctive `figure` by limiting its caption to fifty percent width when viewed on medium and large screens:

```
.figure--reverse figcaption {  
  max-width : 50%; }
```

Drinking at a seedy bar on a rainy night, Hammer notices a man come in with an infant. The man, named Decker, cries as he kisses the infant, then walks out in the rain and is shot dead. Hammer shoots the assailant as he searches Decker's body.



Small details like this on common elements are so often overlooked, but they can make the difference between an ordinary design and an interesting one.

We'll return to the interesting uses flexible box layout's two axes can be put to later in this book, when we develop more complex designs using flexbox.


Creating a flex axis

Float behaviour has become ingrained in the way we think about layouts. For example, when the width of two floated elements exceeds the width of their parent container, one element will drop down below the other. Flexible box layout has a different model and flex-items behave differently in relation to a flex-container's width.

I'll illustrate the fundamental differences between the float model and flexbox by putting four `article` element boxes inside a `section`:


```
<section class="hb-shelf">
  <article class="item"> [...] </article>
  <article class="item"> [...] </article>
  <article class="item"> [...] </article>
  <article class="item"> [...] </article>
</section>
```

With nothing more than some basic styling, those articles will stack vertically.




1. Police Detective Cases

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.




2. The Spicy Detective Magazine

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.



3. Off Beat Detective Stories

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.



4. Snappy Mystery Stories

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.

We have little or nothing to do to prepare this list for smaller screens.

Now let's apply `display: flex` to the `section` and watch as the browser creates a horizontal main axis and arranges our articles along it. Without us needing to specify any more flexbox properties, our browser sizes the articles so that, combined, they fit all of the width available in the `section`.



Arranging articles in a row along a `section` element's default main axis.

You might be surprised by what happens when we specify a width for each `article`.

```
.item {  
width : 400px; }
```

In a float-based layout, our browser will display as many articles as there is space to fit in a row before starting the next row.

As we increase the articles' width, fewer of them will be displayed on one row. However, in flexbox layout, `flex-wrap: nowrap` value overrides the `width` property on these articles.

Wrapping flex-items

Unlike float-based layouts, flexbox will automatically expand the width of a flex-container to match the combined width of flex-items within it. That's because the people who designed flexible box layout made a smart choice when they decided that the initial `flex-wrap` value should be `nowrap`. Change that `flex-wrap` value to `wrap` and something entirely different happens, as our browser now respects the width that we gave our flex-item articles and wraps them onto new lines according to the available space in the `section` flex-container:

```
.hb-shelf {  
  flex-wrap : wrap; }
```



1. Police Detective Cases

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.



2. The Spicy Detective Magazine

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.



3. Off Beat Detective Stories

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.



4. Snappy Mystery Stories

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.

Change `flex-wrap` from the initial `nowrap` to `wrap` and the layout resembles floats, but with some significant differences.

If you've been paying attention to our example articles, you might have noticed that they're numbered 1–4. Whether these articles have been set to `nowrap` or `wrap`, they're displayed in the same order as the source, starting in top-left corner. Flexbox gives us even more control over the way these articles wrap, and changing the `flex-wrap` value to `wrap-reverse` starts the articles from the bottom-left corner instead:


```
.hb-shelf {
flex-wrap : wrap-reverse; }
```



3. Off Beat Detective Stories

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.



4. Snappy Mystery Stories

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.



1. Police Detective Cases

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.



2. The Spicy Detective Magazine

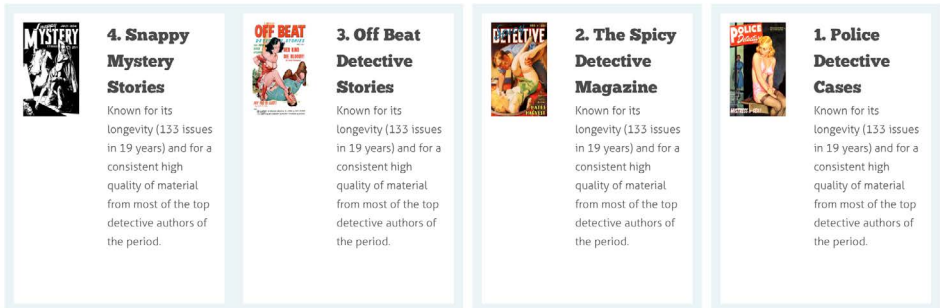
Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.

Wrapping flex-items in reverse alters their starting position from when we change their **flex-direction**.

I can guess what you might be thinking now: “How is this result different from using **row-reverse** value for **flex-direction**?” That’s a good question, so let’s change the **flex-wrap** property back to **wrap** and instead change the **flex-direction** to **row-reverse**:

```
.hb-shelf {
flex-wrap : wrap;
flex-direction : row-reverse; }
```

With these values the flex-items start wrapping in the top-right instead of the bottom-left.



Seeing the difference between **flex-wrap** and **flex-direction** gives us a better understanding of the creative potential of flexible box layouts.

The flex-flow property

The **flex-flow** property is shorthand, combining **flex-direction** and **flex-wrap**. You might remember the initial value for **flex-direction** is **row** and for **flex-wrap** it's **nowrap**, so this is one shorthand declaration you shouldn't ever need to write:

```
.hb-shelf {
  flex-flow : row nowrap; }
```

Both **flex-direction** and **flex-wrap** are powerful layout properties on their own, but combined they can make layouts that would've been extremely difficult or even impossible using older methods. Flexbox is only getting started and when you see the incredible creative potential of sizing and ordering flex-items, I'm sure you'll be as excited as I am about implementing designs using flexible box layouts.

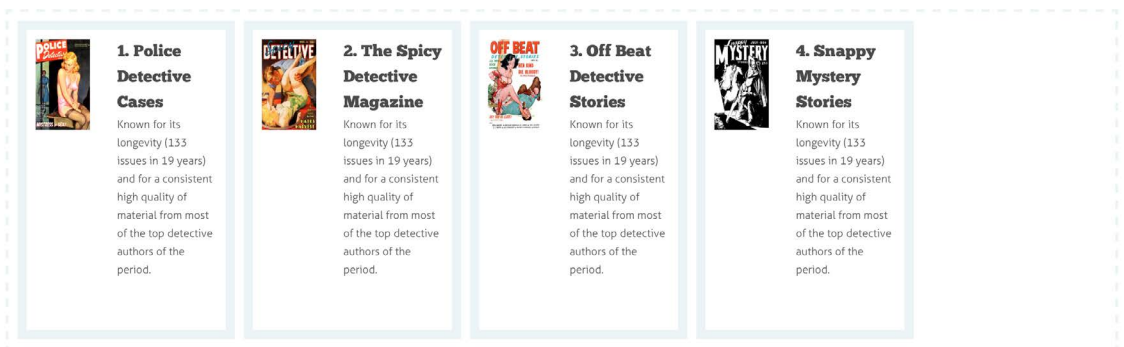
Sizing flex-items

If your method for making layouts using floats was anything like mine, it included more mathematics than we'd like. Even the simplest of layouts required us to calculate the width of our items based on how many of them were to fit inside a parent container. Two elements, fifty percent. Three elements, thirty-three percent and so on. Margins between elements affected our calculations too, making developing layouts more a mathematical challenge than a creative one.

Flexible box layout changes all that, making implementing interesting designs much simpler but, most importantly, more responsive. To illustrate how, let's return to our previous boxes. This time we've applied a width to our flex-item articles:

```
.item {  
width : 240px; }
```

At narrower viewport widths, our articles resize so that they fit equally along the main axis.



Making our `section` a flex-container arranges items along the main axis, but sometimes we leave space available.

When the flex-container width is wider than the combined width of the flex-items, space opens up on the right-hand side of our layout. This may well be acceptable for some designs, but in others it would make better sense for those articles to grow to fill that available space. Flexible box layout includes properties that give us options for how flex-items will grow or shrink according to space available inside their flex-containers.

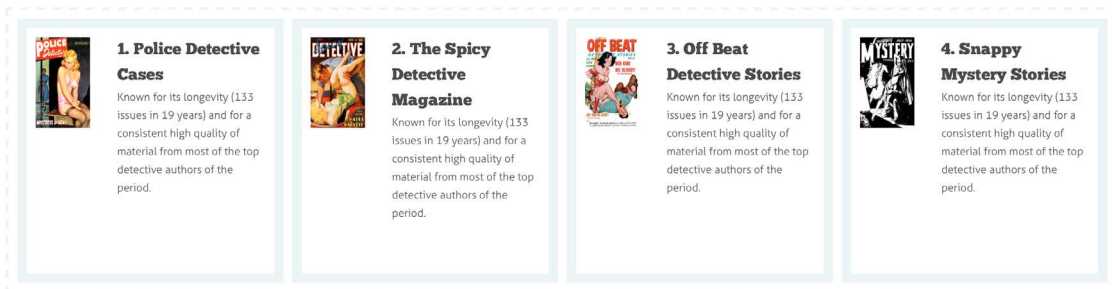
The flex-grow property

`flex-grow` sets the grow factor of a flex-item. This grow factor number determines how much a flex-item will grow, proportionally to other flex-items, when additional space is available inside a flex-container. `flex-grow` is one of the trickiest parts of the flexbox model to understand, so let's use those `article` elements again to illustrate how it works.

In our last example, we set a width of 240px on each of our four articles. Below 960px, `display: flex` on the flex-container ensured that the articles all shrank by an equal amount. Above 960px, space appears to the right of our layout.

Giving all flex-items the same `flex-grow` factor — in this first instance, a grow factor of `1` — instructs those items to expand by an equal amount to take over all available space within a layout.

```
.item {  
  flex-grow : 1; }
```



Giving all flex-items the same **flex-grow** factor makes them expand equally to fill any available space.

Our articles all expanded wider than their set 240px width until all the available space in our layout was filled. But what if we want those flex-items to be different sizes? What if we want the available space to be distributed in different proportions? To illustrate this, in our next example we'll allocate twice the amount of available space to the second flex-item. Here, all our items receive a grow factor of **1**, then the second item receives a grow factor of **2**.

```
.item {
  flex-grow : 1; }

.item:nth-of-type(2) {
  flex-grow : 2; }
```

Using the flex shorthand

There's more to sizing elements with flexible box layout than just the **flex-grow** property. In a moment we'll learn about **flex-shrink** — the opposite of **flex-grow** — and **flex-basis**. For now, we just need to know that in most cases **flex-grow** is used in conjunction with these other properties and written using the shorthand **flex** property. From now on we'll use **flex** instead of the longhand values.



Allocating different proportions of available space to individual elements is a fundamental principle of implementing layouts using flexbox.

This second article has been allocated twice the amount of available space as its siblings and is therefore now twice their width.

The flex property sees some action

To help us understand **flex-grow** and to reinforce that flexible box layout can be used on all kinds of HTML elements, we'll put it to use making some layouts for **figure** images and their associated captions. This is an ideal application for flexbox as with just a few lines we can transform a simple **figure** with a design that — while popular in magazines and newspapers — is rarely seen on the web. Our markup isn't anything out of the ordinary:

```
<figure class="figure--horizontal">
  
  <figcaption>Pulp magazines were inexpensive fiction magazines
published until the '50s.</figcaption>
</figure>
```

Until we intervene, our `figcaption` will be displayed in its default position underneath the image, but we can do better than that. At medium screen sizes we'll first turn our `figure` into a flex-container using `display: flex`. As the initial values are `row` and `nowrap`, we don't need to declare those values.

```
@media (min-width: 48rem) {  
  .figure--horizontal {  
    display : flex; }  
}
```

The figure's image and caption have now become flex-items, arranged evenly along the flex's main axis line. So far, so good. As the image is first in our source order, it will appear on the left — because the initial `flex-direction` is `row` — and the `figcaption` comes second and appears on the right.

I'd like to allocate four times the amount of the figure's width to our image than to the caption, so we'll add a `flex-grow` factor of `4` to the image and another of `1` to the caption:

```
@media (min-width: 48rem) {  
  
  .figure--horizontal img {  
    flex : 4; }  
  
  .figure--horizontal figcaption {  
    flex : 1; }  
}
```



Mike Hammer wakes up being questioned by the police in the same hotel room as the body of an old friend from World War II. His friend, Chester Wheeler, has apparently committed suicide with Hammer's own gun after they had been drinking all night.

Arranging the figure's image and associated caption along the flex's main axis line and sizing them both using proportions of available space is simpler than using older layout methods.

What if we'd like to shake things up a little bit and swap the visual order of the image and its caption? There's no need for us to change the order in our markup, as we simply need to change the initial `flex-direction` from `row` to `row-reverse`:

```
@media (min-width: 48rem) {  
  .figure--horizontal-reverse {  
    flex-direction : row-reverse; }  
}
```

Mike Hammer wakes up being questioned by the police in the same hotel room as the body of an old friend from World War II. His friend, Chester Wheeler, has apparently committed suicide with Hammer's own gun after they had been drinking all night.



Changing a layout's visual order doesn't mean changing its source order any more.

The flex-basis property

The `flex` shorthand property is one of the most powerful in flexbox because it packs in three flexible box layout properties: `flex-grow`, `flex-basis` and `flex-shrink`. While we commonly use the powerful shorthand to distribute space within a flexbox layout, it's important that we understand the principles behind all three properties, starting with `flex-basis`.

So far we've dealt with how flex-items grow and shrink as their flex-containers change size inside a responsive layout. We've allowed our items to be completely fluid and not specified their size in any way. There will be situations where our designs demand that a flex-item should start at a particular size, before we allow it to either grow or shrink. In flexbox, we use the `flex-basis` property to set the starting size for an element before it flexes.

To illustrate this, we'll return to our earlier example, but this time there are only two boxes:

```
<section class="hb-shelf">
  <article class="item"> [...] </article>
  <article class="item"> [...] </article>
</section>
```

Start by turning the section into a flex-container by applying `display: flex`. We won't need to specify `row` as the `flex-direction`, or `nowrap` instead of `wrap` for `flex-wrap`, as these are the initial values:

```
.hb-shelf {
  display : flex; }
```

Whereas before our flex-item articles started flexing from their initial width, this time we'll specify that they are both 420px wide:

```
.item {flex-basis : 420px; }
```

In horizontal layouts, `flex-basis` acts in precisely the same way as `width`. Notice that on screens larger than the flex-item's combined width of 840px there's still space available on the right of our layout.



Setting a `flex-basis` gives us control over the size of our flex-items.

We'll distribute that available space between our two flex-items. Look what happens when we distribute that space to just the first of our items: that first flex-item expands to fill all the available space, while the second remains at the size of its `flex-basis`.

```
.item:first-of-type {  
  flex-grow : 1; }
```



1. Police Detective Cases

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.



2. The Spicy Detective Magazine

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.

Distributing space to just one flex-item increases its size while other items remain at the size we specified using `flex-basis`.

The flex-shrink property

While `flex-grow` specifies how flex-items should increase in size when a flex-container is larger than the combined sizes of items, `flex-shrink` specifies how flex-items should decrease in size when the flex-container's width is less than their combined width. In other words, while `flex-grow` specifies the proportions of space that a flex-item will gain, `flex-shrink` specifies the proportion of space that an item will lose.

When we set the `flex` shorthand property to `1`, we're specifying that all flex-items should grow and shrink in the same proportions. So the shorthand declaration:

Changing the size of images inside a flexbox layout can sometimes change their proportions. Noah Blon has written a thorough explanation of how to use the `flex-shrink` property to avoid this.¹⁴

¹⁴ codepen.io/noahblon/post/practical-guide-to-flexbox-dont-forget-about-flex-shrink

```
.item {  
flex : 1; }
```

Gives the same results as:

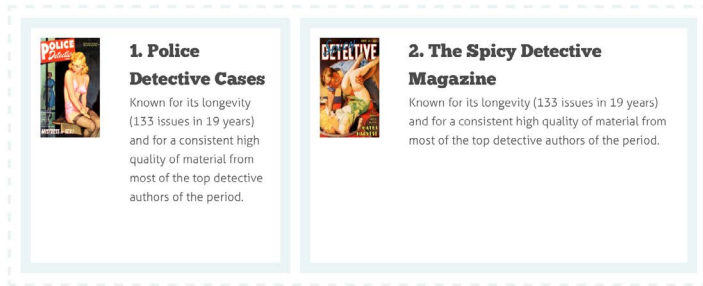
```
.item {  
flex-grow : 1;  
flex-shrink : 1; }
```

That's because in flexbox, when there's not enough space in a flex-container to display the `flex-basis` of all items, the normal behaviour is to divide space evenly between flex-items. When that won't give us the layout we need, we can alter that behaviour and change the way flex-items shrink using `flex-shrink`.

This can be one of the most difficult flexible box layout concepts to understand, so to help explain it we'll add a large number value to the `flex-shrink` property on the first of our flex-items. To begin with we'll go with 8:

```
.item:first-of-type {  
flex-shrink : 8; }
```

Adjust the width of your browser and you should notice that as long as the width of the flex-container is sufficient to display both flex-items at their `flex-basis` or larger, there'll be no change in what you see. The magic happens when the width of the container isn't sufficient. Then, although our second item maintains its `flex-basis`, the first reduces in size due to its `flex-shrink` value.



Specifying the proportions of space that a flex-item will lose helps us to control layouts at smaller screen sizes as well as larger ones.

Now reduce that `flex-shrink` value to just `2` and you should notice the first flex-item increase in size, and we're specifying that it lose proportionally less of its size as screen sizes reduces.

```
.item:first-of-type {
  flex-shrink : 2; }
```

Using `flex-shrink`, we control the ratio by which flex-items shrink. Experiment with different `flex-shrink` values and watch how the proportions of our layout change.

Understanding the flex property shorthand

As I mentioned earlier, we can and should combine `flex-grow`, `flex-shrink` and `flex-basis` into a single `flex` property, and browsers interpret these values in that order:

```
.item {
  flex : 1 1 420px; }
```

If we leave out `flex-shrink`, browsers use the initial value of `1`. When we omit `flex-basis`, browsers default to `0%`.

Ordering flexible boxes

Before we leave flexible box layout for now — don't worry, there's still plenty to learn and we'll cover those topics in the context of other hardboiled CSS properties — it would be wrong not to learn about one of the most powerful features of flexbox. It's something that designers and developers have wanted for many years and that I couldn't have imagined would be possible when I wrote the first edition of this book. It's the ability to visually reorder content without changing its order in the HTML source.

If you're looking for more practical examples of using flexbox, Landon Schropp has written the excellent 'Unraveling Flexbox' ebook¹⁵ and video series.

Now, you might be wondering if we've seen this before with the `flex-direction` property — and we have — but `order` gives us much more precise control and allows us to move a flex-item to any position in a `flex-direction`.

To demonstrate flexbox `order`, we'll use a series of flex-item `article` elements inside a flex-container section. This example should be familiar to you by now:

```
<section class="hb-shelf">
  <article class="item">1 [...] </article>
  <article class="item">2 [...] </article>
  <article class="item">3 [...] </article>
  <article class="item">4 [...] </article>
</section>
```

There are no `id` attributes to identify these articles, but each one has a number, placing it in order inside what's known as an ordinal group. The first article has an ordinal value of `1`, the second `2` and so on.

¹⁵ unravelingflexbox.com

In flexbox, flex-items are displayed in the same order as they appear in the document, but there are plenty of instances where we might want to reorder our flex-items to improve our layout at particular responsive viewport sizes. We'll start by turning our **section** into a flex-container and giving it a **flex-direction** value of **column**.

```
.hb-shelf {
display : flex;
flex-direction : column; }
```



1. Police Detective Cases

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.



2. The Spicy Detective Magazine

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.



3. Off Beat Detective Stories

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.



4. Snappy Mystery Stories

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.

Stacking an ordinal group of flex-items into a vertical column. Each article has an ordinal value that we can change using the flexbox **order** property.

You might be wondering why we need to use `display:flex`, especially as block-level elements naturally stack to form a column, but we'll need that property so we can reorder the flex-items inside it. Now let's go ahead and make the last `article` element display first, using the flexbox `order` property. Notice that unlike other flexbox properties, `order` isn't prefixed with `flex-`.

```
.item:last-of-type {  
  order : -1; }
```



4. Snappy Mystery Stories

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.



1. Police Detective Cases

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.



2. The Spicy Detective Magazine

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.



3. Off Beat Detective Stories

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.

Changing the order a flex-item is displayed visually doesn't change its position in the DOM, so screen readers will continue to use the source order.

The initial value for all flex-items inside an ordinal group is zero (0) so we don't have to set that on every flex-item. Any `order` value we add to specific flex-items starts their ordering after the initial group, so in that example we need to apply an `order` value of minus one (-1) to make it display before the rest of the group.

If we prefer to start all `order` values from 1 and not use negative values except for a particular effect, we can give all flex-items the same initial value:

```
.item {  
  order : 1; }
```

With a small number of flex-items in this group, it's very possible that we might give each one its own `order` value and display them in an order that's very different from how they're ordered in our markup:

```
.item:nth-of-type(1) {  
  order : 3; }  
  
.item:nth-of-type(2) {  
  order : 4; }  
  
.item:nth-of-type(3) {  
  order : 1; }  
  
.item:nth-of-type(4) {  
  order : 2; }
```




3. Off Beat Detective Stories

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.



4. Snappy Mystery Stories

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.



1. Police Detective Cases

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.



2. The Spicy Detective Magazine

Known for its longevity (133 issues in 19 years) and for a consistent high quality of material from most of the top detective authors of the period.

Using `nth-of-type` pseudo-selectors can be an ideal way to target flex-items without adding unnecessary `class` or `id` attribute values.

Let's put these simple box examples behind us and move on to using flexbox order for something I'm particularly excited about: changing the order of page sections across responsive breakpoints so the layout is the most appropriate for different screen sizes.

The order property sees some action

There have been countless times when I've needed to change the display order of a page for a responsive website design. One example that springs to mind immediately is changing the visual position of a website's navigation across viewport sizes. Typically, the order of page sections looks something like this:

```
<header> [...] </header>
<nav> [...] </nav>
<section> [...] </section>
<footer> [...] </footer>
```

I can't think of many reasons why we'd need to change the display order of either our **header** or **footer** in this layout, but I have seen plenty of instances where it would help people using smaller screens by moving a **nav** from near the top of a page to near the bottom. This makes it unnecessary for someone to scroll past what could be a long list of links to get to the content **section**.

The flexbox **order** property is ideal for doing this, so let's start by adding applying **display: flex** to the body of the page, turning it into a flex-container. As we're changing the order vertically, we'll also specify the body's **flex-direction** as column:

```
body {
  display : flex;
  flex-direction : column; }
```

Now, we'll give each of our flex-items a specific **order** value:

Developer Wes Bos has created a free, fun and informative video series about flexbox that I highly recommend. It covers everything from flexbox basics to practical examples like this one.¹⁶

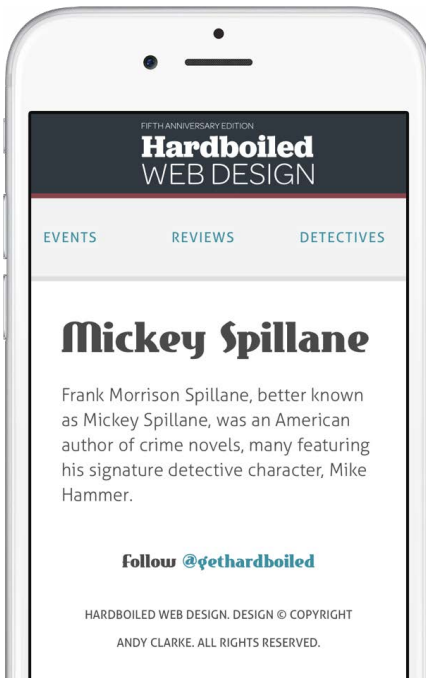
¹⁶ flexbox.io

```
header {  
  order : 1; }
```

```
nav {  
  order : 2; }
```

```
section {  
  order : 3; }
```

```
footer {  
  order : 4; }
```

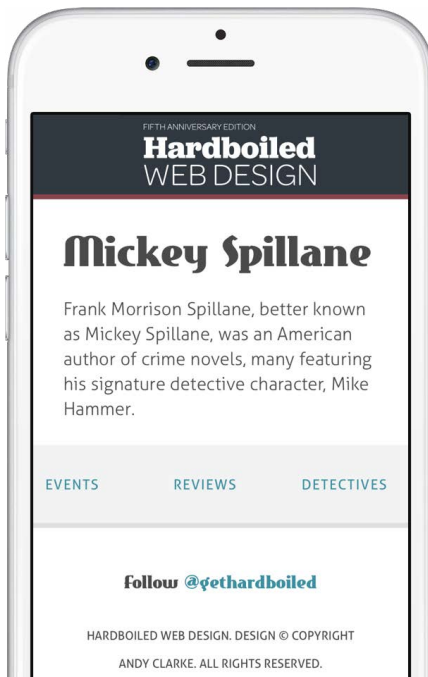


Ordering flex-items to match their order in our markup isn't necessary, so this example is purely to demonstrate how flexbox **order** works.

We can swap the visual position of the `nav` and `section` elements, placing our navigation between the `section` and `footer` which helps the layout work better on smaller screens:

```
nav {  
  order : 3; }
```

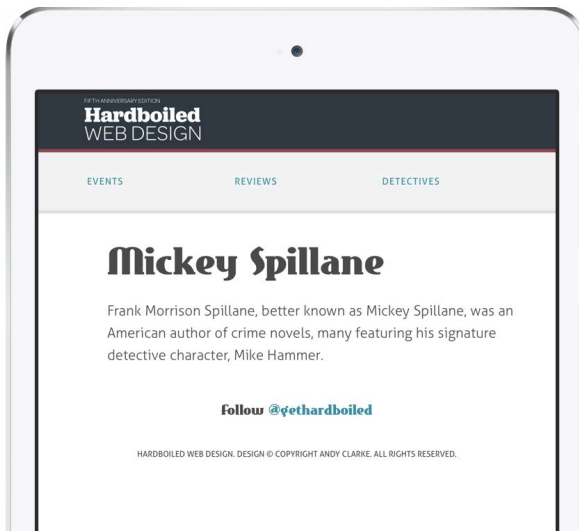
```
section {  
  order : 2; }
```



Swapping the display position of our `nav` prevents people from having to scroll past it to get to the content `section` on smaller screens.

This order works well for people who view our websites on smaller screens, but on larger displays I doubt anyone will expect to find navigation at the bottom of a page. For people using medium and larger screens, we'll reset the order of our elements by using the original `order` values inside a media query:

```
@media (min-width: 48rem) {  
  nav {  
    order : 2; }  
  
  section {  
    order : 3; }  
}
```



Resetting the order of our elements for people using medium and larger screens.

Cross-browser flexible box layout

Flexible box layout had a long and well-documented development process and went through several iterations and revisions to its syntax. If I had written this edition of *Hardboiled Web Design* two years ago, I might have recommended we make use of all available vendor-specific prefixes to ensure that our flexbox layouts work across different browsers.

Fortunately, today it's a very different situation as we have almost uniform support for flexbox across all contemporary mobile and desktop browsers. Unless you need to support versions of Microsoft Internet Explorer older than 11, Safari 8 on Mac OS X and iOS, you'll need no vendor prefixes at all. At Stuff & Nonsense, we include only ~~-webkit-~~ vendor prefixes for Safari and deploy other methods when our clients need to include older browsers. Here's a typical flexbox declaration, with vendor-specific prefixes first, followed by the standard syntax:

```
.hb-shelf {  
display : -webkit-flex;  
-webkit-flex-direction : column;  
-webkit-flex-wrap : wrap;  
display : flex;  
flex-direction : column;  
flex-wrap : wrap;  
flex : 1; }
```

Autoprefixer

Like many people, we rarely type these long lists of prefixed properties by hand. Instead, we use Autoprefixer,¹⁷ a plugin that parses our style sheets and adds the appropriate vendor prefixes according to data from Can I Use.¹⁸ Autoprefixer runs on the command line, but if you're Terminal-phobic like me, you'll find it built in to tools like Codekit.¹⁹

Modernizr

Modernizr²⁰ is the feature detection library that played such a big part in the first edition of *Hardboiled Web Design*. It detects support in browsers for all variations of flexbox syntax from its different development versions and outputs class attribute values based on a browser's level of support. These values help us quarantine properties designed for legacy browsers from the contemporary browsers that don't require them:

```
.hb-shelf {  
display : flex; }  
  
.no-flexbox .hb-shelf {  
display : table; }
```

Although there is now less need to use Modernizr to detect support for what were cutting-edge CSS properties five years ago, it remains an incredibly useful tool for developing alternatives to flexbox for the legacy browsers that require them.

Modernizr (modernizr.com) and flexbox go together like detectives and redheads, and Zoe Mickley Gillenwater explains how in her article 'Using Modernizr with Flexbox'²⁰

¹⁷ github.com/postcss/autoprefixer

¹⁸ caniuse.com/#search=flexbox

¹⁹ incident57.com/codekit

²⁰ zomigi.com/blog/using-modernizr-with-flexbox

Breaking it up

Throughout my professional career working with CSS, like so many other designers and developers I've got to know, I've been frustrated by the fragility and limitations of developing page layouts using properties that were never designed for the applications that we put them to today. For years we've needed a better way to arrange elements within a layout and we're fortunate that now we have flexible box layout and widespread support for it among contemporary browsers. Flexbox is packed with properties that make responsive designs more flexible, but we mustn't stop there. We should use flexbox to help us push our creativity as well as our technical abilities.

Responsive typography

No. 12

INFORMATION ARCHITECTS' OLIVER REICHENSTEIN wrote in 2006 that “Web Design is 95% Typography”.¹ I’m not 110% sure if that’s true, but I do know that much of the web content I consume every day consists of the written word. When we first published this book, many people still relied on a limited set of commonly installed fonts — those usual suspects like Arial, Georgia, Times, Verdana and others — and so typography on the web was, not to put too fine a point on it, often incredibly dull. Skip ahead five years and we’re now incredibly fortunate that our typographic horizons have been expanded beyond that rudimentary selection of fonts to include vast libraries of typefaces that are now available as what we call web fonts.

A short history of web fonts

In the late 1990s both Netscape and Microsoft released browsers that allowed us to embed fonts into a web page. Given the competition between them, they didn’t make it easy. I can remember trying — but mostly failing — to use their incompatible TrueDoc and Embedded OpenType (EOT) formats. That makes me feel old as I bet that many people reading this never used a Netscape browser.²

¹ ia.net/know-how/the-web-is-all-about-typography-period

² Netscape launched its first web browser in 1994 and Netscape Navigator was once the dominant browser with over 90% usage share. After competition from Microsoft Internet Explorer, Netscape’s market share dropped to less than 1% by the end of 2006 and its final browser, Netscape Navigator 9 (which was by then based on Firefox), was released in 2008.

Netscape lost the browser war and although Internet Explorer continued to support font embedding, its EOT format was never implemented by other browser makers. For a decade, web fonts stalled and it wasn't until ten years later — ten years, dammit! — that Apple's Safari 3.1 became the first browser to support embedding fonts in TrueType and OpenType formats (but not EOT). Mozilla followed and so did Opera. When Google launched Chrome and later Android, and Apple launched Safari on its iOS platform, they both included support for web fonts.

Why web fonts matter

Web fonts offer a way to use more varied fonts. With all contemporary browsers now supporting web fonts, we can largely assume that people will experience our type designs as we intend them to.

Web fonts are simple to implement and when we use them our text stays accessible, selectable and friendly to search engines. To embed a web font we need three things:

1. A font file in a format that browsers will understand. We can use any copyrighted or licence-free font.
2. A `@font-face` declaration at the start of our style sheet. This will define the `font-family` name, where the font file is hosted, and its format (WOFF2 in the following example). A simple declaration looks like this:

```
@font-face {  
  font-family : 'Aller Light';  
  src : url('fonts/aller_std_lt.woff2') format('woff2');}
```

3. A `font-family` property, which applies the embedded font to an element, id, class, child, sibling, attribute, pseudo- or any other CSS selector. Let's make top-level headings look hardboiled with a typeface called Eastmarket, from Font Squirrel:³

```
h1 {  
  font-family : Eastmarket; }
```

We'll work through the details of embedded web font syntax in just a moment.

Web font formats

There are six font formats that have been used widely on the web — EOT, OpenType, SVG, TrueType, WOFF and WOFF2 — and different browsers typically support some, but not all, of these formats. The font formats we choose to use depends entirely on the browsers we need to support. The newer our target browsers, the fewer formats we need. Conversely, when we have older browsers in our supported browser list, we'll need to serve more varied formats.

Embedded OpenType (EOT)

Designed by Microsoft specifically for embedding web fonts, EOT contains a wrapper for TrueType that makes it more difficult to download, extract and reuse an embedded font. This makes it easier to uphold font licences, or at least that's the theory. Although Microsoft submitted EOT to the W3C in 2007, it has never been part of any standard.

³ You can use Font Squirrel's online web font generator¹ to create EOT versions of your fonts: fontquirrel.com/tools/webfont-generator

OpenType (OTF)

OpenType is an extension of TrueType that offers better control over typography because it provides up to 65,000 different glyphs and has better rendering of complex script typefaces.

SVG

SVG isn't a font format at all — it's a technology for making scalable vector graphics — but we can include font information inside an SVG document and link to that in the same way we would any other type of font. We can convert fonts to SVG using Font Squirrel's web font generator. Web font services such as Typekit⁴ also provide an option for serving fonts as SVG.

TrueType (TTF)

Apple introduced TrueType in the late 1980s as an alternative to Adobe's PostScript Type 1 format. With TrueType, all aspects of a typeface — including its kerning and hinting information — are contained within a single file. This can make some TrueType font files large and impractical to use as web fonts.

Web Open Font Format (WOFF)

WOFF isn't strictly a font format. It's a compressed wrapper for TrueType and OpenType fonts — a transfer format, similar to a compressed ZIP file. This makes it small in size and therefore eminently suitable for using on the web. Because WOFF includes ownership information, it's more attractive to font foundries that are concerned about protecting their intellectual properties.

⁴ typekit.com

WOFF2

WOFF2 is the latest generation of the WOFF format and is notable for its higher compression rates and, therefore, its better performance, especially on mobile devices. WOFF2 is clearly a future web font standard⁵ and before long could easily be the only web font format we need.

Including @font-face in a style sheet

To link web fonts to our style sheets, the first step is to specify the name of the font file and where it's hosted. We'll link to the Aller Light web font we downloaded from Font Squirrel. Aller Light is the typeface we're using for body copy in our 'Get Hardboiled' example files.

```
@font-face {  
  font-family : 'Aller Light';  
  src : url('fonts/aller_std_lt.woff2') format('woff2'); }
```

In that example we're only linking to the WOFF2 format of our font which, in an ideal world, should be all we need. Sadly the world isn't ideal and even today some of the most modern browsers — including Safari on iOS and Mac OS X don't support WOFF2. To help them we'll also include a font in the previous version of WOFF which currently has wide support in contemporary browsers except Opera Mini. We'll separate the two formats with a comma in our new declaration:

```
@font-face {  
  font-family : 'Aller Light';  
  src : url('fonts/aller_std_lt.woff2') format('woff2'),  
        url('fonts/aller_std_lt.woff') format('woff'); }
```

⁵ The State of Web Type is a comprehensive reference of support for type and typographic features on the web. Choose a font property and the site will tell you which browsers and versions support that feature: stateofwebtype.com

For wider support among older browsers, Android and Safari on iOS, we could include a TrueType format font in our declaration:

```
@font-face {  
  font-family : 'Aller Light';  
  src : url('fonts/aller_std_lt.woff2') format('woff2'),  
        url('fonts/aller_std_lt.woff') format('woff'),  
        url('fonts/aller_std_lt.ttf') format('truetype'); }
```

Marcin Wichary's 'Using System UI Fonts In Web Design: A Quick Practical Guide'⁶ is an excellent primer for using an operating systems' fonts.

And for the most complete support that includes even legacy versions of Microsoft Internet Explorer, we could include the EOT format, too:

```
@font-face {  
  font-family : 'Aller Light';  
  src: url('aller_std_lt.eot');  
  src: url('aller_std_lt.eot?#iefix') format('embedded-opentype'),  
        url('aller_std_lt.woff2') format('woff2'),  
        url('aller_std_lt.woff') format('woff'),  
        url('aller_std_lt.ttf') format('truetype'); }
```

Place this new **@font-face** declaration at the top of a style sheet (or the top of your typography section) so that it's available to any declarations below. The **font-family** name we choose needn't match a font's file name: we can name it anything that makes referencing it in our style sheets easier. This web font font should appear first in our font stack, followed by a backup made from commonly installed system fonts:

```
body {font-family : 'Aller Light', Helvetica, Arial, sans-serif; }
```

Specifying commonly installed backup fonts in our stack is essential as we can't always rely on our own web fonts being loaded into a visitor's browser. Likewise, we can't assume 100% uptime from the web font delivery services we use.

In August 2015, a problem at Amazon S3 (where Adobe Typekit stores its font and kit configuration files) left thousands of Typekit's customer websites without their carefully chosen web fonts and exposed the fact that many hadn't specified a backup font stack.⁷

⁶ smashingmagazine.com/2015/11/using-system-ui-fonts-practical-guide/

⁷ smashed.by/outage

Web fonts and performance

We may be tempted to include several web fonts when implementing our designs, but we must remember that every font represents an extra file for a user to download. Go heavy-handed on web font use and our pages will soon become heavyweights, so ask yourself if any font you want to include is really necessary.

Font files can be large and as well as their weight, web fonts present another challenge to performance-minded designers and developers. Many browsers hide all the text content on a page until they have downloaded a web font. This means our visitors can be left staring at a blank page of hidden content for up to three seconds. If our font hasn't loaded in this time, a browser stops downloading it and displays a system font instead.

To help work around the problem of font loading, Scott Jehl of the Filament Group has written about their approach to loading web fonts and avoiding the FOIT (flash of invisible text).⁹

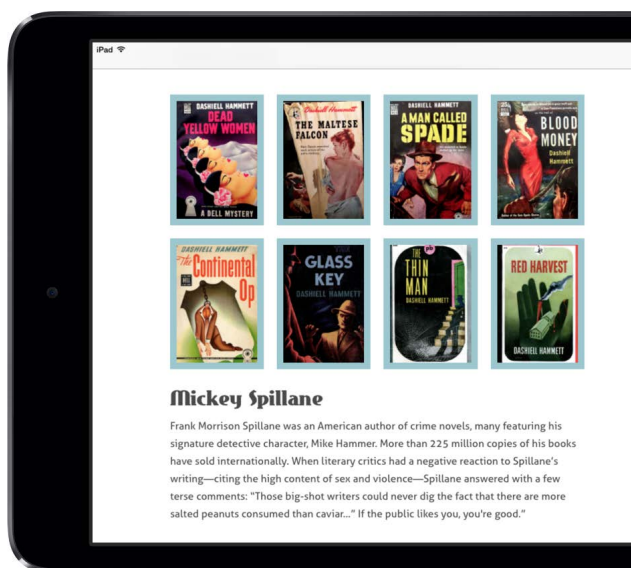
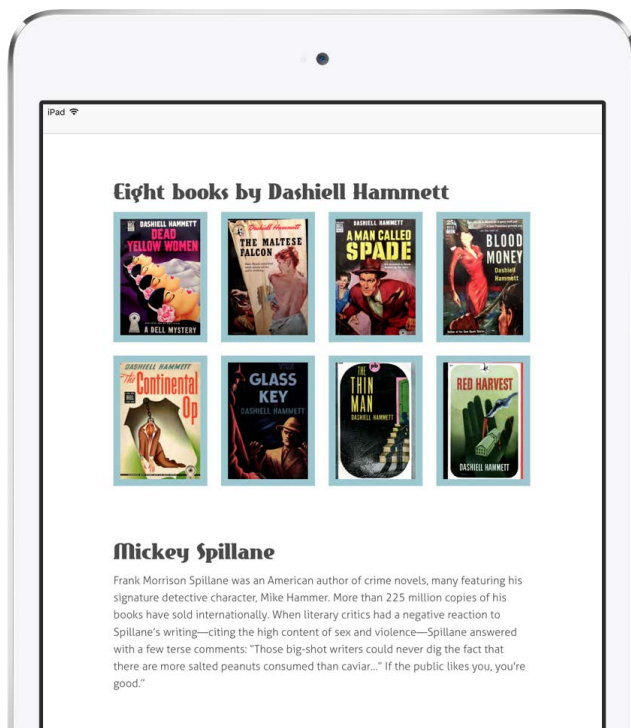
Bram Stein's presentation⁸ on web font performance issues and how they block page rendering explores workarounds while still delivering a good user experience.

Designing type for high-resolution displays

There can be no doubt that Apple's introduction of higher resolution Retina displays (first on the iPhone 4, then iPad, MacBook Pro and 5K iMac), followed by similar displays from other smartphone and PC manufacturers, made web designers' and developers' jobs more complicated. Not only do we often need to serve different size (2x) images to browsers on high-resolution devices, we also need to pay attention to how our fonts render differently on both low- and high-resolution screens.

⁸ speakerdeck.com/bramstein/web-fonts-performance

⁹ filamentgroup.com/lab/font-events.html



We can't assume that everyone reading our text will have a high-resolution display, so it's important to test our typography on different resolution screens. Top: iPad mini with Retina display. Bottom: low-resolution iPad mini.

Lightweight typefaces with thin arcs, ascenders and descenders can look stunning on high-resolution displays, but can render poorly and provide an even poorer reading experience when viewed on low resolution screens.

When we choose to use thin typefaces, it's always good practice to check font rendering across resolutions and, where necessary, serve different weights to different screen types: a heavier weight for lower resolutions and a lighter weight for higher ones. We can do this easily using a `min-resolution` media query.

First, we'll specify the web font we're serving to low-resolution screens. In our example, this is the regular weight of our chosen Aller typeface:

```
body {  
  font-family : 'Aller Regular', Helvetica, Arial, sans-serif; }
```

Next, we'll specify a minimum resolution threshold for the lighter, thinner version of Aller, in our case 192dpi. Devices with screens above that resolution will display the thinner Aller Light typeface:

```
@media  
(min-resolution: 192dpi) {  
  font-family : 'Aller Light', Helvetica, Arial, sans-serif; }
```

Currently, Safari on both iOS and Mac OS X supports only the non-standard, vendor-prefixed `max-device-pixel-ratio` alternative property, so we'll need to include that in our style declaration to serve those light typefaces to iOS devices and Macs:

```
@media  
(-webkit-min-device-pixel-ratio: 2),  
(min-resolution: 192dpi) {  
  font-family : 'Aller Light', Helvetica, Arial, sans-serif; }
```

In our example, we've specified the resolution in dots per inch. A screen typically has a lower number dpi — as low as 72dpi for a conventional screen — whereas print is typically higher. Dots per inch isn't the only resolution unit. We can also use:

- **dpcm**: number of dots per centimetre.
- **dppx**: number of dots per pixel unit.

Internet Explorer 9–11 support only dpi, as does Opera Mini at the time of writing.

Sourcing web fonts

There are now plenty of options of where to find, license and serve web fonts to include in our designs. In the five years between editions of this book, many type foundries and their resellers have licensed versions of their fonts for use on the web. For example, Hoefler & Co.¹⁰ has made their popular Gotham, Knockout, Whitney and other fonts available to purchase for desktop and license for web use.

Adobe Typekit¹¹ and Fontdeck¹² — a partnership between UK design studio Clearleft and OmniTI — are both popular services that sell and serve libraries of thousands of typefaces from many designers and type foundries.

Typekit and Fontdeck have slightly different pricing models. While Fontdeck charges an annual fee for every font you use, Typekit offers a limited set of fonts that are free to use, then either a paid-for plan or one that's included in their Creative Cloud subscription. Both paid-for options include using all the fonts in their library.

¹⁰ typography.com

¹¹ typekit.com

¹² fontdeck.com

Websites such as Font Squirrel have become popular sources of licence-free web fonts. Font Squirrel also offers an online generator for converting desktop fonts to web fonts. This generator is a useful service, but always be sure to check the end user licence agreement (EULA) of the fonts you're converting as it's neither fair nor legal to use fonts that aren't licensed for use on the web.

Finally, Google provides a smaller but nevertheless useful selection of fonts.

Web fonts' 404 adventure

I hope that you're not feeling lost among all this talk of web fonts, because now we'll put what you learned to work on a 404 page for 'Get Hardboiled'. This page uses two web fonts, an image and a splattering of CSS. Don't worry that one or two of the CSS properties aren't supported by all browsers. We'll make sure that everyone who finds themselves on this page will get an appropriate experience.

This design doesn't need much markup — two divisions, a heading and a couple of paragraphs:

```
<div class="splatter">
  <div class="splatter__content">
    <h1 class="splatter__heading">404</h1>
    <p class="splatter__lead">You dumb mug!</p>
    <p>You can look all you want, but what you're looking for
just ain't here. Did you click a link that I bumped off? Maybe
that page is hot? Either way, don't be a bunny.</p>
  </div>
</div>
```



404

YOU DUMB MUG!

YOU CAN LOOK ALL YOU WANT, BUT WHAT
YOU'RE LOOKING FOR JUST AIN'T HERE. DID
YOU CLICK A LINK THAT I BUMPED OFF?
MAYBE THAT PAGE IS HOT? EITHER WAY,
DON'T BE A BUNNY.

With its mixture of web fonts,
images and utter rudeness,
this is a 404 page that people
won't forget in a hurry.

Our first job is to add a bloody background image to the outer `splatter` division. We'll make sure the whole splash will always be visible by setting a minimum height on that division:

```
.splatter {  
  min-height : 900px;  
  background-image : url(blood.png);  
  background-repeat : no-repeat;  
  background-position : 50% 0; }
```

Centre the content division horizontally and make sure that it's only just wide enough for the large heading to fit inside. It should also fit neatly into the width of a small screen.

```
.splatter__content {  
  width : 280px;  
  margin : 0 auto; }
```

Now link to two bloody typefaces — ChunkFive and Boycott. We'll use just three formats: TrueType, WOFF and WOFF2:

```
@font-face {  
  font-family : 'ChunkFive';  
  src : url('fonts/chunkfive.woff2') format('woff2'),  
  url('fonts/chunkfive.woff') format('woff'),  
  url('fonts/chunkfive.ttf') format('truetype'); }
```

```
@font-face {  
  font-family : 'Boycott';  
  src : url('fonts/boycott.woff2') format('woff2'),  
  url('fonts/boycott.woff') format('woff'),  
  url('fonts/boycott.ttf') format('truetype'); }
```

With these links in place, let's style the main heading with ChunkFive in white:

```
.splatter__heading {  
  font-family : ChunkFive;  
  font-size : 16rem;  
  text-align : center;  
  color : rgb(255,255,255); }
```

Next, we'll style the two paragraphs using Boycott in a very light grey to emphasise the heading above it:¹³

```
p {  
  font-family : Boycott;  
  font-size : 1.6rem;  
  text-align : center;  
  color : rgb(224,224,224); }
```

To complete the design for all web font-capable browsers, we'll style the "You Dumb Mug!" paragraph.

```
.splatter__lead {  
  font-family : ChunkFive;  
  font-size : 3rem;  
  text-transform : uppercase; }
```

Experimental WebKit properties

In the past, browsers implemented experimental properties using vendor-specific prefixes, and even though these might not be standards, they can be extremely useful to add visual flourishes to a design. We'll use a WebKit-specific property, `-webkit-text-stroke`, to add a subtle stroke to our text that will be rendered by Chrome, Opera and Safari.

```
.splatter__heading,  
.splatter__lead {  
  -webkit-text-fill-color : transparent;  
  -webkit-text-stroke : 4px rgb(255,255,255); }
```

Want to preview how web fonts look on any website, even one that's not yours? WebFonter from FontShop does just that with its bookmarklet, Chrome extension and online tool¹³

¹³ webfonter.fontshop.com



404

YOU DUMB MUG!

YOU CAN LOOK ALL YOU WANT, BUT WHAT
YOU'RE LOOKING FOR JUST AIN'T HERE. DID
YOU CLICK A LINK THAT I BUMPED OFF?
MAYBE THAT PAGE IS HOT? EITHER WAY,
DON'T BE A BUNNY.

We should always be cautious when using experimental CSS properties that aren't part of an ongoing standards process.

Text shadows

Although using shadows has fallen out of fashion as designers have followed the trend towards flatter designs, text shadows are still an effective tool to add depth or to simply enhance the legibility of our text against more complex backgrounds. Here's a simple `text-shadow` declaration applied to a heading.

```
h1 {  
  text-shadow : 2px 2px 0 rgb(204,211,213); }
```

Let's break down those `text-shadow` values.

The first `2px` value is the shadow's horizontal offset, and the second its vertical offset. In our example, those two values are the same, but they can be different depending on the lighting effect that we're creating. The greater the offset, the further away a shadow will appear from the text.

Our third value (`0`) is the shadow's blur radius. In this example, the radius is small, resulting in a very hard shadow. The greater the blur radius, the softer our shadow will look, as if we're moving a light source closer to our text.

Finally, we'll declare the colour of the shadow. We can use either solid or semi-transparent colours. If you're not familiar with `RGBA` colour, you'll learn about how to use it in the next chapter.

My Gun Is Quick

In this example, we'll add just one primary shadow to our text.

Next, we'll make a softer shadow by increasing the vertical offset to three pixels and the blur radius to six pixels.

```
h1 {  
  text-shadow : 2px 3px 6px rgb(204,211,213); }
```

My Gun Is Quick

This primary shadow is softer as we've increased its blur radius from zero to ten pixels.

Text shadows can accept negative as well as positive values, so in this next example we'll change the vertical offset to minus five pixels to move the light source and cast a shadow above the text.

```
h1 {  
  text-shadow : 2px -3px 6px rgb(204,211,213); }
```

My Gun Is Quick

By changing the horizontal and vertical offsets we can cast a shadow on all sides of our text.

Working with multiple text shadows

If our designs demand a more natural-looking result, we can layer multiple shadows, separating their sets of values using a comma:

```
h1 {  
  text-shadow :  
    2px 2px 0 rgba(125,130,131,.75),  
    2px 5px 10px rgba(125,130,131,.65); }
```

We can even create three-dimensional text objects by using three shadows. Here, we'll cast one white shadow above the text and two darker shadows beneath.

```
h1 {  
  text-shadow :  
    2px 2px 0 #0f2429,  
    2px 5px 10px rgba(15,36,41,0.5) ,  
    2px -2px 5px rgb(56,143,162); }
```

My Gun Is Quick

`text-shadow` can be used to create many different effects and, when combined with web fonts, they reduce our need to use images of text enormously.

Breaking it up

After a decade of promise and frustration, web fonts have definitely arrived. Mature delivery services such as Fontdeck and Adobe Typekit, along with libraries from Font Squirrel, Google and dozens more font foundries and designers, mean that we don't have to design with a limited set of typefaces. We can use (almost) any font we choose. Web fonts are easy to apply and style in contemporary browsers, and although there are (and will likely remain) differences in the ways that browsers handle type, there are very few reasons not to be using web fonts in our websites and applications.

RGBa and opacity

No. 13

THERE'S MORE THAN ONE WAY TO DEFINE A COLOUR in CSS — colour names, hexadecimal values, RGB/RGBa and HSL/HSLa — but no matter which you choose, the colour displayed by a smartphone, tablet, PC, Mac or TV screen is made from a combination of red, green and blue transmitted light (RGB), these days usually in 24-bit.

In 24-bit RGB, zero indicates no light and 255 indicates the maximum. So when red, green and blue channels are all zero, the result is black. When they're all 255, the result is white — with a wide gamut of over 16 million colours in between.

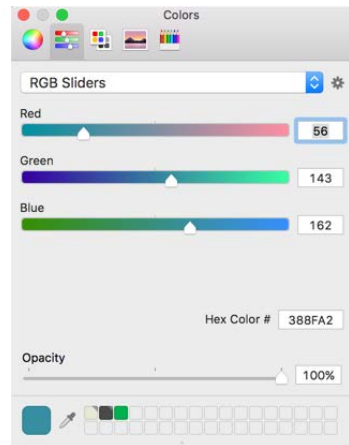
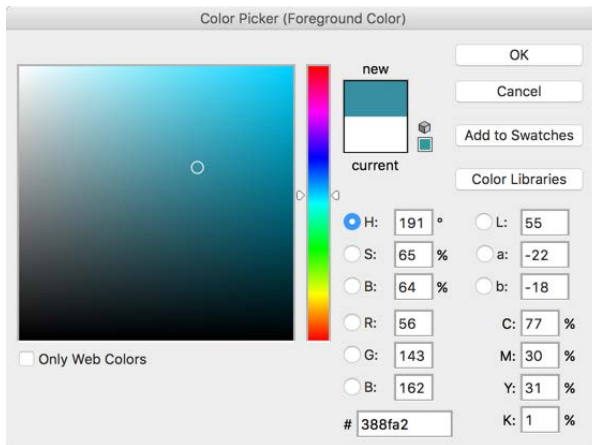
Switching to RGB

Open either Photoshop or Sketch and one of the options in the colour picker is the familiar hexadecimal value, where white is defined as `#ffffff` and black as `#000000`.

```
a {  
  color: #388fa2; }
```

In CSS, we can describe that same blue using RGB, first declaring the colour space and then, in parentheses, the quantities of red, green and blue in values between 0 and 255:

```
a {  
  color : rgb(56, 143, 162); }
```

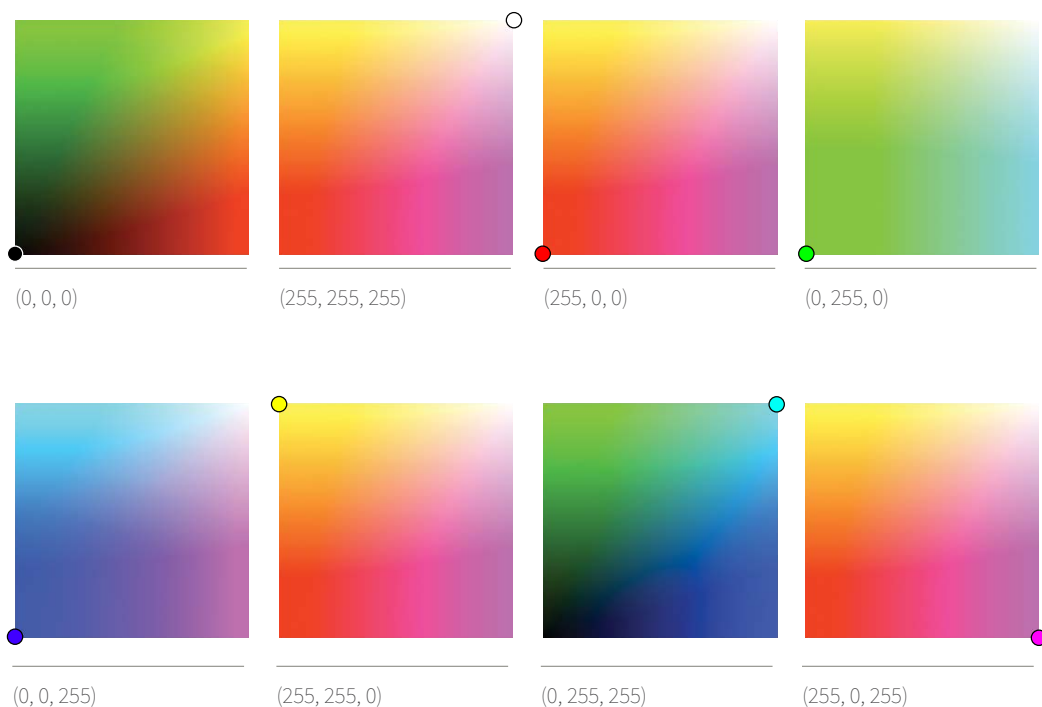


The blue colour I've chosen for the links on 'Get Hardboiled' is represented in hexadecimal as #388fa2.

You might ask why you should choose RGB over hexadecimal. There's no technical reason to use RGB colour over hexadecimal — after all, every colour we see on screen is RGB — but when we're choosing colours for our style sheets, hex values can be tougher to visualise. Pub quiz question: what colour is #003399? Stumped?*

When we understand that for each channel, 0 is no colour and 255 is the maximum, RGB becomes a piece of cake to visualise.

* #003399 is a deep blue.



Layering colour with RGBa

At art school, subtlety wasn't my style — but then, you've probably guessed that by now. My friend Ben, on the other hand, made exquisite paintings because he laid on hundreds of layers of paint. In CSS, RGBa values help us to layer colour and add depth in a similar way.

RGBa is short for red, green, blue, plus a fourth channel — an alpha channel — which defines the transparency of the resulting colour. This alpha value can range between zero (fully transparent) and one (fully opaque). If you use either Photoshop or Sketch you'll be used to working with alpha-transparency.

In the ‘Get Hardboiled’ site, the background colour of our overlaid panels has ninety-five percent transparency. We’ll add a fourth value, an alpha channel value of `0.95`, to our CSS declaration to turn RGB into RGBA.

```
.item__description {
background-color :
rgba(223,225,226,0.95); }
```

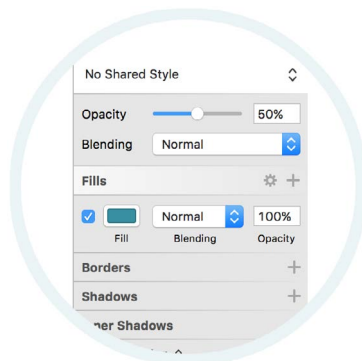
Our ability to use RGBA to subtly adjust colour transparency levels in this way opens up a world of elegant design possibilities.

RGBA vs. opacity

There’s another CSS property we can use to make elements appear semi-opaque — it’s the `opacity` property.

In CSS, both RGBA and opacity vary the alpha channel, but there are subtle differences between them. While RGBA changes the transparency of just one colour on one element, opacity affects an element and all of its children. To demonstrate this, we’ll use those overlaid panels from ‘Get Hardboiled’. We’ll replace the RGBA background colour with a ninety-five percent opacity level:

```
.item__description {
opacity : .95; }
```



Making a layer semi-transparent in Sketch.



Opacity works in a similar way to how we adjust the transparency of a layer in Photoshop or Sketch.

ADD TO CART

With **opacity**, all children are now semi-opaque.

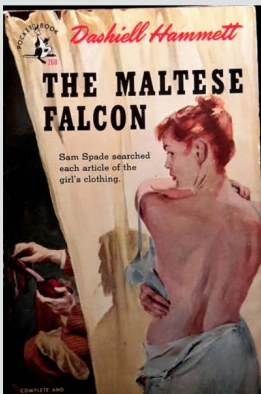
ADD TO CART

Using RGBa, only the background colour is affected.

Opacity gets us out of a tight spot

Let's head back to the 'Get Hardboiled' website. We'll make a grid of eight images hide a secret.

Eight books by Dashiell Hammett



The Maltese Falcon

"The Maltese Falcon," first published as a serial in the pulp magazine Black Mask, is the only full-length novel in which Spade appears. The novel's atmosphere is dense as a San Francisco fog, and its descriptions of locations are so accurate that many can be pinpointed on a map

ADD TO CART

For this interface, we'll arrange the images into a grid and then hide their associated descriptions using absolute positioning and opacity.

You don't have to go too far before you stumble across someone telling you not to use IDs in your HTML. Ignore them. They're crazy people. While using IDs does give an element higher CSS specificity, using them to address elements individually, as in a fragment identifier, is perfectly acceptable.

To build this interface, we need only a tiny amount of hardboiled HTML. We'll start with a container that will hold all of our items. All the example modules in this book, including this one, have an **hb-** prefix so that you can spot them easily if you use them in your projects.

```
<div class="hb-target"> [...] </div>
```

Our items have one division each and we'll add a class attribute value of **item**. To make our CSS-only interactions work, we'll also give an **item** a unique **id** so we can address each of them individually:

```
<div class="item" id="hb-target-01"> [...] </div>
<div class="item" id="hb-target-02"> [...] </div>
<div class="item" id="hb-target-03"> [...] </div>
<div class="item" id="hb-target-04"> [...] </div>
<div class="item" id="hb-target-05"> [...] </div>
<div class="item" id="hb-target-06"> [...] </div>
<div class="item" id="hb-target-07"> [...] </div>
<div class="item" id="hb-target-08"> [...] </div>
```

Now add two more divisions inside each item. One will contain an image, the second a description or some other information about that item.

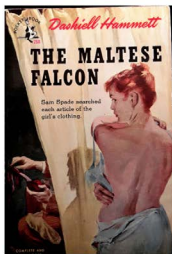

```

<div class="item" id="hb-target-01">
  <div class="item__img">
    
  </div>
  <div class="item__description">
    <h3 class="item__header">The Scarlet Menace</h3>
    <ul class="list--plain">
      <li>Vol. 1 Number 3</li>
      <li>Issue #3</li>
      <li>May '33</li>
    </ul>
    <a href="cart.html" class="btn">Add to cart</a>
  </div>
</div>

```

Make a mental note of this markup pattern as we'll be reusing it several times.

Our designs need to look good and work well across every size and type of screen, from the largest to the smallest. As the smallest screens are often on mobile devices, to help our websites and applications load quickly on them, we should start by styling them using the minimum amount of CSS.



[The Maltese Falcon](#)

"The Maltese Falcon," first published as a serial in the pulp magazine Black Mask, is the only full-length novel in which Spade appears. The novel's atmosphere is dense as a San Francisco fog, and its descriptions of locations are so accurate that many can be pinpointed on a map.

[Add to cart](#)

It's a good idea to check your HTML in a browser before you start styling it. Ask yourself what's the minimum amount of styling you can add.

For people who use devices with smaller screens, our aim should be to reduce complexity, so we'll develop a simple and stylish list of items. We'll use flexbox to arrange both our image and description divisions along a horizontal axis:

```
.item {  
  display : flex; }
```

Now let's give our items a little style with some margin to separate them, and padding and borders to frame their content:

```
.item {  
  margin-bottom : 1.35rem;  
  padding: 10px;  
  border: 10px solid rgb(235,244,246); }
```

Define a **flex-basis** for our image divisions that's appropriate for a smaller screen, then a little margin on the left to help separate them from the description. We should add a border to our images too, to reflect the style of our items:

```
.item__img {  
  margin-right : 20px;  
  flex: 0 0 133px; }  
  
.item__img img {  
  border: 10px solid rgb(235,244,246); }
```

I love how easy flexbox makes laying out modules like these. With just a few lines of CSS we've transformed our simple HTML markup into a good-looking list of pulp detective magazines. But although that list would look just fine even on larger screens, we can do better than that. In the next section we'll turn that list into an interactive module that uses changes in opacity to hide, then show our descriptions and bring our design to life.

Eight books by Dashiell Hammett



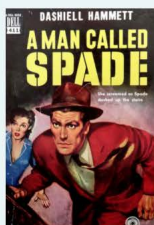
Dead Yellow Women

The Op is hired by a wealthy Chinese-American woman to investigate a robbery-murder-kidnapping that occurred at her San Mateo County mansion.

[ADD TO CART](#)

The Maltese Falcon

"The Maltese Falcon," first published as a serial in the pulp magazine Black Mask, is the only full-length novel in which Spade appears. The novel's atmosphere is dense as a San Francisco fog, and its descriptions of locations are so accurate that many can be pinpointed on a map

[ADD TO CART](#)

A Man Called Spade

Sam Spade is a fictional private detective and the protagonist of Dashiell Hammett's 1930 novel, "The Maltese Falcon." Spade also appeared in three lesser-known short stories by Hammett.

[ADD TO CART](#)

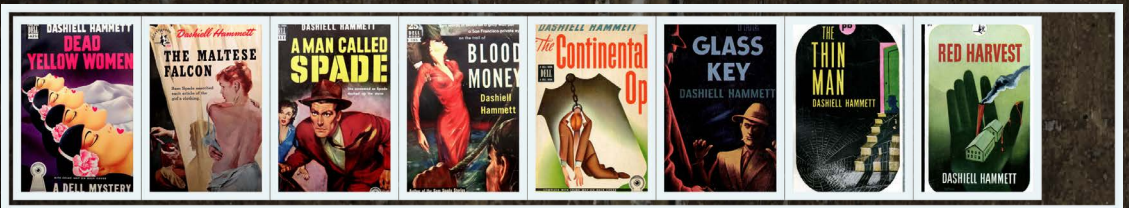
Our design for smaller screens is a simple but stylish list of items.

Flexbox justify-content

In all of our flexbox examples so far, we've arranged flex-items along the along the main axis line of a flex-container. In a similar way to justifying blocks of text to either the left, centre or right of a column, we can also justify items in a flex-container in several different ways using the `justify-content` property:

```
.item {  
  justify-content : flex-start; }
```

When our design demands that we justify the content in different ways, `flex-end` justifies content to the opposite point from where the flex starts. When `flex-direction` is set to `row`, this will be on the right. When it's set to `column`, this will be at the bottom — and I'll bet you can already guess what `center` will do.

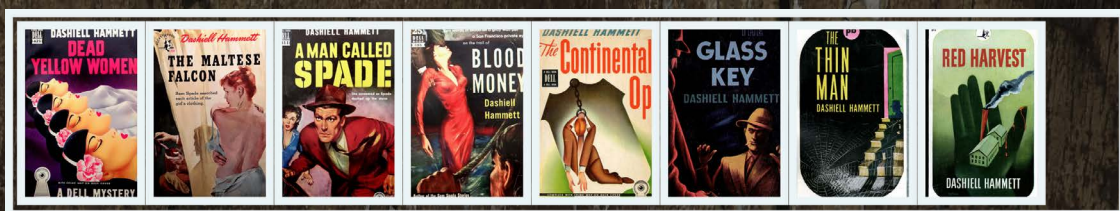


We shouldn't need to type that declaration very often, though, as `flex-start` is the initial value.



Using the `justify-content` property with values like `flex-end`, we can change where content is packed along the main axis line.

There are two more values with names that you might not recognise. They are `space-around` and `space-between`. With `space-around`, flex-items are evenly distributed along the main axis line. A browser uses the width of the space between each item to calculate a half-size space that's added before the first and after the final flex-item.



With `space-around`, flex-items are evenly distributed along the main axis line.

Using `space-between`, flex-items are again evenly distributed along the main axis line with the first item positioned at the `flex-start` position and the final one at the `flex-end`. Any remaining space is distributed evenly between the flex-items.



Remaining space between flex-items is calculated automatically by browsers.

Adapting to larger screens

With our hardboiled HTML all set and our smaller screen styling in place, we'll now give our design an extra level of fidelity and interaction that makes the most of the space available on larger screens. We'll redevelop our vertical list into a grid of eight magazine covers that reveal their descriptions when we press on them. We can do this simply by applying relative positioning, but no horizontal or vertical offsets:

```
@media (min-width: 48rem) {  
  .hb-target {  
    display : flex;  
    flex-wrap : wrap;  
    position : relative;  
    max-width : 700px; }  
}
```

We'll style our flex-items by allowing them to grow from a basis of 130px, and set margins that will space them evenly, horizontally and vertically:

```
@media (min-width: 48rem) {  
  .item {  
    display : block;  
    flex : 1 0 130px;  
    margin : 0 20px 20px 0; }  
}
```

We won't need any right margins on the fourth and fifth items, so we'll remove that using `:nth-of-type` pseudo selectors:

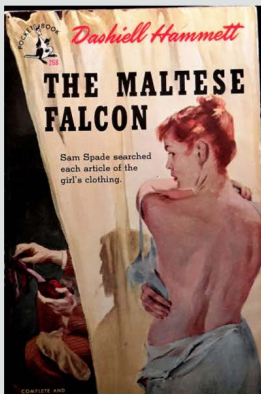
```
@media (min-width: 48rem) {  
  .item:nth-of-type(4) {  
    margin-right : 0; }  
  .item:nth-of-type(5) {  
    margin-right : 0; }  
}
```

To make this interface load faster, we can use each image twice: once for the main grid, and again as a **background-image** on each item's description overlay.

Now it's time to turn our attention to the descriptions. We'll position them absolutely to the **top** and **left** of each item and give them zero (0) opacity. This makes them completely transparent:

```
@media (min-width: 48rem) {
  .item__description {
    opacity : 0;
    position : absolute;
    top : 0;
    left : 0; }
}
```

Eight books by Dashiell Hammett



The Maltese Falcon

"The Maltese Falcon," first published as a serial in the pulp magazine *Black Mask*, is the only full-length novel in which Spade appears. The novel's atmosphere is dense as a San Francisco fog, and its descriptions of locations are so accurate that many can be pinpointed on a map

ADD TO CART

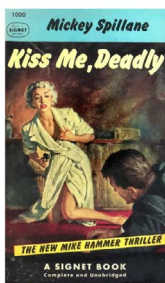
Here's how our finished grid interface should now look. Each of the description divisions is invisible, hidden by **opacity**.

Flexbox align-items

I'd like you to cast your mind back to when we first came to terms with flexible box layout. I mentioned then that when we make an element flex, we arrange its descendants along a main axis, or another axis that crosses it — and sometimes both. This gives us the ability to create layouts that are impossible to make when using floats. So far, however, everything we've developed has made use of only the main axis and that cross axis has gone untouched. It's time to put that right by learning about the `align-items` property.

```
.item {  
  align-items : stretch; }
```

We shouldn't need to type that declaration very often, though, as `stretch` is the initial value.



Kiss Me Deadly

Kiss Me, Deadly (1952) is Mickey Spillane's sixth novel featuring private investigator Mike Hammer.



The Big Kill

Drinking at a seedy bar on a rainy night, Hammer notices a man come in with an infant. The man, named Decker, cries as he kisses the infant, then walks out in the rain and is shot dead.

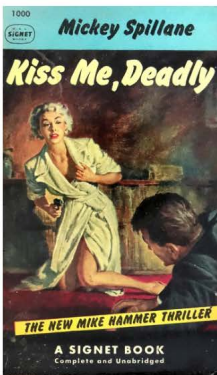


Vengeance Is Mine

Mike Hammer wakes up being questioned by the police in the same hotel room as the body of an old friend from World War II. His friend, Chester Wheeler, has apparently committed suicide with Hammer's own gun after they had been drinking all night.

Stretching flex-items along the cross axis is one of the most useful things about flexbox.

`align-items` is similar in concept to `justify-content`, but whereas `justify-content` aligns flex items along a main axis line, `align-items` uses the cross axis. There are four useful values that enable us to create interesting designs using flexbox. `flex-start` packs flex-items at the start of the cross axis line. When `flex-direction` is set to `row`, the start of the cross axis will be at the top. When it's set to `column`, this will be on the left.



Kiss Me Deadly

Kiss Me, Deadly (1952) is Mickey Spillane's sixth novel featuring private investigator Mike Hammer.



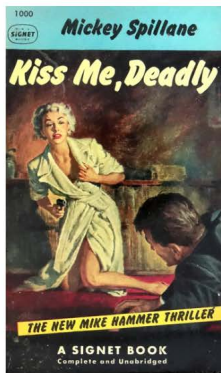
The Big Kill

Drinking at a seedy bar on a rainy night, Hammer notices a man come in with an infant. The man, named Decker, cries as he kisses the infant, then walks out in the rain and is shot dead.



Vengeance Is Mine

Mike Hammer wakes up being questioned by the police in the same hotel room as the body of an old friend from World War II. His friend, Chester Wheeler, has apparently committed suicide with Hammer's own gun after they had been drinking all night.



Kiss Me Deadly

Kiss Me, Deadly (1952) is Mickey Spillane's sixth novel featuring private investigator Mike Hammer.



The Big Kill

Drinking at a seedy bar on a rainy night, Hammer notices a man come in with an infant. The man, named Decker, cries as he kisses the infant, then walks out in the rain and is shot dead.



Vengeance Is Mine

Mike Hammer wakes up being questioned by the police in the same hotel room as the body of an old friend from World War II. His friend, Chester Wheeler, has apparently committed suicide with Hammer's own gun after they had been drinking all night.

Achieving layouts like this, where flex-items are aligned to the end of a cross axis, is something designers have wanted to do for years.

When we specify flex-items will align to the `center`, they'll be packed towards the centre of the cross axis. When we combine this with `justify-content:center` we can achieve horizontally and vertically centred layouts more easily than ever before.

Targeting with pseudo-class selectors

When we apply a unique `id`, we turn any element into a uniquely addressable fragment of a page. We can even target these fragments from links on the same page. The CSS `:target` pseudo-class selector allows us to change the styles applied to these elements when a user follows a link pointing to them. In this hardboiled interface, the `:target` pseudo-class selector changes the look of interface elements without using JavaScript.

Next, we'll change the styling properties of the description divisions using the `:target` pseudo-class selector. We'll give them dimensions, padding, and background and border properties and — most importantly — reset their `opacity` back to fully opaque (1):

```
@media (min-width: 48rem) {  
  .item:target .item__description {  
    opacity : 1;  
    width : 100%;  
    height : 480px;  
    padding: 40px 40px 40px 280px;  
    background-color: rgb(223,225,226);  
    background-repeat : no-repeat;  
    background-position : 40px 40px;  
    border: 10px solid rgb(236,238,239);  
    box-shadow: 0 5px 5px 0 rgba(0, 0, 0, 0.25), 0 2px 2px 0 rgba(0,  
    0, 0, 0.5); }  
}
```

Why so much padding on the left? We'll place a `background-image` into that space, reusing the same image we used to form the grid. To apply these background images, use a selector that descends from the `id` we applied to each item:

```
@media (min-width: 48rem) {  
  #hb-target-01:target .description {  
    background-image : url(target-01.jpg); }  
  #hb-target-02:target .description {  
    background-image : url(target-02.jpg); }  
  #hb-target-03:target .description {  
    background-image : url(target-03.jpg); }  
  #hb-target-04:target .description {  
    background-image : url(target-04.jpg); }  
  [...]  
}
```

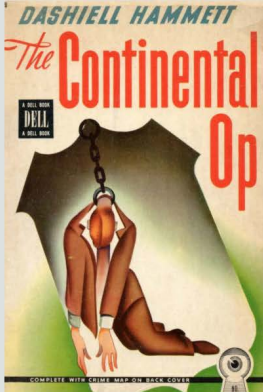
To complete this design, we need to include a way to hide the description division to reveal the image grid. We can achieve this by providing a link that points back to the outer items container thereby resetting the interface to its original state:

```
<a href="#hb-target"></a>
```

We'll use an attribute selector to position that link outside at the top-right of the design:

```
@media (min-width: 48rem) {  
  a[href="#hb-target"] {  
    position : absolute;  
    top : -20px;  
    right : -20px;  
    display : block;  
    width : 26px;  
    height : 26px; }
```

Eight books by Dashiell Hammett



The Continental Op

The Continental Op made his debut in an October 1923 issue of Black Mask, making him one of the earliest hard-boiled private detective characters to appear in the pulp magazines of the early twentieth century. He appeared in 36 short stories, all but two of which appeared in "Black Mask."

ADD TO CART

Our JavaScript-free, `opacity` and `:target` pseudo-class interface is now complete.

Breaking it up

Our designs don't have to appear flat and two-dimensional as we can use RGBA and opacity to give our designs depth. This richness, part of a design's atmosphere, can transcend responsive breakpoints. Unlike opacity in graphic design tools such as Sketch, we can use both RGBA and opacity to bring our designs to life with little more than a few simple lines of CSS. Now that's hardboiled.

No. 14

Borders

Borders are an integral part of a design's atmosphere, but it's always been hard to get overexcited about them. Yet CSS borders can be exciting because they include properties that open up a wealth of creative opportunities. These properties are `border-radius` to give (almost) any element those rounded corners our clients love so much, and `border-image` for using images inside those borders. Let's investigate.

Rounding corners with border-radius

You don't have to go too far on the web before you'll find rounded corners. We use them to make irregular shapes, style links so they look like buttons, and chamfer the sharp edges off boxes. In the past we used images to create these rounded corners and that meant first carving out images. Thankfully, we don't need to abuse images any more because `border-radius` makes it easy to add uniform or non-uniform rounded or elliptical corners to almost any element.

Pushing the right buttons

Using `border-radius` we can round every corner of a box uniformly using either pixels, ems or percentages based on the size of the box. We'll start by styling links on the 'Get Hardboiled' store to make them look more like buttons. Here's the HTML:

```
<a href="cart.html" class="btn">Add to cart</a>
```

Now transform that link into a faux button. We'll add padding specified in rems (to allow its proportions to scale up and down when a user changes the text size in their browser), a background colour and a darker border at the bottom:

```
.btn {  
padding : 1rem 1.25rem .75rem;  
background-color: rgb(188, 103, 108);  
border: 5px solid rgb(140, 69, 73); }
```

Complete the look by rounding every corner with a uniform, rem-based `border-radius` that will also scale along with the text:

```
.btn {  
border-radius : 1rem; }
```

Rounding selected corners

If we round every corner of this description box, it will look out of place next to the square-cornered book covers that appear below it. Fortunately, we're able to specify individual radii for each corner:

```
div {  
border-top-left-radius : 1rem;  
border-top-right-radius : 1rem;  
border-bottom-left-radius : 0;  
border-bottom-right-radius : 0; }
```

Even though contemporary browsers offer page zooming, it's still important to make designs that are flexible by using rem units and percentages where possible, so that our designs are responsive to any size or type of screen.



The tiniest details matter and there is something not quite right about the rounded corners on the bottom of this box.



By selectively rounding only the top-left and top-right corners, we visually link the book covers to their descriptions.

Making irregular shapes

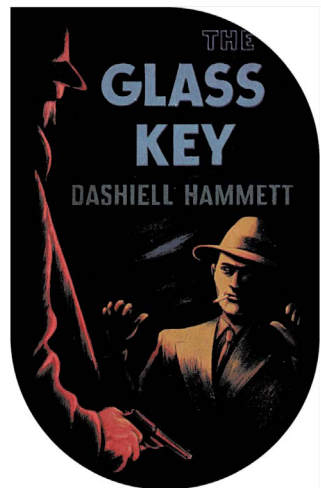
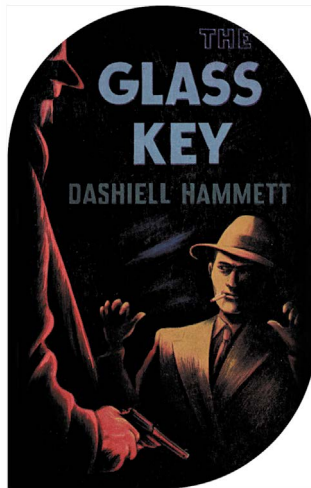
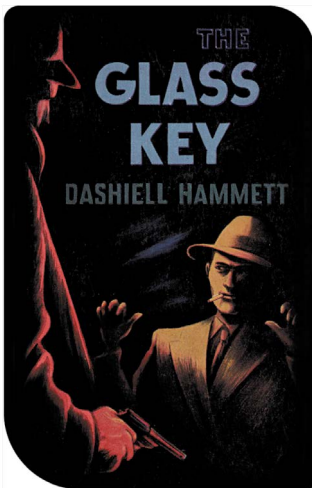
Rounded corners don't have to be circular and we can use twin radius values to create ellipses, where the first value sets a horizontal radius and the second a vertical radius. In this next declaration, the same two radii are applied to all four corners.

```
.h-card {
border-radius : 30px 60px; }
```

We can also create more complex shapes by specifying twin values individually for each corner.

```
.h-card {
border-top-left-radius : 5px 30px;
border-top-right-radius : 30px 60px;
border-bottom-left-radius : 80px 40px;
border-bottom-right-radius : 40px 100px; }
```


By selectively styling each corner with different `border-radius` properties, we can create even more complex shapes.



Shorthand properties

Writing longhand `border-radius` declarations is inconvenient, so it's lucky we can use shorthand values to crush that last example back to just one line:

```
.h-card {  
border-radius : 15px 30px 45px 60px; }
```

When we need to combine elliptical corners in a shorthand declaration, we specify all horizontal values before a forward slash and all vertical values after:

```
.h-card {  
border-radius : 60px / 15px; }
```

Translucent box-shadow with RGBa

When we want design to stand out we could add a subtle shadow by combining `box-shadow` with RGBa. The `box-shadow` syntax is easy to learn as the first and second values apply horizontal and vertical offsets respectively, the third applies blur-radius and finally we set the shadow colour inside parentheses:

```
.item__description {  
box-shadow : 0 1px 3px rgba(0,0,0,.8); }
```

Unless it's noon and you're in the middle of a desert, everything you see around you has more than one shadow. To create a more natural three-dimensional effect, add a second, softer shadow. This one should have a greater vertical offset, a wider blur-radius and be more transparent. The values for each shadow should be separated using a comma:

```
.item__description {  
box-shadow :  
0 1px 1px rgba(0,0,0,.8),  
0 6px 9px rgba(0,0,0,.4); }
```

As in nature, we can cast light onto an element from any direction, so to throw shadows either above or to the left, use negative values in our shadows:

```
.item__description {  
box-shadow :  
0 -1px 1px rgba(0,0,0,.8),  
0 -6px 9px rgba(0,0,0,.4); }
```

For individual radii, the values are set clockwise starting from the top-left, so: top-left; top-right; bottom-right; then bottom-left. When we omit bottom-left, its radius will be the same as top-right. If we omit bottom-right, it will be the same as top-left and so on. To set an elliptical value on each radius individually, you'd use something like this:

```
.h-card {  
border-radius: 5px 30px 80px 40px / 30px 60px 40px 100px; }
```

Adding images to borders

When I wrote the first edition of *Hardboiled Web Design*, the only choices designers had for their borders were: dotted, dashed, solid and double; and groove, ridge, inset and outset. OK, put your hands in the air those of you who have recently used any of the last four. Ever. Me neither.

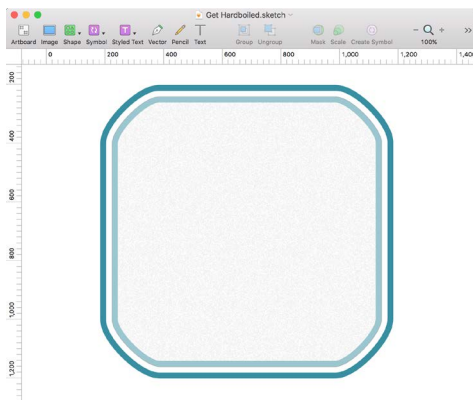
Back then, CSS `border-image` had only just enabled designers to add images — either bitmaps, SVGs or even CSS gradients — within an element's border space and I was very excited about the possibilities that this new property would bring. After all, we can add images to the borders of any element, even table cells and rows (unless they've been set to collapse their borders).

So how did that work? Did we see the web flooded with clever border designs? No. Not at all. In fact, when I've asked attendees at my CSS workshops these past five years whether they've used `border-image`, only a small minority have ever raised their hands.

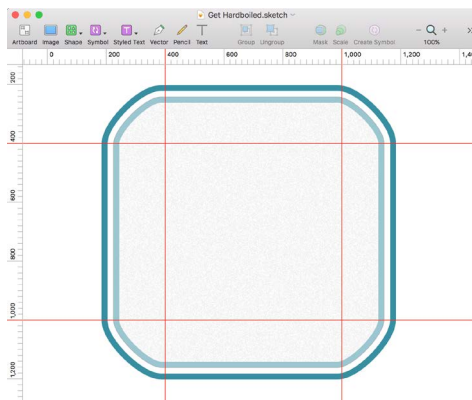
I wonder why could this be, because `border-image` opens up new avenues for creativity. It's also perfectly suited to the demands of responsive web design, where we need to keep our downloads light and make the most we can out of the smallest assets.

It's entirely possible that people steer clear of **border-image** because its syntax can be a little tough to learn, so I'll guide you through it as painlessly as possible. Let's start by using **border-image** to style a box containing comments on a blog entry. Here's our HTML:

```
<div class="media h-review">
  <div class="media__figure"></div>
  <div class="media__content"> [...] </div>
</div>
```



The CSS **border-image** property is a powerful tool for making tiny images stretch and repeat to create interface elements of any size. It can be particularly effective in fluid layouts and on designs for mobile devices where every byte counts.



border-image slices up an image into nine parts using slice guides similar to those in our graphics tools. Slice guides can be set any distance from the top, right, bottom and left sides of an image.

Slicing border images

The concept of **border-image** is to take a tiny asset — one that's as small as we can make it — and then, using only CSS, slice and use its corners and sides to style what could be much larger elements. Just as we can slice up images using graphics software, **border-image** slices up any image into nine parts of a 3×3 grid.

The image that we're using is only 60×60 pixels and weighs in at only a few bytes. We're going to use its four corners as the corners of any element that we apply them to. The top-left of our image will be used in the top left corner of our element's border. The bottom-right will be used in the bottom right and so on.

The `border-image-source` property specifies the URL of the image we're slicing up to insert into the borders of our comments. In this case we're using a bitmap image:

```
.h-review {  
border-image-source : url(h-review.png); }
```

And `border-image-slice` sets the positions of our CSS slice guides.

```
.h-review {  
border-image-slice : 20 20 20 20; }
```

You'll notice we don't need to add units to the `border-image-slice` values as we're using a bitmap image and the browser automatically assumes we're using pixels. We'll cover units for other types of border images as we explore them.

Don't forget to set the width of these borders as without it there will be nowhere for our border images to display:

```
.h-review {  
border-width : 20px 20px 20px 20px; }
```

What's created between those slice guides become the parts of our border: four corners, four sides and the central part of the tiny image, if we choose to use it.

For the blog entry comments we're designing, we'll set slice guides twenty pixels from each side and the browser will apply those values clockwise from the top (top, right, bottom, left).

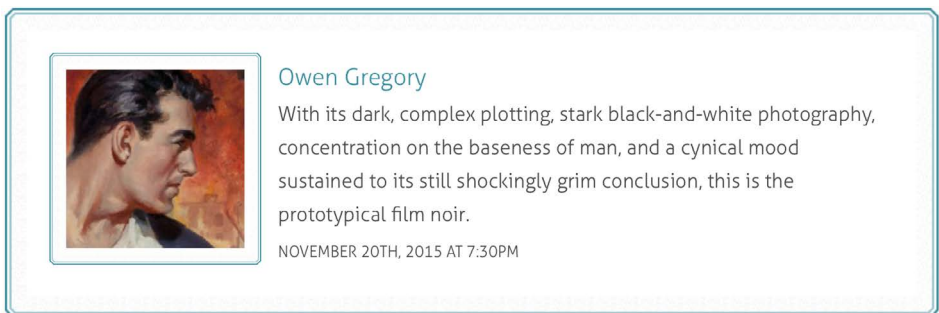
Writing all those declarations was a little long-winded, so instead of specifying `border-image-source` and `border-image-slice` separately, this time we'll combine them into one shorthand `border-image` property:

```
.h-review {  
border-image : url(h-review.png) 20;  
border-width : 20px 20px 20px 20px; }
```

We can also combine duplicated values into either pairs or even a single value, just as we would when writing CSS margins and padding:

```
.h-review {  
border-image : url(h-review.png) 20;  
border-width : 20px; }
```

With our `border-image` slices decided and space made for them inside the element's border, the browser pushes the sliced corners of our image into place in the corners of the element we're styling.



The corners of our blog entry comments have been filled with slices from our tiny border image.

When we use only one value, that value will be used for all four borders. If we omit a `border-bottom` value, a browser will use the same value as `border-top`. Likewise when we omit a `border-left` value, a browser will use the same value as `border-right`.

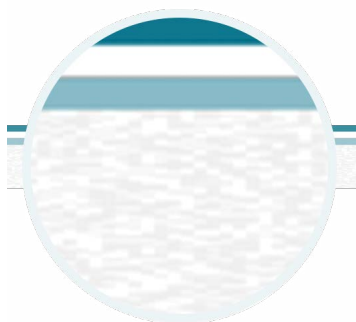
Neither the images we slice nor the position of the slice guides need to be symmetrical. Slice guides don't have to be set at equal distances from the four sides of an image. To make borders asymmetrical, we can specify separate values for each of the slice guides. For our next example, slice guides will be set at: ten pixels (top); twenty pixels (right); forty pixels (bottom); and eighty pixels (left) to create the irregular shape of this border.

```
.h-review {  
border-image : url(asymmetrical.png) 10 20 40 80;  
border-width : 10px 20px 40px 80px; }
```

Asymmetrical border images can take on any shape or size our flexible designs demand, while at the same time reducing the size of the images we need to download.

Styling between the borders

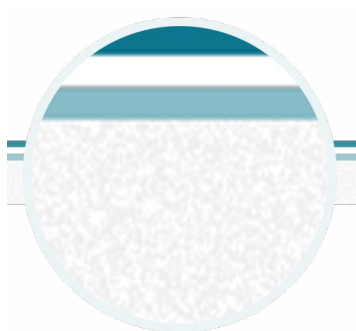
With the corners of our border images in place, let's turn our attention to the sides between those corners. As you might imagine, the border at the top of our image will be placed in the top border of the element that we're styling. The same will be true of the other three borders. Of course, in a responsive design, we never know just how wide or how tall the elements we're styling are going to appear, so we need to take care to fine-tune how images will repeat or even stretch when they fill a border:



The border image *stretched* to fill a border.

Stretch: When the image we've sliced is flat or smooth, we might stretch it to fill the available width. Our twenty pixel wide original slice might be stretched to hundreds or thousands of pixels wide without degrading.

```
.h-review {  
border-image-repeat : stretch; }
```



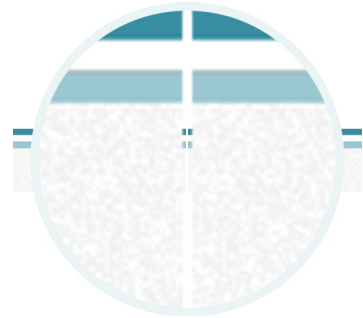
The border image *repeated* to fill a border.

Repeat: If our border image has texture such as noise, stretching it isn't an option, so we might repeat it to fill the available width. With textured images like this we'd shouldn't need to worry about matching the edges of that repeat.

```
.h-review {  
border-image-repeat : repeat; }
```


Round: If our border image has a pattern and not only can't be stretched, but we need to match the edges of the repeat, we can specify that repeat to be **round**. The browser will re-size the border image as needed so that only whole pieces will display inside the border.

```
.h-review {  
border-image-repeat : round; }
```



Resizing slices to ensure that only whole pieces fill the border space.

Space: Similar to **round**, when using the **space** property only whole pieces will display inside the border. But instead of resizing the border image, the browser will add space between the repeat.

```
.h-review {  
border-image-repeat : space; }
```



Repeating whole slices and adding space between tiles so that an area is evenly filled.

When we need to specify separate **stretch**, **repeat**, **round** or **space** values for each border, we can write multiple keywords on the same line.

```
.h-review {  
border-image-repeat : stretch round; }
```

Outsetting a border image

There can be times when we'd like a border's image to extend beyond the normal boundaries of the element's border-box. Using the `border-image-outset` property, we can do just that. The simplest syntax extends the border evenly on all sides by `5px`:



Comparing the non-extended border (top) with one extended by 5px (bottom).

```
.h-review {  
border-image-outset : 10px; }
```

But of course, there being four possible borders on every element, we can specify how much each one extends individually:

```
.h-review {  
border-image-outset : 10px 0 10px 0; }
```

We can also combine duplicated values into either pairs or even a single value, just as we would when writing CSS margins and padding:

```
.h-review {  
border-image-outset : 10px 0; }
```

The `border-image-outset` property accepts any CSS length value including the most commonly used `px`, `em`, `rem` and even `vh` and `vw`, or you may choose to use a simpler unitless number.

Filling in the centre

So far we've used all four corners and all four sides of our small border image, but what about the centre? By default, the browser will ignore the centre of an image after it's been sliced. But we can put it to good use on the bordered element by adding the `fill` keyword to our `border-image-slice` declaration:

```
.h-review {  
border-image-slice : 20 fill; }
```



Owen Gregory

With its dark, complex plotting, stark black-and-white photography, concentration on the baseness of man, and a cynical mood sustained to its still shockingly grim conclusion, this is the prototypical film noir.

NOVEMBER 20TH, 2015 AT 7:30PM

Filling in the centre of our blog entry comments with subtle noise that repeats across the background.

Using alternatives to bitmaps

Border images are perfectly suited to the demands of responsive web design as they allow us to take the tiniest bitmap images and use them to style borders of elements of any size. But the images we use in our borders need not be bitmaps at all, as we can also use scalable vector graphics (SVGs) and even gradients made from pure CSS. The simplest way to use vector images inside borders is to use `border-image-source` in exactly the same way as we would when using a bitmap:

```
.h-review {  
border-image-source : url(h-review.svg); }
```

This method is well supported and every browser that has implemented `border-image` allows us to set SVG as a `border-image-source`.

Using CSS gradients for borders¹⁵ is perhaps the most interesting alternative to bitmaps as it opens up a wealth of new creative opportunities. CSS gradients add a negligible amount of weight to our pages and being included within a style sheet file add no extra requests, making them perfect for responsive web design.

Don't worry if you've not used CSS gradients before as we'll be covering them in detail later. For now, let's add a striped pattern made by adding a repeating linear gradient to our border:

```
.h-review {  
border-image-repeat : repeat;  
border-image-source : repeating-linear-gradient(-45deg, white,  
white 3px, #ebf4f6 3px, #9Bc7d0 6px);  
border-image-slice : 10;  
border-width : 10px; }
```

¹⁵ CSS Tricks created a collection of CSS gradients that we can use as inspiration for our own gradient borders: css-tricks.com/examples/GradientBorder



Owen Gregory

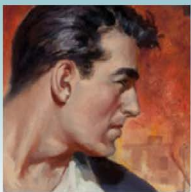
With its dark, complex plotting, stark black-and-white photography, concentration on the baseness of man, and a cynical mood sustained to its still shockingly grim conclusion, this is the prototypical film noir.

NOVEMBER 20TH, 2015 AT 7:30PM

Repeating a gradient border is a perfect example of how two CSS properties — border images and gradients — can be combined to help keep our responsive designs fast and flexible. Sara Soueidan wrote a useful article¹⁶ about **border-image** that explains its shorthand syntax in more detail than we're able to do here.

Of course, we needn't stop there, as we're able to combine border images and gradients to create effects that are more difficult to achieve using other CSS properties. For our next example we'll use a simple linear gradient that starts at the top with a darker blue and fades into a lighter blue over the height of the element:

```
.h-review {  
border-image-source : linear-gradient(to bottom, #9Bc7d0,  
#ebf4f6 100%);  
border-image-slice : 10;  
border-width : 10px; }
```



Owen Gregory

With its dark, complex plotting, stark black-and-white photography, concentration on the baseness of man, and a cynical mood sustained to its still shockingly grim conclusion, this is the prototypical film noir.

NOVEMBER 20TH, 2015 AT 7:30PM

Adding a linear gradient to a border can help us create designs that are difficult to achieve using other CSS properties.

¹⁶ tympanus.net/codrops/css_reference/border-image

Our gradient fades from top to bottom over the height of the element and, of course, this height will vary depending on the volume of content it contains. To help keep the gradient borders consistent across all of our blog entry comments, we'll change that variable 100% value in our gradient declaration to a consistent, but still flexible, 8rem:

```
.h-review {
border-image-source : linear-gradient(to bottom, #9Bc7d0,
#ebf4f6 8rem); }
```



Owen Gregory

With its dark, complex plotting, stark black-and-white photography, concentration on the baseness of man, and a cynical mood sustained to its still shockingly grim conclusion, this is the prototypical film noir.

NOVEMBER 20TH, 2015 AT 7:30PM

Changing gradient values from percentages to flexible rem units helps us keep the gradients consistent across different height elements.

Styling a hardboiled business card

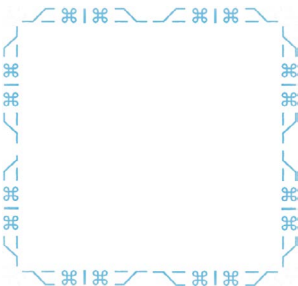
We'll round off this chapter by using border images to implement a hardboiled business card. We don't need any fancy HTML to make this card because, as it contains contact information, we should use the h-card microformat which looks like this:

```
<div class="h-card">
<h3 class="p-name">S.A.Fari</h3>
<p class="p-role">Web Inspector</p>
<h4>Checking all elements</h4>
<p>Dial <span class="p-tel">4.0.4 5531.21.10</span></p>
<p>Member of the WebKit team since 2006</p>
</div>
```



This isn't just any dog-eared scrap of cardboard. It has a decorative border that's been made from Apple keyboard symbols.

Start with a small PNG image, only 160×160px and weighing in at only 3Kb. We'll use border images to style an element that could be an infinite number of different sizes.



Starting with a tiny image containing the four corners and a pattern that we'll use to style the sides of our business card.

We'll first set slicing guides that are an even twenty pixels from each side of the image we're using to style our border. Then we'll set our borders' width to the same twenty pixels:

```
.h-card {  
border-image-source : url(safari.png);  
border-image-slice 20;  
border-width : 20px; }
```

So far, so good, as those declarations push the corners of the source image to the four corners of our new business card — but what about the sides? With this intricate design, we must take care when controlling how those sides are displayed.

stretch is out of the question for a design like this, as is a simple **repeat** which could cause mismatches where patterns in the sides join the corners. We won't want **space** to insert space between our patterns as they repeat, so we'll choose **round**. This will slightly adjust the size of the repeating pattern so that only whole pieces of it are displayed. To complete our design and lift our hardboiled business card off the page background, let's add two shadows: the first harder and darker, the second lighter and softer.



```
.h-card {
  box-shadow : 0 2px 5px
    rgba(0,0,0,.5),
    0 20px 30px rgba(0,0,0,.2); }
```

By using the **round** keyword, we instruct browsers to resize the parts of the decorative image so that only whole pieces of it will fit inside the border.

Changing a border image's width

In every **border-image** example until now, we've made the width of a border precisely match the size of an image slice, but what happens when they are different?

When we change a border's width we can control how large the images they contain will appear. To see this effect in action, reduce the border's width down to only ten pixels and watch as a browser scales the image to match the new border width.

```
.h-card {
  border-image : url(safari.png) 20 round;
  border-width : 10px; }
```


Making a border's width larger than the size of a slice has the opposite effect. Scale up a border's width in several increments to see the increase in size of the border's image.



`border-width : 30px;`



`border-width : 40px;`



`border-width : 50px;`



`border-width : 60px;`

Breaking it up

Whether we make our corners rounded, elliptical or fill them with images, with `border-radius` and `border-image`, CSS borders can be interesting. These properties save us time, solve common implementation problems and open up new creative possibilities, so start making your borders hardboiled.

No. 15

Background images

Not too long ago, setting more than one background image on a single element led to presentational junk in our markup. We treated HTML like a goon just to satisfy our selfish need for a visual design, but we can quit abusing our markup because all contemporary browsers allow us to apply more than one background image. We're also able to change the origin point and size of the backgrounds we apply, which helps to open up new creative opportunities. Let's get started by making a design using multiple background images.

Multiple background images

For this design we're going to use background images to give the illusion that a heading wraps around the area containing an article. In the past, we would have needed two nested elements to create this illusion, applying a different background image to each one:

```
<div class="left">
  <div class="right"> [...] </div>
</div>
```

Fortunately, our markup can stay hardboiled as we need use only a single HTML `section` element and apply two background images to that.

```
<section> [...] </section>
```

I've made two background images for our design, one to position to the left, one to the right. We can specify both in a single `background-image` value, separating the source of each image with a comma:

```
section {
  background-image :
    url(section-left.png),
    url(section-right.png); }
```

At this stage we should also specify the position and repeat for both background images. We can do that in the same way, separating each value with a comma:

```
section {  
  background-position : 0 0, 100% 0;  
  background-repeat : no-repeat, no-repeat; }
```

To save a few bytes, we can write those values in shorthand by combining source, repeat and position for both images into a single declaration:

```
section {  
  background :  
  url(section-left.png) no-repeat 0 0,  
  url(section-right.png) no-repeat 100% 0; }
```

Overlapping background images

When multiple background images overlap, you might think that their order follows the CSS positioning stacking order, where the element furthest down the source appears highest, or closer to the viewer (unless `z-index` determines otherwise.) Something like this:

```
section {  
  background :  
  url(background.png) no-repeat 0 0,  
  url(middle-ground.png) no-repeat 0 0,  
  url(foreground.png) no-repeat 0 0; }
```

You'd be wrong. The first image in a declaration will appear closest to the viewer and for very good reason. If an older browser doesn't support multiple background images it will display only the first image before it chokes on the first comma.

```
section {  
  background :  
    url(foreground.png) no-repeat 0 0,  
    url(middle-ground.png) no-repeat 0 0,  
    url(background.png) no-repeat 0 0; }
```

Everything old is new again

I bet that the box model was one of the first things you learned about CSS. It may also have been one of the first things to trip you up because, in the traditional box model, padding and borders are added to, and not subtracted from, the size of an element.

Add ten pixels padding and a five-pixel border to a one hundred pixels square box and the resulting width and height will be 130 pixels (100px + 20px + 10px = 130px.) This is the default box model in all modern browsers and CSS3 now calls this the **content-box**.

In fixed-width designs, this traditional box model rarely causes any problems. But when we're developing responsive designs, this box model can cause headaches because CSS never made it easy to mix percentages with fixed units like pixels and ems.

To illustrate this, imagine a box that fills one hundred per cent of the browser window. If that box needs ten pixels padding, what width should we give it? If the same box then needs a five-pixel border, how wide will the box be now? Historically, to work around these difficulties we resorted to nesting one element that used pixels inside another that used percentages.

To help solve the problem of mixing pixel and percentage units on the same element, CSS introduced a second box model type — a **border-box** — where padding and borders are subtracted from, not added to, a box's dimensions. This makes it easy to use a one hundred percent width plus padding and borders set in pixels for this **section**.

```
section {  
width : 100%  
padding : 10px;  
border : 5px solid rgb(235, 244, 246);  
box-sizing : border-box; }
```



In this example, padding and borders are added to the dimensions of a **content-box**.



Whereas with **border-box**, padding and borders are subtracted.

Does the way `border-box` draws an element sound familiar? You must be as old as I am, because that was the way Microsoft calculated box sizes up until Internet Explorer 6.

Clipping backgrounds

When we combine a background image or colour with a border, by default the background extends underneath the border and out to the edges of a box. CSS3 calls this default behaviour a `border-box` and the `background-clip` property gives us control over this behaviour:

```
.h-card {  
  background-image : url(h-card.png);  
  border : 10px dashed rgb(0,0,0);  
  background-clip : border-box; }
```

If we specify a box to be a `padding-box`, any background colour or image will be clipped to the outer edges of the box's padding and won't extend behind its border:

```
.h-card {  
  background-clip : padding-box; }
```



A browser's default behaviour is to draw a box's border over the top of its background colour or image, but this isn't always desirable. Luckily, `background-clip` gives us the power to change that behaviour.

Defining a background image's origin

You'll no doubt already know about CSS's `background-position` property.¹⁷ Browsers position a background image relative to the outer edges of an element's padding, inside its border. CSS3 calls this origin a `padding-box` and extends creative possibilities by providing properties with several `background-origin` values:

One of these `background-origin` properties positions a background image relative to an element's outer edges, beneath its border. It's called, unsurprisingly, a `border-box`:

```
.h-card {  
  background-origin : border-box; }
```



With a `content-box`, a background image's origin will be relative to the outer edge of any content, inside its padding:

```
.h-card {  
  background-origin : content-box; }
```



¹⁷ A refresher on `background-position` values: `0 0` is the same as `left top`; `50% 0` is fifty per cent horizontally but still at the top; and `100% 100%` is the same as right bottom. You can specify a background image's position using keywords (`left`, `top`, `right`, `bottom`), percentages, pixels and any other CSS units.

Sizing background images

Working with large background images can often be a headache and I can't count the number of times in the past that I resized images in Photoshop while working on a design. CSS has a `background-size` property that gives us far greater control over background image sizes. This can save time and open up a world of creative opportunities.

The `background-size` property takes horizontal and vertical pixel or percentage values, plus optional keyword values of `cover` and `contain`:

```
.item__img {  
  background-size : 100% 50% contain; }
```

Let's start with a box. Its dimensions are 200×310 pixels and we'll add a background image that's the same size as that box:

```
.item__img {  
  width : 200px;  
  height : 310px;  
  background-image : url(magazine.jpg); }
```

When both sets of dimensions are identical there's no problem, but do you hear that? It's the sound of a client changing their mind about a design. Don't worry, `background-size` will take care of it for us and save us a trip back into our graphics software.

Pixel units	Size a background image using pixels (width and height).
Percentages	Specify a background image size as a percentage of the size of the element it is attached to (width and height).
<code>cover</code>	A background image's aspect ratio <i>covers</i> an element's background.
<code>contain</code>	A background image's aspect ratio is <i>contained</i> inside an element.

Sizing background images using pixels

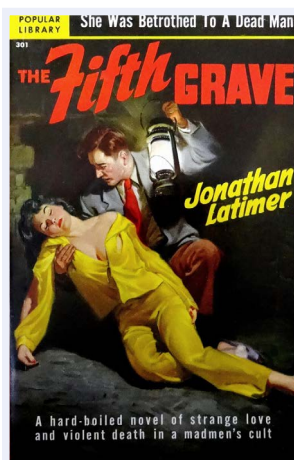
The `background-size` property allows us to specify the exact size of a background image using pixels, like this:

```
.item__img {  
  background-size : 200px 310px; }
```

The first value defines the width, the second is height. When we don't specify a height, a browser will automatically choose `auto` and maintain the intrinsic aspect ratio of the background image. In the example, these three values all produce identical results:

```
.item__img { background-size : 200px 310px; }  
.item__img { background-size : 200px auto; }  
.item__img { background-size : 200px; }
```

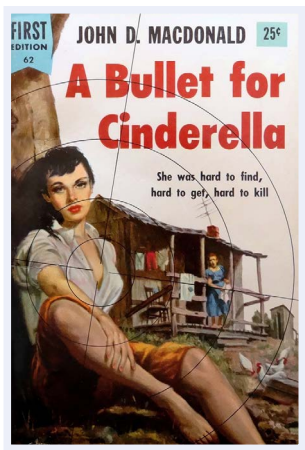
If an element changes size, perhaps to 240×350px, we can apply those new sizes to a background image and it will scale or stretch to fit. We could even specify a background size that is very different from an element. Here are three examples:



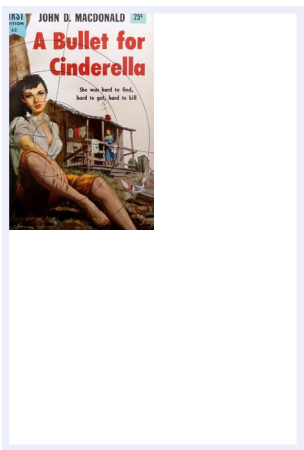
`background-size : 240px 350px; background-size : 120px 175px; background-size : 60px 87px;`

Sizing background images in percentages

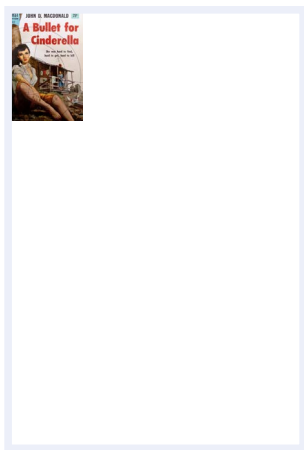
CSS enables us to scale a background image using percentages. In the following series of examples, the first value defines an image's width, the second its height. When we don't specify a height, a browser will automatically choose `auto` and maintain an image's aspect ratio.



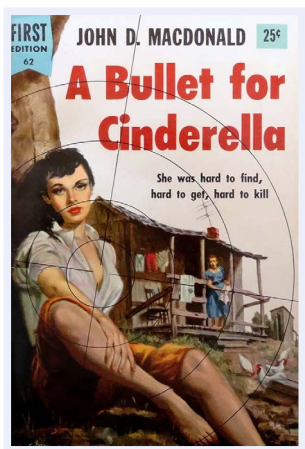
`background-size : 100% 100%;`



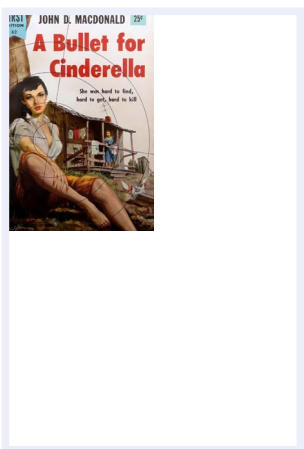
`background-size : 50% auto;`



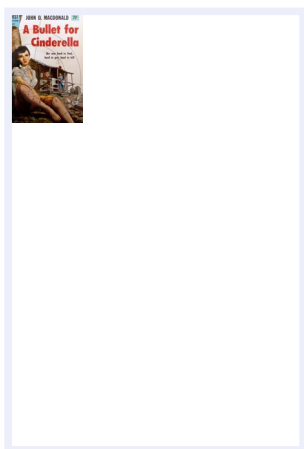
`background-size : 25%;`



`background-size : auto 100%;`



`background-size : auto 50%;`



`background-size : auto 25%;`

Cover and contain

Let's make something a little more adventurous. It's a promotional panel for the 'Get Hardboiled' site that's designed to promote a special book. Start with hardboiled HTML: one **article** that contains a heading and a paragraph:

```
<article class="item">
  <h1 class="item__header">The Phantom Detective</h1>
  <p class="item__description">The Phantom Detective was the
second pulp hero published after The Shadow. The first issue was
released in February 1933. The title continued until 1953, with
a total of 170 issues.</p>
</article>
```

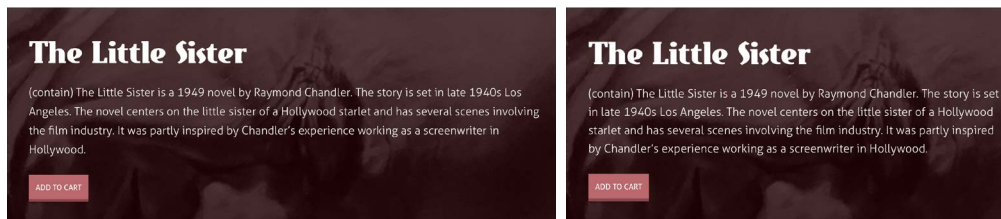
This **article** will span one hundred per cent of the width of its container, but we'll also need to set padding in pixels, a combination that's difficult to pull off. Don't worry, by declaring **border-box** we'll make it easy to mix those percentages with pixels:

```
.item {
width : 100%;
padding : 40px 80px 40px 280px;
box-sizing : border-box; }
```

If you're wondering why the large amount of left padding is needed, hold that thought — we'll get to that in just a minute. Now apply a large background image. It's the key to this design and we'll centre it horizontally and fix it to the bottom of the section:

```
.item {
width : 100%;
padding : 40px 80px 40px 280px;
background: url(scene.jpg) no-repeat 50% 100%;
background-size : 200px 300px;
box-sizing : border-box; }
```

The result's looking good, but it's not perfect because when a user narrows their browser window, they'll cut off both sides of the background image.



On the right, the background image is cut off when someone reduces the size of the browser window.

CSS has two more **background-size** keywords. They both scale an image while maintaining its aspect ratio, which is perfect for just this situation. Somewhat confusingly though, these keywords are called **cover** and **contain**. First, the **contain** keyword, which scales an image so that both its width and height are contained inside the element and not clipped:



This background image is contained inside its element.

With the `cover` keyword, both the background image's width and height scale to cover the background.

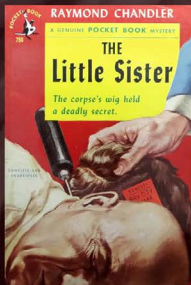


This background image will always cover the element, even when that element changes size — perfect for responsive web designs.

To finish our promotional panel design, we'll add a second background image of a book cover. Any ideas where we'll position it? You guessed it: in the space left by the large amount of left padding, forty pixels from the top and forty pixels from the left.

Separate the position, repeat and size values of each image using commas — and remember, the image we specify first will be the one that appears closest to the viewer.

```
.item {  
  background-image :  
    url(cover.jpg) 40px 40px no-repeat,  
    url(scene.jpg) 50% 100% no-repeat;  
  background-size : 200px 300px, cover; }
```



The Little Sister

(cover) The Little Sister is a 1949 novel by Raymond Chandler. The story is set in late 1940s Los Angeles. The novel centers on the little sister of a Hollywood starlet and has several scenes involving the film industry. It was partly inspired by Chandler's experience working as a screenwriter in Hollywood.

ADD TO CART

Our final result: a responsive web design accomplished using two background images. The second scales to fit any size container while at the same time maintaining its aspect ratio; the first appears at its native size. Now that's hardboiled.

Breaking it up

When we need to apply more than one background image to an element, we can keep our HTML hardboiled using CSS backgrounds. Background properties give us precise control over the size of our background images and how they're rendered behind our elements. Are you using them yet? What are you waiting for? Christmas?

Gradients

No. 16

AS YOU LOOK AT THE STATE OF WEBSITE DESIGN in 2015, you could be forgiven for thinking that web browsers are only capable of displaying flat colours. A flat design aesthetic – possibly inspired (but certainly fuelled) by the design of recent operating systems such as iOS and Windows – has become the norm. Almost every site I see includes large, flat areas of colour, often laid out across horizontal bands, almost always the full width of our screens, with flat or outlined buttons, and icon graphics that are also flat. I hope designers will soon move on from the mediocrity this flat aesthetic epitomises and that we'll see web design that's rich and full of life.

Gradients help bring a flat, two-dimensional design to life. Making them in Photoshop or Sketch isn't difficult, but in the era of responsive web design where we're more concerned than ever about flexibility and performance, making gradients using CSS simply makes sense.

It's been possible to make gradients using SVG for some time, but SVG isn't the simplest of technologies to use for making gradients. Thankfully, creating gradients of all kinds – linear, radial and repeating – using CSS is much more convenient and in this chapter we'll do just that.

Gradients are background images

I have to admit that I was very surprised when I first learned that CSS gradients were a type of background image — just like bitmap images or SVG — and not distinct like a background gradient property would've been. It took me a while to realise that one of the benefits of gradients being background images is our ability to mix them with other image formats in a multiple `background-image` declaration.

Linear gradients

A linear gradient is possibly the most common and useful type of gradient and in CSS it consists of a gradient axis and two or more colours. That axis can be horizontal, vertical or at any angle we choose across an element's background. The concept and syntax of CSS gradients shouldn't be difficult to grasp, particularly if you're experienced using Photoshop or Sketch.

We'll start writing a vertical linear gradient to style a button, a gradient made up of two colours from the 'Get Hardboiled' brand colours:

```
div {  
  background-image : linear-gradient(  
    #fed46e,  
    #ba5c61); }
```

We can define a gradient's colour values using either keywords, hexadecimal values, RGB and RGBA, or HSL and HSLA, and separate each of the colours in our gradient using a comma.

Next, we'll specify a gradient's direction, simply by stating where we want the gradient 'to' end. This could be on the left or on the right, at the bottom or at the top. We don't need to specify where the linear gradient starts as this is implied from where it ends.

Our first example ends at the top:



This one ends on the right:



This next gradient ends on the left:



The `to` syntax doesn't only work for the top, right, bottom or left sides of an element; it works from the four corners of an element too, enabling us to create diagonal gradients.

This gradient ends at the bottom-right:



```
div {  
  background-image : linear-gradient(  
    to bottom right,  
    #fed46e,  
    #ba5c61); }
```

The next gradient ends bottom-left:



```
div {  
  background-image : linear-gradient(  
    to bottom left,  
    #fed46e,  
    #ba5c61); }
```

This gradient ends top-left:



```
div {  
  background-image : linear-gradient(  
    to top left,  
    #fed46e,  
    #ba5c61); }
```

Our final gradient ends top-right:



```
div {  
  background-image : linear-gradient(  
    to top right,  
    #fed46e,  
    #ba5c61); }
```

When we need to specify the precise angle of a gradient in degrees, we can follow the same pattern, replacing `to` with a number of degrees.

This next example includes a thirty degree gradient:



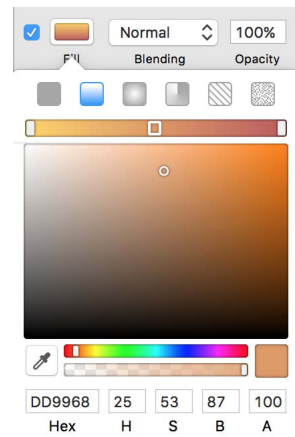
When needed, we can reverse the angle of a gradient using a negative number of degrees:



Adding colour stops

Simple gradients are created from two colours, but our designs will often require more complex gradients that include one or more colour stops. To help us visualise what a colour stop is, let's head back into familiar territory, graphics software, in this case Sketch. Here, we can add colours to a gradient by double-clicking on the gradient fill bar.

Adding a colour stop to a CSS gradient works exactly the same way and when we specify one or more colour stops, a browser will blend smoothly between them.



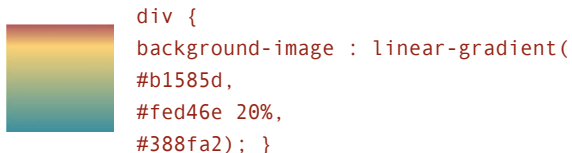
Adding colour stops to a gradient in Sketch.

In our next example, the linear gradient flows from the top to the bottom and blends from red through yellow and ends with blue:

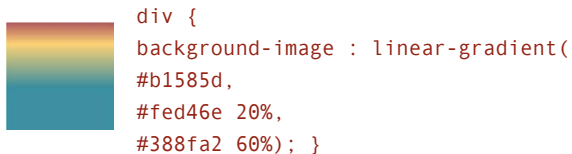


As we haven't yet specified the positions where we'd like these colours to blend, they'll blend evenly across the gradient's axis. When we'd like precise control over where our colours blend, we can introduce a colour stop at the position where we'd like a colour to start blending. In this example, we'll specify that the second, yellow colour starts twenty percent from the start of the gradient axis.

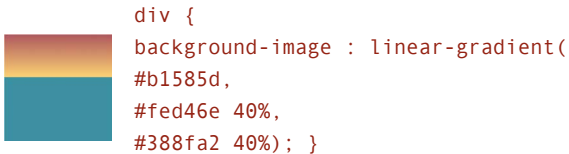
Look closely and you'll see a 20% next to our yellow colour value:



We're able to add colour stops to every colour in our gradients, so we might also specify that our final blue colour starts blending sixty percent from the start of the gradient axis.



In every example so far, we've blended our colours gradually along a gradient's axis, but sometimes our designs mean that we need to change abruptly from one colour to another. CSS gradients make this trivial.¹ To add a sudden change in colour, simply give two colours the same colour stop value; in the next example, forty percent:



Colin Keany's Blend is a beautifully designed and very useful online tool¹ for creating gradients. Choose two colours from a selection of palettes and grab code that's ready to use.

Linear gradients see some action

It's time for gradients to see some action and we'll make this happen by creating the kind of "We'll be right back" sticky note that you might see on a hardboiled detective's door. Of course, you could use a note like this on a website's holding page. Our markup is hardboiled, just a lonesome `article` element containing a semantic heading and a list:

```
<article>  
  <h1>Back soon!</h1>  
  <ul>  
    <li><del>Gone for smokes</del></li>  
    <li><del>Getting booze</del></li>  
    <li>On a job (yeah, really)</li>  
  </ul>  
</article>
```

We'll start by giving our `article` some dimensions, a little padding and a solid background colour that people using browsers incapable of rendering gradients will see:

¹ colinkeany.com/blend

```
article {
width : 280px;
height : 280px;
padding : 22px;
background-color : #fed46e;
box-sizing : border-box;
text-align : center; }
```



Now to make our sticky note more realistic by adding a diagonal gradient with two colours that will blend towards to the top-right corner of the note, with a colour stop at 60%:

```
article {
background-image : linear-gradient(
to top right,
#fed46e 60%,
#bf9f53); }
```



Browsers that have implemented CSS gradients, with or without a vendor-specific prefix, will render them. Those that aren't capable will render the solid background colour we specified earlier.

Finally, to give a greater feeling of depth, we'll add a subtle shadow to our note:

```
article {
box-shadow : 0 2px 5px
rgba(0,0,0,.5); }
```



Radial gradients

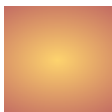
In the first edition of *Hardboiled Web Design*, I kept this section about radial gradients “intentionally brief” as at that point browser vendors still disagreed about the syntax for writing them. I wrote:

“*To watch every punch and counter-punch as these standards develop, follow the CSS Working Group on Twitter or keep up with the minutes of their meetings on their blog. Pretty, it ain’t.*”

Fortunately, those battles are now far behind us and all contemporary browsers now fully support the W3C standard for all types of gradients.

Defining a gradient type

Just like linear gradients, radial gradients are values we can use on the `background-image` property. We’ll keep the same two colours from our linear examples, but specify the gradient type as radial this time:



```
div {  
  background-image : radial-gradient(  
    #fed46e,  
    #ba5c61); }
```

With this simplest of radial gradients, the first colour will blend into the second from the centre of the element to its furthest edge. This means that unless an element’s height and width are the same, the gradient will be an `ellipse`, the default shape for a radial gradient.

When we need our radial gradients to be circles, we can override the default elliptical shape by adding the `circle` keyword to our declaration, separated from our colours by a comma:



```
div {  
  background-image : radial-gradient(  
    circle,  
    #fed46e,  
    #ba5c61); }
```

Look closely at that last example and you should notice that the gradient circle extends to the farthest edge of the element, meaning that we see an incomplete circle. When our designs mean that we need the gradient circle to be fully enclosed with the element — in effect stopping at its closest side — we can make that happen by adding the `closest-side` keyword along with `circle`:



```
div {  
  background-image : radial-gradient(  
    circle closest-side,  
    #fed46e,  
    #ba5c61); }
```

Of course, there are other keywords we can use to vary which side or even corner we'd like our gradient to end.

This circle ends at the closest corner from the centre:

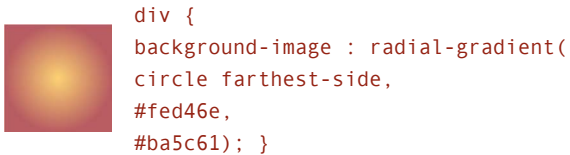


```
div {  
  background-image : radial-gradient(  
    circle closest-corner,  
    #fed46e,  
    #ba5c61); }
```


This one at the farthest corner:



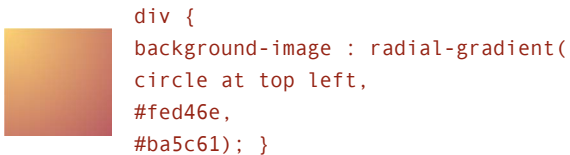
And this the farthest side:



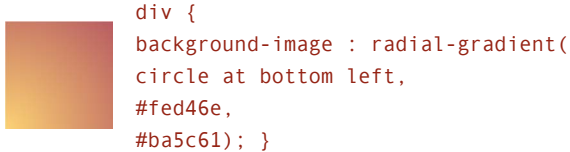
Changing the gradient origin

By default, radial gradients start at the centre of an element's background and blend their colours outward. I can think of many occasions where we might need to change that default and we're able to do exactly that using the `at` keyword, followed by either a position or some other value.

This gradient starts in the top-left of an element's background:



While this one starts at the bottom-left:



We can start gradients in the top-right, too:

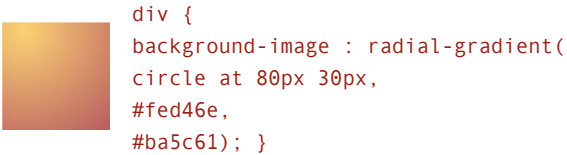


And, of course, in the bottom-right:

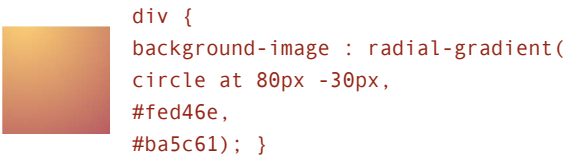


If you crave even more creative control over your gradients, you're really in luck because in addition to those `at` keywords, we can precisely control a gradient's origin position using CSS units, including pixels and percentages — ideal when making responsive web designs.

Let's start by positioning the centre of our next gradient eighty pixels from the left and thirty pixels from the top of our element:

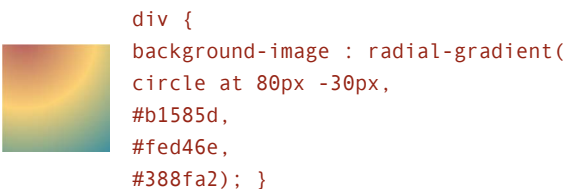


If you'd like the centre of your gradient outside of the element itself, you can even use negative numbers. In this example, the centre is thirty pixels outside the top of the element:

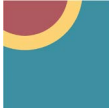


Adding colour stops

As with linear gradients, simple radial gradients are created from two colours, but our designs will often require more complex gradients that include one or more colour stops. Next, we'll add a third colour to that last gradient:



As we haven't yet specified the positions where we'd like these colours to blend, they'll blend evenly across the gradient. When we'd like precise control, we can introduce colour stops at the positions where we'd like colours to start blending.

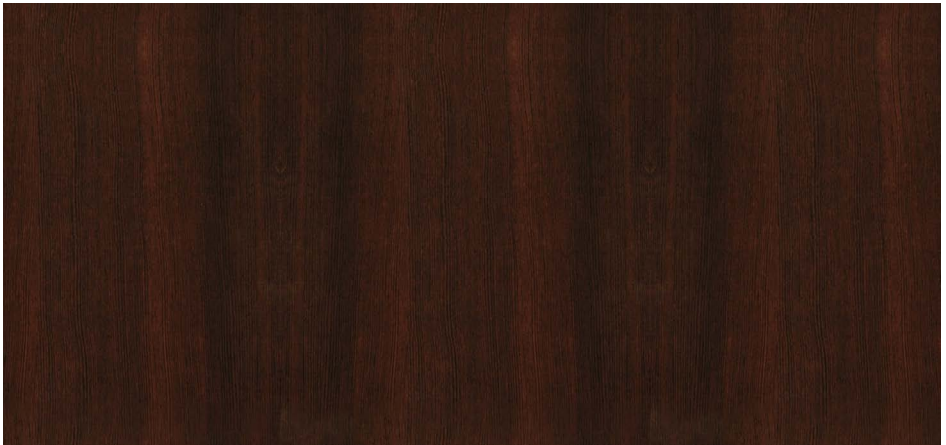


```
div {  
  background-image : radial-gradient(  
    circle at 80px -30px,  
    #b1585d 30%,  
    #fed46e 30%,  
    #fed46e 40%,  
    #388fa2 40%); }
```

Radial gradients in the limelight

It's time to put radial gradients in the limelight by combining them with RGBa to shine a spotlight on the door of the 'Get Hardboiled' office. First, let's style the door. We'll apply a dark background colour and a wood panel background image:

```
.hb-about {  
  background-color : #332115;  
  background-image : url(about-wood.jpg);  
  background-position : 50% 50%;  
  min-height : 100vh; }
```



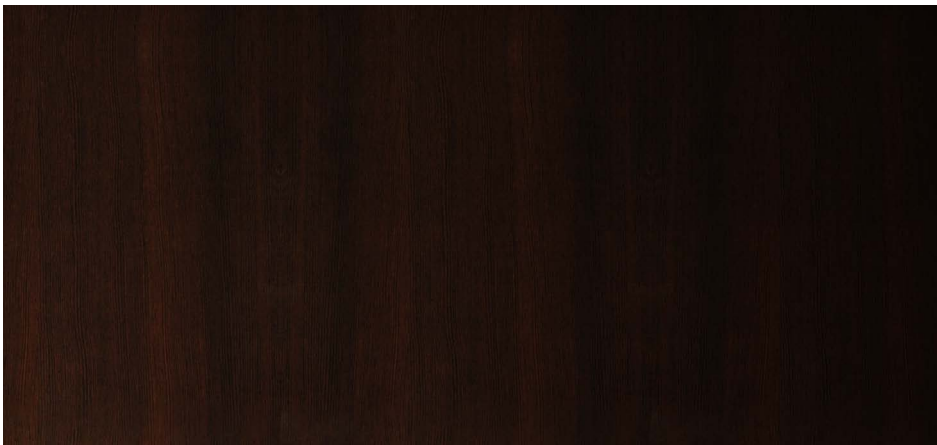
This smart wooden panelling will give a good impression to all visitors to the 'Get Hardboiled' office. We'll welcome even those who are using a less capable browser and forget to wipe their feet.

Because CSS gradients use the `background-image` property, we can use them in multiple backgrounds, including bitmap background images or other CSS gradients. First, we'll add a radial gradient to our `background-image`, and because this comes first in the declaration, it will appear over the door's wooden pattern:

```
.hb-about {  
background-image : url(about-wood.jpg);  
background-position : 50% 50%; }
```

Now add `background-position` and `background-repeat` values for the gradient, separating them with a comma from those styling the wooden pattern image:

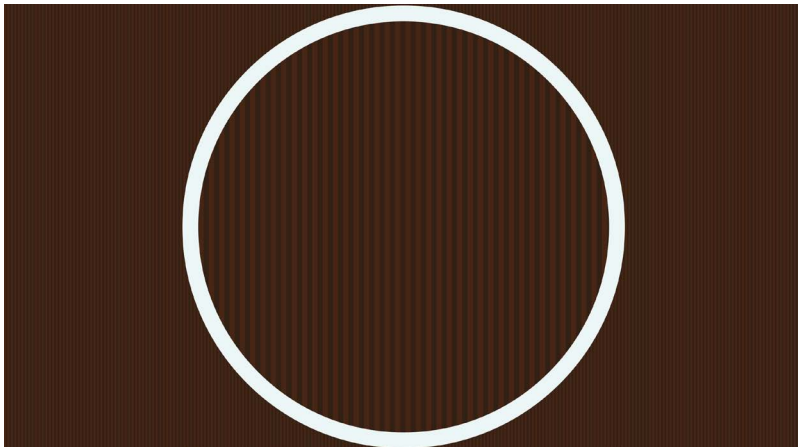
```
.hb-about {  
background-image :  
radial-gradient(  
circle at bottom left,  
transparent,  
rgba(0,0,0,.8)),  
url(about-wood.jpg);  
background-position : 0 100%, 50% 50%;  
background-repeat : no-repeat, repeat; }
```



Now our door looks ready to kick down, but before we put the boot in we need to ask ourselves whether our design is hardboiled enough. Although the bitmap wood grain image we've used optimises down to only 50Kb, that still means an extra download and HTTP request when someone downloads that image.

Hardboiled CSS is all about making the most from very little, so let's first replace that bitmap using a partly transparent **linear-gradient** combined with the **background-size** property.

```
.hb-about {  
  background-image :  
  linear-gradient(  
    90deg,  
    #472615 50%,  
    transparent 50%);  
  background-size : 6px; }
```



Our gradient runs top to bottom from a wooden brown colour to transparent. Because both colours have the same fifty percent colour stop, they meet in a sharp line no matter what the background's size:

This gradient creates vertical stripes that look like the edges of stained plywood, but the background doesn't resemble our wood grain image just yet, so let's use a different type of gradient, a repeating one.

Repeating gradients

So far we've learned about linear and radial gradients that both blend across the entire size of an element. But what if we want a gradient to repeat across an element's background, to create a pattern made of nothing more than a few simple lines of CSS? Well, we can do just that with a repeating gradient.²

There are two types of repeating gradient, `repeating-linear-gradient` and `repeating-radial-gradient`. Here's how to specify a repeating gradient that will be linear:

```
div {background-image : repeating-linear-gradient(); }
```

Whereas a radial gradient that repeats looks like this:

```
div {background-image : repeating-radial-gradient(); }
```

We'll start writing a repeating linear gradient made up of two colours from the 'Get Hardboiled' colour palette. As we want our gradient to run vertically, we'll set the angle of the gradient to ninety degrees.

```
div {background-image : repeating-linear-gradient(90deg); }
```

² On CSS Tricks, Ana Tudor answers the question, "Why Do We Have repeating-linear-gradient Anyway?" Her explanation includes some devilishly clever examples of how to use gradients to create effects that you wouldn't think possible using CSS: css-tricks.com/why-do-we-have-repeating-linear-gradient-anyway

Now add our two ‘Get Hardboiled’ brand colours along with colour stops that create hard edges between the colour blends:



That last colour stop value performs a very important role as it effectively controls the size of the gradient background we’re going to repeat. Change that in proportion to our colour stop values and we can alter the look of our background dramatically. This next repeating gradient has a tight pattern set at forty-five degrees:



Now let’s switch that gradient to `-45deg` and open out the repeating pattern:



So far we've looked at repeating linear gradients, but repeating gradients can include circles or ellipses too. Our next gradient is a circle whose origin is at the centre-bottom of the element:



```
div {  
  background-image : repeating-radial-gradient(  
    circle at 50% 100%,  
    #fed46e,  
    #ba5c61 20px); }
```

That final colour stop again controls the size of the repeating background, so let's increase it and change the position of the gradient's origin to centre-top:



```
div {  
  background-image : repeating-radial-gradient(  
    circle at 50% 0,  
    #fed46e,  
    #ba5c61 40px); }
```

Finally, we'll change the circle to an **ellipse** and position the gradient's origin to right-centre:



```
div {  
  background-image : repeating-radial-gradient(  
    ellipse at 100% 50%,  
    #fed46e,  
    #ba5c61 40px); }
```

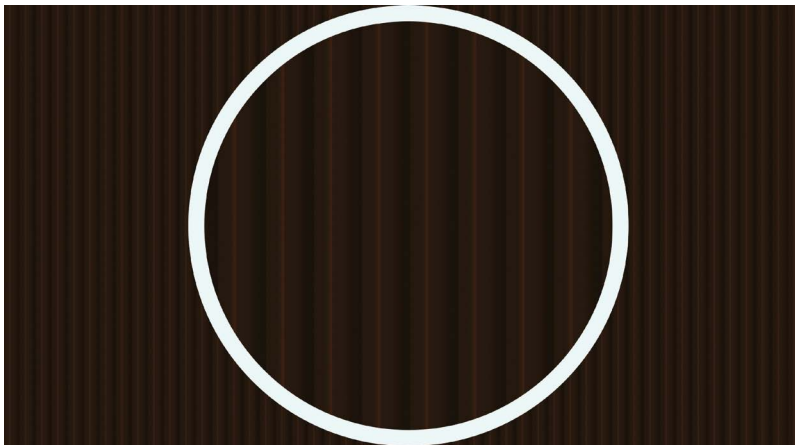
Lea Verou collects some inspiring examples of patterns made from CSS gradients, with code to copy, paste and adapt. You can even submit your own gradient patterns to the gallery.³

Repeating gradients see some action

Let's take our knowledge of repeating gradients back to the 'Get Hardboiled' office door. Our last attempt using a straight linear gradient was close, but no pack of Luckies. This time we'll use a repeating gradient to replace the bitmap wood grain image. We'll mix up six colours with several different colour stops to create a more natural-looking result:

```
.hb-about {  
background-image :  
repeating-linear-gradient(  
90deg,  
#24170b,  
#24170b 6px,  
#291A0b 8px,  
#3e2010 10px,  
#281A11 11px,  
#281A11 12px,  
#25170a 18px,  
#180f06 24px,  
#180f05 24px,  
#180f05 28px);  
}
```

Now our office door just got more hardboiled. Although on close



inspection it doesn't look exactly like it's made of hardwood, when someone views our design on a smaller screen they may not notice the difference, but they will appreciate the smaller size of their download. This leaves us free to reintroduce the background image at a larger breakpoint when we absolutely will need that extra level of detail:

```
@media (min-width: 48rem) {  
  .hb-about {  
    background-image :  
    radial-gradient(  
    circle at bottom left,  
    transparent,  
    rgba(0,0,0,.8)),  
    url(about-wood.jpg);  
    background-position : 0 100%, 50% 50%;  
    background-repeat : no-repeat, repeat; }  
}
```

Breaking it up

The flat design aesthetic may be all the rage today, but I'm old and ugly enough to know that fashions in web design change as fast as hipsters change coffee shops. Mark my words, gradients will be back and — whether you like yours linear, radial, repeating or with multiple background images — you'll need to know how to handle them.



A stylized illustration on a dark red background. On the left, a black fist is clenched, with a white flame-like shape emerging from it. The flame has several pointed, wavy edges and extends towards the bottom right of the page. The overall style is graphic and bold.

MORE HARDBOILED CSS

You now know that hardboiled CSS helps us to leave behind some of the ways of working we've become accustomed to, making our websites lighter, faster and more responsive as a result. Now it's time to turn up the heat again.

In **More Hardboiled CSS**, you'll learn about the latest background blends and CSS filters, how to translate, scale, rotate and skew elements using CSS transforms in two and three dimensions. You'll find out how to make state changes smoother with a host of CSS transitions, and finish off by discovering how to add columns to your layout without a extra division in sight. Now that will be hardboiled.

No. 17

Background blends and filters

The rapidly increasing pace of change in what we make is reflected not only in the tools we use but also the speed in which new tools and technologies are being developed. Nowhere is this truer than in the time it takes for emerging CSS properties to be implemented across a range of contemporary browsers. In the past, designers and developers waited year after year for the simplest CSS technologies, such as `border-radius`, to be implemented reliably across browsers; today, new properties go from idea to design to implementation – and even to specification – in a fraction of the time.

In almost every way, this change in the pace of development is a good thing for designers and developers, businesses and brands, and the internet in general. It means there's a likelihood that even recent browser versions may not be in step with developments, but we can't slow down the pace of progress. Instead, we should push the web forward by using emerging technologies, not just on experimental projects, but in the work we're paid to do every day.

Peter-Paul Koch (PPK) wrote an article about wanting a year-long moratorium on new browser features to “free us from the churn of ever more features and ever more tools.” That article stirred considerable controversy and some intelligent and respectful replies from Jake Archibald² and Bruce Lawson.³

¹ quirksmode.org/blog/archives/2015/07/stop_pushing_th.html

² jakearchibald.com/2015/if-we-stand-still-we-go-backwards

³ dev.opera.com/articles/on-a-moratorium-on-new-browser-features

CSS shaders

Much of the innovation in CSS over the last decade has been instigated by browser makers, but in the past few years Adobe — maker of Illustrator and Photoshop and owner of Typekit — has inspired some of the most interesting graphical effects in CSS. In 2011, Adobe announced what it called CSS shaders,⁴ advanced visual effects for the web. Reaction to the proposal to bring to browsers the type of filters we use in Photoshop was universally well received, and in late 2014 Adobe's filters were included in the working draft of the W3C's Filter Effects module.⁵

The rate of adoption for blending modes and filters has been astonishing, with every major browser now supporting CSS filters in one form or another. Chrome, Opera and Safari all require the `-webkit` vendor-specific prefix, and Microsoft Edge has supported filters under the “Enable CSS filter property” flag.

CSS filters

Not to be confused with Microsoft's proprietary filters from the dark days of the browser wars, CSS filters are powerful new tools that make available within a web browser some of what's been possible in graphics and photography software for some time. The `filter` property puts effects such as blurring, image adjustment and even full drop-shadows into our browsers.

`blur`
`drop-shadow`
`invert`
`sepia`

`brightness`
`grayscale`
`opacity`

`contrast`
`hue-rotate`
`saturate`

Microsoft's proprietary filters were implemented from Internet Explorer version 4, all the way to version 8. Microsoft is now adopting the standard `filter` property in the Edge browser.

⁴ adobe.com/devnet/archive/html5/articles/css-shaders.html

⁵ w3.org/TR/filter-effects/

As you might expect from technology that has its roots in photographic retouching software, these properties are mostly used to manipulate images, although it's possible to use them on any element and even apply them to an entire page if you feel so inclined. I hope that in the near future we'll be able to apply filters to backgrounds and borders too.

The syntax for filters is simple: the `filter` property followed by a filter type, such as `blur`, then its various values inside parentheses:

```
.filter {  
  filter : blur(5px); }
```

Filters are easy to use and because of their relative novelty as properties in CSS, still fun to experiment and play with. We'll work through a range of filter types, learn about the values they accept and look at the effects they create. Let's start with blurring an element.

Blur

To apply a Gaussian blur to an element using a filter, we need only specify `blur` as the filter type, then a value that represents the radius of the blur. To demonstrate this we'll apply `blur` to the illustrated banner background on the Stuff & Nonsense website:

```
.filter {  
  filter : blur(5px); }
```




Blurring a banner's background division on the Stuff & Nonsense website.⁶

Blur filters accept any CSS unit as their radius, so we can use pixels, em, rem and even cm if you're feeling adventurous. The higher the number, the larger the radius we apply and the stronger the filter effect. The unit we can't use is a percentage. If we enter an invalid value, the browser will apply `none` as a value instead.

Blur is one of the first to come to mind when we think about filters, and applying them using CSS is now very simple. What's often still difficult, though, is judging the correct amount of blur to achieve a natural-looking result.

⁶ Illustration by Josh Cleland (joshcleland.com)

brightness and **contrast** accept numbers as well as percentages.

1 produces the same visual effect as **100%**, **2** the same effect as **200%**, and so on. This handy shortcut value works for any filter that accepts percentages.

Brightness and contrast

As you might expect, **brightness** increases or decreases the lightness of any element that it's applied to. While you might at first think that **brightness** can only be applied to photographs or other images, you can apply it to any element, everything from text elements to entire sections of a page. In this next example, we'll reduce the brightness of another illustrated Stuff & Nonsense header by fifty percent, as if we're dimming the lights:

```
.filter {  
  filter : brightness(50%); }
```



The **brightness** filter accepts percentage values, with **100%** leaving an element with its original look. Values between **0%** and **100%** turn down that **brightness** towards black, while values over 100% turn it up, and up and up, until the element looks burned out.

The **sepia** filter also replaces colour with shades of grey, but also adds a warm tone that's reminiscent of old photographs. Knowing this, I predict the next trend will be websites that look like Victorian photo albums.

Using identical values to **brightness**, the **contrast** filter can also be applied to any element. To compare it to **brightness**, we'll change the contrast of that same illustrated banner.

```
.filter {  
  filter : contrast(50%); }
```



contrast accepts the same percentages. **0%** brings highlight and shadow contrast together to create a flat grey; between **0%** and **100%**, contrast is increased; at **100%** the element appears unchanged. Increasing contrast over **100%** creates some very interesting results as the previous montage illustrates.

Grayscale and saturate

A **grayscale** filter progressively replaces colour with shades of grey. Values start at **0%** which leaves the element unchanged, all the way to **100%**.

When I'm writing HTML and CSS to develop the layout for a new design, I often apply a **100% grayscale** filter so I can concentrate on layout, especially in relation to typography, without being distracted by colour.

```
.filter {  
  filter : grayscale(100%); }
```



On the other hand, the **saturate** filter leaves the mix of colour values intact and changes the amount of them all. The values for **saturate** are different to the **grayscale** filter: before, **0%** left an element unaltered; in **saturate**, **0%** makes an element appear completely devoid of colour.

```
.filter {  
  filter : saturate(25%); }
```



100% saturation produces the initial look for any element, whereas values over **100%** oversaturate it.

Invert

The invert filter inverts any colour present in an element by the amount we specify. Use a value of **0%** and the element will look unchanged.

```
.filter {  
  filter : invert(100%); }
```



Progressively increasing that percentage will invert the colours by ever greater amounts until the **100%** maximum is reached, with completely inverted colours.

Opacity

Now you may be wondering why we would use a relatively new **opacity** filter when there's been an **opacity** property available to us for years. It's true that this filter works in exactly the same way as the original. **0%** makes an element fully transparent; **100%** makes it fully opaque. With both we can use numbers instead of percentages with **.75** giving the same result as **75%**.

```
.filter {  
  filter : opacity(.75); }
```


So what are the benefits of this new filter over the previous property? There are two that spring to mind. The first, as you'll learn in a moment, is that the `opacity` filter can be combined with other filters for interesting creative effects. Second, some later browsers use hardware acceleration on CSS filters to improve the speed of page rendering.

Combining multiple filters

For even more interesting creative effects, we can combine two or more filters. The syntax for these combinations always catches me off guard. If, like me, you expect a list of filters to be separated by commas, you'd be wrong. In the next example we'll combine filters for higher `brightness` and lower `contrast` with undersaturation to give an aged photograph feel. The syntax for combining filters looks like this:

```
.filter {  
  filter : brightness(1.25) contrast(.75) saturate(40%); }
```

The order in which we write our filters matters and it might help you to read a string of filters from left to right. In that previous example, the element's brightness is increased to `125%` before having its contrast reduced to `75%`. Finally, the result of combining those two filters is reduced to `40%` saturation.

One word of caution when we use filter effects to add a little extra visual interest to `:active`, `:focus` or `:hover` states, we must repeat every property across each state. For example, we might want to adjust the saturation value of the previous example on hover, but leave brightness and contrast looking the same.

```
.filter {  
  filter : brightness(1.25) contrast(.75) saturate(40%); }  
  
.filter:hover {  
  filter : saturate(10%); }
```

Watch out! The declaration we applied to `:hover` removed both `brightness` and `contrast` from the element in that state. To maintain all filter properties across states, we must repeat our values across all of them.

Drop-shadow vs box-shadow

Once again you may be wondering why we have a relatively new `drop-shadow` filter and what the differences might be between it and the older CSS `box-shadow` property. You may be even more confused when you find out that both take the same arguments: a horizontal (x) offset, a vertical (y) offset, a blur radius, a spread radius and a shadow colour value. Here's the syntax for the newer `drop-shadow` filter:

```
.filter {  
  filter : drop-shadow(5px 5px 5px rgba(0,0,0,.5)); }
```

It's unlikely you'll see the difference between `drop-shadow` and `box-shadow` until you apply them to an image that contains alpha transparency. When an image has an alpha value, the `drop-shadow` filter detects it and applies its shadow inside the image space, just as it would if we were adding a drop-shadow in Adobe Photoshop or Sketch.



The effects of **drop-shadow** (above) and **box-shadow** (below).



In contrast, the **box-shadow** property only recognises the outer edges of an image and applies its shadow to them.

Let's see how those horizontal and vertical offsets affect the result of our **drop-shadow** filter as we move left to right and increase the offsets.



Increasing offsets.

Although a drop-shadow's horizontal and vertical offsets are required, other shadow values aren't. Next, we'll increase the optional blur radius from small to large. When we don't specify a blur radius, the shadow will be hard and its edges sharp.



Adjusting blur radii.

Having observed the standards development process for many years and been frustrated by the slow pace of change over those years, I'm both amazed and excited that new technologies such as CSS filters are now finding their way into web browsers so rapidly. This is great news for designers and developers, especially as we continue to embrace responsive web design.

The rise of mobile and the associated necessary focus on performance have made it clear that we need to reduce the number of requests to a server and the weight of downloaded assets. The more visual aspects of a design we can move from images into CSS, the better our sites will perform. CSS filters and their related blending modes are a strong step towards doing just that.

Background blends

You might not realise it looking at today's generally flat website and application designs, but CSS is capable of giving us depth and subtlety. You could be forgiven for thinking that the elements we add to our pages live on one plane, but in fact elements lie on top of one another, overlapping to form a stacking order. There's even a stacking order within elements' multiple background images too. Personally, I hope the current flat trend and rather soulless period of website will pass and we'll go back to making designs that demonstrate such depth.

If you're familiar with photo retouching or graphic design software such as Adobe Photoshop, Affinity Photo or Pixelmator, you'll be familiar with blend modes.⁸ In those applications, blend modes give us the ability to blend or merge separate parts of an image to create a wide variety of different effects.⁹

While support for CSS filters in browsers has come quickly and for the most part completely, CSS blend modes are taking a little longer to achieve solid support. Microsoft's Edge browser hasn't yet implemented blend modes, and Apple's Safari for both iOS and Mac OS X are missing support for **hue**, **saturate**, **color**, and **luminosity** blend modes.

⁸ Sara Soueidan wrote a fabulous explanation of compositing and blending that includes a useful introduction to compositing operations that determine which portions of source and destination elements will be affected by blends: smashed.by/blendcss

⁹ Speaking of Sara Soueidan, her CSS Blender demonstrates the **background-blend-mode** property by uploading an image and seeing the results of different blend modes immediately in the browser: sarasoueidan.com/demos/css-blender/

The same is now possible natively in browsers, thanks to two types of CSS blend mode: `background-blend-mode` and `mix-blend-mode`. We'll look at each one, starting with `background-blend-mode`, a property designed to blend together the background properties of a single element.

Background-blend

In the CSS box model, an element's `background-color` lies behind any `background-image`, and borders lie on top of both of them. As you might have guessed from its name, the first of our two blend modes controls how a single element's `background-color` and images blend together. When an element has just a single `background-image`, `background-blend-mode` controls how that image blends with the `background-color` behind it.

To apply a background blend, use the `background-blend-mode` property followed by the blend type value, in this instance `lighten`:

```
.blend {  
  background-color : #8c4549;  
  background-image : url(blend-01.jpg);  
  background-blend-mode : lighten; }
```



Left: `normal`. Right: `lighten`.

Blend modes

CSS has sixteen blend types: **normal** (no blending applied); **color**, **color-dodge** and **color-burn**; **difference**, **exclusion**, **hue**, **luminosity**, **multiply**, **overlay**, **saturate** and **screen**; **lighten** and **darken**; **hard-light** and **soft-light**.

Each will give a different visual result, and the look of the source **background-image** will be changed by the blend type¹⁰ we choose and the **background-color** of the element.



multiply: multiplies colours in the source **background-image** with colours in the destination **background-color**. The effect is almost always a darker source **background-image**.



exclusion: Measures the brightness in both source and destination, and subtracts the colour with the greater brightness from the other. Exclusion is similar to **difference**, but results in a lower contrast effect.



lighten: Lightens the source by measuring both source and destination and choosing the lighter of the two. **darken** is the mathematical opposite and has, unsurprisingly, the opposite effect.

¹⁰ It may or may not be the “ultimate” but Pye’s guide to Photoshop blend modes is certainly comprehensive and includes very useful visual examples:
srlounge.com/school/photoshop-blend-modes/



overlay: A complex blend mode, **over**lay either multiplies or screens colours depending on the destination colour. Lighter colours get lighter, darker colours become darker.



saturation: Produces a result with the saturation of the source colour and the hue and luminosity destination colour. **sat**urate is similar to **hue**, but uses alternate properties.



color-dodge: brightens the destination colour to reflect the source colour.

Blending multiple background images

When we give an element multiple background images, we can apply different blend modes to each individual image. Each image blends with the images below it in the list and finally with the element's **background-color**.

```
.blend {  
  background-color : #8c4549;  
  background-image : url(blend-01.jpg), url(blend-02.jpg);  
  background-blend-mode : lighten, multiply; }
```

In this example, the second *blend-02.jpg* image blends with the **background-color**, using the **multiply** mode; then the first *blend-01.jpg* image blends with the second image and then the **background-color** using the **lighten** mode.

There's certainly a lot more to using **background-blend-mode** than blending between a single background image and a colour. I hope we'll soon see designers blending multiple background images to create designs that are rich and full of depth, so I'm excited about the creative possibilities that this new property makes possible.

Mixing image types

In our next example we'll blend a page background colour with both a radial and then a repeating linear gradient to create a different lighting effect on our 'Get Hardboiled' detective's desktop.



The properties we blend need not even be images in the traditional sense; they can be any type of gradient that we've generated using CSS.

Let's first apply a radial gradient to the `body` element of our page. This gradient starts transparent, then transitions to a red colour to simulate neon light falling across the desk:

```
.hb-bg-light {  
background-color : #332115;  
radial-gradient(circle at bottom left, transparent, #f00); }
```

To ensure that the body extends to the full height of the viewport, no matter how much or how little content it contains, we'll set the `min-height` to `100%` of the viewport:

```
.hb-bg-light {  
min-height : 100vh; }
```

Next we'll add a repeating linear gradient to create texture in the design:

```
.hb-bg-light {  
background-image :  
radial-gradient(circle at bottom left, transparent, #f00),  
  
repeating-linear-gradient(  
90deg,  
#24170b,  
#24170b 6px,  
#291A0b 8px,  
#3e2010 10px,  
#281A11 11px,  
#281A11 12px,  
#25170a 18px,  
#180f06 24px,  
#180f05 24px,  
#180f05 28px); }
```




A repeating linear gradient helps to add texture and visual interest to the design.

Now it's time to blend those gradient background images with each other and with the background of the page itself. We'll choose two blend modes. The first, **color**, will blend the radial gradient spotlight effect; the second, **screen**, will blend the repeating gradient texture of the desktop with the **background-color** behind:

```
.hb-bg-light {
background-blend-mode : color, screen; }
```

Because the result of blend modes depends so heavily on the colours we're blending, finding the mode that gives us the precise effect we're looking for can sometimes be tricky and other times require a little good old-fashioned trial and error. Experiment by switching blend modes and by changing the background colour to see what these properties can create. Not always knowing the outcome can often give us an unexpected, but pleasant, result.



Not one, but two background modes that together blend the gradient texture of the desktop.

Testing support for background-blend-mode

Not all browsers support `background-blend-mode`, and while this shouldn't necessarily deter a hardboiled designer, there may be some cases where we need to consider using alternatives for browsers with no or only partial support.

Modernizr's tests and resulting class attribute values — so that we can write specific selectors with alternative styles — are the obvious choice for dealing with Microsoft Internet Explorer and Edge. When a browser supports `background-blend-mode`, Modernizr appends a `backgroundblendmode` class to the `html` element:

```
.backgroundblendmode .blend {
background-color : #8c4549;
background-image : url(blend-01.jpg), url(blend-02.jpg);
background-blend-mode : lighten, multiply; }
```

When there's no support, Modernizr appends a `no-backgroundblendmode` class so that we might serve a different image to compensate:

```
.no-backgroundblendmode .blend {  
background-image : url(blend-alt.jpg); }
```

Useful though Modernizr is, it can also be a blunt instrument as it tests only that a browser supports a CSS property and not individual values of that property. For example, Apple's Safari has partial support for `background-blend-mode` as it's missing the `color`, `hue`, `luminosity` and `saturate` blend modes. Apple's browser has implemented the `@supports` feature query that we learned about earlier and this at-rule is an ideal solution to the problem of partial support.

We'll chain several feature queries together, first testing for `color`, then `hue`, then `luminosity` and finally `saturate`. We'll use both the `not` operator to target browsers that lack support for these specific pairs of properties and values, and the `or` operator to make sure we cover several possible missing blend types:

```
@supports not (background-blend-mode:color)  
or not (background-blend-mode:hue)  
or not (background-blend-mode:luminosity)  
or not (background-blend-mode:saturate) {  
  
  .blend {  
background-image : url(blend-alt.jpg); }  
}
```

Mix-blend

While `background-blend-mode` enables us to affect the way that one or more background images visually interact with a background colour inside a single element, with another blend mode property we can affect the ways that elements visually interact with one another — and even with the page itself. This new property is called `mix-blend-mode` and it opens up new opportunities to be creative in website and application design.¹¹

The syntax for `mix-blend-mode` is no more complicated than it was for `background-blend-mode`:

```
.blend {  
  background-colour : #a20b30;  
  mix-blend-mode : multiply; }
```

In this example, our element won't blend within itself, but with other elements below it in the stacking order and the page itself.

Mixing blend modes

If an element contains multiple background images, we can either assign the same blend mode to all of them, like this:

```
.blend {  
  background-image : url(blend-01.jpg), url(blend-02.jpg), url(-  
  blend-03.jpg);  
  mix-blend-mode : multiply; }
```

CSSgram is a tiny (and fun) JavaScript library that recreates popular Instagram photo filters using CSS filters and blend modes. Personally, I love how it's now possible to be creative with image filters in browsers and not just in Photoshop¹¹

¹¹ una.im/CSSgram

Or we can specify a different blend type for each individual

`background-image:`

```
.blend {  
background-image : url(blend-01.jpg), url(blend-02.jpg),  
url(blend-03.jpg);  
mix-blend-mode : multiply, screen, luminosity; }
```

In that example, *blend-01.jpg* will use the blend mode `multiply`, *blend-02.jpg* will use `screen`, and so on. We therefore need to pay attention to the order we specify for both background images and their corresponding blend modes.

The `mix-blend-mode` property uses the same blend types as `background-blend-mode` so we can achieve the same blending effects between elements as we have within them.

Breaking it up

In almost every way, the rapidly increasing pace of change in what we make and how we make it is a good thing for designers and developers, businesses and brands, and the internet in general. New technologies like CSS filters and background blends are not only being introduced faster, but they're being implemented in browsers and turned into standards faster too. Now's not a time to kick back — it's a time to use these exciting new tools to make creative work with depth and subtlety, work that's hardboiled.

No. 18

Transforms

Despite our very best efforts, CSS layouts can sometimes be a little strait-laced. CSS even calls its basis for layout a box model. While some CSS layout specifications are slowly making their way through the standardisation process inside the W3C, two-dimensional and three-dimensional transforms can help our designs break out of the box.

Two-dimensional transforms

Two-dimensional CSS transforms are supported in all current browsers, so using them is a real no-brainer. The basic syntax for transforms is simple:

```
transform : transform type;
```

There are a number of ways that we can **transform** an element:

- **translate**: moves an element horizontally and vertically.
- **skew**: distorts an element horizontally and vertically.
- **rotate**: rotates an element.
- **scale**: increases or decreases the size of an element.

Transform: translate

We'll start by moving elements with **translate**. In many respects, this behaves in a similar way to relative positioning, where an element is offset visually but keeps its position in the document's normal flow.

`translate` moves elements on the *x*- and *y*-axes. We can specify how far they move by using pixels, ems or percentages that are relative to the size of the element. For example, a 100 pixel box translated by 150% moves 150 pixels. Percentages can be particularly useful in flexible designs and on dynamic sites where the size of elements changes.

We'll `translate` a 'Get Hardboiled' business h-card 100 pixels to the right using `translateX` and a distance inside parentheses.

```
.h-card {  
transform : translateX(100px); }
```

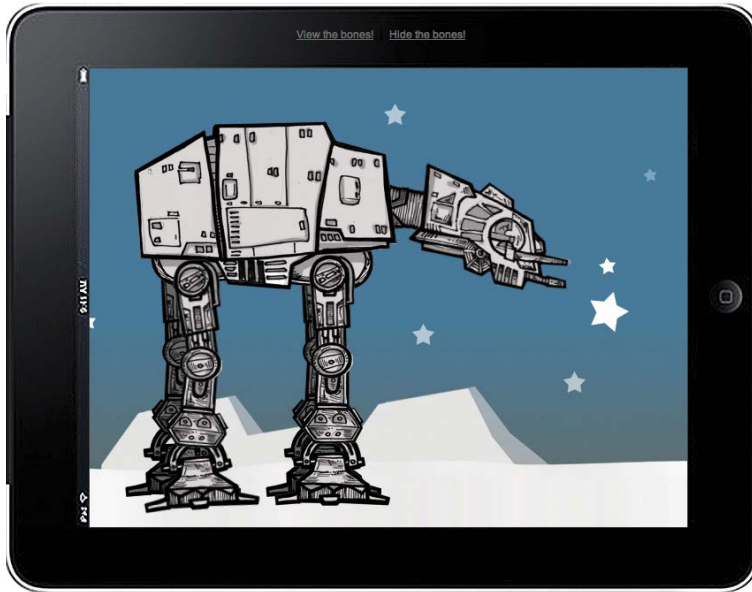
We might also `translate` that card down by 50% with `translateY`:

```
.h-card {  
transform : translateY(50%); }
```

Finally, combine `translateX` and `translateY` into a single `translate` value.

```
.h-card {  
transform : translate(100px 50%); }
```

Like CSS positioning, transformed elements also create a new instance of normal flow and become positioning contexts for any absolutely positioned child elements

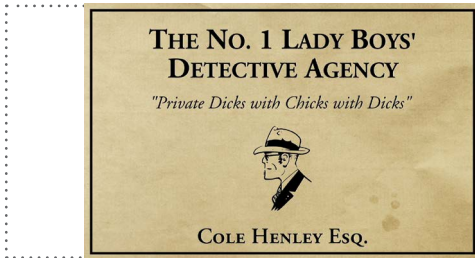


Anthony Calzadilla uses `translate` and `rotate` to create his 'Pure CSS3 AT-AT Walker'¹² from Star Wars. Be honest, admit it – Star Wars was the real reason you wanted to learn CSS.

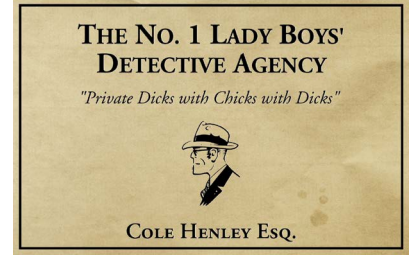
If another element already occupies the space, any translated element will overlap it; it will appear in front of the element if it comes later in the source order, otherwise it will appear behind. As with relative positioning, when we use `translate` the document's normal flow stays unaltered and nothing can flow in to occupy any vacated space.

The best way to learn transforms is to see them in action, so we'll `translate` another business card in several different directions using pixels and percentages. In each example, the dotted box shows the card's original position.

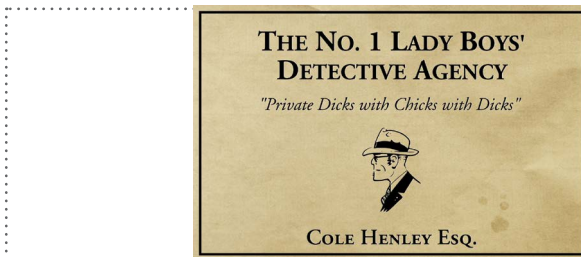
¹² anthonymcalzadilla.com/css3-ATAT/index.html



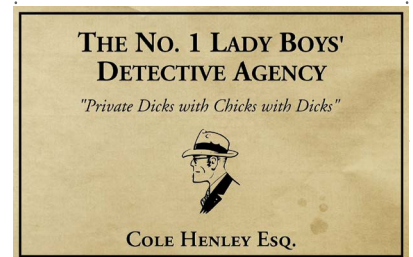
```
.vcard { transform : translateX(50px); }
```



```
.vcard { transform : translateY(50px); }
```



```
.vcard { transform : translateX(50%); }
```



```
.vcard { transform : translateY(50%); }
```

Transform: scale

When we use the `scale` value, we make elements appear larger or smaller. By how much and on which axis is determined by a scaling factor, which can range between `0.99` and `0.01` to make an element smaller, or `1.01` and above to make it larger. A scaling factor of `1` maintains the intrinsic size of an element. Other elements remain blissfully unaware of the new size and so don't reflow around it.

You can `scale` elements along the horizontal or vertical axis, or a combination of the two. Next, we'll scale a card horizontally by 150% using `scaleX`. The scaling factor is in parentheses:

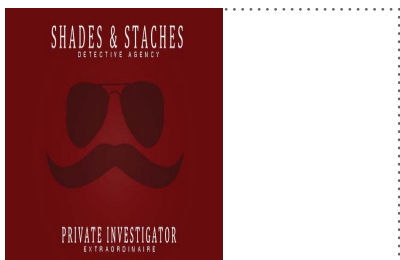
```
.h-card {
transform : scaleX(1.5); }
```

Now use `scaleY` to also increase its height by 50%:

```
.h-card {
transform : scale(1.5, .5); }
```

Look sharp! There's something there that could trip us up if we don't keep our wits about us. Inside those parentheses, the two values must be separated by a comma.

To see `scale` in action, we'll change the size of another business card in several ways. The dotted box is there to remind us of their original sizes:



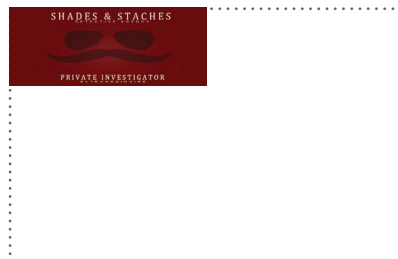
```
.vcard { transform : scaleX(.5); }
```



```
.vcard { transform : scaleY(.5); }
```



```
.vcard { transform : scale(.25, .5); }
```



```
.vcard { transform : scale(.5, .25); }
```

Transform: rotate

We can **rotate** an element between 0 and 360 degrees (clockwise) and even use negative values to **rotate** an element anticlockwise. The syntax is quick to learn. First, declare the **rotate** value, then the angle — in this case forty-five degrees (**45deg**) — inside parentheses:

```
.h-card {  
  transform : rotate(45deg); }
```

When an element is rotated, other elements on a page remain unaware of any change in angle and don't reflow around it. To see **rotate** in action, we'll change the angle of another card by different degrees. The dotted box is there to remind us of the original position:



```
.vcard { transform : rotate(-30deg); }
```

```
.vcard { transform : rotate(30deg); }
```



```
.vcard { transform : rotate(60deg); }
```



```
.vcard { transform : rotate(90deg); }
```

Transform: skew – the twisted thing

skew distorts an element on the horizontal axis, vertical axis or both. The syntax is simple, so to demonstrate, we'll skew a card horizontally by first declaring **skewX**, then the amount, thirty degrees (**30deg**), inside parentheses:

```
.h-card {  
  transform : skewX(30deg); }
```

Now let's combine two axes by also skewing the card vertically by fifteen degrees (**15deg**) using **skewY**. Longhand values look like this:

```
.h-card {  
  transform : skewX(30deg);  
  transform : skewY(15deg); }
```

We can also use the shorthand `skew` property:

```
.h-card {  
  transform : skew(30deg, 15deg); }
```

The best way to learn skews is to see them in action, so we'll put the skews on another business card by skewing it horizontally and vertically, positively and negatively to demonstrate the effects. In each example, the dotted box shows the card's original shape.



```
.vcard { transform : skewY(30deg); }
```



```
.vcard { skew(-15deg, -15deg); }
```

Setting the origin of a transform

Translating (moving), scaling, rotating and skewing are powerful tools for controlling the finer details in a design, but for even greater control we can define the origin of a `transform` on any given element.

Define a `transform-origin` by using either keywords like `top`, `right`, `bottom`, `left` and `center`, or by using pixels, ems or even percentages. Origins normally consist of two values: the first is a point on the horizontal axis; the second is on the vertical. In the next example, we'll `transform` a card around its right, uppermost corner:

```
.h-card {  
  transform-origin : right top; }
```

The declaration below will give us the same results using percentages:

```
.h-card {  
  transform-origin : 100% 0; }
```

When we use just one value, a browser will automatically assume that the second is `center`.

One of the best ways to understand transform origins is to see their effects in action, so in the next set of examples, the card is rotated by minus thirty degrees (`-30deg`) anticlockwise around different origin points. You sussed it: the dotted boxes shows us the card's original position:



```
.vcard {
transform : rotate(-30deg);
transform-origin : 0 0;
}
```



```
.vcard {
transform : rotate(-30deg);
transform-origin : 50% 0;
}
```



```
.vcard {
transform : rotate(-30deg);
transform-origin : 100% 0;
}
```



```
.vcard {
transform : rotate(-30deg);
transform-origin : 0 100%;
}
```

Combining two or more transforms

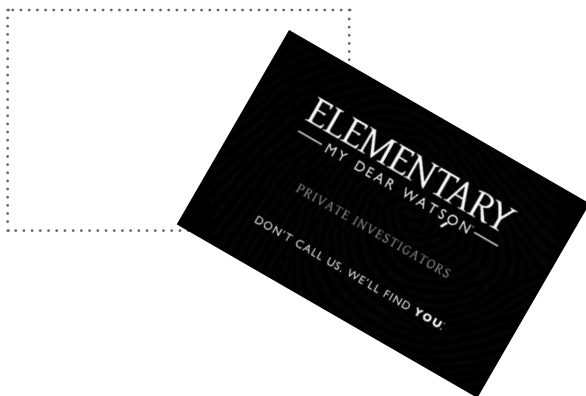
Occasionally a design will require us to set two or more transforms on one element. To set multiple `transform` values, string them together and separate each with a space. In this next example, the card is both rotated by two degrees (`2deg`) and scaled five percent (`1.05`) above its original size:

```
.h-card {  
  transform: rotate(2deg) scale(1.05); }
```

A browser applies these transforms in order, reading from the left. In that last example, it will first `rotate` the card by two degrees clockwise (`2deg`) before increasing its size by five percent (`1.05`). Watch as we apply a series of transforms, each one building on the last.



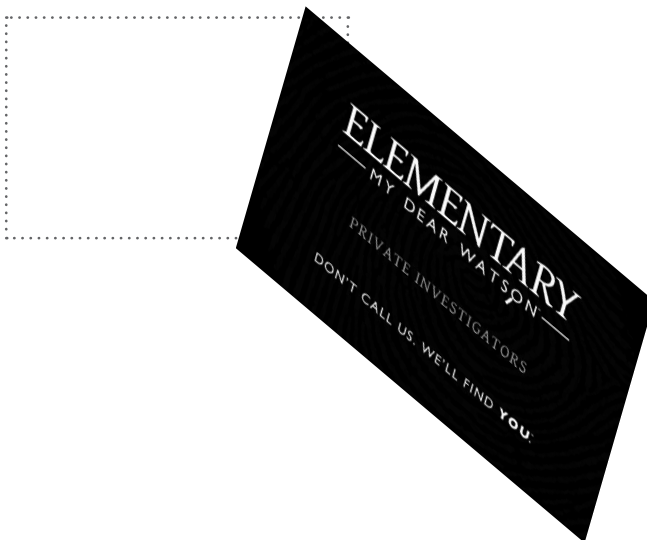
```
.vcard {  
  transform : translate(100px, 50%);  
}
```



```
.vcard {  
  transform : translate(100px, 50%);  
  transform : rotate(30deg);  
}
```




```
.vcard {  
  transform : translate(100px, 50%);  
  transform : rotate(30deg);  
  transform : scale(1.05);  
}
```



```
.vcard {  
  transform : translate(100px, 50%);  
  transform : rotate(30deg);  
  transform : scale(1.05);  
  transform : skew(-15deg, -15deg);  
}
```

2D transforms see some action

It's time to put transforms to work by scattering some 'Get Hardboiled' business cards across the detective's desk. To achieve the off-kilter design, use `rotate` transforms and fine-tune their origins using the `transform-origin` property.

The HTML we'll use is strictly hardboiled and you won't find a single presentational element or attribute no matter how hard you look. There are nine microformat h-cards, each with its own set of values to describe every detective's contact information. You won't even find a unique `id` on any of the cards. Now that's hardboiled.

```
<div class="h-card">
<h3 class="p-org">The No. 1 Detective Agency</h3>
</div>
```

```
<div class="h-card">
<h3 class="p-name p-org">
Shades & Staches Detective Agency</h3>
</div>
```

```
<div class="h-card">
<h3 class="p-name p-org">Command F Detective Services</h3>
</div>
```

```
<div class="h-card">
<h3 class="p-name">The Fat Man</h3>
</div>
```

```
<div class="h-card">
<h3 class="p-name p-org">Hartless Dick</h3>
</div>
```

```
<div class="h-card">
<h3 class="p-name p-org">Nick Jefferies</h3>
</div>
```

```
<div class="h-card">
<h3 class="p-name p-org">Elementary, My Dear Watson</h3>
</div>

<div class="h-card">
<h3 class="p-name p-org">Shoes Clues</h3>
</div>

<div class="h-card">
<h3 class="p-name p-org">Smoke</h3>
</div>
```

Let's start by writing styles that will be common to every h-card. We'll give all cards the same dimensions and apply the `background-size` property to ensure that background images will always scale to fit, no matter how large we make the cards:

```
.h-card {
width : 300px;
height : 195px;
background-position : 50% 50%;
background-repeat : no-repeat;
background-size : 100% 100%; }
```

To crack this design wide open, we'll apply different background images to each — but how? Remember, there wasn't a single presentational `id` or `class` attribute value anywhere in our HTML. It's time to get hardboiled again and select each card with the rarely used `:nth-of-type` pseudo-element selector.

Uncovering :nth-of-type

You've probably used an `:nth-` pseudo-element selector before. Maybe the last time was `:last-child` to remove a border from the final item in list. Perhaps it was adding a border to a paragraph that comes at the start of an article using `:first-child`, like this:

```
p:first-child {  
  padding-bottom : 1.5rem;  
  border-bottom : 1px solid #ebf4f6;  
  font-size : 1rem; }
```

So far, so good. Then some deadbeat goes and drops in another element before the paragraph, maybe a list or a quotation. Those styles? Poof!

`:nth-child` selectors are fine in predictable situations (list items in an unordered list, or rows in a table), but there's a more flexible option when we need to style elements whose position we can't predict. Wouldn't it be better to target an element based on its type and position in the document? That's exactly what an `:nth-of-type` pseudo-element selector does, making it one of CSS's best-kept secrets.

Want to target a first paragraph, no matter where it appears in the document order? Not a problem. How about the thirteenth item in the fourth instance of an unordered list? `:nth-of-type` will help you out there, too. Target any element, wherever it appears, without needing `id` or `class` attributes. Pretty damn powerful and very hardboiled, don't you think?

For an incredibly clever explanation of how to use these selectors to create quantity-aware responsive layouts, read Heydon Pickering's 'Quantity Queries' for CSS.¹³

¹³ alistapart.com/article/quantity-queries-for-css

:nth-of-type arguments

:nth-of-type will accept one of several arguments: keywords like **odd** and **even**, a number or an expression. Sound complicated? Not really. I'll walk you through a few examples.

Imagine you want to add a border under each odd-numbered item in a list (first, third, fifth, seventh, etc.). :nth-of-type makes that easy. You won't need to add classes to your HTML or use a JavaScript hack, just the **odd** keyword:

```
li:nth-of-type(odd) {  
  border-bottom : 1px solid #ebf4f6; }
```

In the next example, an :nth-of-type selector makes the first paragraph in an article bold, no matter what comes before or after it in the document flow:

```
article p:nth-of-type(1) {  
  font-weight : bold; }
```

Expressions are more complicated and we all scratch our heads the first time we encounter them. My tip is read expressions in reverse, from right to left; in the example below, **3n+1** will match the first instance of a table row (**1**) followed by every third row (**3n**) after that:

```
tr:nth-of-type(3n+1) {  
  background-color : #fff; }
```

6n+3 will match the third element, then every sixth one after that.

```
tr:nth-of-type(6n+3) {  
  opacity : .8; }
```

SitePoint published a thorough explanation of expressions. Read it with a whisky and maybe some painkillers (Legal disclaimer: I advise readers not to mix alcohol and drugs with CSS).¹⁴

¹⁴ reference.sitepoint.com/css/understandingnthchildexpressions

Now's a great time to use those `:nth-of-type` pseudo-element selectors to add background images to each card:

```
.h-card:nth-of-type(1) {
  background-image : url(card-01.jpg); }
```

```
.h-card:nth-of-type(2) {
  background-image : url(card-02.jpg); }
```

```
.h-card:nth-of-type(3) {
  background-image : url(card-03.jpg); }
```

```
.h-card:nth-of-type(4) {
  background-image : url(card-04.jpg); }
```

```
.h-card:nth-of-type(5) {
  background-image : url(card-05.jpg); }
```

```
.h-card:nth-of-type(6) {
  background-image : url(card-06.jpg); }
```

```
.h-card:nth-of-type(7) {
  background-image : url(card-07.jpg); }
```

```
.h-card:nth-of-type(8) {
  background-image : url(card-08.jpg); }
```

```
.h-card:nth-of-type(9) {
  background-image : url(card-10.jpg)); }
```

As we only want the background images to show and not the HTML text, indent every element inside those cards to move them off-screen:

```
.h-card * {
  text-indent : -9999px; }
```



On smaller screens, the cards stack into a tidy pile.

Adding transforms

Those cards look sweet, albeit a little stiff. Let's break that design out of the box by applying some **rotate** transforms. We won't apply these transforms to specific cards, though; we'll be reckless and use **:nth-of-type(n)** selectors to give our design a more random look:¹⁵ Let's loosen it up by rotating odd-numbered cards anticlockwise by two degrees (**-2deg**):

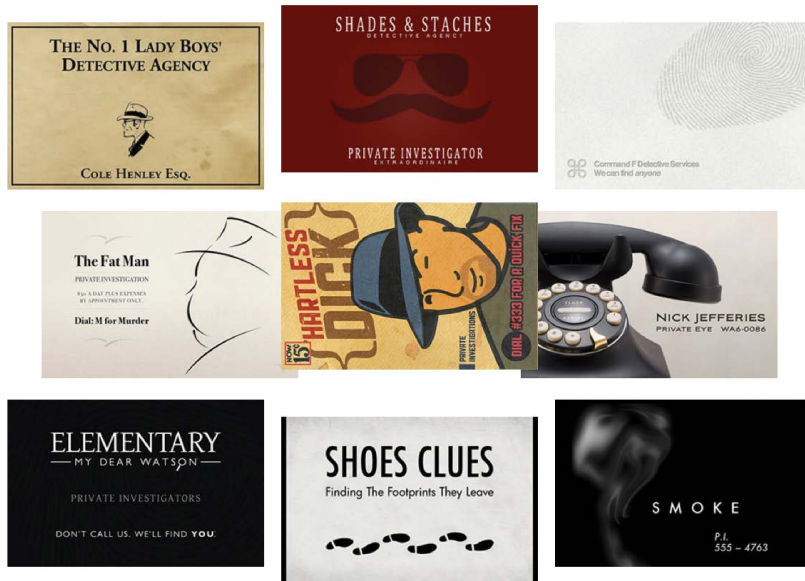
```
.h-card:nth-child(odd) {  
  transform : rotate(-2deg);  
  transform-origin : 0 100%; }
```

Now let's shake things up again, giving every third, fourth and sixth card different **rotate** values and every sixth card **translate** values that will nudge them off-centre:

```
.h-card:nth-child(3n) {  
  transform : rotate(2deg) translateY(-30px); }  
  
.h-card:nth-child(4n) {  
  transform : rotate(2deg);  
  transform-origin : 0 100%; }  
  
.h-card:nth-child(6n) {  
  transform : rotate(-5deg);  
  transform-origin : 0 0; }
```

If you're still confused by **:nth-child**, try CSS Tricks' *:nth Tester* to input expressions and watch as they're applied instantly.¹⁵

¹⁵ css-tricks.com/examples/nth-child-tester



Our stack of cards is getting messed up, thanks to [transform's](#) and pseudo-element selectors.

On smaller screens, those hardboiled business cards fit well and the vertical layout suits the format perfectly. On medium to large screens however, a vertical stack isn't the best use of the available space, so for them we'll use those same pseudo-element selectors, positioning and more transforms to arrange the cards into a grid.

Let's wind back a little and apply absolute positioning to every card. We don't need this positioning applied to smaller screens, so we'll use a media query to apply these styles to medium and larger screens only:

```
@media (min-width: 48rem) {
  .h-card {
    position : absolute; }
}
```


Let's put that positioning to use by giving each card its own **top** and **left** values to form them into a loose grid:

```
@media (min-width: 48rem) {  
  .h-card:nth-of-type(1) {  
    top : 100px;  
    left : 0; }  
  
  .h-card:nth-of-type(2) {  
    top : 80px;  
    left : 320px; }  
  
  .h-card:nth-of-type(3) {  
    top : 100px;  
    left : 640px; }  
  
  .h-card:nth-of-type(4) {  
    top : 320px;  
    left : 40px; }  
  
  .h-card:nth-of-type(5) {  
    top : 270px;  
    left : 570px; }  
  
  .h-card:nth-of-type(6) {  
    top : 320px;  
    left : 600px; }  
  
  .h-card:nth-of-type(7) {  
    top : 540px;  
    left : 0; }  
  
  .h-card:nth-of-type(8) {  
    top : 560px;  
    left : 320px; }  
  
  .h-card:nth-of-type(9) {  
    top : 540px;  
    left : 640px; }  
}
```



By applying `rotate` and `translate` values to a few of the cards, we make the design appear more natural.

I bet you've spotted my deliberate mistake. The fifth card has a portrait orientation whereas all the others are landscape. Fix this by rotating that errant card ninety degrees clockwise (`90deg`). The `transform-origin` rotates the card around its top-left corner:

```
@media (min-width: 48rem) {
  .h-card:nth-of-type(5) {
    transform : rotate(90deg);
    transform-origin : 0 0; }
}
```



The lonesome portrait format card looks best when we rotate it by ninety degrees clockwise (**90deg**) so it overlaps other cards.

Now it's time for a few finishing touches. Add not one, but two RGBa shadows.

```
.h-card {
  box-shadow :
    0 2px 1px rgba(0,0,0,.8),
    0 2px 10px rgba(0,0,0,.5); }
```



Zooming in on the design. Over on the left, soft RGBa shadow adds depth.

Designing alternatives

Let's head back outside the 'Get Hardboiled' office. That note is still stuck on the door, but someone's been by and added another. Remember our HTML? An **article** element? We could make the new note using an **aside** element as it's related to the first note in some way:

```
<article>
  <h1>Back soon!</h1>
  <ul>
    <li><del>Gone for smokes</del></li>
    <li><del>Getting booze</del></li>
    <li>On a job (yeah, really)</li>
  </ul>
</article>
<aside>
  <p>Something on your mind or just want to say hello,
  tweet @gethardboiled</p>
</aside>
```

Let's stick that first note **article** back on the door, this time skewed with a **transform**:

```
article {
  transform : skew(-5deg, -2deg); }
```

Now we'll position the **aside** note and **skew** that too, placing it on top of the first note so that it stands out on the office door:

```
aside {
  position : absolute;
  top : 100px;
  left : 70%;
  z-index : 10;
  transform : skew(5deg, -5deg); }
```



Experiment with skew values as the tiniest change in angles can have a dramatic effect.



With just a few simple lines of CSS, we've transformed these two semantic elements into a design that's totally appropriate for our 'Get Hardboiled' theme.

Three-dimensional transforms

In 2009, Apple announced three-dimensional transforms in Safari running on Mac OS X 10.6 Snow Leopard. These properties position elements in a three-dimensional space to add even more depth to our designs.

Apple's proposals have been adopted by the W3C and, at the time of writing (November 2015), three-dimensional transforms are supported by all contemporary browsers.

Putting it all into perspective

Perspective is key in making elements appear three-dimensional. It takes `transform` properties and places them within a three-dimensional space. To enable `perspective`, we must apply it to a parent element and not to transformed elements themselves. To demonstrate `perspective` we'll build a new example. We won't need special 3D HTML, just a division for each item and a parent `hb-3d` division:

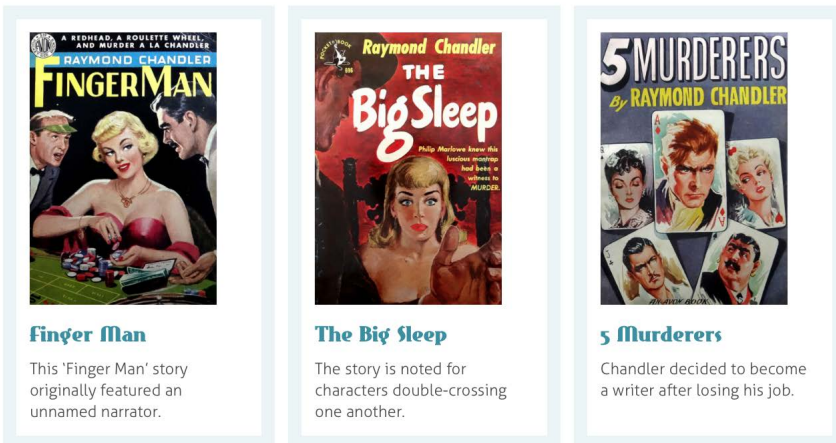
```
<div class="hb-3d">
<div class="item"> [...] </div>
<div class="item"> [...] </div>
<div class="item"> [...] </div>
<div class="item"> [...] </div>
</div>
```

Inside each item, we'll add two further divisions that contain a cover image and its description:

```
<div class="item__img">
  
</div>
<div class="item__description">
  <h3 class="item__header">Finger Man</h3>
  <p>This Finger Man story originally featured an unnamed
narrator.</p>
</div>
```

We'll start by adding styles that will be seen by people using smaller screens — a simple two-dimensional layout that arranges items horizontally. We'll add `display:flex;` to the parent `hb-3d` division, and then `flex:1;` to each item along with margins, padding and a wide blue border:

```
.item {
flex : 1;
margin-right : 10px;
margin-bottom : 1.35rem;
padding : 10px;
border : 10px solid #ebf4f6; }
```



This interface looks neat and tidy, but it won't set your head spinning.

When the browser window gets wide enough for the three-dimensional layout to be appropriate, we'll start setting up those styles. Inside a media query, add a forty-five degree **rotate** transform to all items. We won't be needing those margins, padding and borders on larger screens, so we'll remove those too:

```
@media (min-width: 48rem) {
.item {
transform : rotateY(45deg);
margin : 0;
padding : 0;
border-width : 0; }
}
```



Finger Man

This 'Finger Man' story originally featured an unnamed narrator.



The Big Sleep

The story is noted for characters double-crossing one another.



5 Murderers

Chandler decided to become a writer after losing his job.

When we rotate these items in two dimensions they appear compressed.

```
@media (min-width: 48rem) {
  .hb-3d {
    perspective : 500; }
}
```

Raising and lowering **perspective** has this effect on each item:



Finger Man

This 'Finger Man' story originally featured an unnamed narrator.



The Big Sleep

The story is noted for characters double-crossing one another.



5 Murderers

Chandler decided to become a writer after losing his job.



Finger Man

This 'Finger Man' story originally featured an unnamed narrator.



The Big Sleep

The story is noted for characters double-crossing one another.



5 Murderers

Chandler decided to become a writer after losing his job.

```
.item {
  perspective : 300; }
```

```
.item {
  perspective : 1200; }
```


Changing our viewpoint

When we look at an element that's transformed in three dimensions, our default perspective is in the centre, both horizontally and vertically. We can change this **perspective-origin** using either keywords, pixel or em values, or percentages. In percentage terms, **0 50%** places the perspective viewpoint on the left, halfway down; while **50% 0** places it halfway horizontally and at the very top:

```
@media (min-width: 48rem) {
  .hb-3d {
    perspective-origin : 50% 50%; }
}
```

Watch how a different origin changes our perspective on these items:



Finger Man
This 'Finger Man' story originally featured an unnamed narrator.



The Big Sleep
The story is noted for characters double-crossing one another.



5 Murders
Chandler describes a detective who never sleeps a wink.

```
.item {
  perspective-origin : 0 0; }
```



Finger Man
This 'Finger Man' story originally
featured an unnamed narrator.



Finger Man
This 'Finger Man' story originally
featured an unnamed narrator.



Finger Man
This 'Finger Man' story originally
featured an unnamed narrator.



Finger Man
This 'Finger Man' story originally
featured an unnamed narrator.



Finger Man
This 'Finger Man' story originally
featured an unnamed narrator.



Finger Man
This 'Finger Man' story originally
featured an unnamed narrator.

```
.item {  
  perspective-origin : 0 100%; }
```

```
.item {  
  perspective-origin : 50% 100%; }
```

Hardboiled in 3D

First, CSS2 gave us the ability to position elements and a stacking order so we can arrange them using **z-index**. Then CSS3 introduced **translate**, which moves elements along *x*- and *y*-axes. Now, three-dimensional transforms give us **translateZ**, moving an element closer to or away from the viewer.

To demonstrate **translateZ** we'll carry on building that three-dimensional 'Get Hardboiled' interface. First, we'll style those magazine cover images, giving them a wide border:

```
@media (min-width: 48rem) {

.item__img img {
border-color : #9bc7d0; }

.item__img img:hover {
border-color : #eceeef; }
```

Next, style the descriptions by adding **width** and **padding** and position them relatively to move them up by 150 pixels. We'll also add a background colour and borders:

```
@media (min-width: 48rem) {

.item__description {
position : relative;
top : -150px;
padding : 11px;
width : 160px;
background-color : #dfe1e2;
border : 10px solid #ebf4f6; }
}
```



These simple styles will be applied by browsers of all capabilities

Scaling in three dimensions

CSS3 includes other three-dimensional `transform` properties: `rotateZ` and `scaleZ`. `scaleZ` allows us to `scale` the element in exactly the same way as `scaleX` and `scaleY`, but along the z-axis. Or we can use the combined `scale3d` property to specify scaling along all three axes at once:

```
.item {  
  transform : scale3d(scaleX, scaleY, scaleZ); }
```

With our foundations steady, we'll work through the final components that make the interface appear three-dimensional.

Preserving 3D

By default, when we apply perspective to an element, its children lie flat against a two-dimensional plane. The `transform-style` property gives us the option to either maintain this flattened behaviour, or raise elements off that plane using a value of `preserve-3d`.¹⁶

For this design, we'll apply `preserve-3d` to every item, then lift their descriptions into three-dimensional space using `translateZ`. This will make the descriptions appear to be closer to the viewer by eighty pixels:

```
.item {  
  transform-style : preserve-3d; }  
  
.item__description {  
  transform : translateZ(80px); }
```

¹⁶ All elements lie flattened against a two-dimensional plane by default. Applying `transform-style : flat`; sets this value explicitly.

Enhancing depth with box-shadow

To enhance the feeling of depth in this design, add two RGBa shadows to the descriptions and images:

```
.item__img img {
  box-shadow: 0 5px 5px 0 rgba(0, 0, 0, 0.25),
    0 2px 2px 0 rgba(0, 0, 0, 0.5); }

.item__description {
  box-shadow: 0 5px 5px 0 rgba(0, 0, 0, 0.25),
    0 2px 2px 0 rgba(0, 0, 0, 0.5); }
```



We can enhance the appearance of depth using **box-shadow** for browsers that are capable of rendering three-dimensional transforms.

Adding interactivity

Our interface is almost complete, but the eagle-eyed among you will have spotted that the items become difficult to read the more that **perspective** increases. To fix this, swing the items back to face the user when they mouse-over them. Do this by resetting the y-axis rotation to zero on hover:

```
.item:hover {
transform : rotateY(0); }
```

For good measure, we'll also reduce the amount of `translateZ` from eighty pixels to just five and move the descriptions to the right by twenty pixels:

```
.item:hover .item__description {
transform : translateZ(5px) translateX(20px); }
```

When these descriptions move to their new positions, the shadows they cast fall in the wrong places. Help these shadows appear more natural by altering their blur radii and transparency values.

```
.item:hover.item__img img {
box-shadow : 0 5px 15px rgba(0,0,0,.25); }
```

```
.item:hover .item__description {
box-shadow : 0 10px 15px rgba(0,0,0,.5); }
```



Now open this 'Get Hardboiled' interface and watch as the items change their rotations in three-dimensional space as your mouse passes over them.

Taking a hike

Finally, smooth the transitions between all of the state changes.

```
.item {  
  transition-property : transform;  
  transition-duration : .5s;  
  timing-function : ease-in-out; }  
  
.item__description {  
  transition-property : transform, box-shadow;  
  
  transition-duration : .25s;  
  timing-function : ease-in-out; }
```

Wait... what's that?

That is what we call a cliffhanger.

No. 19

Transitions

In web pages and applications, changes in state can have a huge impact on how it feels to use an interface. Make a change too fast and an interaction can feel unnatural. Make it too slow, even by a few milliseconds, and an interface will feel sluggish.

When we transform links into faux buttons, we often change their appearance on hover simply by changing their backgrounds:

```
.btn {  
  background-color : #bc676c; }
```

```
.btn:hover {  
  background-color : #a7494f; }
```

By default, these style changes happen instantly, but using transitions, we can make them change over a specified period of time and control acceleration and delay:

```
.btn {  
  transition-property : background-color; }
```

We can trigger transitions using dynamic pseudo-class selectors like `:hover`, `:focus`, `:active` and `:target`. Our first step is to specify which property or properties we want to transition. On our faux buttons, we need transition only the background colour, so use `transition-property` to specify this:

```
.btn {  
  transition-property : background-color;  
  transition-duration : .25s; }
```


Transition duration

Transitions change one or more styles over any number of seconds (s) or milliseconds (ms). These time units have, until now, only been used in aural style sheets. To smooth the transition over a quarter of one second, add a duration of `.25s`.

```
.btn { transition-duration : .25s; }
```

If we set duration at zero (0) or omit the property altogether, there will be no transition and any state changes will happen immediately.

Notice how we include transition declarations on the element to be transitioned and not on a state change such as `:hover`. Now change the background colour for the link's `:hover` state:

```
.btn {  
  transition-property : #a7494f; }
```

With this declaration in place, the button's background colour will now transition smoothly over a quarter of a second between those two shades of red.

We can apply transitions to any block-level or text-level element, as well as to `:before` and `:after` pseudo-elements. Here are some ideas of what you can do with them:

Background	Transition a background-color , CSS gradient or the background-position of a background-image on mouse-over.
Border	Emphasise a warning message by transitioning border-color , border-width or border-radius . We can also use outline for similar effects.
Colour	Smoothly transition text color when an element is moused over, active or in focus.
Dimensions	Transition width , height , min-width , max-width , min-height and max-height to make dynamic interfaces.
Font	Ease the transition between font-family , font-size or font-weight . For more control over typography, transition between letter-spacing , word-spacing and line-height values.
Margin and padding	Draw attention to an element by transitioning to new margin and padding values on :target .
Opacity	Add smooth fades and reveals by changing either the opacity or visibility of an element.
Position	Move smoothly between top , right , bottom and left and transition between z-index values to make simple animations from positioned elements.
Transform	Add transitions to transform property types like translate , scale , rotate and skew to bring interfaces to life.

Combining transitions

When we need two or more properties to transition — for example, both a background colour and a text colour — separate each property with a comma.

```
.btn {  
background-color : #bc676c;  
color : #fff;  
transition-property : background-color, color; }
```

We could otherwise group multiple properties into a single declaration using the `all` keyword:

```
.btn {  
transition-property : all; }
```

Delaying a transition

In the physical world, many objects we interact with don't turn on immediately when pressing a button or flipping a switch. By default, CSS transitions start from the moment they're activated — we'll call that the zero point. We can add a sense of physical reality by adding a delay between the zero point and the start of a transition. Specify the amount of delay in either milliseconds (`ms`) or seconds (`s`).

```
.btn {  
transition-property : background-color;  
transition-duration : .25s;  
transition-delay : .1s; }
```

Here, we added a delay of only a tenth of one second (`.1s`), from the zero point to the start of the background colour change. The same delay will also be applied when a property returns to its original state.

Accelerating a transition

Acceleration depends on the `transition-timing-function` we choose. For example, a linear transition will maintain a constant speed across its entire duration, whereas ease will gradually slow it down across the style change. Three more keywords are available to vary acceleration still further. They are:

<code>ease-in</code>	Starts slowly and gradually increases speed.
<code>ease-out</code>	Starts quickly and reduces speed over time.
<code>ease-in-out</code>	Accelerates quickly, reaches a peak and then tails off.

On our button links we'll specify a linear timing function:

```
.btn {  
  transition-property : background-color;  
  transition-duration : .25s;  
  transition-delay : .1s;  
  transition-timing-function : linear; }
```

Applying multiple transitions

When we need two or more properties to transition, we can group them into a comma-separated list, then specify duration, delay and timing function values for each. First, we'll write these multiple transitions in longhand:

The W3C's CSS3 Transitions Module also includes the ability to plot a `transition-timing-function` along a custom bezier curve. This mathematical approach to timing is fascinating, but beyond the scope of this book.

```
.btn {  
  transition-property : background-color, color;  
  transition-duration : .25s, .25s;  
  transition-delay : .1s, .1s;  
  transition-timing-function : linear, linear; }
```

When the duration, delay or timing function values are the same, we only need write their value once:

```
.btn {  
  transition-property : background-color, color;  
  transition-duration : .25s;  
  transition-delay : .1s;  
  transition-timing-function : linear; }
```

We can also combine all values into one comma-separated string.

```
.btn {  
  transition : background-color .25s .1s linear, color .25s .1s ↵  
  linear; }
```

Transitions see some action

In the previous chapter we built a 3D interface for the ‘Get Hard-boiled’ website and I left you with a cliffhanger. Glance back over your shoulder if you need a recap, because now it’s time to add transitions and rotate items forty-five degrees (**45deg**) in three-dimensional space. Then, to ensure that users can read our text, we’ll turn the items back to face the viewer on hover:

When we need our transitions to happen one after the other, in sequence, give each transition a different delay value.

It’s important to remember that when you include delay in a transition string, it must come after duration.

```
.item {  
  transform : rotateY(45deg);  
  transform-style : preserve-3d; }  
  
.item:hover {  
  transform : rotateY(0); }
```

Set up this way, the transition will happen instantly. To make our interface feel more fluid, add transitions, first by defining `transform` as the property to transition:

```
.item {  
  transition-property : transform; }
```

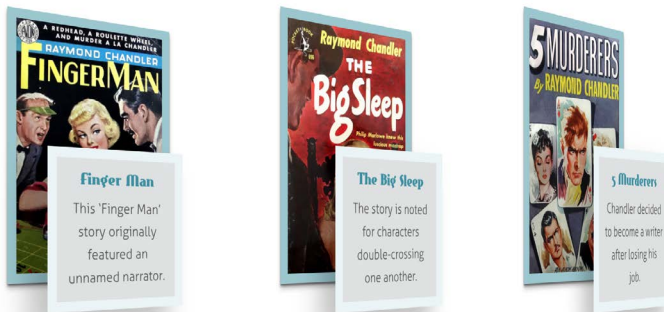
Next, specify that the transition will take three quarters of one second (`.75s`) with an `ease-in-out` timing function:

```
.item {  
  transition-duration : .75s;  
  timing-function : ease-in-out; }
```

When we need to shave a few bytes off our style sheets, we can combine these properties into a single shorthand declaration:

```
.item {  
  transition : transform .75s ease-in-out; }
```

To give this 3D interface added realism, use `translateZ` to make the descriptions appear to be closer to the viewer by eighty pixels. Then move it back and to the left, adjusting the strength of its shadow on hover:



```
.item div {transform : translateZ(80px);
box-shadow : -20px 20px 30px rgba(0,0,0,.25); }
```

```
.item:hover .item__description {
transform : translateZ(5px) translateX(20px);
box-shadow : 0 10px 15px rgba(0,0,0,.5); }
```

We'll transition all of our state changes over half of one second (.5s) and delay them by a fifth of one second (.2s).

```
.item__description {
transition-property : transform, box-shadow;
transition-duration : 5s, 5s;
transition-delay : .2s, .2s;
timing-function : ease-in-out, ease-in-out; }
```

Both transitions share the same duration, delay and timing function values, so we could simplify this declaration by combining two values into one:

```
.item__description {
transition-property : transform, box-shadow;
transition-duration : .5s;
timing-function : ease-in-out; }
```



Our design now feels more fluid and a user's interaction with it is more akin to what they might experience in the physical world.

Pulp fiction

If you or your clients aren't ready for three-dimensional interfaces yet, we can use transitions to create an entirely different 'Get Hardboiled' page. This new interface won't require any changes to our HTML. Here's a reminder:

```
<div class="hb-opacity">
<div class="item">
<div class="item__img"> [...] </div>
<div class="item_description"> [...] </div>
</div>
```

We'll start this design by styling the element for people who use smaller screens. For them, we'll implement a simpler layout where the items are stacked vertically and the magazine cover images are placed on the right. Here's a preview of the final smaller screen design.

We'll use flexbox to develop this design. To start building its foundations, apply `display:flex` to all items. We'll also add a little margin and padding and give those items a thick border:

```
.item {
display : flex;
margin-bottom : 1.35rem;
padding : 10px;
border : 10px solid #ebf4f6; }
```

This smaller screen design is different from the others that we've developed as we need to place the magazine cover images on the right, not on the left as they occur in the source order. Luckily, this is trivial using flexbox as we only need specify `row-reverse` as our `flex-direction`:

```
.item {
flex-direction : row-reverse; }
```

Now we'll turn our attention to what's inside those items by adding styles for the images. We'll give them width and a thick border to match their parent items:

The Phantom Detective

The Scarlet Menace

Vol. 1 No. 3
Issue #5
Jly '33

ADD TO CART



The Jewels Of Doom

Vol. 1 No. 3
Issue #5
Jly '33

ADD TO CART



The Yellow Murders

Vol. 4 No. 1
Issue #10
Dec '33

ADD TO CART



The interface we're developing, as it will be seen by people whose screens are smaller.

```
.item__img {
margin-left : 20px;
width : 133px; }

.item__img img {
border : 10px solid #ebf4f6;
box-sizing: border-box; }
```

To make the descriptions take up all the space that's left over by the image width and margin, use the `flex-grow` property with a value of `1`:

```
.item_description {
flex-grow : 1; }
```

Now that we've styled our items for small screens, it's time to build on that by adding styles for medium and larger size screens. Any styles added from this point will be nested within media queries to apply them only to devices that need them.

The Phantom Detective

The Scarlet Menace

Vol. 1 No. 3
Issue #5
Jly '33

ADD TO CART



The Jewels Of Doom

Vol. 1 No. 3
Issue #5
Jly '33

ADD TO CART



The Yellow Murders

Vol. 4 No. 1
Issue #10
Dec '33

ADD TO CART



A simple stack of items is perfect for people who use smaller screen devices.

To start laying the foundations for this design, set the outer container to `display:flex` so that all its direct descendants will be arranged along a horizontal axis:

```
@media (min-width: 48rem) {  
  .hb-opacity {  
    display : flex; }  
}
```

Now remove the `display:flex` we perviously added to all items and replace it with `display:block`. We'll also overwrite the margin, padding and border that we set previously. We'll add `position:relative` to establish each `item` as a positioning context as we'll need that in just a moment:

```
@media (min-width: 48rem) {  
  .item {  
    display : block;  
    margin : 0 20px 0 0;  
    padding : 0;  
    border-width : 0;  
    position : relative; }  
}
```

Forget everything you've read about absolute positioning being inflexible or unsuitable for dynamic content. With careful planning, absolute positioning can give us precise control, even in the most demanding situations. Now make the descriptions wider than their containers and use negative absolute position values to move them to the left:

```
@media (min-width: 48rem) {  
  .item_description {  
    position : absolute;  
    width : 200px;  
    left : -40px; }  
}
```

Finish styling them by adding padding, borders and a semi-transparent background colour that allows the elements behind the description to peek through:

```
@media (min-width: 48rem) {
  .item_description {
    padding: 20px;
    background-color: rgba(223, 225, 226, 0.95);
    border: 10px solid #ebf4f6;
    box-sizing: border-box; }
}
```



Layering descriptions over images.

To create the bubbles, reposition the descriptions above the top of the images. To ensure that active bubbles always appear closest to the viewer, give them a higher **z-index**:

```
@media (min-width: 48rem) {
  .item:hover .item_description {
    top : -80px;
    z-index : 3; }
}
```

Next, add depth with two RGBa shadows, separating the values of each with a comma:

```
@media (min-width: 48rem) {  
  .item:hover .item_description {  
    box-shadow : 0 5px 5px 0 rgba(0,0,0,0.25),  
    0 2px 2px 0 rgba(0,0,0,0.5); }  
}
```



Adding depth with box shadows.

With our description bubbles inflated, hide them from view simply by making them fully transparent. We can reveal them again on hover:

```
@media (min-width: 48rem) {  
  .item .item_description {  
    opacity : 0; }  
  
  .item:hover .item_description {  
    opacity : 1; }  
}
```

By default, the changes in position and **opacity** will happen instantly, but we can use transitions to make them feel more fluid. First, define **top** and **opacity** as the two properties to transition, followed by a half-second (.5s) duration and a timing function that slows the transitions as they progress:

```
@media (min-width: 48rem) {  
  .item .item_description {  
    transition-property : top, opacity;  
    transition-duration : .25s;  
    transition-timing-function : ease-out; }  
}
```

The Phantom Detective



Now our bubbles slide and fade into view when a user mouses-over an item.

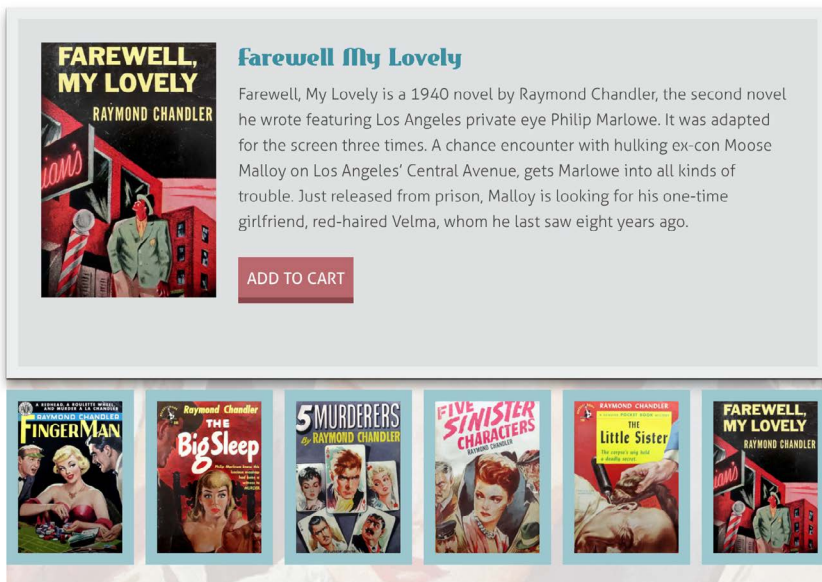
These bubbles now render correctly in all current browsers. But what about less capable ones, those that don't implement either transitions or **opacity**? How do they handle this interface?

Browsers that don't support transitions will safely ignore them and we should remember — as Dan Cederholm reminded us — websites don't need to be experienced exactly the same in every browser.

Panel game

This next interface has an entirely different look and feel. Clicking on a book reveals a panel that contains its description. We'll build the panels using CSS positioning, **opacity** and transitions. We can reuse our HTML from the last example, but this time we'll need a unique **id** for each item so that we can address their fragments directly:

Six books by Dashiell Hammett



Here's an early look at the 'Get Hardboiled' interface we're building.

```
<div class="hb-transitions">
  <div id="hb-transitions-01" class="item">
    <div class="item__img"> [...] </div>
    <div class="item__description"> [...] </div>
  </div>
</div>
```

We'll also need an anchor that points back to its parent item:

```
<div id="hb-transitions-01" class="item">
<a href="#hb-transitions-01"></a>
</div>
```

As we're starting by designing for people who use smaller screens, we'll style those items into a simple, vertical list. Once again, we'll use flexbox to develop our layout. As we need to display the magazine cover images on the right instead of their place in the source order, we'll also set `row-reverse` as our `flex-direction`:

```
.item {
display : flex;
flex-direction : row-reverse;
margin-bottom: 1.35rem;
padding: 10px;
border: 10px solid #ebf4f6; }
```

Inside each item we'll give the images a little left margin to separate them from the description, and a width:

```
.item__img {
margin-left : 20px;
width : 133px; }
```

To make the descriptions take up all the space that's left over by the image width and margin, use the `flex-grow` property with a value of `1`:

```
.item_description {
flex-grow : 1; }
```


With our design now appropriate for people who use smaller screens, we'll turn our attention to those using larger screens. Any styles added from this point will be nested within media queries to apply them only to devices that need them.

Start by adding dimensions to the `hb-transitions` division, then establish it as a positioning context for any positioned child elements by applying relative positioning without any offsets.

```
@media (min-width: 48rem) {
  .hb-transitions {
    position : relative;
    height : 500px;
    width : 710px; }
}
```

Next, size those inline images and position them so that they fit neatly at the bottom of the panel. Later we'll use those same images as backgrounds, so they need to be larger than they appear initially:

Six books by Dashiell Hammett

Finger Man

This "Finger Man" story originally featured an unnamed narrator, identified as "Carmady" in subsequent stories, and later renamed Marlowe for book publication.

ADD TO CART



The Big Sleep

The story is noted for its complexity, with many characters double-crossing one another and many secrets being exposed throughout the narrative. The title is a euphemism for death; it refers to a rumination in the final pages of the book about "sleeping the big sleep."

ADD TO CART



5 Murderers

Raymond Thornton Chandler was a British/American novelist and screenwriter. In 1932, at age forty-four, Chandler decided to become a detective fiction writer after losing his job as an oil company executive during the Great Depression.

ADD TO CART



This stack of items works well on smaller screen sizes.

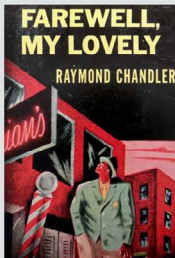
```

@media (min-width: 48rem) {
  .item__img {
    position : absolute;
    top : 330px;
    width : 110px;
    height : 160px; }

  #hb-transitions-01 .item__img { left : 0; }
  #hb-transitions-02 .item__img { left : 120px; }
  #hb-transitions-03 .item__img { left : 240px; }
  #hb-transitions-04 .item__img { left : 360px; }
  #hb-transitions-05 .item__img { left : 480px; }
  #hb-transitions-06 .item__img { left : 600px; }
}

```

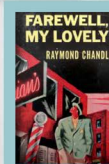
Six books by Dashiell Hammett



Farewell My Lovely

Farewell, My Lovely is a 1940 novel by Raymond Chandler, the second novel he wrote featuring Los Angeles private eye Philip Marlowe. It was adapted for the screen three times. A chance encounter with hulking ex-con Moose Malloy on Los Angeles' Central Avenue, gets Marlowe into all kinds of trouble. Just released from prison, Malloy is looking for his one-time girlfriend, red-haired Velma, whom he last saw eight years ago.

ADD TO CART



With planning, absolute positioning gives us fine control, even in the most demanding situations.

We don't want our descriptions to show until a user clicks on a book cover, so make every description small enough to position behind its respective cover. Setting `overflow` to `hidden` will make sure that long content won't escape and ruin our design:

```
@media (min-width: 48rem) {  
  .item__description {  
    z-index : 1;  
    position : absolute;  
    top : 335px;  
    left : 5px;  
    width : 100px;  
    height : 150px;  
    overflow : hidden; }  
  
  #hb-transitions-01 .item__description { left : 0; }  
  #hb-transitions-02 .item__description { left : 120px; }  
  #hb-transitions-03 .item__description { left : 240px; }  
  #hb-transitions-04 .item__description { left : 360px; }  
  #hb-transitions-05 .item__description { left : 480px; }  
  #hb-transitions-06 .item__description { left : 600px; }  
}
```

Now tuck the descriptions behind the images by giving them a lower `z-index` and, to make sure they're not seen until we want them, set `opacity` to zero (0) so they'll be fully transparent:

```
@media (min-width: 48rem) {  
  .item__img {  
    z-index : 2; }  
  
  .item__description {  
    z-index : 1;  
    opacity : 0; }  
}
```

Earlier, we twisted the knife by pointing an anchor to its parent item. It's this anchor and the `:target` pseudo-class selector that make it possible to trigger the transformation of each description. Reset the descriptions' `opacity` and position, then resize them to fill the top of the listing panel. Add padding including a wide space on the left that will soon be filled with a background image.

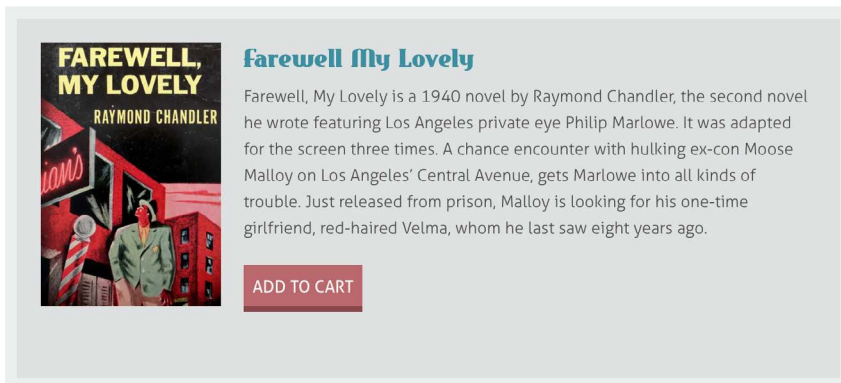
```
@media (min-width: 48rem) {  
  .item:target .item__description {  
    opacity : 1;  
    top : 0;  
    left : 0;  
    width : 100%;  
    height : 320px;  
    padding : 20px 20px 0 190px; }  
}
```

Now to set background and border properties, common to every description:

```
@media (min-width: 48rem) {  
  .item:target .item__description {  
    background-color: #dfe1e2;  
    background-origin: padding-box;  
    background-position: 20px 20px;  
    background-repeat: no-repeat;  
    background-size: auto 220px;  
    border: 10px solid #eceeef;  
    box-sizing: border-box; }  
}
```

Next, add a unique book cover background image to each description:

```
@media (min-width: 48rem) {  
#hb-transitions-01:target .item__description {  
background-image : url(transitions-01.jpg); }  
  
#hb-transitions-02:target .item__description {  
background-image : url(transitions-02.jpg); }  
  
#hb-transitions-03:target .item__description {  
background-image : url(transitions-03.jpg); }  
  
#hb-transitions-04:target .item__description {  
background-image : url(transitions-04.jpg); }  
  
#hb-transitions-05:target .item__description {  
background-image : url(transitions-05.jpg); }  
  
#hb-transitions-06:target .item__description {  
background-image : url(transitions-06.jpg); }  
}
```



The panels are almost complete. When a user presses on a book cover, a panel that contains its description will appear above.

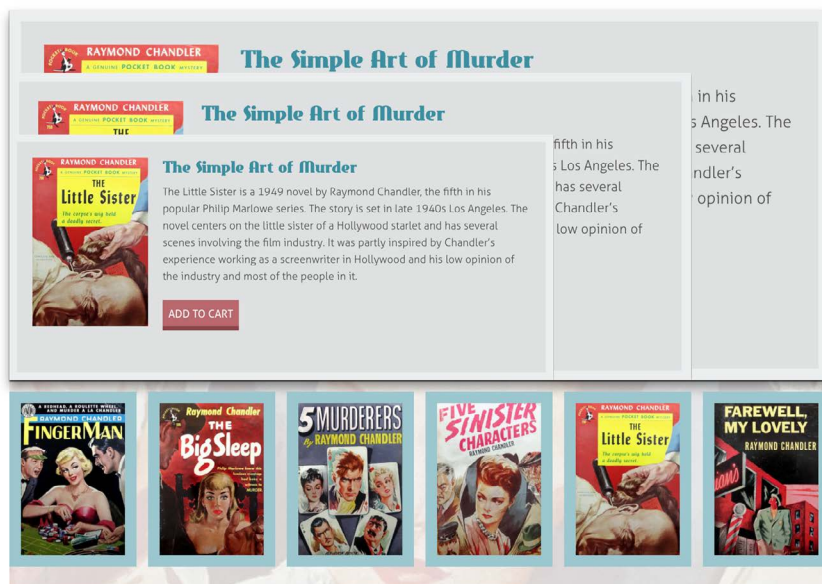
Now we'll use transitions to make the interaction seem smoother and bring our interface to life. For each description, we'll transition four properties — **top**, **width**, **height** and **opacity** — separating them using a comma:

```
@media (min-width: 48rem) {
  .item__description {
    transition-property : top, width, height, opacity; }
}
```

Finally, set a duration for each property:

```
@media (min-width: 48rem) {
  .item__description {
    transition-duration : .5s, .5s, .75s, .5s; }
}
```

Six books by Dashiell Hammett



The changes to **top**, **width** and **height** will last half of one second (.5s) – the **opacity** change will last three quarters of one second (.75s)

Designing for landscape and portrait orientations

When our HTML is hardboiled, we can more easily adapt our designs to satisfy the demands of different browsing environments, including devices such as tablets that can switch between portrait and landscape orientations. While the wider screen layout we just made works fine in portrait, it doesn't fit so well into a tablet's landscape format.

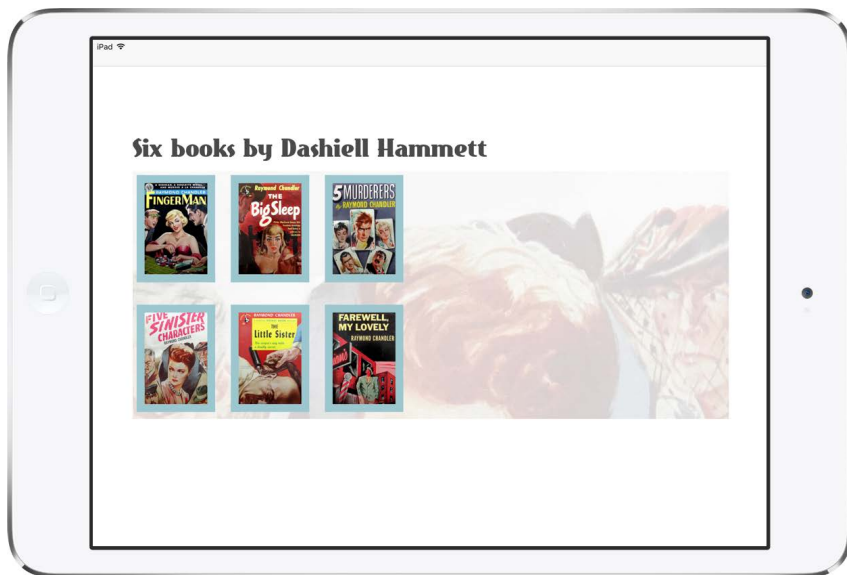
We'll start this alternative layout by changing the height of the parent division:

```
@media (min-width: 48rem) {  
  .hb-landscape {  
    position : relative;  
    width : 760px;  
    height : 500px; }  
}
```

Next, resize those inline images and position them to form a new grid on the panel's left side:

```
@media (min-width: 48rem) {  
  .item__img {  
    position : absolute;  
    width : 100px;  
    height : 150px; }  
  
  #hb-landscape-01 .item__img {  
    top : 0;  
    left : 0; }  
  
  #hb-landscape-02 .item__img {  
    top : 0;  
    left : 120px; }
```

```
#hb-landscape-03 .item__img {  
  top : 0;  
  left : 240px; }  
  
#hb-landscape-04 .item__img {  
  top : 170px;  
  left : 0; }  
  
#hb-landscape-05 .item__img {  
  top : 170px;  
  left : 120px; }  
  
#hb-landscape-06 .item__img {  
  top : 170px;  
  left : 240px; }  
}
```



Laying out images into a grid.

Now we need to make the descriptions small enough to position behind their respective images:

```
@media (min-width: 48rem) {  
  .item .description {  
    position : absolute;  
    width : 100px;  
    height : 10px;  
    overflow : hidden; }  
  
  #hb-landscape-01 .item_description {  
    top : 0;  
    left : 0; }  
  
  #hb-landscape-02 .item_description {  
    top : 0;  
    left : 120px; }  
  
  #hb-landscape-03 .item_description {  
    top : 0;  
    left : 240px; }  
  
  #hb-landscape-04 .item_description {  
    top : 170px;  
    left : 0; }  
  
  #hb-landscape-05 .item_description {  
    top : 170px;  
    left : 120px; }  
  
  #hb-landscape-06 .item_description {  
    top : 170px;  
    left : 240px; }  
}
```

Set up the basis for transitions by giving each description a lower **z-index** than the corresponding images and set their **opacity** to zero (0):

```
@media (min-width: 48rem) {  
  .item__img {  
    z-index : 2; }  
  
  .item_description {  
    z-index : 1;  
    opacity : 0; }  
}
```

Use the **:target** pseudo-class selector to reset the **opacity** to zero (0), and reposition and resize the descriptions to fill the right side of the panel. Add padding, a background colour and a thick border to complete the look:

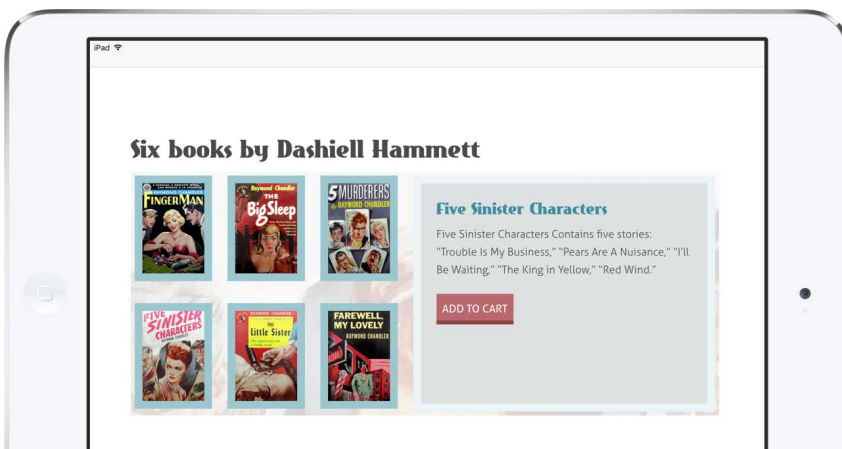
```
@media (min-width: 48rem) {  
  .item:target .item_description {  
    top : 0;  
    left : 360px;  
    width : 390px;  
    height : 280px;  
    padding : 20px;  
    opacity : 1;  
    background-color: #dfe1e2;  
    border : 10px solid #ebf4f6; }  
}
```

For this version, we'll use just two transition properties — **height** and **opacity**:

```
@media (min-width: 48rem) {  
  .item_description {  
    transition-property : height, opacity; }  
}
```

Set a duration for each transition property: half a second (**.5s**) for the change in **height**; and three quarters of one second (**.75s**) for **opacity**:

```
@media (min-width: 48rem) {
  .item_description {
    transition-duration : .5s, .75s; }
}
```



Providing an alternative layout for a landscape orientation.

Breaking it up

How a web page or application feels can have a huge impact on how often people use it. You learned how to add subtle transitions that can both delight a user and add the element of surprise to make using an interface more enjoyable.

On three different interfaces for the 'Get Hardboiled' store, the same HTML underpinned three very different interfaces using transitions tailored to make them appropriate for people who use small screen devices, without compromising the experience of people who use the biggest screens. Now that's hardboiled.

No. 20

Multicolumn layout

I know web designers are constantly reminded that web is its own medium, that it isn't print; but there's so much about print design – in the form of magazines and newspapers – that can and should inspire our work on the web. The different ways that magazine designers use columns of text to make their publications individual are an enormous inspiration to me, which is why I'm constantly surprised by how unimaginative most website layouts are, particularly in the responsive web design era. That needs to change and CSS multicolumn layout is one way to help influence it.

We've been able to use CSS multicolumn layouts to create columns of text with no presentational markup for ten years. I wrote about them in *Transcending CSS* and again, five years ago, in the first edition of this book. I teach CSS columns at all my workshops and I'm dismayed that every time I ask for a show of hands from people who've used them, only a few are raised. I hope this third time's a charm and that I can inspire you to start using CSS multicolumn layouts.

You've probably guessed from its title that the CSS multicolumn layout module¹⁷ provides a way to create multicolumn layouts with just CSS and no additional markup, floats or other layout methods.

¹⁷ w3.org/TR/css3-multicol

We'll dive right in with a multicolumn layout example from the 'Get Hardboiled' entry page. For this design we'll use columns to reduce the measure,¹⁸ making the content more readable. To achieve this, we'd traditionally add a number of divisions to divide blocks of content, then float them to create columns.

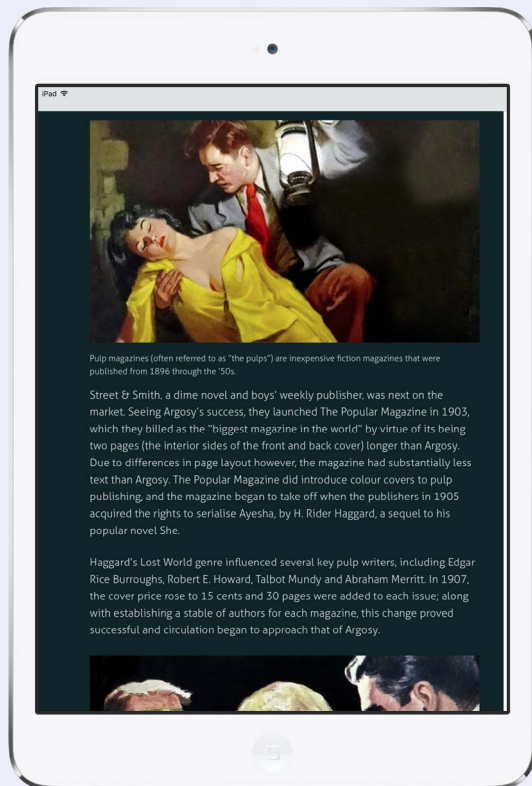
```
<div class="col">
<p>Raymond Thornton Chandler was an American novelist and
screenwriter. In 1932, at age forty-four, Chandler decided to
become a detective fiction writer after losing his job as an oil
company executive during the Great Depression.</p>
</div>
<div class="col">
<p>Chandler published seven novels during his lifetime (an
eighth in progress at his death was completed by Robert B.
Parker). All but Playback have been made into motion pictures,
some several times. In the year before he died, he was elected
president of the Mystery Writers of America.</p>
</div>
```

There's nothing inherently wrong with this familiar technique. It's easy to implement and, for the most part, it's reliable. That's no doubt why we see it being used on countless websites. Today though, in the era of responsive web design where we must consider many different sizes of screens, the advantages of this simple technique are far outweighed by the disadvantages.

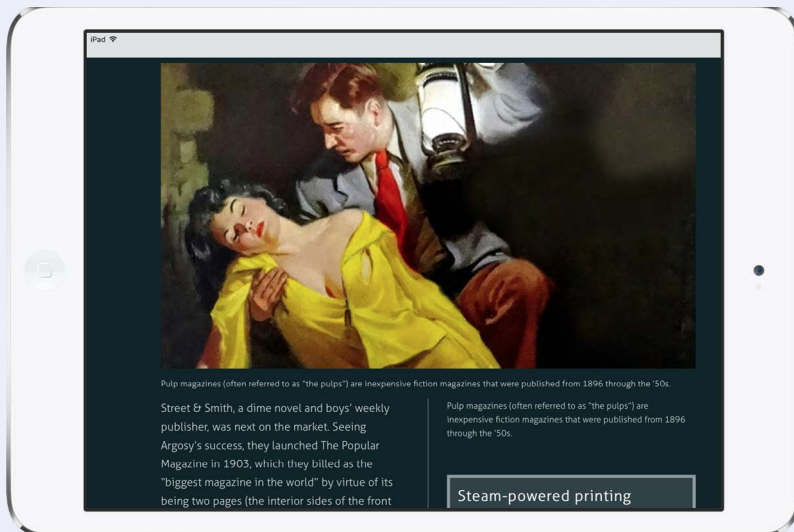
¹⁸ In typography parlance, the measure is the measurement in characters of a line of text's length.

Columns in your hand

First, the iPad, in its various sizes and configurations, then other devices like it, have changed what many people describe as a computer. Tablets do many things very well, but something I use them for most is demonstrating layout changes across both responsive breakpoints, and landscape and portrait screen orientations. I use iPads so often for this that I sometimes give them to our clients to keep after a project. When we hold an iPad in portrait orientation (or look at a low-resolution monitor) it makes sense to present content on our 'Get Hardboiled' home page in a single column, as it makes good use of space and our text is comfortable to read.



This measure works well with a single column in iPad's portrait orientation.



Two columns and a narrower measure make text more readable in the iPad's landscape orientation.

Turn an iPad to landscape orientation (or use a larger monitor) and that one column doesn't work as well because the lengths of the lines now make reading less comfortable. To improve the reading experience in landscape orientation, we'll use two columns and create a narrower measure.

Wouldn't it be incredible if our layouts could automatically change the number of columns and therefore optimise a user's reading experience? Guess what? With CSS multicolumn layout they can.

Column widths and counts

Changing the number and width of columns so our layouts adapt to different screen widths and orientations is easy when we use CSS columns. We can implement columns in two ways: the first by defining the number of columns; and the second by specifying the width of columns.

First things first, let's rewrite the HTML of our 'Get Hardboiled' article, removing those presentational divisions to leave only structured content in an HTML `section` element:

```
<section>
  <p>Raymond Thornton Chandler was an American novelist and
  screenwriter. In 1932, at age forty-four, Chandler decided to
  become a detective fiction writer after losing his job as an oil
  company executive during the Great Depression. His first short
  story, 'Blackmailers Don't Shoot', was published in 1933 in
  Black Mask, a popular pulp magazine. His first novel, 'The Big
  Sleep', was published in 1939. In addition to his short stories,
  Chandler published seven novels during his lifetime (an eighth
  in progress at his death was completed by Robert B. Parker). All
  but Playback have been made into motion pictures, some several
  times. In the year before he died, he was elected president of
  the Mystery Writers of America.</p>
</section>
```

We didn't need those divisions in our markup as we can now create columns in our style sheet instead. First, we'll specify our new columns' width using the `column-width` property. We can do this using several units including pixels, but I prefer to size my columns using `rem` units linked to the size of my text:

```
section {
  column-width : 32rem; }
```


Raymond Chandler

Raymond Thornton Chandler was an American novelist and screenwriter. In 1932, at age forty-four, Chandler decided to become a detective fiction writer after losing his job as an oil company executive during the Great Depression. His first short story, 'Blackmailers Don't Shoot,' was published in 1933 in *Black Mask*, a popular pulp magazine. His first novel, 'The Big Sleep,' was published in 1939. In addition to his short stories, Chandler

published seven novels during his lifetime (an eighth in progress at his death was completed by Robert B. Parker). All but *Playback* have been made into motion pictures, some several times. In the year before he died, he was elected president of the Mystery Writers of America.

If a parent gets wider, a browser will add new columns. When it narrows, a browser will remove them one at a time — all the while reflowing text to fit.

I've chosen 32rem because — at my chosen type size of 1.6rem — it creates a comfortable reading line length of between 45 to 75 characters inside the columns. A browser will start with just one column on a small screen. When a screen's wide enough to display more than one 32rem column, a browser will dynamically display first two, then three, then more columns.

Writing vendor prefixes

Firefox and WebKit both implemented CSS columns using their own vendor-specific prefixes, so we'll need to add them, followed by the W3C's official syntax:

```
section {  
  -moz-column-width : 32rem;  
  -webkit-column-width : 32rem;  
  column-width : 32rem; }
```

At the time of writing, Microsoft Edge, Opera Mini and Safari on iOS and Mac OS X have implemented CSS multicolumn layout prefix-free.

Column count

I can think of several design situations where, instead of specifying a column's width, we need to define how many columns we need. We'll use the `column-count` property for this.

For small screen screens, our `section` needs only one column. We don't need to specify that, since a browser will display it natively. As it makes more sense to use columns on medium and large screens, we'll place our `column-count` declaration inside a media query:

```
@media (min-width: 48rem) {  
  section {  
    column-count : 2; }  
}
```

When a browser's width is greater than 48em, our text will flow into two columns. Likewise, when we need three columns we'll place that next declaration inside another, wider minimum width media query:

```
@media (min-width: 64rem) {  
  section {  
    column-count : 3; }  
}
```

Raymond Chandler

Raymond Thornton Chandler was an American novelist and screenwriter. In 1932, at age forty-four, Chandler decided to become a detective fiction writer after losing his job as an oil company executive during the Great Depression. His first short story,

'Blackmailers Don't Shoot,' was published in 1933 in *Black Mask*, a popular pulp magazine. His first novel, 'The Big Sleep,' was published in 1939. In addition to his short stories, Chandler published seven novels during his lifetime (an eighth in progress at

his death was completed by Robert B. Parker). All but *Playback* have been made into motion pictures, some several times. In the year before he died, he was elected president of the Mystery Writers of America.

In a responsive layout, the width of these new columns will vary to fit the width of their parent container, but the number of columns will stay the same.

Columns shortcut

As the `column-width` and `column-count` properties do not overlap, it makes sense to combine both of them into a shorter `columns` property, like this:

```
@media (min-width: 48rem) {  
  section {  
    columns: 32rem 2; }  
}
```

Column gaps

White space is an incredibly important factor in improving readability, and gaps between columns help to define reading areas. We'll insert gaps between our columns. We could specify gaps using pixels, but it's better practice in responsive web design to use a flexible unit like rems. Our gaps will be `4rem` wide:

```
@media (min-width: 48rem) {  
  section {  
    column-gap : 4rem; }  
}
```

To help our design stay connected to the screen we're viewing, we'll increase the width of our gaps on larger screens:

```
@media (min-width: 64rem) {  
  section {  
    column-gap : 6rem; }  
}
```

Raymond Chandler

Raymond Thornton Chandler was an American novelist and screenwriter. In 1932, at age forty-four, Chandler decided to become a detective fiction writer after losing his job as an oil company executive during the Great Depression. His first short story,

'Blackmailers Don't Shoot,' was published in 1933 in *Black Mask*, a popular pulp magazine. His first novel, 'The Big Sleep,' was published in 1939. In addition to his short stories, Chandler published seven novels during his lifetime (an eighth in progress at

his death was completed by Robert B. Parker). All but *Playback* have been made into motion pictures, some several times. In the year before he died, he was elected president of the Mystery Writers of America.

Great responsive web design is about more than adapting layout. It includes making tiny changes to many elements across responsive breakpoints, to help a design to stay connected to the screen it's being viewed on.

Column rules

Horizontal rules are so important in web design that they warrant their own element, `hr`, but vertical rules are equally as important. Although they don't have their own HTML element, they do have a CSS multicolumn property. First, we'll specify the width of a `column-rule`. I usually define mine using pixels:

```
section {  
  column-rule-width : 2px; }
```

I've also often been known to increase the width of my column rules across responsive breakpoints. The wider the screen, the wider my rules:

```
@media (min-width: 64rem) {  
  section {  
    column-rule-width : 3px; }  
}
```

Of course, we can specify the colour of rules, too.

```
section {  
  column-rule-color : #ebf4f6; }
```

Finally, we can define a style for rules. Dashed, dotted and solid are staple rule styles, but you can also use any **border-style** value as a **column-rule-style**. I know you'll be pleased to know that **groove**, **ridge**, **inset** and **outset** are available too. No? Just me then.

```
section {  
  column-rule-style : solid; }
```

Raymond Chandler

Raymond Thornton Chandler was an American novelist and screenwriter. In 1932, at age forty-four, Chandler decided to become a detective fiction writer after losing his job as an oil company executive during the Great Depression. His first short story,

'Blackmailers Don't Shoot,' was published in 1933 in Black Mask, a popular pulp magazine. His first novel, 'The Big Sleep,' was published in 1939. In addition to his short stories, Chandler published seven novels during his lifetime (an eighth in progress at

his death was completed by Robert B. Parker). All but Playback have been made into motion pictures, some several times. In the year before he died, he was elected president of the Mystery Writers of America.

CSS columns are fast and easy to implement and have excellent support in contemporary browsers.

CSS columns see some action

I've often wondered, considering that CSS columns are easy to implement and have widespread browser support, why so few people use them. I understand that support was patchy in the past and that may have dissuaded people from working with them, but I honestly think people don't use columns because they lack the imagination as to where to use them.

From conversations with fellow designers, I know that most people first think about using CSS columns to divide blocks of body copy to create layouts that are reminiscent of those in magazines and newspapers. I don't blame people for that, it's an obvious choice; but while columns of text work well in print media, they're not always the best choice for the web. Take this example of an article on 'Get Hardboiled' where we've divided the main content into columns.

Classic Hardboiled stories

Pulp magazines (often referred to as "the pulps") are inexpensive fiction magazines that were published from 1896 through the 1950s. The term pulp derives from the cheap wood pulp paper on which the magazines were printed; in contrast, magazines printed on higher quality paper were called "glossies" or "slicks." The typical pulp magazine had 128 pages with ragged, untrimmed edges.

Lurid and exploitative stories

In their first decades, pulps were most often priced at ten cents per magazine, while competing slicks cost 25 cents a piece. Pulps were the successors to the penny dreadfuls, dime novels, and short fiction magazines of the 19th century. Although many respected writers wrote for pulps, the magazines were best known for their lurid and exploitative stories and sensational cover art. Modern superhero comic books are sometimes considered descendants of 'hero pulps'; pulp magazines often featured illustrated novel-length stories of heroic characters, such as 'The Shadow,' 'Doc Savage,' and 'The Phantom Detective.'

The first pulp was Frank Munsey's revamped Argosy Magazine of 1896, with about 135,000 words (192 pages) per issue, on pulp paper with untrimmed edges, and no illustrations, even on the cover.

The steam-powered printing press had been in widespread use for some time, enabling the boom in dime novels; prior to Munsey, however, no one had combined cheap printing, cheap paper and cheap authors in a package that provided affordable entertainment to young working-class people. In six years Argosy went from a few thousand copies per month to over half a million.



Pulp magazines (often referred to as "the pulps") are inexpensive fiction magazines that were published from 1896 through the '50s.

Street & Smith, a dime novel and boys' weekly publisher, was next on the market. Seeing Argosy's success, they launched The Popular Magazine in 1903, which they billed as the "biggest magazine in the world" by virtue of its being two pages (the interior sides of the front and back cover) longer than Argosy.

Due to differences in page layout however, the magazine had substantially less text than Argosy. The Popular Magazine did introduce colour covers to pulp publishing, and the magazine began to take off when the publishers in 1905 acquired the rights to serialise Ayesha, by H. Rider Haggard, a sequel to his popular novel She.

Haggard's Lost World genre influenced several key pulp writers, including Edgar Rice Burroughs, Robert E. Howard, Talbot Mundy and Abraham Merritt. In 1907, the cover price rose to 15 cents and 30 pages were added to each issue; along with establishing a stable of authors for each magazine, this change proved successful and circulation began to approach that of Argosy.



Pulp magazines (often referred to as "the pulps") are inexpensive fiction magazines that were published from 1896 through the '50s.

Steam-powered printing

At their peak of popularity in the 1920s and 1930s, the most successful pulps could sell up to one million copies per issue. The most successful pulp magazines were Argosy, Adventure, Blue Book and Short Stories, collectively described by some pulp historians as "The Big Four." Among the best-known other titles of this period were Amazing Stories, Black Mask, Dime Detective, Flying Aces, Horror Stories, Love Story Magazine, Marvel Tales, Oriental Stories, Planet Stories, Spicy Detective, Startling Stories, Thrilling Wonder Stories, Unknown, Weird Tales and Western Story Magazine.

Although pulp magazines were primarily an American phenomenon, there were also a number of British pulp magazines published between the Edwardian era and World War II. Notable UK pulps included Pall Mall Magazine, The Novel Magazine, Cassell's Magazine, The Story-Teller, The Sovereign Magazine, Hutchinson's Adventure-Story and Hutchinson's Mystery-Story. The German fantasy magazine Der Orchideengarten had a similar format to American pulp magazines, in that it was printed on rough pulp paper and heavily illustrated.

Dividing large blocks of content into columns isn't always the best choice and can create a less enjoyable reading experience.

There are two remarkable things to notice about this columnised design. First — and possibly most important — by adding columns we've provided a unconventional and possibly less convenient reading experience. While in a magazine or newspaper we're very familiar with reading down a column to its bottom, then moving our eyes to the top of the next, we're not used to doing that on the web.

This reading experience is made less enjoyable on smaller and medium-sized screens where the columns can extend outside the viewport forcing people to scroll up and down the page to continue reading.

Spanning columns

Fortunately, there are CSS multicolumn properties that help to make people's reading experience better, and all that's needed is some careful thought when using them. Let's rewind to that previous example. Long passages of columnised content work well when we can define the height of the overall layout, but that's not possible in responsive web design. Shorter passages of columnised content can be extremely effective and can give a design a more distinctive look.

To achieve these shorter columnised sections, we don't need additional elements, just the `column-span` property applied to strategic elements; for example, major headings or perhaps `figure` elements:

```
figure {  
  column-span : all; }
```

This makes an element span any number of columns. While `column-span` accepts several values — including `inherit`, `initial`, `none` and `unset` — only `all` is of any practical use. To demonstrate how effective using `column-span` can be, let's add it to major headings and figure elements using a BEM-based `columns__span` class attribute value:

Classic Hardboiled stories

Pulp magazines (often referred to as "the pulps") are inexpensive fiction magazines that were published from 1896 through the 1950s. The term pulp derives from the cheap wood pulp paper on which the

magazines were printed; in contrast, magazines printed on higher quality paper were called "glossies" or "slicks." The typical pulp magazine had 128 pages with ragged, untrimmed edges.

Lurid and exploitative stories

In their first decades, pulps were most often priced at ten cents per magazine, while competing slicks cost 25 cents a piece. Pulps were the successors to the penny dreadfuls, dime novels, and short fiction magazines of the 19th century. Although many respected writers wrote for pulps, the magazines were best known for their lurid and exploitative stories and sensational cover art. Modern superhero comic books are sometimes considered descendants of 'hero pulps'; pulp magazines often featured illustrated novel-length stories of heroic characters, such as 'The Shadow,' 'Doc Savage,' and 'The Phantom Detective.'

The first pulp was Frank Munsey's revamped Argosy Magazine of 1896, with about 135,000 words (192 pages) per issue, on pulp paper with untrimmed edges and no illustrations, even on the cover.

The steam-powered printing press had been in widespread use for some time, enabling the boom in dime novels; prior to Munsey, however, no one had combined cheap printing, cheap paper and cheap authors in a package that provided affordable entertainment to young working-class people. In six years Argosy went from a few thousand copies per month to over half a million.



Pulp magazines (often referred to as "the pulps") are inexpensive fiction magazines that were published from 1896 through the '50s.

Street & Smith, a dime novel and boys' weekly publisher, was next on the market. Seeing Argosy's success, they launched The Popular Magazine in 1903, which they billed as the "biggest magazine in the world" by virtue of its being two pages (the interior sides of the front and back cover) longer than Argosy. Due to differences in page layout however, the magazine had substantially less text than Argosy. The Popular Magazine did introduce colour covers to pulp publishing, and the magazine began to take off when the publishers in 1905 acquired the rights to serialise Ayesha, by H. Rider Haggard, a sequel to his popular novel She.

Haggard's Lost World genre influenced several key pulp writers, including Edgar Rice Burroughs, Robert E. Howard, Talbot Mundy and Abraham Merritt. In 1907, the cover price rose to 15 cents and 30 pages were added to each issue; along with establishing a stable of authors for each magazine, this change proved successful and circulation began to approach that of Argosy.



Pulp magazines (often referred to as "the pulps") are inexpensive fiction magazines that were published from 1896 through the '50s.

Steam-powered printing

At their peak of popularity in the 1920s and 1930s, the most successful pulps could sell up to one million copies per issue. The most successful pulp magazines were Argosy, Adventure, Blue Book and Short Stories, collectively described by some pulp historians as "The Big Four". Among the best-known other titles of this period were Amazing Stories, Black Mask, Dime Detective, Flying Aces, Horror Stories, Love Story Magazine, Marvel Tales, Oriental Stories, Planet Stories, Spicy Detective, Startling Stories, Thrilling Wonder Stories, Unknown, Weird Tales and Western Story Magazine.

Although pulp magazines were primarily an American phenomenon, there were also a number of British pulp magazines published between the Edwardian era and World War II. Notable UK pulps included Pall Mall Magazine, The Novel Magazine, Cassell's Magazine, The Story-Teller, The Sovereign Magazine, Hutchinson's Adventure-Story and Hutchinson's Mystery-Story. The German fantasy magazine Der Orchideengarten had a similar format to American pulp magazines, in that it was printed on rough pulp paper and heavily illustrated.

```
.columns__span {  
  column-span : all; }
```

Using `column-span` elements strategically to create shorter columnised sections can improve the usability of a design.


From a higher-level view of the page, we can see how dividing the columnised content into shorter sections gives the layout more structure. It also reduces the distance that a reader's eyes must travel between columns, which dramatically improves their reading experience.

Breaking columns

When we divide our content across columns, it will automatically be evenly balanced across them. In practice, this can lead to some unpredictable results.

Fortunately, we can ensure that elements stay together by using the `break-inside` property.¹⁹ We'll apply it to a box using using a BEM-based `columns__break` class attribute value, like this:

Street & Smith, a dime novel and boys' weekly publisher, was next on the market. Seeing Argosy's success, they launched The Popular Magazine in 1903, which they billed as the "biggest magazine in the world" by virtue of its being two pages (the interior sides of the front and back cover) longer than Argosy.



Pulp magazines (often referred to as "the pulps") are inexpensive fiction magazines that were published from 1896 through the '50s.

Steam-powered printing

At their peak of popularity in the 1920s and 1930s, the

most successful pulps could sell up to one million copies per issue. The most successful pulp magazines were Argosy, Adventure, Blue Book and Short Stories, collectively described by some pulp historians as "The Big Four." Among the best-known other titles of this period were Amazing Stories, Black Mask, Dime Detective, Flying Aces, Horror Stories, Love Story Magazine, Marvel Tales, Oriental Stories, Planet Stories, Spicy Detective, Startling Stories, Thrilling Wonder Stories, Unknown, Weird Tales and Western Story Magazine.

Although pulp magazines were primarily an American phenomenon, there were also a number of British pulp magazines published between the Edwardian era and World War II. Notable UK pulps included Pall Mall Magazine, The Novel Magazine, Cassell's Magazine, The Story-Teller, The Sovereign Magazine, Hutchinson's Adventure-Story and Hutchinson's Mystery-Story. The German fantasy magazine Der Orchideengarten had a similar format to American pulp magazines, in that it was printed on rough pulp paper and heavily illustrated.


Balancing columns can sometimes mean that related areas of content, like this box, become separated.

```
.columns__break {
  break-inside : avoid; }
```

¹⁹ Using the `break-inside` property helps to keep related content together.

Street & Smith, a dime novel and boys' weekly publisher, was next on the market. Seeing *Argosy's* success, they launched *The Popular Magazine* in 1903, which they billed as the "biggest magazine in the world" by virtue of its being two pages (the interior sides of the front and back cover) longer than *Argosy*.

Haggard's *Lost World* genre influenced several key pulp writers, including Edgar Rice Burroughs, Robert E. Howard, Talbot Mundy and Abraham Merritt. In 1907, the cover price rose to 15 cents and 30 pages were added to each issue; along with establishing a stable of authors for each magazine, this change proved successful and circulation began to approach that of *Argosy*.



Pulp magazines (often referred to as "the pulps") are inexpensive fiction magazines that were published from 1896 through the '50s.

Steam-powered printing

At their peak of popularity in the 1920s and 1930s, the most successful pulps could sell up to one million copies per issue. The most successful pulp magazines were *Argosy*, *Adventure*, *Blue Book* and *Short Stories*, collectively described by some pulp historians as "The Big Four." Among the best-known other titles of this period were *Amazing Stories*, *Black Mask*, *Dime Detective*, *Flying Aces*, *Horror Stories*, *Love Story Magazine*, *Marvel Tales*, *Oriental Stories*, *Planet Stories*, *Spicy Detective*, *Startling Stories*, *Thrilling Wonder Stories*, *Unknown*, *Weird Tales* and *Western Story Magazine*.

Although pulp magazines were primarily an American phenomenon, there were also a number of British pulp magazines published between the Edwardian era and World War II. Notable UK pulps included *Pall Mall Magazine*, *The Novel Magazine*, *Cassell's Magazine*, *The Story-Teller*, *The Sovereign Magazine*, *Hutchinson's Adventure-Story* and *Hutchinson's Mystery-Story*. The German fantasy magazine *Der Orchideengarten* had a similar format to American pulp magazines, in that it was printed on rough pulp paper and heavily illustrated.

Sadly, at the time of writing, we need to take a less convenient approach to ensure elements stay together in all browsers. This involves using three different properties: the first for Blink and WebKit-based browsers (including Google Chrome, Opera and Safari); the second for Firefox; and the third for Internet Explorer versions 10 and 11:

```
.columns__break {
  -webkit-column-break-inside : avoid;
  page-break-inside : avoid;
  break-inside : avoid; }
```

`columns__span` and `break-inside` are both useful properties that can help solve readability issues when using CSS multicolumn layout on passages of long body copy. But the best way to ensure that columns improve a person's reading experience and don't spoil it is careful planning and a little imagination.

Breaking lists into columns

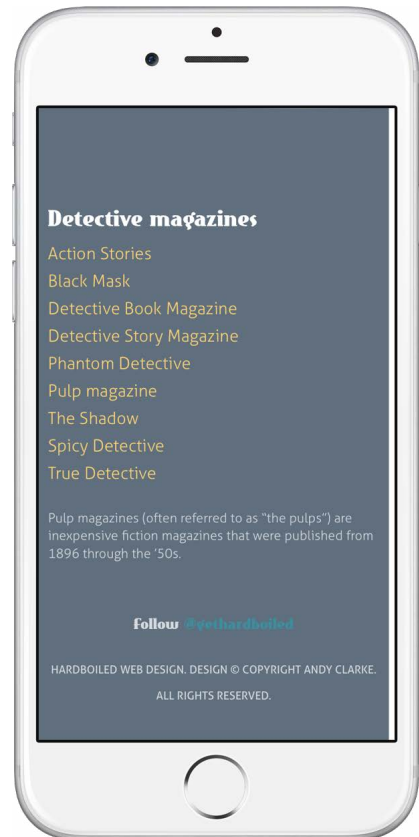
It shouldn't take too much to imagine that we can also use CSS columns to improve both the look and usability of content modules as well as long body copy. In fact, columns are incredibly useful for making the most of what would otherwise be empty space in a layout at certain responsive breakpoints.

One example that immediately springs to mind is a list, and ours contains the names of famous pulp detective magazines.

```
<ul class="list--columns">
<li>Action Stories</li>
<li>Black Mask</li>
<li>Detective Book Magazine</li>
<li>Detective Story Magazine</li>
<li>Phantom Detective</li>
<li>Pulp Magazine</li>
<li>The Shadow</li>
<li>Spicy Detective</li>
<li>True Detective</li>
</ul>
```

We don't need much styling to make the design of this list appropriate for smaller screens. Its default, vertical layout is perfect.

Displaying our list of pulp detective magazines on a smaller screen.



That same vertical layout is also perfectly suited when our list is displayed in a narrow column, such as a sidebar within a larger screen layout. However, on medium-sized screens — including smartphones in landscape orientation and tablets of various sizes — this vertical layout creates a large amount of white space. This isn't the best use of what could be limited space, so let's combine CSS multicolumn layout with media queries to improve it.

We've already added a `list--columns` class attribute value to our list and we don't need any special styling for smaller screens. When someone's browser is of medium size, we'll divide our list into three columns. A `column-rule` might not be appropriate, but `column-gap` will help separate list items, particularly those that contain longer text :

```
@media (min-width: 48rem) {  
  .list--columns {  
    column-count : 3;  
    column-gap : 4rem; }  
}
```



Making better use of the available space on medium-sized screens.

Our list now extends across what would have otherwise been empty space at this breakpoint. This layout isn't going to work at larger screen sizes, though, when its narrow column container shifts position to become a sidebar. Adapting the layout for this breakpoint is easy, however, and we'll simply reset the `column-count` to `1` using another media query:

```
@media (min-width: 64rem) {  
  .list--columns {  
    column-count : 1; }  
}
```

Now our simple list of pulp magazines shifts its layout across three responsive breakpoints to make the most of available space. For me, this type of attention to detail can make the difference between an average design and a fabulous one, and it is exactly what responsive web design should be all about.

Improving figures by adding caption columns

As someone who loves newspaper layouts, I'm often disappointed when I see unimaginative designs of figures and their captions on the web. Considering the current trend for full-width images, you might imagine that designers would be creative with their caption designs. Sadly, most designs stick to the conventional format of image first, single column caption second.

We've already seen how using flexbox to change the position of captions in relation to images can make an enormous impact on the design of figures. Now we can go a step further and make them more distinctive using columns.

We won't need to do anything different with our figure's markup except add a `figcaption__columns` class attribute value to our `figcaption`:

```
<figure>
  
  <figcaption class="figcaption__columns">Hardboiled heroes are
almost always down at heel, usually broke, often drunk and liv-
ing on a diet of black coffee and smokes - hey, that sounds like
most web designers I know. They have a good woman to help them
stay on the straight and narrow but don't always treat her as
well as they should. When a glamorous redhead walks in the room,
a hardboiled hero can't help but turn his head.</figcaption>
</figure>
```



Hardboiled heroes are almost always down at heel, usually broke, often drunk and living on a diet of black coffee and smokes—hey, that sounds like most web designers I know. They have a good woman to help them stay on the straight and narrow but don't always treat her as well as they should. When a glamorous redhead walks in the room, a hardboiled hero can't help but turn his head.

The line length in this figure's caption makes it difficult to read comfortably.

At first glance, that `figure` looks acceptable, but look a little closer and you might notice that the smaller size of the `figcaption` means that there are a lot of characters per line. More than I'd find comfortable to read, even for a few lines. We could increase the size of the `figcaption` text to adjust the measure, but that may make it visually indistinct from normal body copy. Instead, we'll use `columns` to improve the measure while keeping the text size the same.

This time, we'll specify that our columns should be 32rem wide and the browser will create as many as it can fit within the available space. Smaller screens won't benefit from columns, so we'll introduce them at medium screen sizes using a media query:

```
@media (min-width: 48rem) {  
  .figure--classic figcaption {  
    column-width : 32rem;  
    column-gap : 4rem; }  
}
```



Dividing the caption text into columns makes it more comfortable to read and more visually interesting.

Hardboiled heroes are almost always down at heel, usually broke, often drunk and living on a diet of black coffee and smokes—they, that sounds like most web designers I know. They have a good woman

to help them stay on the straight and narrow but don't always treat her as well as they should. When a glamorous redhead walks in the room, a hardboiled hero can't help but turn his head.

We've given our `figcaption` column gaps that are 4rem wide, and to add an extra layer of visual interest at larger screen sizes, we'll also add that same 4rem as a margin to the left of the `figcaption`:

```
@media (min-width: 64rem) {  
  .figure--classic figcaption {  
    margin-left : 4rem; }  
}
```



Hardboiled heroes are almost always down at heel, usually broke, often drunk and living on a diet of black coffee and smokes—hey, that sounds like most web designers I know. They have a good

always treat her as well as they should. When a glamorous redhead walks in the room, a hardboiled hero can't help but turn his head.

Adding a little margin equal to the gap gives this figure a more interesting look.

Developing for older browsers

Up until now, we've concentrated on developing for contemporary browsers that all support CSS multicolumn layout, but what about older browsers that don't support them? What should we do? The answer couldn't be simpler. Nothing. You needn't do anything because browsers without column support will ignore their styles and instead display a single column of text. That might seem a little too hardboiled, but it's fair because people using those less capable browsers won't know what they're missing.

We could use the `@supports` CSS feature query to adjust a caption's text size between those browsers that do and those that don't support columns:

```
.figure--classic figcaption {
  font-size : 1.6rem; }

@supports ( column-width : 32rem ) {
  .figure--classic figcaption {
    font-size : 1.4rem; }
}
```


However, it's unlikely that `@supports` will be understood by older browsers with no support for CSS columns. Instead, when a design dictates that we must differentiate between browsers, we can use Modernizr to detect support and then serve alternative versions of our design.

As we're only interested in serving alternatives to browsers that don't support CSS multicolumn layout, we'll use Modernizr's detected class attribute value (`.no-csscolumns`) to quarantine these styles from all other browsers, like this:

```
.no-csscolumns {  
  .figure--classic figcaption {  
    font-size : 1.6rem; }  
}
```

Breaking it up

For years I've been disappointed that so few web designers and developers use CSS columns, because for just as long they've had good support across contemporary browsers. They've also always degraded beautifully in browsers that don't implement them. While I understand that people cite usability issues as reasons not to adopt columns, I think that all it takes is a little careful planning and imagination to overcome these problems and make columnised designs that are distinctive and more interesting. I hope that I've inspired you to use CSS columns and that your next design will make good use of them.

That was a breeze

In **More Hardboiled CSS**, background blends and filters added depth to our designs. Transforms translated, scaled, rotated and skewed elements in not two but three dimensions, creating designs that were previously not possible using CSS alone. CSS transitions made state changes smoother and simple animations possible and brought designs to life in the best browsers. Finally, CSS multicolumn layout made our typography responsive to many screen sizes and type of devices.

It's time to get hardboiled

No. 21

I HAVE A CONFESSION TO MAKE. When I sat down to update this book for the fifth anniversary edition I expected the process to take only a few weeks. When I asked several of my friends to reread the first edition and to tell me which aspects needed updating, they all told me how well the book had stood the test of time. I understood that I'd have to update the examples and lessons. I expected to make dozens of new images. I also knew there was content I wanted to replace, because I thought that a chapter teaching flexbox was more important than one that taught CSS animations. I expected all of this, but I wasn't prepared for the scale of the changes I have actually made for this new book. Instead of the two new chapters I anticipated, there are five. I haven't replaced dozens of images, I've made over 350. Every example and lesson needed updating to bring it up to date with today's responsive code and techniques.

I underestimated all of this because I hadn't appreciated just how different the work we do to make the web is today from what it was when I wrote *Hardboiled Web Design* five years ago.

The websites and applications we design today must be responsive by default and we should design them to adapt to any number of screen sizes and types of device. Performance is more of an issue today than it has been since the days of the 28k modem, and designers, developers and the people we work for need to understand the importance that speed plays in improving both user experience and results.

These changes aren't easy to deal with; they require us to think differently about the processes and tools we use to design. The closer collaboration we need between designers and developers means that many creative and technical companies need not only think differently about how they structure their projects, but how they structure their entire businesses. The processes we now commonly use when designing — processes like designing atoms and elements and building web design style guides — mean our bosses and clients have had to change their expectations about the stages a project will pass through. None of this change happened overnight; it changed gradually and cumulatively, but the results have been the same. The websites and applications we make and how we make them are different and this has affected everyone involved in designing and developing for the web.

To help us cope with the changes in our industry, many of us now rely on familiar responsive design patterns. When we're designing for an unpredictable world, we crave predictability, so we've created processes to help make our work more reliable. A process is a tried and tested method we've used to do something we've done before. But what's the point of following a formula? A formula will lead to a predictable, but ordinary result — and who wants to make something ordinary? I hope you don't.

For the hardboiled heroes in the detective stories that I love, rules are there to be remade when that means going to work, catching the killer, seeing that justice is done by any means necessary. As web professionals, jamming a pistol into a guy's temple or ramming a fist into his guts isn't all part of a day's work, but we can learn a lot from those hardboiled heroes.

We don't have to follow rules for making high performance, responsive websites and applications. And even when we do, who makes those rules? We do. Who lives by those rules? We do. Whose responsibility is it to make damn sure that the work we do on the web is the best that it can be, so that everyone — we, our bosses and clients and their customers — will benefit? It's ours. It's what we're paid to do. It should be our passion.

Some people say that formative and summative research, qualitative and quantitative data and analysis, psychology, anthropology and human-computer interaction, wireframes, prototypes, functional specifications and flowcharts, are more important than creativity. To hell with that. This attitude makes many of us think that efficient processes are more important than ideas. The processes we adopt and the technologies we use are there to help us express our ideas, not limit them.

These technologies are now even more powerful than they were five years ago and we're fortunate to have contemporary browsers that support them faster and more consistently than ever before. We're lucky to have clients who now not only expect our designs to be responsive, they understand that websites needn't look the same in every browser. All these factors are opportunities to be more creative, opportunities for better business.

The time's never been better to seize those opportunities, grab them with both hands — and get hardboiled.