



A BOOK APART



Tim Brown

FLEXIBLE TYPESETTING

FOREWORD BY Jessica Hische

MORE FROM A BOOK APART

Going Offline

Jeremy Keith

Conversational Design

Erika Hall

The New CSS Layout

Rachel Andrew

Accessibility for Everyone

Laura Kalbag

Practical Design Discovery

Dan Brown

Demystifying Public Speaking

Lara Hogan

JavaScript for Web Designers

Mat Marquis

Practical SVG

Chris Coyier

Design for Real Life

Eric Meyer & Sara Wachter-Boettcher

Git for Humans

David Demaree

Visit abookapart.com for our full list of titles.

Copyright © 2018 Tim Brown
All rights reserved

Publisher: Jeffrey Zeldman
Designer: Jason Santa Maria
Executive Director: Katel LeDû
Managing Editor: Tina Lee
Editor: Caren Litherland
Technical Editors: Juliette Cezzar and Ray Schwartz
Copyeditor: Katel LeDû
Proofreader: Katel LeDû
Book Producer: Ron Bilodeau

ISBN: 978-1-937557-71-3

A Book Apart
New York, New York
<http://abookapart.com>

TABLE OF CONTENTS

1	<i>Introduction</i>
5	CHAPTER 1 What Is Typesetting?
24	CHAPTER 2 Preparing Text and Code
55	CHAPTER 3 Selecting Typefaces
86	CHAPTER 4 Shaping Text Blocks
143	CHAPTER 5 Crafting Compositions
174	CHAPTER 6 Relieving Pressure
205	<i>Conclusion</i>
207	<i>Acknowledgements</i>
210	<i>Resources</i>
212	<i>References</i>
218	<i>Index</i>

FOREWORD

I SPEND MOST OF MY TIME working as a lettering artist and illustrator, but I cut my professional creative teeth learning and practicing graphic design—book design in particular. When I say “book design,” you might immediately think of a book cover. Perhaps you encountered one recently that’s still fresh in your mind, or maybe there’s a book on your shelf that’s remained in your life for years.

I love the power of a good book cover—it entices you from across the room. With just a glance, you feel a flutter of excitement about what lies ahead when you inevitably open the book and immerse yourself in the text. The cover design can feel like the most impactful part of a book project—the one aspect of it that people will remember, that will drive sales, that will earn recognition as a designer. But it’s the seemingly invisible work that leaves the biggest impact (from the cover to everything in between).

As a reader, you know the importance of well formatted, expertly styled text, whether or not you’ve ever stopped to think about it. In fiction and other literary texts, good typesetting makes the act of reading recede into the background—you’re not scanning letters on a page, you’re visualizing worlds. In manuals, articles, labels, and other workaday texts, skillful typesetting allows readers to absorb information quickly and reference it easily in the future. Good typesetting is often invisible.

In this book, Tim sets out not only to celebrate the invisible but important role of good typesetting for the web; he also he introduces us to the world of typography-driven design. He explains complex typographic concepts—from overarching principles to practical and immediately applicable tips and tricks—without condescension, encouraging us all the while. By the end, I hope you will have fallen in love with this wonderfully arduous, invisibly important, indulgently nerdy world and work.

—Jessica Hische

INTRODUCTION

THANKS FOR READING THIS BOOK. I wrote it for two reasons. First, I want to help you understand and enjoy typesetting—it's the most important part of typography, and a secret cornerstone of flexible, responsive design. I want to help you hone your skills and mentally organize the decisions you make, so that as you prepare text for reading, you'll do it in a methodical, meaningful way.

Second, I wrote this book to illustrate how the web has transformed typography. As I explain in Chapter 1, the web has catalyzed the single biggest change in typographic history—a change that has profoundly affected not only the experience of reading but also the practice of designing that experience: typesetting.

Granted, typography and typesetting have changed before. These practices have existed for centuries, over which time new technologies, new media, and new ideas have bloomed, flourished, and withered. And through each season, old-growth customs persist. Our traditions are deep and lush. However, the web is neither a new technology, nor a new medium, nor a new idea; the web represents an evolutionary step for all technologies, all media, and all ideas.

I'm not asking you to forget the past—on the contrary. The web has changed typography, certainly, but only by returning to our typographic roots can we truly understand how flexible design works, and how best to serve readers.

Who will benefit from this book

This book will be most helpful for web designers and developers. If you code, but don't have a background in design, that's okay! I'll explain how to look at your work with a designer's eye. If, on the other hand, you design in software like WordPress or prototyping tools, where coding is minimal, that's okay too! I'll keep the code we write in this book manageable, with examples you can review in a web browser (more about that in a minute).

Students and teachers, I hope this book will help bring the web to your typography instruction—and strengthen the typographic curriculum in your web courses. Bosses and managers, I hope this text will help you understand the challenges faced by your smartest, most talented designers as they try to do beautiful work with text.

I also wrote this book for people who do print-based work. Before I knew anything about the web, I studied and practiced typography for print—and wow, did I love it. In college, I worked part-time in the publications office, laying out posters and small books. I knew all the QuarkXPress key commands. I would jog back and forth to the print shop to check on jobs and improve my prepress skills. The web demands a similar working knowledge, but we need to recognize that the differences between web and print typography involve more than technical know-how—beyond that, there is something inherently difficult about designing for the web. This book attempts to shed light on that difficulty.

Examples

Throughout this book, we'll use a website I made for the St. Remy Volunteer Fire Department (where, incidentally, I'm a firefighter) as an example project.

You can see the finished site online at stremyfd.com, but we'll edit in-progress snapshots of its code to focus on each step. Those snapshots are all on CodePen (<http://bkaprt.com/ft/00-01/>). You'll get the most out of this book if you follow along and edit the snapshots with me. CodePen is free; I recommend signing up so you can save any changes you make. You can view and edit code just fine without an account, though. In case you need to brush up on HTML and CSS to help you follow along here, I recommend *Learning Web Design* by Jen Robbins for a solid introduction.

Also, while you're getting set up, make sure you have Google's Chrome web browser installed (<http://bkaprt.com/ft/00-02/>). For studying and editing CodePen snapshots throughout the book, you can use any modern browser; for layout review in the later chapters, however, we'll use Chrome's powerful Developer Tools. (Note: these tools are not available in mobile versions of Chrome, so you'll need to use a laptop or desktop computer.)

The St. Remy site offers a simple, concrete example to help us focus on typesetting basics, but it's just that—an example. Even when not directly applicable, the advice in this book is worth considering for many different kinds of websites, as well as apps, virtual- and augmented-reality environments, and design systems.

The workflow presented in this book is also just an example. It's not a prescription for typesetting, but rather a walk-through of my process, designed to illuminate the nature of flexible typography. We'll begin by studying the philosophical and technological underpinnings of typesetting in Chapters 1 and 2. Then, in Chapters 3 through 5, we'll develop a strong awareness of balance and relationships in text and layouts.

Finally, in Chapter 6, I'll introduce a metaphorical ladder to show how the book's process can be used recursively for improvement and refinement (**FIG 0.1**). In fact, if you have a finished website you want to improve, you might consider starting with Chapter 6; its patterns identify many common pressures that crop up in the reading experience and will lead you to relevant parts of earlier chapters.

However you choose to begin, I hope you'll read this book again and again. I hope it will encourage you to do the hard work great typesetting requires, foster your appreciation for our practice, guide your contributions to the craft, improve your day-to-day design work, and maybe even help you see the web in a new way.

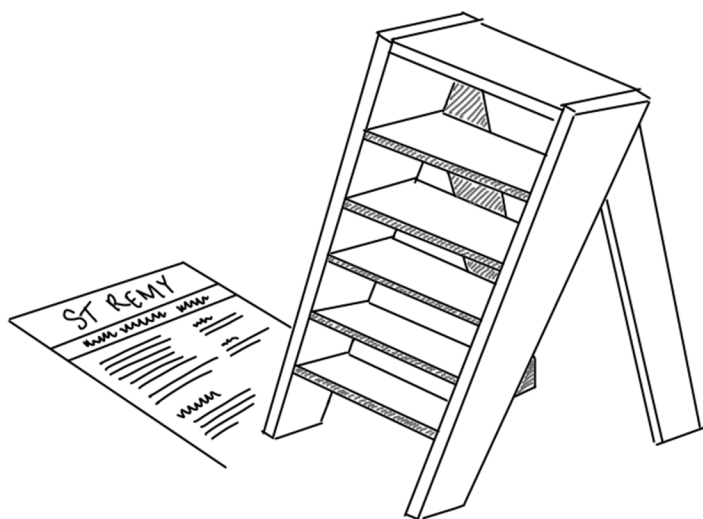


FIG 0.1: Climb up and down the “ladder” to evaluate pressure and see how typographic adjustments relate to one another.

1 WHAT IS TYPESETTING?

TYPESETTING IS THE MOST IMPORTANT part of typography, because most text is meant to be read, and typesetting involves preparing text for reading.

You're already great at typesetting. Think about it. You choose good typefaces. You determine font sizes and line spacing. You decide on the margins that surround text elements. You set media query breakpoints. All of that is typesetting.

Maybe you're thinking, *But Tim, I am a font muggins. Help me make better decisions!* Relax. You make better decisions than you realize. Some people will try to make you feel inferior; ignore them. Your intuition is good. Practice, and your skills will improve. Make a few solid decisions; then build on them. I'll help you get started.

In this chapter, I'll identify the value of typesetting and its place within the practice of typography. I'll talk about *pressure*, a concept I use throughout this book to explain why typeset texts sometimes feel awkward or wrong. I'll also discuss how typesetting for the web differs from traditional typesetting.

WHY DOES TYPESETTING MATTER?

Typesetting shows readers you care. If your work looks good and feels right, people will stick around—not only because the typography is comfortable and familiar, but also because you show your audience respect by giving their experience your serious attention (**FIG 1.1**).

Sure, you could buy the “it” font of the moment (you know, the font all the cool people are talking about). You could use a template that promises good typography. You could use a script that spiffs up small typographic details. None of these things is necessarily bad in and of itself.

But when you take shortcuts, you miss opportunities to care about your readers, the text in your charge, and the practice of typography, all of which are worthwhile investments. Spending time on these things can feel overwhelming, but the more you do it, the easier and more fun it becomes. And you can avoid feeling overwhelmed by focusing on the jobs type does.

Imagine yourself in a peaceful garden. You feel the soft sun on your arms, and take a deep breath of fresh, clean air. The smell of flowers makes you feel happy. You hear honeybees hard at work, water trickling in a nearby brook, and birds singing. Now imagine that this garden needs a website, and you’re trying to find the right typeface.

Sorry to spoil the moment! But hey, if you do this right, the website could give people the same amazing feeling as sitting in the garden itself.

If you’re anything like me, your first instinct will be to recall sensations from the imaginary garden and look for a typeface with shapes that evoke similar sensations. But this is not a good way to choose among thousands upon thousands of fonts, because it’s too easy to end up with typefaces that—as charming as they may seem at first—don’t do their jobs. You’ll get disappointed and go right back to relying on shortcuts.

Finding typefaces that are appropriate for a project, and that evoke the right mood, is easier and more effective if you know they’re good at the jobs you need them to do. The trick is to eliminate type that won’t do the job well (**FIG 1.2**).

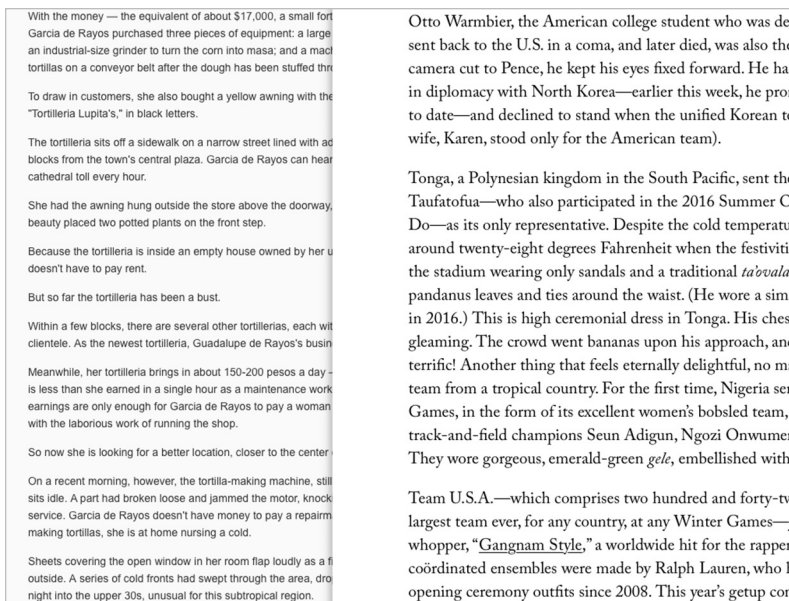


FIG 1.1: Glance at these two screenshots. Which one would you rather read? Which publisher do you think cares more about your experience?

Less Fanfare

Then on the Monday came news that Padraic Pearse had surrendered, and that the Commandants under him were accepting the order, though reluctantly. The first week's strain was released, but the mood of the people began to make a slow change, such a change as Pearse had foreseen. Already in the first week that change has appeared; but the news now told of defeat, an ancient tale in Ireland, full of old honour. On Tuesday the mail was resumed. Papers came and were passed eagerly from hand to hand. The people were afraid, but sullen.

FIG 1.2: Hatch, a typeface by Mark Caneso, is fun to use large, but not a good choice for body text.

Depending on the job, some typefaces work better than others—and some don't work well at all. Detailed, ornate type is not the best choice for body text, just as traditional text typefaces are not great for signage and user interfaces. Sleek, geometric fonts can make small text hard to read. I'll come back to this at the beginning of Chapter 3.

Considering these different jobs helps you make better design decisions, whether you're selecting typefaces, tending to typographic details, or making text and layout feel balanced. We'll do all of that in this book.

Typesetting covers type's most important jobs

Typesetting, or the act of *setting type*, consists of typographic jobs that form the backbone of a reading experience: body text (paragraphs, lists, subheads) and small text (such as captions and asides). These are type's most important jobs. The other parts of typography—which I call *arranging* and *calibrating* type—exist to bring people to the typeset text, so they can read and gather information (FIG 1.3).

Let's go over these categories of typographic jobs one by one. *Setting type* well makes it easy for people to read and comprehend textual information. It covers jobs like paragraphs, subheads, lists, and captions. *Arranging type* turns visitors and passersby into readers, by catching their attention in an expressive, visual way. It's for jobs like large headlines, titles, calls to action, and "hero" areas. *Calibrating type* helps people scan and process complicated information, and find their way, by being clear and organized. This is for jobs like tabular data, navigation systems, infographics, math, and code.

Arranging and calibrating type, and the jobs they facilitate, are extremely important, but I won't spend much time discussing them in this book except to put them in context and explain where in my process I usually give them attention. They deserve their own dedicated texts. This book focuses specifically on setting type, for several reasons.

First, typesetting is critical to the success of our projects. Although the decisions we make while typesetting are subtle almost to the point of being unnoticeable, they add up to give

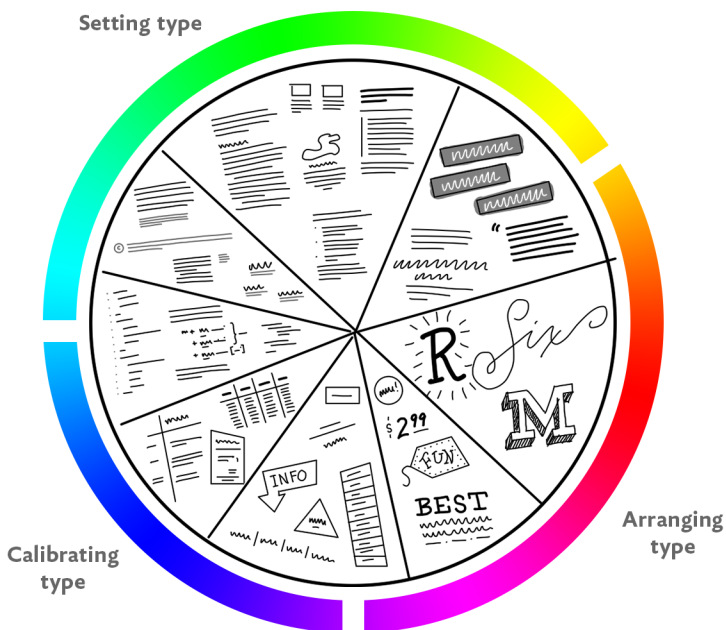


FIG 1.3: Think of these typographic activities as job categories. In Chapter 3, we'll identify the text blocks in our example project and the jobs they need to do.

readers a gut feeling about the work. Typesetting lays a strong foundation for everything else.

It also happens to be more difficult than other parts of typography. Good type for typesetting is harder to find than good type for other activities. Good typesetting decisions are harder to make than decisions about arranging type or calibrating type.

Furthermore, typesetting can help us deeply understand the web's inherent flexibility, which responsive web design has called attention to so well. The main reason I make a distinction between typesetting, arranging type, and calibrating type is because these different activities each require text to *flex* in different ways.

In sum, typesetting matters because it is critical for readers, it supports other typographic activities, the difficult decisions

informing it take practice, and its nature can help us understand flexibility and responsiveness on the web. A command of typesetting makes us better designers.

Why do some websites feel wrong?

It's not hard to find websites that just feel, well, sort of wrong. They're everywhere. The type they use is not good, the font size is too small (or too big), lines of text are too long (or comically short), line spacing is too loose or too tight, margins are either too small or way too big, and so on (FIG 1.4).

It's logical to think that websites feel wrong because, somewhere along the line, a typographer made bad decisions. Remember that a *type designer* is someone who makes type; a *typographer* is someone who uses type to communicate. In that sense, we are all typographers, even if we think of what we do as designing, or developing, or editing.

For more than 500 years, the job of a typographer has been to decide how text works and looks, and over those years, typographers have made some beautiful stuff. So if some websites feel wrong, it must be because the typographers who worked on them were inexperienced, or lazy, or had no regard for typographic history. Right?

Except that even the best typographers, who have years of experience, who have chosen a good typeface for the job at hand, who have made great typesetting decisions, who work hard and respect tradition—even those people can produce websites that feel wrong. Websites just seem to look awful in one way or another, and it's hard to say why. Something's just not quite right. In all likelihood, it's the typesetting. Specifically, websites feel wrong when they put *pressure* on typographic relationships.

Typographic relationships

Have you ever chosen a new font for your blog template, or an existing project, and instinctively adjusted the font size or line spacing to make it feel better?

ve persons at the table, and you hinder them all
et a proper value upon time, this is a great evil.
a meal, never wait to finish what you are doing,
and so sit down to your food like a heathen. Th
isy clamor. The younger members of the family
It does not appear well for a very young person
wish to make an inquiry or a remark, do it in a m
reful not to interrupt any other person. Sensible
e at table. Yet you should never appear inattent
to observe the movements of others, or to hear

waste fifteen minutes of pre
er value upon time, this is a
may as easily be at your se
meal, never wait to finish w
e it, and proceed to your pla
the blessing, and so sit do
table is a place for easy, ch

FIG 1.4: Some typesetting just looks wrong. Why? Keep reading.

FAR FROM AN IRISHMAN'S
EMOTIONAL CONSCIOUSNESS

Such were the days of an anxious week. None knew what to believe, what to trust, or what to distrust. Work was impossible. Sleep even was almost impossible. We could but drift about and wait, when to do so seemed almost like a tragic cowardice. What proved finally to be well-grounded of the rumours that flew were disbelieved. What proved to be false were the only matters in which any reliance was placed. None doubted, for instance, that Cork and Limerick were "up," or that Wexford County was in a blaze, or that Ballina, quite near home, had captured Killala Bay.

None placed much reliance in the rumours of fierce fighting round Boland's Mill and Jacob's Factory. None doubted that Athlone Bridge had been blown up and that the Galway boys were retreating from the town, contesting every foot of the way against a large English force. None believed in the landing and capture of Casement. One of the county papers published a special edition on Thursday recording all the rumours. "The Mayo News," however, refused in its edition on the Saturday to print or give

FAR FROM AN IRISHMAN'S
EMOTIONAL CONSCIOUSNESS

Such were the days of an anxious week. None knew what to believe, what to trust, or what to distrust. Work was impossible. Sleep even was almost impossible. We could but drift about and wait, when to do so seemed almost like a tragic cowardice. What proved finally to be well-grounded of the rumours that flew were disbelieved. What proved to be false were the only matters in which any reliance was placed. None doubted, for instance, that Cork and Limerick were "up," or that Wexford County was in a blaze, or that Ballina, quite near home, had captured Killala Bay.

None placed much reliance in the rumours of fierce fighting round Boland's Mill and Jacob's Factory. None doubted that Athlone Bridge had been blown up and that the Galway boys were retreating from the town, contesting every foot of the way against a large English force. None believed in the landing and capture of Casement. One of the county papers published a special edition on Thursday recording all the rumours. "The Mayo News," however, refused in its edition on the Saturday to print or give

FAR FROM AN IRISHMAN'S
EMOTIONAL
CONSCIOUSNESS

Such were the days of an anxious week. None knew what to believe, what to trust, or what to distrust. Work was impossible. Sleep even was almost impossible. We could but drift about and wait, when to do so seemed almost like a tragic cowardice. What proved finally to be well-grounded of the rumours that flew were disbelieved. What proved to be false were the only matters in which any reliance was placed. None doubted, for instance, that Cork and Limerick were "up," or that Wexford County was in a blaze, or that Ballina, quite near home, had captured Killala Bay.

None placed much reliance in the rumours of fierce fighting round Boland's Mill and Jacob's Factory. None doubted that Athlone Bridge had been blown up and that the Galway boys were retreating from the town, contesting every foot of the way against a large English force. None believed in the

Franklin Libre (default)

Kepler

Kepler, adjusted

FIG 1.5: Replacing this theme's default font with Kepler made the text seem too small. Size and line-spacing adjustments felt necessary.

Those typesetting adjustments help because the typeface itself, as well as its font size, *measure* (a typographic term for the length of lines of text), and line spacing all work together to make a text block feel balanced. (We'll return to text blocks in more detail in Chapter 3.) This balance is something we all instinctively notice; when it's disrupted, we sense pressure.

But let's continue for a moment with this example of choosing a new font. We sense pressure every time we choose a new font. Why? Because each typeface is sized and positioned in unique ways by its designer (**FIG 1.6**).

In Chapter 2, we'll take a closer look at *glyphs*, which are instances of one or more characters. For now, suffice it to say that glyphs live within a bounding box called the *em box*, which is a built-in part of a font file. Type designers decide how big, small, narrow, or wide glyphs are, and where they are positioned, within this box. The em box is what becomes our CSS-specified font size—it maps to the CSS content area.

So when we select a new typeface, the visible font size of our *text block*—the chunk of text to which we are applying styles—often changes, throwing off its balance. This means we need to carefully adjust the font size and then the measure, which depends on both the typeface and the font size. Finally, we adjust line spacing, which depends on the typeface, font size, and measure. I'll cover how to fine-tune all of these adjustments in Chapter 4.

Making so many careful adjustments to one measly text block seems quite disruptive, doesn't it? Especially because the finest typographic examples in history—the work we admire, the work that endures—commands a compositional balance. *Composition*, of course, refers to a work of art or design in its entirety. Every text block, every shape, every space in a composition relates to another. If one text block is off-kilter, the whole work suffers.

I'm sure you can see where I'm headed with this. The web puts constant pressure on text blocks, easily disrupting their balance in myriad ways.

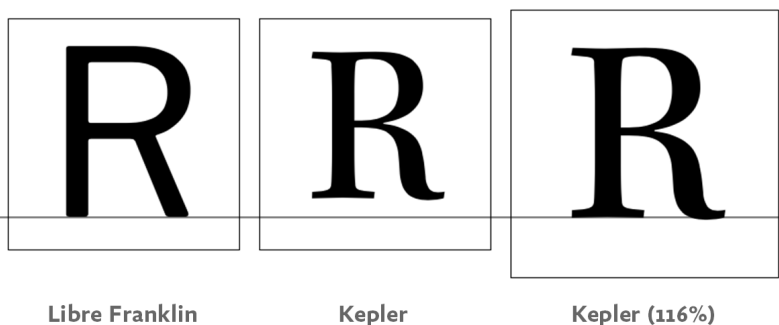


FIG 1.6: Glyphs are sized and positioned within a font’s em box. When we set a font size, we are sizing the em box—not the glyph inside it.

Pressure

There are no “correct” fonts, font sizes, measures, or line heights. But relationships among these aspects of a text block determine whether reading is easier or harder. Outside forces can apply pressure to a balanced, easy-to-read text block, making the typesetting feel wrong, and thus interfering with reading.

We just discussed how choosing a new typeface introduces pressure. The same thing happens when our sites use local fonts that could be different for each reader, or when webfonts fail to load and our text is styled with fallback fonts. Typefaces are not interchangeable. When they change, they cause pressure that we have to work hard to relieve.

We also experience pressure when the font size changes (**FIG 1.7**). Sometimes, when we’re designing sites, we increase font size to better fill large *viewports*—the viewing area on our screens—or decrease it to better fit small ones. Readers can even get involved, by increasing or decreasing font size themselves to make text more legible. When font size changes, we have to consider whether our typeface, measure, and line spacing are still appropriate.

Changes to the width of our text block also introduce pressure (**FIG 1.8**). When text blocks stretch across very wide screens, or are squeezed into very narrow viewports, the entire

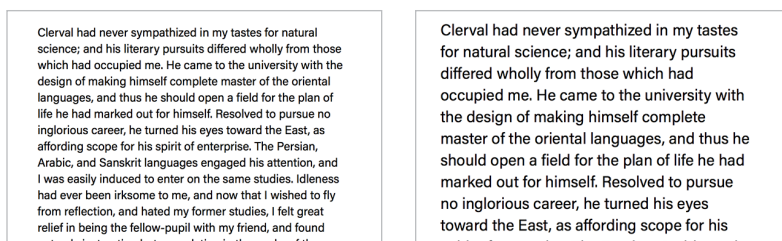


FIG 1.7: Left: a balanced text block. Right: a larger font size causes pressure.

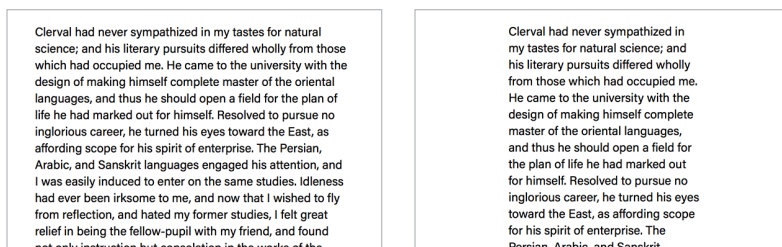


FIG 1.8: Left: a balanced text block. Right: a narrower measure causes pressure.

composition has to be reevaluated. We may find that our text blocks need new boundaries, or a different font size, or even a different typeface, to make sure they maintain a good internal balance—and feel right for the composition. (This may seem fuzzy right now, but it will become clearer in Chapters 5 and 6, I promise.)

We also experience pressure when we try to manage white space without considering the relationships in our text blocks (**FIG 1.9**). When we predetermine our line height with a baseline grid, or when we adjust the margins that surround text as if they were part of a container into which text is poured rather than an extension of the balance in the typesetting, we risk destroying relationships among compositional white spaces—not only the white spaces in text blocks (word spacing, line spacing), but also the smaller white spaces built into our typefaces. These relationships are at risk whenever a website flexes, whenever a new viewport size comes along.

Clerval had never sympathized in my tastes for natural science; and his literary pursuits differed wholly from those which had occupied me. He came to the university with the design of making himself complete master of the oriental languages, and thus he should open a field for the plan of life he had marked out for himself. Resolved to pursue no inglorious career, he turned his eyes toward the East, as affording scope for his spirit of enterprise. The Persian, Arabic, and Sanskrit languages engaged his attention, and I was easily induced to enter on the same studies. Idleness had ever been irksome to me, and now that I wished to fly from reflection, and hated my former studies, I felt great relief in being the fellow-pupil with my friend, and found not only instruction but consolation in the works of the

Clerval had never sympathized in my tastes for natural science; and his literary pursuits differed wholly from those which had occupied me. He came to the university with the design of making himself complete master of the oriental languages, and thus he should open a field for the plan of life he had marked out for himself. Resolved to pursue no inglorious career, he turned his eyes toward the East, as affording scope for his spirit of enterprise. The Persian, Arabic, and Sanskrit languages engaged his attention, and I was easily induced to enter on the same studies. Idleness had ever been irksome to me, and now that I wished to fly

FIG 1.9: Left: a balanced text block. Right: looser line spacing causes pressure.

Typesetting for the web can only be successful if it relieves inevitable pressures like these. The problem is that we can't see all of the pressures we face, and we don't yet have the means (the words, the tools) to address what we *can* see. Yet our natural response, based on centuries of typographic control, is to try to make better decisions.

But on the web, that's like trying to predict the weather. We can't decide whether to wear a raincoat a year ahead of time. What we can do is *get a raincoat* and be ready to use it under certain conditions. Typographers are now in the business of making sure text has a raincoat. We can't know when it'll be needed, and we can't force our text to wear it, but we can make recommendations based on conditional instructions.

For the first time in hundreds of years, *because of the web*, the role of the typographer has changed. We no longer decide; we make suggestions. We no longer choose typefaces, font size, line length, line spacing, and margins; we prepare and instruct text to make those choices for itself. We no longer determine page shape and quality; we respond to our readers' contexts and environments.

These changes may seem like a weakness compared to the command we have always been able to exercise. But they are in fact an incredible strength, because they mean that typeset text has the potential to fit everyone just right. In theory, at least, the web is universal.

The primary design principle underlying the web's usefulness and growth is universality.

—Tim Berners-Lee (<http://bkaprt.com/ft/01-01/>)

We must now practice a *universal* typography that strives to work for everyone. To start, we need to acknowledge that typography is *multidimensional*, *relative* to each reader, and unequivocally *optional*.

MULTIDIMENSIONAL DECISIONS

Take a minute to go find a physical book. Any book will do. Open it up and look at a paragraph. Ask yourself: *What if this paragraph were sitting on a much narrower page? How would the paragraph need to change to look good?* I wrote this book to begin to answer questions like this.

There are many such questions. *What if the paragraphs were much wider? What if the font size were much bigger, or much smaller? What if it were a different font? What if the paper looked sharp and crisp, or coarse and muddy? What if the dimensions of the page were tall and narrow? Or short and wide? Or short and narrow? Or tall and wide? What if the paper were a brighter white, or a dim off-white, or some vivid color? What if the book was so heavy it had to rest on a table?*

This isn't theoretical. People today use a variety of devices with different screens, different settings, and different preferences to read text on the web. That multiplicity of reading experiences means that many people can look at the same typeset text and see it many different ways. All of these people experience *the same typography*, but they each see something a little different than everyone else sees, and the diversity of these unique experiences will only intensify as the web matures (**FIG 1.10**).

Typography now requires us to design for many conditions at once, which is hard because our brains can't directly perceive all of those conditions simultaneously—it's like trying to see the fourth dimension (**FIG 1.11**). But although this might sound extremely complicated, designers have lots of relevant experience. From branding guidelines to advertising campaigns, designers have, as an industry, already tackled the problem of putting the same typographic message into spaces of various size and context. Our new challenge is to articulate those decisions in code. We need to prepare *our work* to answer all kinds of questions, on the fly, as it responds to, and for, readers.

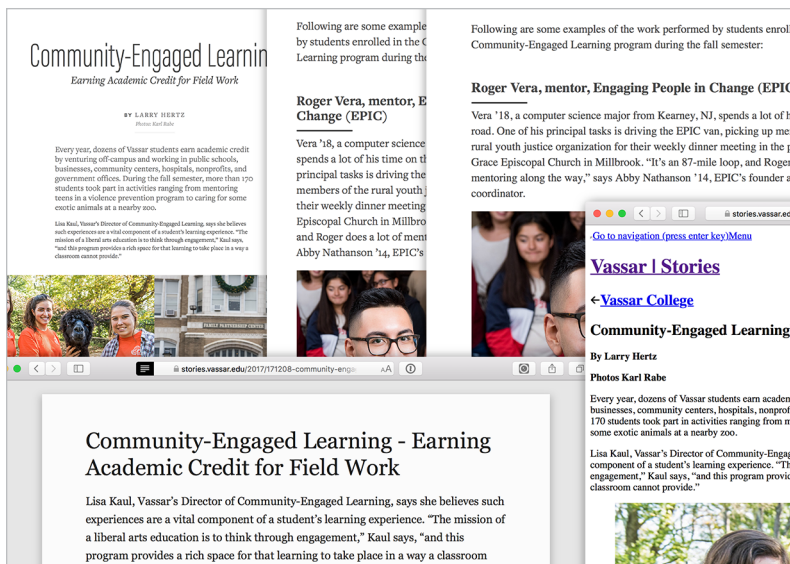


FIG 1.10: Screenshots of an article from Vassar Stories, clockwise from top left: wide layout, narrow layout, with web fonts disabled, with CSS disabled, and in Safari Reader View.

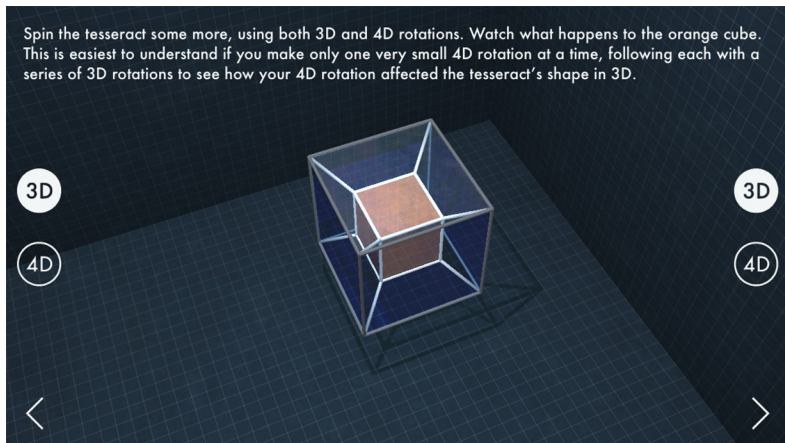


FIG 1.11: Screenshot from The Fourth Dimension (http://bkaprt.com/ft/01-02/), an iOS app with interactive diagrams that helped me understand web design. We build experiences by manipulating their shadows.

ONE WITH THE READER

Typographers don't make absolute decisions anymore. Instead, they make decisions that are relative to each reader.

Good typographers have always gotten to know their readers' contexts, such as the kind of typesetting readers are familiar with, the environment in which they do their reading, and their general level of sightedness. But the web takes this "familiarity and comfort" concept to a whole new level. Familiarity and comfort mean many different things to many different people. The devices we all use are personal. We use them all the time and are very familiar with how text generally looks. There are settings, preferences, and different software versions too, so even people with the same hardware and operating system may still see very different results.

Our decisions about typesetting need to account for this. We do that by providing logic for when conditions are not ideal, but most importantly by starting from each reader's perspective. By using any given reader's default font-size to structure our typesetting and measure our layout, we stand the best chance of making something that feels natural to each reader.

And that's important, because even if we've done all we can to respond to our readers' conditions, not all reader comfort zones will result in beautiful typography—and that's okay. If the reality of a reader's context challenges our design intentions, that's not wrong. It's not even bad typography. Because of the web, typography is now forced into spaces and situations where it *can't* look good. But it can still be beautiful if it works well, and that is good typography.

Get used to the idea that readers are in control, because it's not going away. I have wondered about this. When VR and AR advancements expand our field of view beyond the physical devices we use today, will we revert to giving print-like dimensions to our virtual texts? I don't think so, because reader expectations have already been changed. People won't want to *have* to hold a virtual piece of paper or book—they'll want to summon that text according to their preferences, with minimal effort, and have it look familiar and comfortable.

People now expect text to flex and respond to their settings and preferences.

READERS ARE TYPOGRAPHERS, TOO

Typography is now optional. That means it's okay for people to opt out. Styles for [font-size](#) and [line-height](#), styles for a site's heading hierarchy, and styles for the overall dimensions of a layout—if any of these, or any styles at all, are absent, that's okay. If a webfont file doesn't load, and a reader instead sees a fallback font that you, the designer, have never seen before, that's okay. In fact, it's better than okay.

Deep breaths!

But Tim, you might be thinking, this is nonsense! How text looks is very important. Do I seriously have to defend the value of typography to the author of a book about typography? I hear you. I know how tough it was to read that last paragraph.

Of course typography is valuable. Typography may now be optional, but that doesn't mean it's worthless. Typographic choices contribute to a text's meaning. But on the web, text itself (including its structural markup) matters most, and presentational instructions like typography take a back seat. Text loads first; typography comes later. Readers are free to ignore typographic suggestions, and often prefer to. Services like Instapaper, Pocket, and Safari's Reader View are popular partly because readers like their text the way they like it.

Formal meaning is even devalued by some *presentational* responses to the web's prioritization of semantics: designers have either felt compelled to settle for experiences easily distilled into the format of an academic text, bland tomes speckled with headings and bulleted lists; or they have railed against the web's logic by using plugins and scripts that change its priorities by force, in which case they discard the most essential, amazing aspect of the web: *that it has the potential to work for everyone by default.*

What are we going to do about this devaluation of graphic meaning besides talk? How can we act? What might we do differently as we work on our next project? How can we make

14

15

<h2>Seriously</h2>

16

Seriously

FIG 1.12: On the web, what something is—its structure—matters more than what it looks like. This is a *feature*. The fact that this is a subheading matters more than how the subheading looks.

the optional nature of web typography align with producing excellent work?

One thing we can do is practice. If typography is optional, it has to be *worth* opting into. It has to be great. Readers need to prefer it to experiences that are faster and more convenient. If you have this book, you're probably already honing your skills and working to make your typography as good as it can be for the web. In doing so, you're not only improving the quality of your work—you're moving the entire practice of typography forward.

The other thing we can do is pay attention to performance.

Performance is a word web designers and developers use to describe the speed, loading sequence, and smoothness of a web experience. In the context of typography, performance refers to the speed with which fonts and typographic suggestions load, the order in which they load, and what readers see while that stuff is happening. These factors are in part the result of typographic design decisions.

If we don't pay attention to performance, and if the experience consequently feels slow and sloppy, many people will opt. Furthermore, when we don't pay attention to performance, we're leaving important design work undone. A lot of people have spotty and/or slow internet connections, and we need to design for those experiences too.

We'll touch on performance throughout this book, from the basics of font loading to managing fallback font transitions. For a thorough guide to webfont performance—particularly when it comes to deciding how to spend a budget, and how to optimize type for production sites—read Bram Stein's *Webfont Handbook* (<http://bkaprt.com/ft/01-03/>). And to learn how webfonts fit into a site's overall performance, check out Scott Jehl's *Responsible Responsive Design* (<http://bkaprt.com/ft/01-04/>).

THIS IS ONLY THE BEGINNING

So there you have it. Typesetting matters, but often feels wrong because of pressure exerted by the web's changeable formats and reader participation. To relieve that pressure, designers need to work in a dimension human beings can't directly perceive, consult every single device on earth about our units of measurement, and plan for anyone to contribute opinions that override our own educated design suggestions.

This is fine (**FIG 1.13**). No, really. It's overwhelming and difficult, and it sounds absolutely ridiculous, but it will be okay. We'll figure this out together.

Typesetting for the web—not just sticking some beautiful, static typesetting *on the web*, but crafting something innately *of and for the web*—is difficult because we are at the beginning of a major transition in typographic history.

We used to view typography as a set of fixed decisions, but now we understand it as a continuum of conditional logic. And although we currently think of the web in terms of mobile-device screens and browser windows on personal computers, the crux of this historic transition is not about any particular medium. It's not about today's screens and software. It's about how the web has changed our relationship to text.

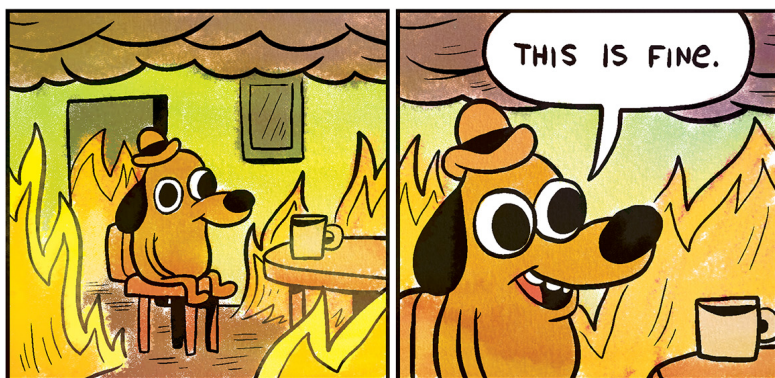


FIG 1.13: Panel from the *Gunshow* strip “On Fire” by KC Green. Reprinted with permission.

Reading on the web is not like picking up a book or a magazine. It’s not like zipping past a billboard on the highway or watching a title sequence crawl through on video. Web-based text is like all of these things at once, and we’ve come to expect it to conform to our personal preferences. We each see text on the web through our own lens, and our ability to customize it invites us to participate in how it looks and behaves. We challenge text to meet us halfway.

Typography is still evolving on the web. There are a number of ways in which it is less capable than print typography, but the gap is closing thanks, in large part, to web standards. *Web standards* are agreed-upon conventions about how things are achieved in code—like how to mark up a heading with HTML, and how to declare `font-weight` in CSS (<http://bkaprt.com/ft/01-05/>). Once upon a time, web browsers competed with features, offering proprietary code and behaving in unique ways. To design for the web back then meant understanding the eccentricities of each context and writing separate codebases for each browser. The introduction of web standards reined in those eccentricities and encouraged collaboration and compatibility among browsers.

Web standards are ours. We—designers, coders, writers—make them, and they exist for everyone’s benefit and use. They are not finished. They will never be finished. New ideas are continually incorporated when there is, as Jeff Veen likes to say, “rough consensus and running code” (<http://bkaprt.com/ft/01-06/>). Successful processes exist for making improvements. I bring this up as a reminder that you and I can participate in shaping the future.

Let me tell you a quick story.

In Chapter 4, I’ll discuss how line spacing looks better tight in narrow text blocks, and loose in wide text blocks. There is broad consensus about this—ask any experienced typographer. In CSS, we can achieve this by changing line-height with media queries, but that’s a bit heavy-handed.

This was bothering me, but then I had an idea for how to make it more fluid. I didn’t know how to script my idea, so I wrote a blog post explaining how a script might work (<http://bkaprt.com/ft/01-07/>). Sharing that idea resulted in several actual working implementations, one of which can be done with ordinary CSS (<http://bkaprt.com/ft/01-08/>). This journey led me to profound ideas about how all white space should flex (which I’ll talk about in Chapters 4 and 5)—ideas that I’m certain benefit typography now that lots of smart people have discussed one such example. That example is something we can show to people who build web browsers and craft web standards. Rough consensus and running code!

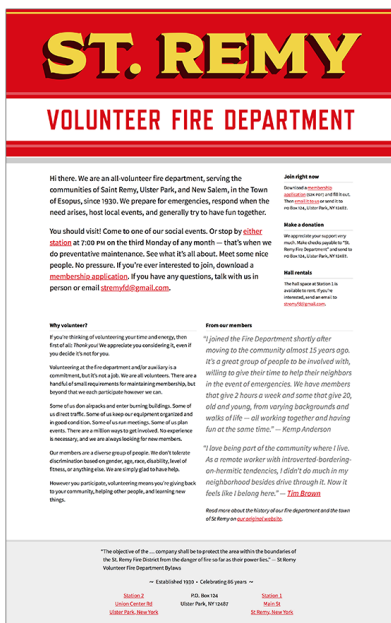
I hope this book will help you orient yourself in this historic moment, so you and I can move things forward together. Don’t worry if it seems complicated. We’ll take it nice and easy. First let’s read some text. It’s the best way to start.

2 PREPARING TEXT AND CODE

DEEP DOWN, WE KNOW THAT successful work takes care and preparation. The word “typography” may conjure up mental images of beautiful typeface specimens, or compositions full of strikingly arranged type, but much of what goes into great typography is not glamorous. No, typography largely involves the arduous work of taking responsibility for text and putting the right code foundation in place.

Let’s make this work less arduous by going over exactly what’s involved. To this end, we’ll look at a website I made for my town’s fire department, stremyfd.com (FIG 2.1). The site is a great example because it contains several different kinds of text blocks, and the lessons it imparts can be scaled up to larger sites with more robust, more complex typographic systems.

Reading and handling the text is our first step. Then we’ll lay the groundwork for typographic suggestions by resetting default styles, frame the composition, and go over basic web-font logistics. Later, in Chapters 3 through 6, we’ll select a text typeface for St. Remy, shape the site’s text blocks, craft the composition, and resolve pressures that arise. Let’s get started.



We'll focus on this part.

FIG 2.1: We'll typeset stremfyd.com from scratch, focusing on its main content area.

TAKE RESPONSIBILITY FOR THE TEXT

You care about the success of your typesetting, or you wouldn't be reading this book, so I'll give it to you straight. The text you typeset—the text itself—is also your responsibility. First, you have to care about its success *as a text*. Only then can you care for its success as a piece of typesetting.

Responsibility is a big word. You obviously can't reshape an entire project, but you can pick your battles. You can decide whether to get involved in a project at all, gauging a text's viability before deciding whether to take it on. You can put guidelines into your contracts, and make sure a competent editor is around to help.

If you're already stuck with a project, you may not be able to improve the text—but if you read and understand the content,

you can offer suggestions to the authors and editors. If you feel that some careful edits and text guidelines would make a big difference, then you can help make a case for it. Typographers—and yes, that means you—are often so close to a text that they understand it as well as anyone else involved. That understanding begins with reading.

Read the text

The text for our example project is on CodePen (<http://bkaprt.com/ft/02-01/>). Fire up your web browser of choice and put the text in front of your face.

As a general rule, if you don't have enough text, ask for more. If it's not the final version, ask for an update. If the text is going to change all the time, like news, find some representative example text and get your hands on any editorial guidelines the writers and editors will use. (I like to think of this as scaffolding for the real text.)

Then read the text. Do more than read it—*listen* to it. Grasp its reason for being. Refer to editorial guidelines (if they exist) to better understand and articulate the goals of the writing. Describe how the text makes you feel. (Writing down adjectives will help you choose, combine, and set type later.)

Read the text again. Put yourself in the reader's shoes. Allow yourself to be absorbed. Or try scanning for information—pretend you're distracted by life outside your screen. Then try to take the perspective of the author and publisher. Consider their business goals, and the day-to-day challenge of acquiring and keeping readers' attention. Take notes about your experience.

If you don't see text until very late in your process, or don't have enough time to read (all of) the text, don't be discouraged. Stick with this chapter to understand how spending more time with text can improve your typesetting. You might also consider picking up a copy of *Nicely Said* by Nicole Fenton and Kate Kiefer Lee (<http://bkaprt.com/ft/02-02/>); in particular, review the chapters on voice and tone for ideas about how to absorb the essence of a person or company.

Consider the text's structure and nuances

Take notes, too, about the structure of the text. Consider its innate hierarchy. How often do headings occur? How is the text divided into sections? Is there a lot of content in each section, or only a little? Are the sections even, in terms of their amount of content? What is the narrative like? Are some sections more powerful than others? How can the typesetting best reflect the story being told? If it's a reference text, you might ask yourself in what sequence, and with what frequency, the reader needs information. Which pieces of text are related?

Jot down observations about any special circumstances. Are there lots of very long words? Abbreviations? Parenthetical citations? How about proper nouns, with their attention-grabbing capital letters? Are there many accented letters? What languages are present? Is there any dialogue? Are there inline quotations or block quotes? What about footnotes? Any specific patterns of punctuation or math symbols?

Here are some of the notes I took while going over the St. Remy Fire Department site:

- It's personal, welcoming, informational, encouraging, brief, clear.
- As a reader, I would find this friendly. The paragraphs are short, easy to read.
- If I were the author, I'd want people read the paragraphs. Maybe the key points would be better as a bulleted list? Or could be emphasized?
- There's not much to the structure here, but the headings make sense. The opening paragraphs and "Why volunteer?" section make up the main text; everything else is supporting information that might be good for quick reference.
- Some special considerations would be block quotes for the testimonials, contact information in a couple of spots, and the location addresses.

The better you know your text, the better your typesetting will be.

Put the text in its place

Once you've read and thoughtfully considered a text, you'll know some things you didn't know before. You'll know the subject of the text and, with a little luck, you'll have a feel for the writing style. Now you can make use of this information by putting the text in its place—identifying its cultural contexts.

Find out how the subject matter has been typeset before. What aesthetic does this kind of text usually have? Look to popular resources in the field. Look to business competition. Look to history. Observe how text generally works and looks for readers who seek out the kind of information your text offers: if a text feels familiar to readers of a given genre, it stands a better chance of success.

Apart from subject matter, find parallels to your text's writing style. Look to entities in other fields that convey the same feelings your text does, relative to their own fields of concern. If you can't think of a parallel, look to the adjectives you noted as you read your text. Think about experiences—not even textual experiences, necessarily—that convey those same adjectives, and study them.

For our St. Remy project, we can look at the websites of other fire departments and volunteer organizations to start getting some ideas about an aesthetic approach (FIG 2.2). We can also study, for example, a good local restaurant, where guests feel welcome and dishes aren't expensive or complicated—perhaps there are stylistic cues or phrases worth borrowing.

HANDLE THE TEXT

We typographers are responsible for maintaining a text's integrity *as text*. Missing characters, incorrect glyphs, spacing oddities, and stylistic oversights can all play a role in undermining a text's integrity.

Let's make sure the text's meaning remains accurate by grooming it (declaring an encoding and setting its language) and marking the text up with sensitivity to typographic details. While we're at it, let's start building our example project's site.



FIG 2.2: Other fire department websites use lots of red, yellow, and black. Engaging volunteer-oriented efforts like Charity: Water feature clear, friendly copywriting.

Declare an encoding and set the language

Did you ever copy and paste text from a word processor and get a weird result? Or notice a glyph on a site that doesn't appear to be in the markup at all, like an invisible character? Maybe you've seen, in text, little boxes (sometimes with an X inside) or question marks (sometimes in a black rhombus shape). These glitches arise when a text's character string contains data that its encoding does not understand.

I've used the phrase "character string" a couple of times. A text's *character string* is simply that: a string of characters. It's what we often think of as the text itself—the selection and arrangement of unique symbols, or *characters*, that we recognize as letters, numerals, punctuation, and other marks. For example, *A* and *¢* are both characters.

Unicode is an international standard that defines every character in every known language by assigning it a hexadecimal value. For example, *A* is defined as [0041](#). *Encoding* refers to how those values get translated into characters. UTF-8 has been the web's dominant encoding since 2009 (<http://bkaprt.com/ft/02-03/>), and it's the one we'll use in this book.

Now, let's start building our example project site. Take a peek at the bare-bones starting point for our HTML. I want to highlight a few parts in particular. First, I specified UTF-8 encoding using the `charset meta` tag, immediately after the opening `head` tag in the HTML:

```
<!DOCTYPE html>
<html dir="ltr" lang="en-US">
  <head>
    <meta charset="utf-8">
    <title></title>
  </head>
  <body>
  </body>
</html>
```

Code example: <http://bkaprt.com/ft/02-04/>

It's important to use the `charset meta` tag soon after the opening `head` tag—ideally in the first byte (the first 127 characters of the markup)—so that browsers see it quickly (<http://bkaprt.com/ft/02-05/>). As soon as browsers sniff the `charset` tag, they stop parsing the page in whatever default encoding they use and start from the beginning using the encoding you've specified. By declaring an encoding quickly, we help pages load faster.

Notice that I also put a couple of attributes in the `html` tag. Using `lang` to declare a language gives speech synthesizers and search engines a head start. It also enables us to make typesetting adjustments based on language, if necessary, using CSS attribute selectors—perhaps specifying different fonts based on their language support, or different glyphs. (Some languages

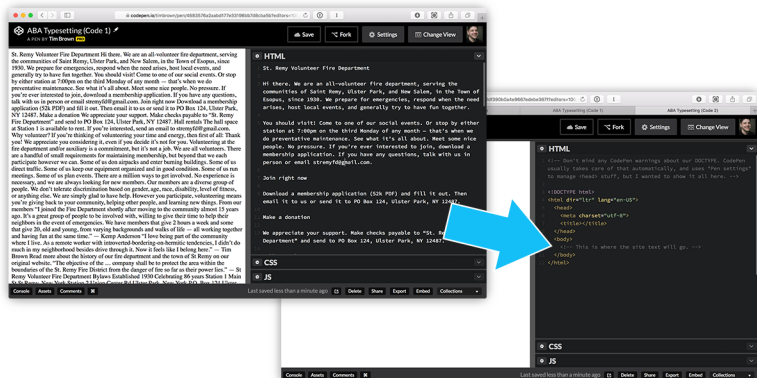


FIG 2.3: After cleaning the text up, you'll drop it into the HTML.

use «guillemets» instead of “quotation marks,” for example.) The `dir` attribute specifies text direction (`ltr` means “left to right”). Although many browsers use `ltr` by default, some might not—either now or in the future. Explicitly declaring text direction ensures that the text flows the way you want it to flow, and also serves as a reminder that text direction is intentional.

Clean up and mark up

As we continue building our example site, we first need to tidy up our text. Then we'll paste it into the bare-bones HTML we just looked at and mark it up (FIG 2.3).

Let's take this one step at a time. Head back to the text in CodePen (<http://bkaprt.com/ft/02-01/>). It's a mess! It needs your help, so start cleaning it up. (Don't worry, I made an “after” example that's all clean and neat.)

Take a look at the following punch list. Going over a set of reminders like this is a good habit to get into whenever you handle text, and you can use this as a starting point for your own text-cleanup checklist. These are recommendations, not hard-and-fast rules.

Interpret structural signals

If the text seems to use punctuation to divide itself up or mark itself up, examine this as you work. Asterisks at the beginning of sequential lines likely represent a bulleted list, while asterisks surrounding a word might mean “emphasize this,” and still another series of asterisks between paragraphs could represent a logical pause in the text. Even empty spaces might hold meaning. Comb through the whole text and resolve any ambiguities or areas where the meaning strikes you as unclear.

Keep in mind that the specific characters the text uses for structural signals may or may not be important to the text’s meaning. When in doubt, ask your editor or author for clarification. You, the typographer, are responsible for how the text looks, so be purposeful with your decisions about specific glyphs and load-bearing graphic devices. For list items, solid circles may work better than asterisks. White space and a fleuron may divide sections more tastefully than a horizontal rule.

Declutter

While you’re deciding how specific symbols and spaces influence the text’s structure, begin tending to the many small chores that a clean text requires. This is monotonous but satisfying work that gives the text consistency.

- Convert double spaces to single spaces. Come on, people: *one* space, not two, after a period.
- Remove tabs, indents, and any other invisible characters that do not belong in the text. Many text editors include a feature for making these visible, so they’re easier to spot.
- Eliminate any line breaks within running text, which are often the result of copy-and-paste problems.
- Remove hyphens in words that shouldn’t need them. Again, this is a common copy-and-paste problem.

Use the right marks

For accessing marks not visible on the keyboard, your operating system or text editor most likely includes keyboard shortcuts, or a menu of choices. You can also copy and paste from a resource like [Charcod.es](https://charcod.es), a Unicode search engine. Let's look at some marks that often need attention:

- Don't strip accents from letters, turning *é* into *e* or *ã* into *a*. If they are missing, restore diacritics to words that have been adopted, untranslated, from other languages (like *résumé* or *jalapeño*).
- Use proper copyright, trademark, and registered trademark entities instead of things like "(c)".
- Use proper math symbols, like \times instead of *x*.
- Pay attention to ellipses. Some style guides suggest converting a sequence of three periods ("...") to an ellipsis character ("…"); others propose using three spaced periods. For further advice, I recommend Robert Bringhurst's brief, thoughtful exploration of the subject, much of which hinges on the context of the typeface in use, in *Elements of Typographic Style*.
- Smarten quotation marks ("that's it" should be "that's it"), but stupefy code quotes (`alt="dog"` should be `alt="dog"`) and use primes as needed (10'6" tall should be 10'6" tall).
- Space with sensitivity. The default space character is one of several available blank glyphs that consume varying amounts of horizontal room. For more, refer to Jost Hochuli's *Detail In Typography*.
- Use an en dash (–) rather than a hyphen (-) to represent a range in strings of digits (99–103), days (Mon–Fri), dates (May 5–8), and times (10:00–11:00). Set these en dashes either unspaced or with very thin surrounding spaces.
- Convert single (-) or double (--) hyphens to proper en (–) or em (—) dashes, and make the text's use of dashes consistent.

Speaking of dashes, there is no shortage of opinions about them. Here's a selection:

The widely used em dash is a blunt line one em long. This is far too much length and invariably spoils any cultivated type area. The only right thing to do is to use...en dashes, and separate them from adjoining words by using the word spacing.

—Jan Tschichold, *The Form of the Book*

En rules...should never be used...because the tendency, then, is for them to look like misplaced hyphens, or minus signs. Em dashes should be separated from the word or words they relate to by a hair space only.

—Geoffrey Dowding, *Finer Points in the Spacing & Arrangement of Type*

Use spaced en dashes – rather than close-set em dashes or spaced hyphens – to set off phrases.

—Robert Bringhurst, *The Elements of Typographic Style*

Note that dash designs differ in length, thickness, slant, and spacing from typeface to typeface. Keep that in mind as you figure out your own position on dashes and other marks. And remember: there is no “right” way. Typographic conventions are ours to hold or, occasionally, to reconsider. The most important thing is consistency.

Designate special strings with markup

Consistency also matters when letters change case, and when abbreviations or numerals come into play. Use markup to target specific character sequences for styling later.

- Make sure text that will eventually be styled as all caps or small caps works primarily as plaintext. For example, headings that you want to style as caps should be encoded in title case (or whatever your editorial guidelines prescribe) and then styled as all caps: `<h2>All-Caps Heading</h2>`.
- Mark up abbreviations with `abbr` tags. Style them independently, applying small caps as needed, using attribute selectors or classes: `5:00<abbr class="time" title="post meridiem">pm</abbr>`.

- Lowercase text can be styled as small caps. (So can uppercase text, and we'll review exactly how to do that when we discuss OpenType features in Chapter 4.)
- Some abbreviations may look better uppercased: `<abbr title="New York">NY</abbr>`; consider this before applying small caps to all `abbr` tags.
- Surround long sequences of numbers with spans, so you can use the CSS `letter-spacing` property to add subtle breathing room: `1,385,600`. (Again, we'll talk about this in more detail in Chapter 4.)
- Plan for case-sensitivity near case changes and numerals. To style characters like hyphens, dashes, and parentheses with case-sensitive forms (alternate glyph designs that help such characters work better in context), you need to account for them in markup.
- For example, keep parentheses like these inside the abbreviation markup: `<abbr>(PDF)</abbr>`. Many fonts bundle case-sensitive forms with caps-related OpenType features.
- Similarly, case-sensitive forms are often bundled with figure features: `(845)555-3333`.
- In headings, mark up ampersands so you have the option of giving them special stylistic treatment, such as italicizing them (sometimes a nice touch): `Hatfields & McCoys`. Note that the `&` character entity is not necessary; UTF-8 encoding includes the ampersand character.
- If certain strings should not break onto two lines (like the testimonial attribution in our example project site), mark those strings up so you can later apply a CSS white-space declaration: `<cite class="nobreak">Somebody Jones</cite>`. Prefer this approach to the hackish practice of adding space entities like ` ` to the markup.
- Use Unicode's fraction characters when possible: `when she was 2½ years old`. (Search [Charcod.es](#) for "fraction" and copy/paste the character you need rather than use the HTML entity.) Mark up any fractions that aren't in Unicode so you can apply fraction-related OpenType features later: `you'll need a 5/16 wrench`.

Automate or script the cleanup

Most of what we just covered about tidying up, using the right marks, and designating special strings can be done manually, but in the long run it's much more efficient to automate cleanup with regular expressions. *Regular expressions* (sometimes called “regex” or “grep”) comb through text with logical precision. CodePen supports them (<http://bkaprt.com/ft/02-07/>), and your text editor's find-and-replace feature probably does too. Regex-One is a nice, tutorial-style introduction to regular expressions (<http://bkaprt.com/ft/02-08/>).

As an added benefit, regular expressions can help you mark text up in HTML. For example, you could do a search for `\n\n` (two simultaneous line breaks), and replace any matches with `</p>\n\n<p>`. Voilà! Much of your text is now surrounded by paragraph tags. Of course, success depends on whether there were two clean simultaneous line breaks, and you'll probably want to swap some of those paragraph tags out for other HTML elements like header tags, but you get the idea.

Scripts like PHP Typography (<http://bkaprt.com/ft/02-09/>) and Typeset.js (<http://bkaprt.com/ft/02-10/>) can also help tackle cleanup. They're especially helpful if you're working on a site that incorporates a content management system. Not all authors will be careful to use typographically appropriate conventions; they may not even realize such conventions exist.

Whether and however you automate, remember that it's your responsibility to know what scripts do (and, if you're using them, what regular expressions do) and to check their output. Scripts will sometimes fail because of messy input, yielding the wrong kind of dash, an extra dot after an ellipsis, or dates formatted as fractions. They may also hold opinions with which you disagree—in which case you should either avoid them or edit them to get the result you want.

Mark the text up

Finally, mark up the elements of the text—paragraphs, headings, lists, block quotes, and so on—as well as any strings that need emphasis. Entire books have been written about HTML,

so I won't delve into it here, but keep in mind that markup is the most basic and possibly the most important *typographic* decision we make. The same sensitivities that drive us to appreciate the subtleties of visual composition can help us realize the importance of what I've come to think of as *invisible composition*: markup. It's our core means of establishing hierarchy, patterns, and meaning.

Here's the "after" CodePen snapshot I mentioned—our project example, now with text that has been cleaned up and marked up as HTML (<http://bkaprt.com/ft/02-11/>). Next, we'll add some CSS.

RESET DEFAULT STYLES

Most browsers and user agents ship with default styles. This is a good thing, because it means that when readers encounter text on the web that is styled incompletely, or not at all, it will be easier to read by default.

But browser default styles can conflict with the styles we typographers carefully write. If a browser specifies margins for paragraphs, that amount of marginal space may not look right in our composition. Furthermore, the measurements used by the browser may not be consistent with other measurements we're using, which may change how our composition looks as it flexes. We may even forget that we ourselves didn't set the paragraph margins.

We need to be aware of every stylistic decision in our work, and purposeful in our response to that awareness. For example, let's say we want our paragraphs to have a bottom margin of `1em`, but we don't want *any* margins or padding to surround our whole composition—we want to fill the whole viewport. We might write CSS like this:

```
body {  
  margin: 0;  
  padding: 0;  
}  
p {
```

```
margin-bottom: 1em;
}
```

Let's compare that to what a browser may do by default:

```
body {
  margin: 10px;
}
p {
  margin-top: 1em;
  margin-bottom: 1em;
}
```

Because of the cascade, any styles we write will override browser default styles. But in the code above, the browser's default styling of our paragraphs' `margin-top` would persist, because we didn't explicitly override it. We can fix this by adding `margin-top: 0;` to our paragraph declaration.

Now, there are two basic approaches we can take to handling browser default styles at scale. One approach is to do what we've done here: know what browsers do by default, allow acceptable defaults to persist, and override selectively. What's nice about this tactic is that it allows us to avoid writing redundant lines of code. If browsers are already specifying `margin-bottom: 1em`, we can inherit that from their default styles. That feels efficient, and it keeps your browser's web inspector looking nice and clean when you peek at the source code.

The difficulty, though, is that we need to familiarize ourselves with the defaults of every browser in order to know which properties and values to inherit, and which to declare ourselves. Tools like Normalize.css (<http://bkaprt.com/ft/02-12/>) can make this easier, but they essentially depend on people paying continuous attention as browsers change.

The second approach—my preference—is to use a *reset stylesheet* instead. Eric Meyer has advocated the use of reset stylesheets since 2007, and I routinely use his latest version as a starting point for my projects. We'll use it in our example project site, too (<http://bkaprt.com/ft/02-13/>).

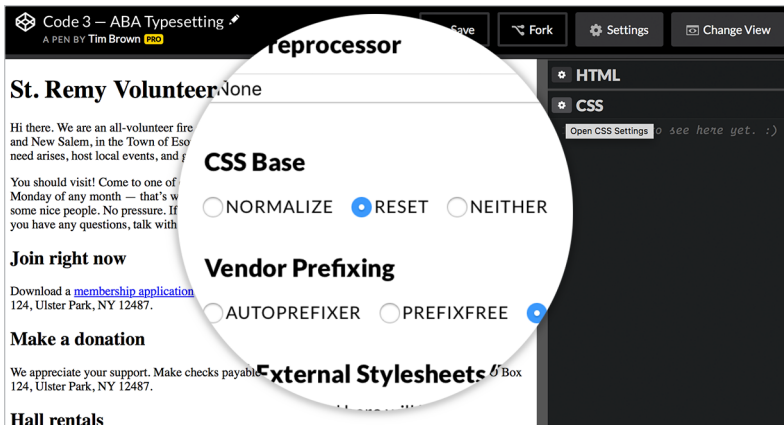


FIG 2.4: CodePen makes using a reset stylesheet extremely easy—we can click to enable it in the CSS settings menu.

A reset stylesheet puts the focus on HTML elements rather than on browser behavior. By removing margins and padding, and giving consistent values to other properties like **font-size** and **line-height**, we create a solid base from which to make design decisions with very few surprises.

Of course, there are also difficulties with the reset approach. Elements the reset stylesheet touches can end up having multiple values that get overridden. This means browsers do extra work in parsing the CSS, and the browser’s web inspector can get a little messy. But for me, the pros far outweigh the cons. Reset is a surer concept. Getting back to our St. Remy project, let’s add a reset stylesheet (**FIG 2.4**).

FRAME THE COMPOSITION

Traditionally, framing a composition has meant choosing the dimensions of a piece of paper or a screen, understanding its sharpness in terms of dots or pixels per inch, and settling on unit conventions like inches, millimeters, points, or pixels (**FIG 2.5**).

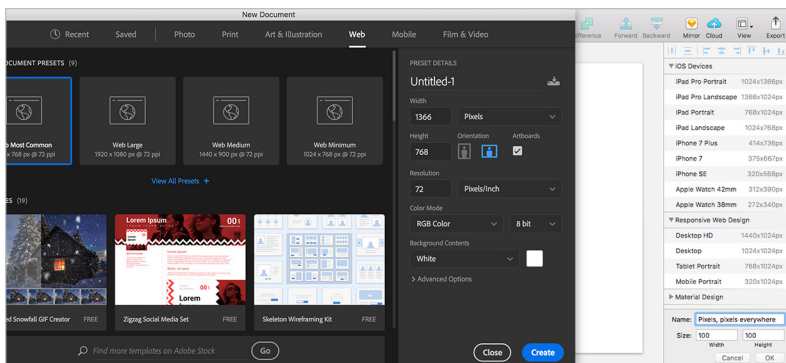


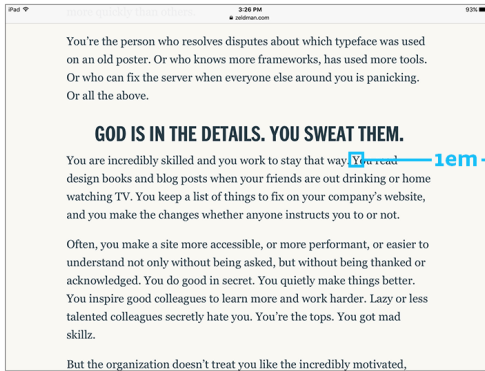
FIG 2.5: Left: Adobe Photoshop’s New Document menu prompts users to choose a canvas size with pixel-based dimensions. Right: Sketch offers many artboard sizes—all using pixels.

The problem is that we need to frame our compositions differently for the web, which has no “frame” per se (**FIG 2.6**). We can’t know dimensions and resolution ahead of time; instead, we ascertain them and respond live. We can’t start the way we traditionally have, because that tradition is not structurally sound. The actual size, proportion, and pixel density of viewports vary wildly from one user to another. Trying to predetermine these things means making assumptions that will almost certainly miss the mark.

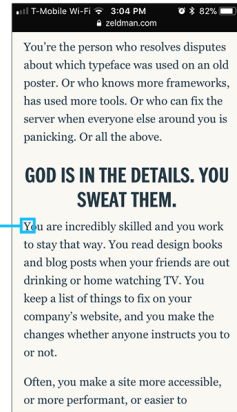
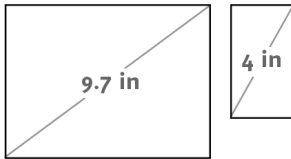
But we do have one reliable building block for typesetting today: default font size. Default font size is a deeply rooted and very personal part of our favorite devices. We can even customize it in our settings and preferences, and many people do. Default font size is a rock. It’s solid, even when nearly everything else about our typography flexes and flows.

By starting with type—starting from each individual reader’s default font size—we can frame compositions from the inside out and generate systems of measurement that feel natural for flexible, responsive compositions. Doing this puts our readers in control of text size, which makes our typesetting feel *just right*, no matter who’s reading.

Here’s how we do it. In CodePen, add this just after the UTF-8 meta tag:



iPad (3rd generation)



iPhone SE

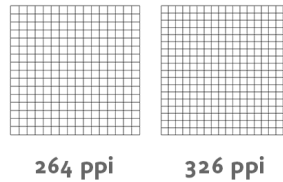


FIG 2.6: Left: a third-generation iPad. Right: an iPhone SE. Their screens' physical dimensions and resolutions differ, but they have one thing in common: each has a default font size.

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

And now in the CSS pane, paste this:

```
@viewport {
  width: device-width;
}
:root {
  font-size: 100%;
}
```

Code example: <http://bkaprt.com/ft/02-14/>

Let's talk about what this code is doing.

Viewport settings

First, we need to make sure readers see our composition at its actual size on any device, without any artificial scaling, while maintaining their ability to zoom in if they want to. We do this with CSS Device Adaptation—the `@viewport` rule.

The technical explanation for `@viewport`'s various properties and values, and their practical implications, could fill an entire book. If you're curious, dig into Peter-Paul Koch's blog post series, *A Tale of Two Viewports* (<http://bkaprt.com/ft/02-15/>), as well as Apple's Safari documentation on configuring the viewport (<http://bkaprt.com/ft/02-16/>) and the W3C's CSS Device Adaptation Module Level 1 working draft (<http://bkaprt.com/ft/02-17/>).

What we're doing here with `width: device-width` is, as one might expect, telling the browser to fit the width of our layout to the width of a person's device. This makes our text actual-size, instead of scaling it to match a specific pixel-based width. The latter can be helpful for showing a zoomed-out view of a static composition, like an old fixed-width website, but we're building a flexible layout here. CSS Device Adaptation is not supported in all browsers yet, though, so we also need to use an HTML `meta` tag that accomplishes the same thing.

Note that it's important to specify *both* the `@viewport` descriptor in CSS and the HTML `meta` tag, because browsers currently use either one or the other (<http://bkaprt.com/ft/02-18/>). Note, too, that vendor prefixes are still necessary for now, in addition to the unprefixed property.

Base font size

The other thing this code does is set our base font size. Although we will use many different font sizes throughout our composition, this is where it all begins:

```
:root {  
  font-size: 100%;  
}
```

The CSS `:root` selector is what the W3C calls a *structural pseudo-class*. It refers to the root element in a document, which in our case means the `html` element. So this is equivalent to using the `html` element selector directly. But I like to use `:root`—it serves as a good reminder that by declaring a base font size, we “root” our composition in the reader.

Percentages are a relative CSS length unit. When we use percentages, we size type *relative* to any given device’s default font size. And when we specify `100%` as our value, we precisely match every one of our readers’ individual, favorite font size. That’s amazing. There is no better way to frame a composition than to fit it to each reader.

It may surprise you to learn that declaring this font size on the root element is unnecessary. Browsers do it for us by default. They use readers’ default font size as the root font size, even if we do nothing. And what we’re doing here is essentially nothing. We’re telling the browser, “Yes, good job, keep doing what you’re doing.” So even though it’s not strictly necessary, this affirmation is good. It reminds us that our loyalty lies with the reader.

Type sizes throughout our composition will be based on this root font size. But you may be wondering where units like pixels, ems, and rems fit in.

Not everyone uses this method of setting the root font size to `100%`, though it has been acknowledged as a best practice for years by forward-thinking designers like Richard Rutter (who introduced the technique) and Ethan Marcotte (who reaffirmed its validity). I think there are a few reasons for that, the main one being that this approach is relatively new and different,

and people haven't understood well enough how important root font size is—both for readers and for designers. Still, reasonable questions about the method remain. Let's tackle them one by one.

My design tool uses pixels. How would I use a relative unit?

Your design tool is working against you. It is stuck in the traditional mindset of absolute measurements. This is precisely one reason why people very good at web design argue that designers should learn to write code. No mainstream design tools—and I work for a company that makes mainstream design tools—are completely appropriate for the practice of typesetting today. Some of the smartest people I know are building more appropriate tools, but for now you'll need to make design decisions by writing and testing code. That's why I wrote this book!

Okay, so never mind design tools. I'm writing code.

Why not 16px as a root font size? Don't browsers all use 16px as their default? And don't all measurements in browsers eventually resolve to pixels?

Yes, everything eventually does resolve to pixels—but dynamically. Design in a relative way, and let browsers handle that dynamic translation to absolute measurement. While it's true that most browsers do use approximately 16px as a default text size, some don't. And who knows what “most” browsers will do in the future? Besides, 16px looks different depending on the screen, text-rendering technology, and settings.

Also, remember that devices ship with various default font sizes, and people can adjust those default font sizes in their settings and preferences. This is critical. Type sized with pixels *does not respect a user's default font-size preference* (<http://bkaprt.com/ft/02-19/>). That means if someone has impaired vision and uses a larger font-size in their browser settings, they're out of luck. Or if someone with a motor impairment sets a very small font size in their user preferences, so they can read without having to scroll as much—no dice. If we size our text in pixels, we risk overriding user settings and preferences.

How about viewport units as a root font-size?

Viewport units like `vw`, `vh`, `vmin`, and `vmax` (<http://bkaprt.com/ft/02-20/>) are relative, but they're relative *to the viewport*—not to a reader's default font size. If you were to use viewport units as a root font size, type would be flexible, but not in an accessible way. Type sized purely in viewport units (even a mix of different viewport units) *does not respect a user's default font size preference* (<http://bkaprt.com/ft/02-21/>).

What about a combination of pixels and viewport units?

Pixel-based font sizes that incorporate viewport units through a CSS `calc()` function like `font-size: calc(10px + 2vw)` can still override user font size settings and preferences. Think about what this equation is ultimately referring to: pixels, which are based on device resolution; and viewport units, which are based on viewport dimensions. What we want to do instead is base our font size on each reader's individual default font size.

But I love using pixels!

I can't stop you from using pixels; I can only try to convince you to get your brain comfortable with ems. Pixels are weird. They're an absolute unit that we expect to behave in a quasi-relative way because of the quirks and conventions of design for screens over the past twenty years. But the issues I'm talking about still exist today—these aren't ancient Internet Explorer bugs. They'll keep popping up, and readers will keep suffering, because “pixels” are loaded with ambiguity.

Can ems or rems be used as the root font-size?

Yes! These are relative units. `100%` is equivalent to `1em` or `1rem` at the root level. In Chapter 4, we'll change our root font size to an em-based value. At that point, we'll also begin using rems. *Rem units*, sometimes called root ems, behave just like ems—except that they are always relative to the root font size. An em unit is relative to the font size of its parent.

Specific use cases for ems and rems will become clearer in Chapters 4 and 5, where we'll discuss flexible text blocks and compositional alterations. But in general, use rems for connected, systematic measurements (like font sizes and layout tailoring), and ems in more insular, self-contained ways (for refinements like letter and word spacing or vertical margins between text blocks). We'll talk about a big exception to this general rule—media queries—in Chapter 5.

Is it okay to change the 100% to a different value, like 125%?

Yes, indeed. And it's also okay to use different em or rem values. But don't do any of that just yet. You are going to love Chapter 4.

Wait, wait—one more. What about a combination of 100%, 1em, or 1rem and viewport units?

Yes again! Percentage-, em-, or rem-based font sizes that incorporate viewport units using a CSS `calc()` function like `font-size: calc(1em + 2vw)` are *accessible*—unlike pure viewport-based sizing, or a combination of viewports and pixels. This approach respects a user's default font size preference. But let me ask *you* a question. Why are flexible font sizes like this so interesting? I bet your answer has something to do with making text look good as layouts grow and shrink. You are going to love Chapter 5.

Please bear with me as we size type with ems and rems, and remember that this book is an example designed to help you think from a sophisticated typographic perspective. Viewport unit calculations are an easy way to make text fill layouts, but we're not painting with broad brush strokes here. This is type-setting. Precision characterizes the craft.

text-size-adjust

Before we move on from this section on framing the composition, there's one more thing you should know about. There's an unofficial CSS property called `text-size-adjust` that, like the `viewport` property and HTML `meta` tag, exists to make life

easier on small screens for websites that are not responsive. Take care to understand the effects of some of this property's different values.

```
text-size-adjust: none;
```

The value `none` prevents type from being resized at all (<http://bkaprt.com/ft/02-22/>). That's a big problem, especially for readers with impaired vision.

```
text-size-adjust: 100%;
```

A value of `100%` prevents type from being resized automatically in iOS when a device's orientation changes. That might be a problem if a reader expects type to get larger in landscape mode for easier reading.

Personally, I avoid `text-size-adjust` entirely. It can cause serious accessibility problems, it is sometimes overridden by the aforementioned viewport settings, and browser implementation is inconsistent right now. (Note that `text-size-adjust` differs from `font-size-adjust`; we'll discuss that property in Chapter 4.)

TACKLE WEBFONT LOGISTICS

If you want to design with webfonts—and I encourage you to do so—you'll need to know where to find them, how to get them, and how to use them. Let's review all of these logistics, so that you know what you're doing when the time comes. I'll be brief, and point to places where you can learn more. (You can put our example project aside for now. We'll pick it back up in Chapter 3.)

Finding webfonts

Webfonts aren't hard to find. But as I'm sure you can imagine, and as this book's forthcoming chapters will make very clear,

selecting and using typefaces successfully is considerably easier if you look for them in the right places.

If you're not sure where to start, check out Adobe Typekit (<http://bkaprt.com/ft/02-23/>) or Fontstand (<http://bkaprt.com/ft/02-24/>). There you can find affordable, diverse collections of quality typefaces with clear information about features and licensing. You'll also find—especially important for the purposes of this book—many typefaces that are especially good for typesetting.

Repositories of free and open-source fonts—even something as convenient as Google Fonts—have significantly fewer choices for text use. In many of these fonts, character sets are incomplete, random glyphs look wrong, critical features are missing, spacing is sloppy, and styles like bold and italic are either nonexistent or incongruous with the main style. Contrast this with the benefits of paying for type: knowing your fonts are fully featured and fully supported; doing better work because you feel invested; having peace of mind regarding licensing; supporting type design as a craft; and forming bonds with type designers (<http://bkaprt.com/ft/02-25/>).

As you spend more time with type, you'll get to know certain type foundries and designers. You can seek them out specifically, and purchase type from them directly, once you have a better idea of what you like and need. Good relationships with type designers will increase your chances of success when using type. I like the analogy Trent Walton made after watching the documentary *Jiro Dreams of Sushi*:

I was impressed with the relationships he has with his food providers. Because sourcing the highest quality ingredients is the foundation for Jiro's craft, he recognizes and trusts each provider as the expert in his particular field—tuna, rice, shrimp, etc. The relationship is symbiotic; each must excel or neither will succeed. I think this mirrors web designers' relationship to type designers.

—Trent Walton, "Jiro, Sushi, & Web Type" (<http://bkaprt.com/ft/02-26/>)

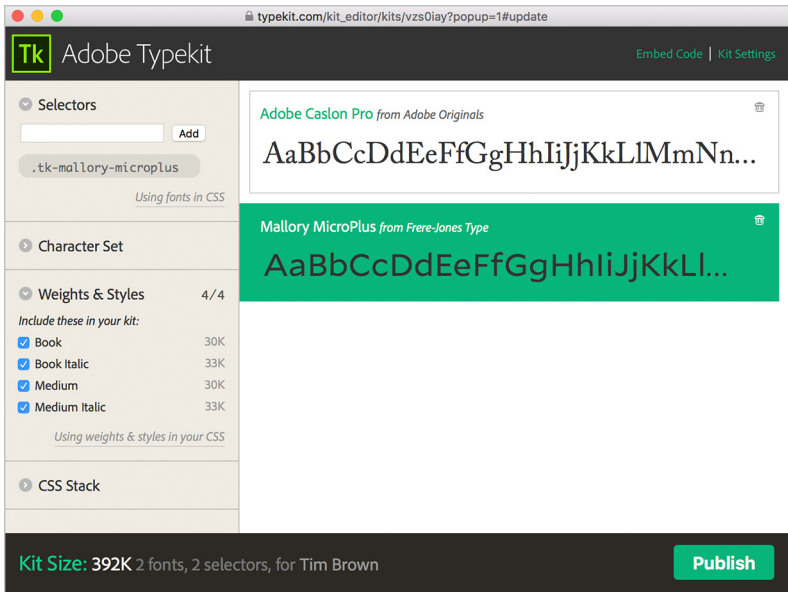


FIG 2.7: In Typekit’s Kit Editor, you can keep all kinds of details organized. Here you can see which families, weights, and styles are included, as well as the specific syntax for using fonts in CSS.

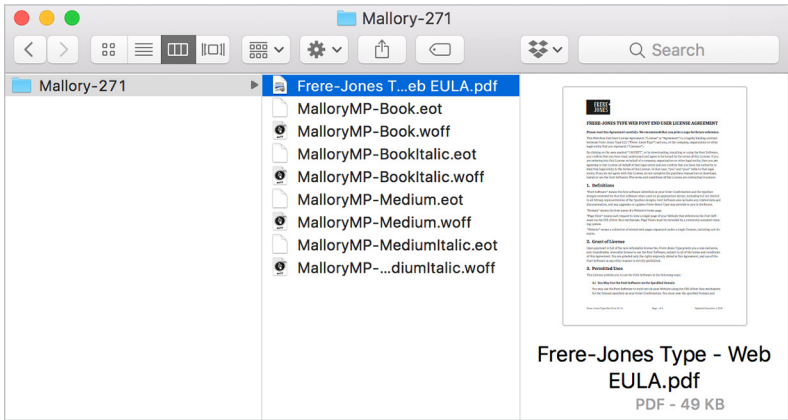


FIG 2.8: When I licensed Mallory directly from Frere-Jones Type, I received font files as well as a EULA.

Getting webfonts

Getting webfonts is easy once you've found them. Let's look at a couple of quick examples: Adobe Typekit (<http://bkaprt.com/ft/02-27/>) and Frere-Jones Type (<http://bkaprt.com/ft/02-28/>).

Fonts you select at Adobe Typekit are added to a “kit” that you publish and connect to your web project. Because Typekit hosts the fonts and offers a standard licensing agreement, there's not really anything you need to “get.”

If you purchase a webfont license directly from a type designer or foundry, make sure you receive font files made for use on the web (WOFF, or preferably WOFF2, which I'll discuss in a minute), as well as a copy of your *End User License Agreement* (EULA) specifying that web use is okay.

Using webfonts

To use webfonts, you need to make sure the fonts are online somewhere, define them in your CSS, and then declare them as you're styling specific elements. The way you go about this depends on how you got the fonts, so let's stick with our two examples of Adobe Typekit and Frere-Jones Type.

Put the fonts online

With Typekit, publishing a kit puts the fonts online. They're hosted on Typekit's *Content Delivery Network* (CDN), separately from the rest of your website. If you get files directly from a type designer or foundry, there are a variety of ways to put them online. Most often you'll need to self-host them.

When you self-host font files, you can put them online anywhere (for example, on your web server or on a CDN, along with your HTML, CSS, JavaScript, and media files). Be sure to take appropriate measures to protect the font files from theft or use by other people. The EULA that came with your fonts likely includes details like this, from Frere-Jones Type:

You further agree to use reasonable efforts to safeguard the Font Software from unauthorized access, use, duplication or

distribution, including but not limited to preventing the use of any process that allows hot-linking, re-serving or re-directing access to the Font Software.

Some type foundries also allow third-party hosting. Fonts licensed directly from Frere-Jones Type can optionally be hosted on Typekit:

You may self-host the Font Software or you may choose to have Typekit host and serve the Font Software. Use of any other third-party hosting service requires the prior written consent of Frere-Jones Type.

Define the fonts

The CSS `@font-face` rule (<http://bkaprt.com/ft/02-29/>) connects font files to stylesheets by defining fonts like so:

```
@font-face {  
  font-family: "Source Sans Pro";  
  font-weight: 400;  
  font-style: normal;  
  font-stretch: normal;  
  src: url("SourceSansPro-Regular.woff2")  
  format("woff2");  
}
```

Here, we're telling our CSS that there's a type family called "Source Sans Pro" and that this file (SourceSansPro-Regular.woff2) is the family's regular weight (`font-weight: 400;`) and regular style (`font-style: normal;`). Every variation of weight and style in a type family needs to be defined like this. For example, here's how we would define the bold weight:

```
@font-face {  
  font-family: "Source Sans Pro";  
  font-weight: 700;  
  font-style: normal;  
  font-stretch: normal;
```

```

src: url("SourceSansPro-Bold.woff2")
format("woff2");
}

```

The font file's extension here, `.woff2`, refers to WOFF 2.0, a font-packaging format designed for webfonts and specified by the W3C (<http://bkaprt.com/ft/02-29/>). It's fairly well supported (<http://bkaprt.com/ft/02-30/>), but you'll need to make sure you have all of the file formats you need to account for the contexts you care about. Here's an example using the "deepest possible browser support" recommendation from CSS-Tricks (<http://bkaprt.com/ft/02-31/>):

```

@font-face {
  font-family: "Source Sans Pro";
  font-weight: 400;
  font-style: normal;
  font-stretch: normal;
  src: url("SourceSansPro-Regular.eot"); /* IE9
  Compat Modes */
  src: url("SourceSansPro-Regular.eot?#iefix")
  format("embedded-opentype"), /* IE6-IE8 */
       url("SourceSansPro-Regular.woff2")
  format("woff2"), /* Super Modern Browsers */
       url("SourceSansPro-Regular.woff")
  format("woff"), /* Pretty Modern Browsers */
       url("SourceSansPro-Regular.ttf")
  format("truetype"), /* Safari, Android, iOS */
       url("SourceSansPro-Regular.svg#svgFontName")
  format("svg"); /* Legacy iOS */
}

```

Typekit writes these `@font-face` definitions for you, serves files that work in a broad range of browsers, and automatically updates your kit over time (<http://bkaprt.com/ft/02-32/>).

If you self-host fonts, you'll need to write `@font-face` definitions yourself (one for each variation) and maintain them over time. There's a lot of sample code online, and foundries will often provide their own sample code, but you should check

to make sure it meets your browser-support needs and isn't outdated. Stein's *Webfont Handbook* (<http://bkaprt.com/ft/01-03/>) will prove its mettle here. The relevant section, "Using Webfonts," is excerpted at *A List Apart* (<http://bkaprt.com/ft/02-33/>). Also, be sure to get files in the appropriate formats directly from the designer or foundry you're working with, because many EULAs forbid conversion of font files to different formats. [CanIuse.com](http://caniuse.com) can help you determine which browsers support which font formats.

Declare the fonts

Now that your font files are online and defined in your CSS, you can use them in declarations like this:

```
p {  
  font-family: "Source Sans Pro";  
}  
p strong {  
  font-style: bold;  
}
```

Because you've defined the Source Sans Pro family, the browser knows which files to use when you refer to that name in a declaration. If you're writing your own [@font-face](#) definitions, the family name can be anything you wish. If you're using Typekit, check the Kit Editor for the family name.

Save optimization for later

There's a lot more to using webfonts than this brief description can cover. Again, check out Stein's *Webfont Handbook* for details on various optimization techniques like font subsetting, conditional loading with [unicode-range](#), reduced requests with browser caching, and asynchronous loading with the CSS Font Loading API.

For the purposes of this book, though, save optimization for later. Use whole font files (not subsets) and as many variations

as you might want to try; also, make sure that all of your fonts' OpenType features are enabled.

Learn about OpenType features

OpenType features are a vital part of font files—in some cases, they're as important as a font's glyphs. They include kerning and ligatures (which both help resolve spacing between specific characters), as well as scores of additional features ranging from contextual alternates (crucial for maintaining the type designer's thoughts about the shapes and spacing in sequences of glyphs) to swashes (cool, but best used in moderation) to small caps. If that stuff is missing or disabled, and if the project you're working on calls for any of it, your typesetting will seriously suffer as a result.

In Chapters 3 and 4, we'll talk about the OpenType features that matter most for body text, and how to use them—but in case you're not very familiar with OpenType features, I encourage you to read my Typekit Practice lesson, "Caring about OpenType features," and try the examples provided (<http://bkaprt.com/ft/02-34/>).

Congratulations, friend! You've successfully prepared text and code. As a reward, go buy yourself some new reading glasses. (*Squints ahead to Chapter 3.*) We're getting to the fun part: selecting typefaces.

3 SELECTING TYPEFACES

“The typeface I use influences so many other parts of the page that until I can settle on which to use, I am unable to carry on. It is the basis for everything else.”

—RICHARD HENDEL, ON BOOK DESIGN

NOW THAT WE KNOW WHAT TYPESETTING IS, have gotten our text under control, and have prepared our code for designing, it's time to dive into the type. Continuing with our example project, let's identify the text blocks we're working with, choose a typeface to serve as an anchor, and then build a type palette around that anchor.

IDENTIFY TEXT BLOCKS (AND JOB OPENINGS)

Every chunk of text you touch is a text block. We've already come across the phrase *text block* several times in this book. It's not a technical term; it's just a name for any bunch of text that shares some formal qualities in a typographic composition.

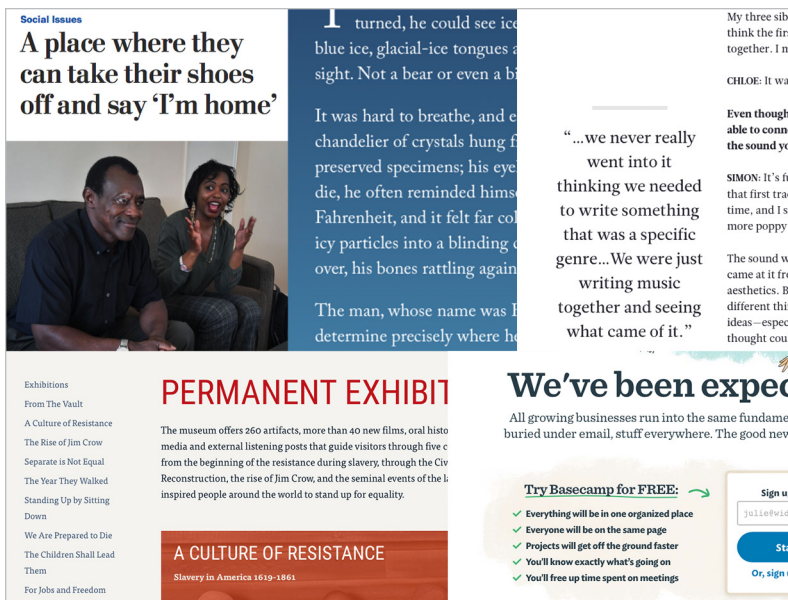


FIG 3.1: Headings, paragraphs, pull quotes, decks, and lists could be described as text blocks. (Clockwise from top left: screenshots from the *Washington Post*, the *New Yorker*, *The Great Discontent*, Basecamp, and the National Civil Rights Museum.)

You could say that a single paragraph is a text block, or that a combination of several elements is a text block (**FIG 3.1**).

Having handled our example project’s text, you’re intimately familiar with the variety of text blocks you’ll be typesetting. Now take an inventory. Comb through the St. Remy markup, sketching and labeling every different chunk of text (**FIG 3.2**). At this point, gather or make any thumbnail sketches you have in mind for layouts—I made some sketches for us (**FIG 3.3**). Ensure that sketches like these are clearly labeled, so you remember which text blocks are which, what they’re called, and how they change from layout to layout.

Once you have an inventory, determine the job each text block needs type to do. This will help you decide which text blocks to focus on. As you’ll recall from Chapter 1, typesetting comprises jobs like body text and small text, but it’s hard to

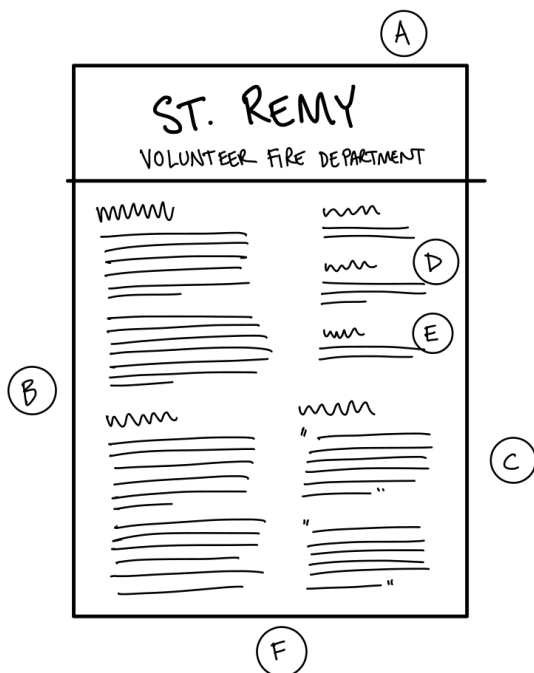


FIG 3.2: St. Remy sketched out, with text chunks labeled. A: Header; B: Main text; C: Testimonials; D: Application PDF and contact information; E: Footer.

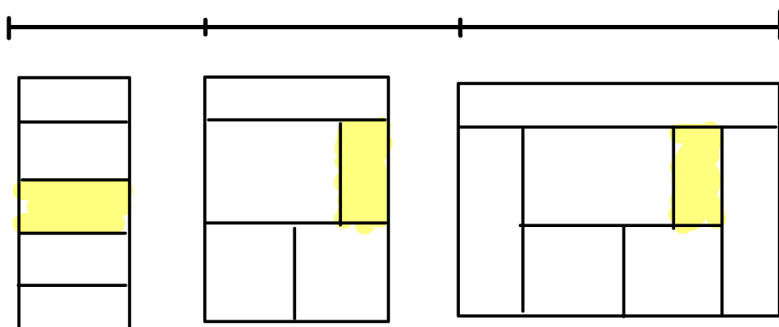


FIG 3.3: At this stage, it's good to begin thinking about different layouts and about how text chunks might flex or be rearranged. Stephen Hay's breakpoint graphs are great for illustrating this (<http://bkaprt.com/ft/03-01/>).

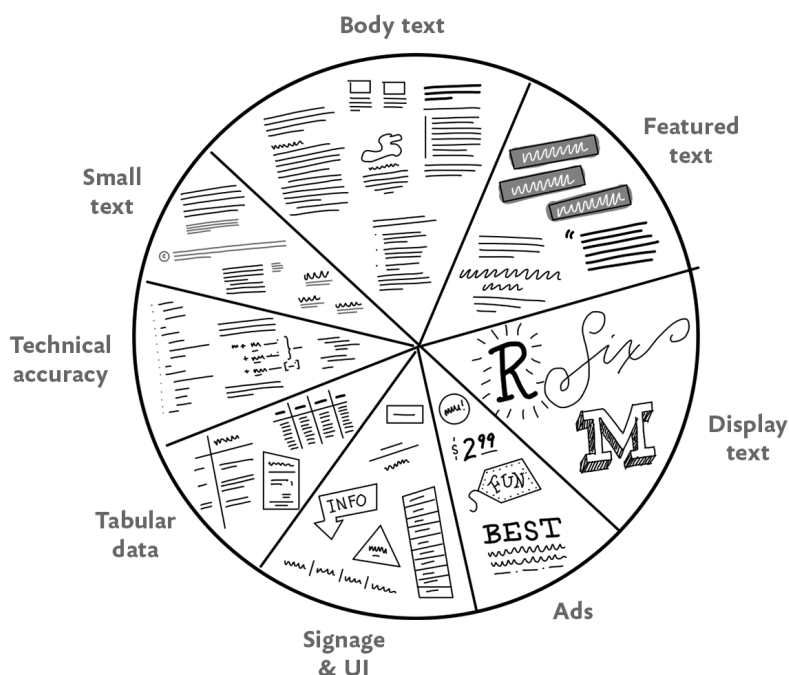


FIG 3.4: A wheel of typographic jobs.

put a name to every kind of typography that exists. Think of these jobs like a color wheel. There are many places along the wheel, and some jobs require a mix of typographic approaches (**FIG 3.4**). Type does a full spectrum of work. Use the following pages to help classify the text blocks in your inventory.

Body text

The purpose of body text is to help people read comfortably and digest information without distraction. Examples of body text include paragraphs, subheadings to keep text organized, decks (an editorial term for a more prominent paragraph that leads from a story's title to its main text), and block quotes (**FIG 3.5**).

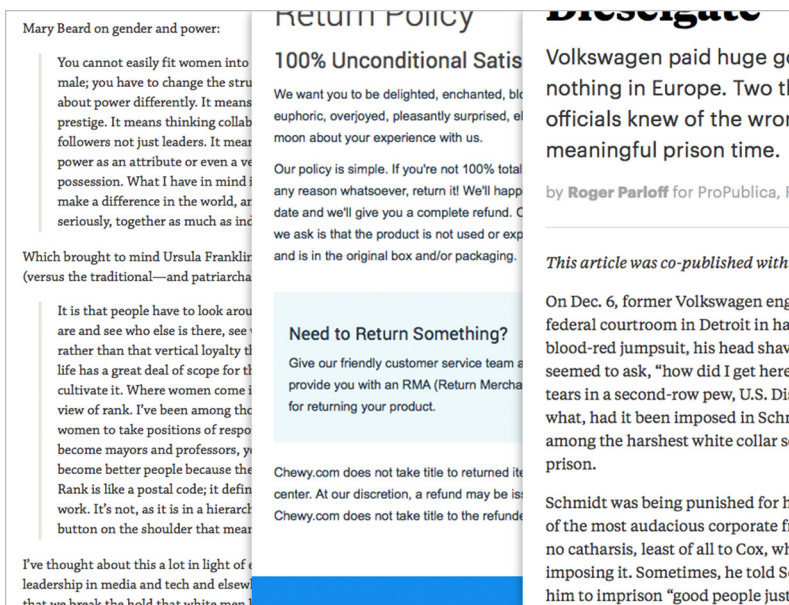


FIG 3.5: From left to right: screenshots from A Working Library, Chewy.com, and ProPublica.

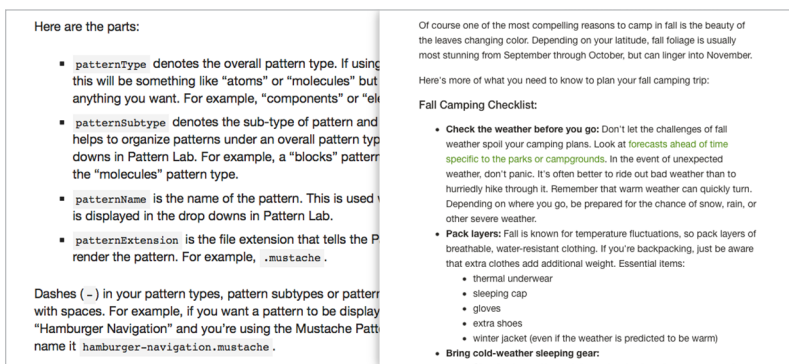


FIG 3.6: Screenshots from Pattern Lab (left) and the Wilderness Society (right).

Lists also qualify as body text most of the time, though it depends on their content and context. If each list item is a small paragraph, or a list of words relevant to the text (like a ten-item checklist of camping supplies in a text about preparing to go camping), treat the list as body text (**FIG 3.6**).

Display text

Display text exists to attract attention, establish tone, and generate enough interest that people will continue to read and investigate. Examples of display jobs include top-level headings, hero text, and logotype lockups. Type often shares display work with lettering (custom glyphs not intended for reuse) and illustration (**FIG 3.7**).

Signage/UI copy

The purpose of signage and *user-interface* (UI) copy is to help people navigate, to find their way around. It needs to be clear and legible. Examples of signage and UI text include website navigation, footers full of links, labels in a grid of products, previous and next links, buttons, and strong calls to action, like the ones often found on product sites (**FIG 3.8**).

Tabular data

Tables of contents, financial data, calendars, and statistics don't flow like body text, though they share some of its traits—like a focus on conveying information. But this kind of text requires the same kind of clarity that signage and UI copy do. People scan tabular data quickly to gather details and make comparisons, so it's important that they're able to do that with minimal effort (**FIG 3.9**).

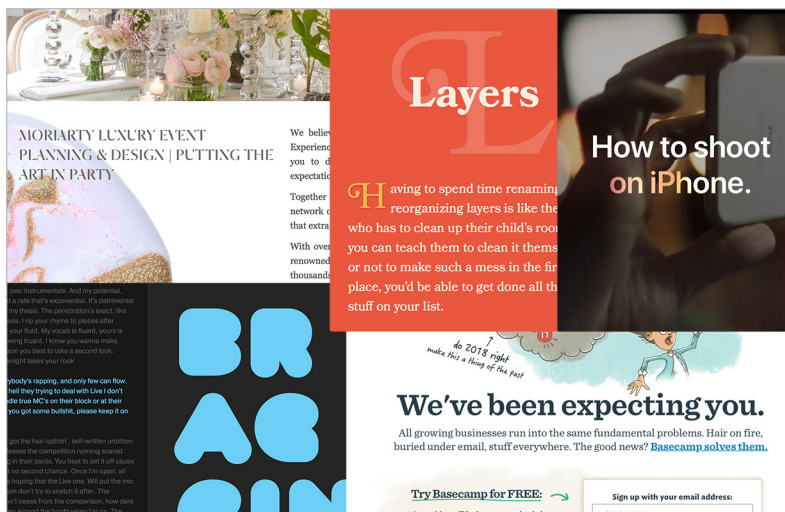


FIG 3.7: Display text in the wild. Clockwise, from top left: Moriarty Events, Photoshop Etiquette, Apple's How to shoot on iPhone, Basecamp, and Hip-Hop Quoted.

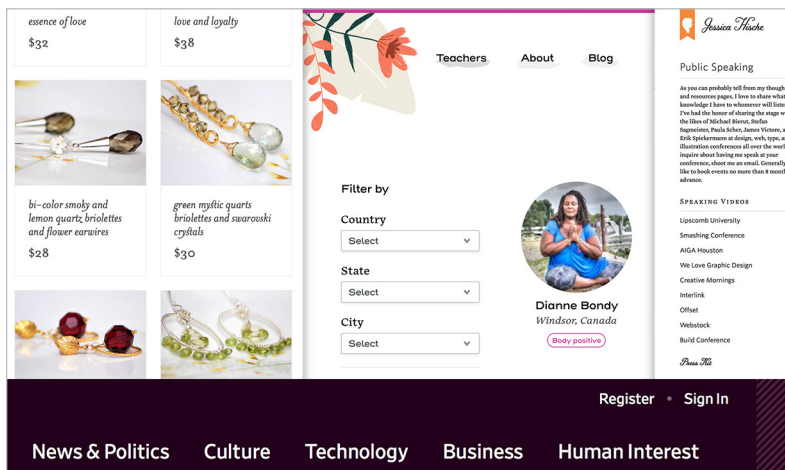


FIG 3.8: Top, from left to right: Simplexpression, Setu, Jessica Hische; bottom: Slate's top navigation.

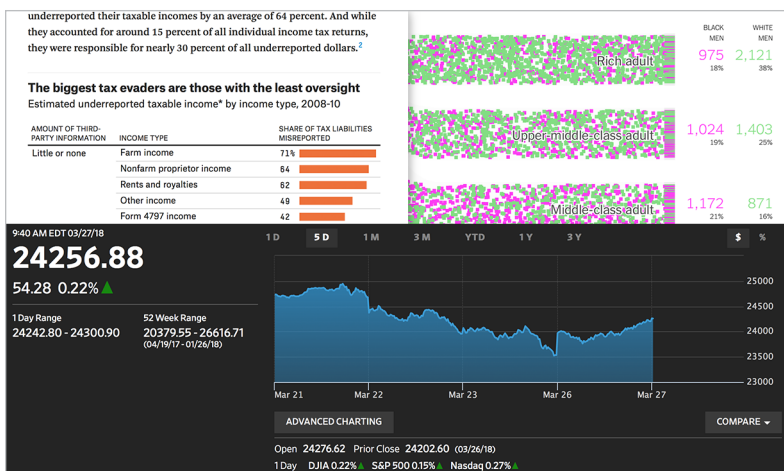


FIG 3.9: Left: FiveThirtyEight. Right: the *New York Times*. Bottom: the *Wall Street Journal*.

Technical accuracy

Some text, because of its nature, requires a level of technical accuracy to be coherent. In code, symbols and their sequence are critical. The same goes for math, which can include complex glyph placement. Poetry, screenplays, and infographics all carry critical information not only through their meaning, but also through the way they are typeset (**FIG 3.10**).

Small text

Small text shares qualities with both body text and signage/UI copy, but it plays a distinct and important role. A typographer may need small text for many reasons—for, say, captions, form labels, help text, or for use when screen real estate is limited, as it is in banner ads (**FIG 3.11**). But small text involves more than simply reducing the font size. Type designed for clarity at small sizes is critical.

No method of classification is perfect. Some text blocks may not fit neatly into the typographic jobs I defined here, and that's okay. I hope this brief exploration of a few primary jobs—as

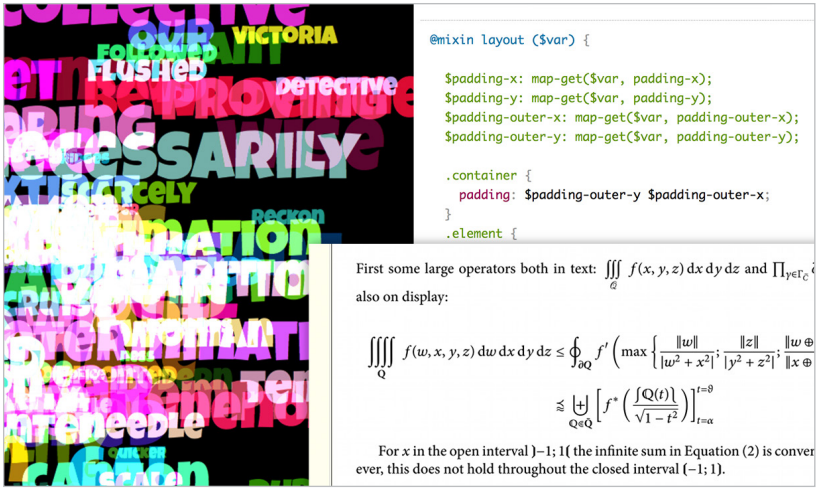


FIG 3.10: Clockwise: “Text Is Flying” by Kabir Shah (<http://bkaprt.com/ft/03-02/>); *A List Apart*; Typoma.

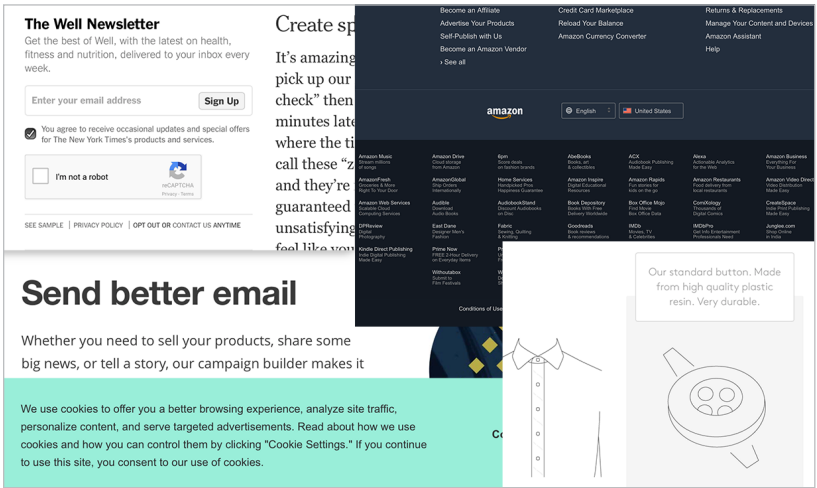


FIG 3.11: Clockwise from top left: screenshots from the *New York Times*, Amazon, Proper Cloth, and MailChimp.

well as secondary jobs that are a mix of those primary considerations—will help serve as a pattern for you to more easily identify and define new jobs.

After comparing the St. Remy text-block inventory with the job descriptions we’ve just covered, I ended up classifying most of the chunks as body text—everything except the header and footer.

For sites that are more complex than our St. Remy example, be sure to account for all of the various templates in use. Pay particular attention to text blocks in different sections that vary only subtly in style; in a site that has grown over time, inconsistencies can quietly become widespread. Nicole Sullivan’s Type-o-matic browser extension for Firefox (<http://bkaprt.com/ft/03-03/>) can help you take a detailed inventory of text elements automatically. Susan Robertson’s article on creating a type style guide might also be of assistance (<http://bkaprt.com/ft/03-04/>).

You may be tempted to start thinking about all of the different typefaces you could use for all of the jobs your text blocks are doing. But don’t reach for that blog post about font pairing just yet! Let’s choose an anchor typeface first, and then use it to style all of the body text we have clearly identified.

CHOOSE AN ANCHOR TYPEFACE

I use the phrase *anchor typeface* to describe a typeface that unites an entire composition, providing stability around which other elements can change and acting as the basis for other decisions.

Body-text typefaces make the best anchors. Of all the jobs that type does, body text is the most critical. It makes up the majority of text you want people to read, and it serves as a structural foundation for the composition by acting as a reference point for size, spacing, and contrast. Body text also establishes a subtle aesthetic that can inspire the whole composition—including a palette of typefaces.

Typefaces made for body text are robust and capable. They’re built to be used at small sizes and to withstand coarse resolutions. They also usually have excellent language support, as well as several crucial OpenType features that make text easier to

read. Good body-text typefaces are hard to find, which is why many designers have favorite “workhorse” typefaces to which they return often.

I’ve already chosen a body-text anchor typeface for our example project—Paul D. Hunt’s Source Sans Pro—but let’s go through the general process of choosing an anchor typeface together. Along the way, I’ll help you start building your stable of workhorses. We’ll do this by looking at the qualities that make a typeface *good* and *appropriate* for body text.

Good type for body text

A good body-text typeface has sturdy shapes, even color, and an active texture. I’ll explain what each of these means in a moment, but first, a few notes:

- To conserve time and effort, evaluate type for goodness *in this order*: first for sturdiness of shapes, then for evenness of color, and finally for the activeness of its texture.
- Know that you can appraise these qualities objectively, even without a project in mind. They’re good things to observe about a typeface, and tuck away for future reference.
- If you haven’t already, now is a good time to get your hands on some fonts. It’s easiest to study them and really get to know them if you have access to them via an affordable subscription or a free trial or rental period. If that’s not possible, you can make do by examining type online if a site offers tools that let you edit text, change font size, adjust line spacing, and control the width of the text block.

Sturdy shapes

Sturdy shapes don’t have frills. In a workhorse body-text typeface, we want extreme subtlety. Close up, good body-text faces tend to look boring—but that lack of conspicuous features is a purposeful design decision (**FIG 3.12**). When people are trying to read, you don’t want them noticing the choice of typeface or, worse, tripping over a design detail.



FIG 3.12: Top: Michał Jarociński’s Macho makes for a surprisingly strong text face, with organic curves and an athletic build. Bottom: the extensive Haboro superfamily’s Book style does not work well for body text.

Take care to avoid weak shapes. What makes a shape weak? Watch out for spots that seem too thin—like delicate lines, scrawny *terminals* (the ends of strokes), or dainty serifs. Think about letters as structures. Do they feel steady, or could they easily be knocked down? Body text needs to stand up to many different contexts.

Comparing typefaces can be especially useful. When making comparisons, you’ll find that good body-text typefaces are often a little bigger for their size (glyphs are larger within the em box, as we discussed in Chapter 1), and wider, with higher x-heights. These qualities make letterforms feel sturdy (a bigger, wider shape is harder to knock down), but they also affect the text block.

When letters are big and wide, they tend to have more interior white space. This means that *letter spaces* (the spaces between letters) should be a bit larger, which may in turn affect word and line spaces (**FIG 3.13**). So in addition to studying shapes for sturdiness up close, observe their effects in blocks of text. Apply the typeface you’re studying to a paragraph, and

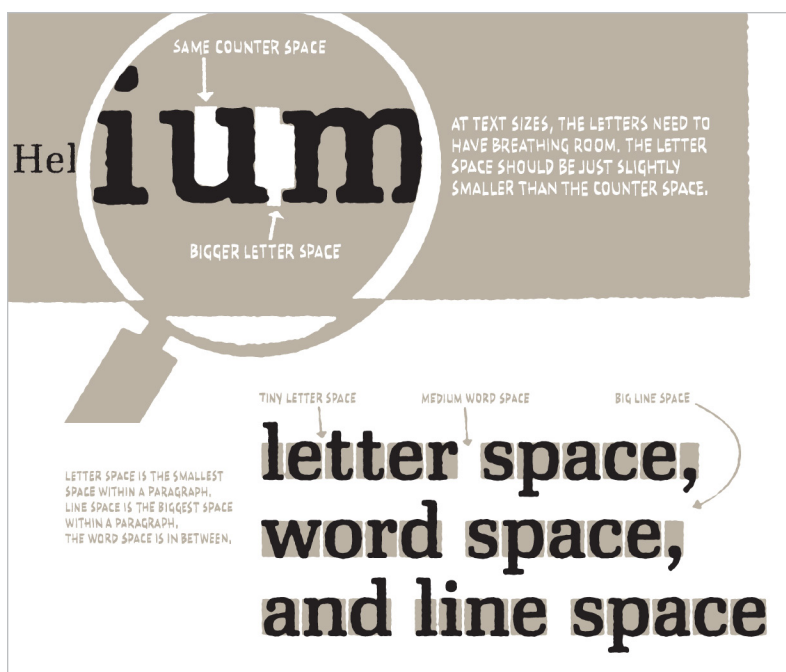


FIG 3.13: Top: in *Inside Paragraphs*, Cyrus Highsmith explains that letter spaces should be bigger at text sizes—but still smaller than the type’s counter spaces. Bottom: Highsmith illustrates different white spaces in text blocks, and describes their relationship (<http://bkaprt.com/ft/03-06/>). Images courtesy of Cyrus Highsmith.

look at the paragraph. It helps to use real content from a project you’re working on; alternatively, grabbing a public-domain text from Project Gutenberg can be handy too (<http://bkaprt.com/ft/03-04/>).

When you think you’ve identified a typeface worth exploring, get a feel for it by messing around with it a bit (**FIG 3.14**). Adjust the font size, line spacing, and letter/word spacing just slightly, and see what you think. While you’re at it, watch for shapes that seem like they don’t need to be there, even if they feel sturdy. In *Reading Letters*, Sofie Beier calls this “irregularity,” and while it can be great for other jobs type does—in signage

gua de Guatemala, the moribund convent held no interest. Occasionally one of the "nuns" whose conscience troubled him would leave an offering of food at the table and mumble a request for prayers of intercession; or the dark-eyed half-Spanish man would stare with something of both fascination and fear at the five white-clad ancient women. Morning and evening, crossed the patio to the chapel: Sister Eulalia on the arm of Sister Teresa, Sister Rose de Lima and Sister Catalina, one on each side of the Mother Superior.

and tear at the five white-clad ancient women who, morning and evening, crossed the patio to the chapel: Sister Eulalia on the arm of Sister Teresa, Sister Rose de Lima and Sister Catalina, one on each side of the Mother Superior. To these two young nuns—their years were but sixty-six and sixty-nine—had fallen, by common consent, the care of the Mother Superior, whose age no one knew, so great it was, and which the nuns loyally concealed. By them her wandering sentences were received.

on and tear at the five white-clad ancient women who, morning and evening, crossed the patio to the chapel: Sister Eulalia on the arm of Sister Teresa, Sister Rose de Lima and Sister Catalina, one on each side of the Mother Superior. To these two young nuns—their years were but sixty-six and sixty-nine—had fallen, by common consent, the care of the Mother Superior, whose age no one knew, so great it was, and which the nuns loyally concealed. By them her wandering sentences were received.

FIG 3.14: Good body-text typefaces often have great letter and word spacing by default—but some require tweaking. Does one of these examples look better to you than the others?

and user interfaces, where unique shapes aid legibility—it is distracting to readers of running text (**FIG 3.15**).

Now that you have a paragraph of type to work with, you can evaluate its color.

Even color

Good body text typefaces also have an even *color*—meaning typographic color, the overall gray value of the text. Sometimes it helps to squint at a text block to zero in on this gray value. What we're looking for is a consistent, pleasant gray. Avoid typefaces that seem to have white spots or dark patches, because these inconsistencies can distract readers. We want a nice, smooth, gray tone (**FIG 3.16–3.17**).

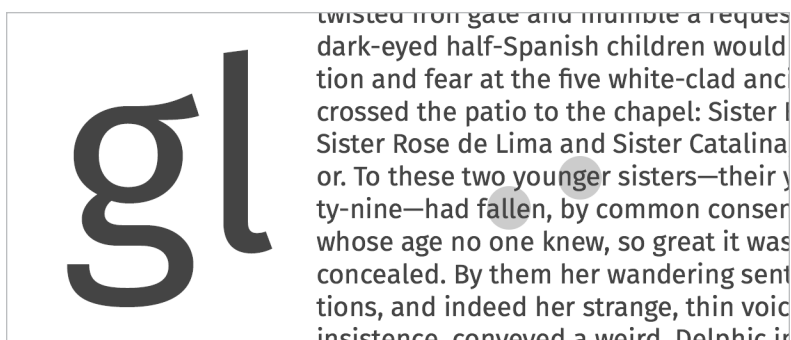


FIG 3.15: Erik Spiekermann designed Fira Sans for user interfaces, where recognizable shapes aid clarity; in body text, these same shapes can be off-putting.

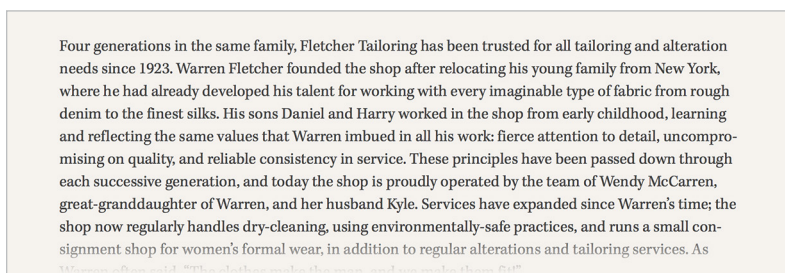


FIG 3.16: Robert Slimbach's Kepler produces a very even typographic color.

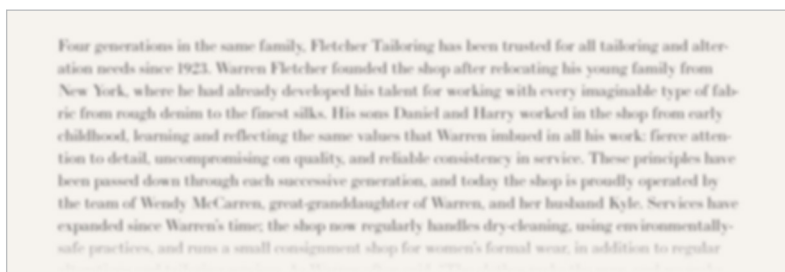


FIG 3.17: Squinting at type makes it easier to gauge typographic color. Here, LTC Bodoni's thick strokes create dark patches that stand out from the rest of the text block. Try the "squint" feature in this Typekit Practice lesson about selecting typefaces (<http://bkaprt.com/ft/03-07/>).

from physical and interpersonal threats to reproductive advantage to those who dream, dreaming evolved to replicate continually practice dealing with them. Therefore, Revonsuo shows that contemporary dreams comprise much more threatening events than people meet in daily non-dream life, and the dreamer usually engages appropriate simulation exclusively.

According to the Threat Simulation Theory, during much of human evolution physical and interpersonal threats were serious, giving reproductive advantage to those who survived them. Therefore, dreaming evolved to replicate these threats and continually practice dealing with them. In support of this theory, Revonsuo shows that contemporary dreams comprise much more threatening events than people meet in daily non-dream life, and the dreamer usually engages appropriate simulation exclusively.

Psychologist Antti Revonsuo posits that "dreaming is threat simulation" exclusively. According to his Threat Simulation Theory, he proposes, during much of human evolution physical and interpersonal threats were serious, giving reproductive advantage to those who survived them. Therefore, dreaming evolved to replicate these threats and continually practice dealing with them. In support of this theory, Revonsuo shows that contemporary dreams comprise much more threatening events than people meet in daily non-dream life, and the dreamer usually engages appropriate simulation exclusively.

Antti Revonsuo posits that dreams are "threat simulation" exclusively. According to the Threat Simulation Theory, during much of human evolution physical and interpersonal threats were serious, giving reproductive advantage to those who survived them. Therefore, dreaming evolved to replicate these threats and continually practice dealing with them. In support of this theory, Revonsuo shows that contemporary dreams comprise much more threatening events than people meet in daily non-dream life, and the dreamer usually engages appropriate simulation exclusively.

sonal threats were serious, giving reproductive advantage to those who survived them. Therefore, dreaming evolved to replicate these threats and continually practice dealing with them. In support of this theory, Revonsuo shows that contemporary dreams comprise much more threatening events than people meet in daily non-dream life, and the dreamer usually engages appropriate simulation exclusively.

at simulation" exclusively. According to his Threat Simulation Theory, he proposes, during much of human evolution physical and interpersonal threats were serious, giving reproductive advantage to those who survived them. Therefore, dreaming evolved to replicate these threats and continually practice dealing with them. In support of this theory, Revonsuo shows that contemporary dreams comprise much more threatening events than people meet in daily non-dream life, and the dreamer usually engages appropriate simulation exclusively.

FIG 3.18: These typefaces all have an active texture. Work Sans (top left, loose and a bit unbalanced) and Gitan Latin (middle right, somewhat bold and tight) feel a bit too active for body text, but could work very well in small doses. Basic Sans (bottom left) and Edita (bottom right) are full of personality—active but not overactive. Europa (middle left) is a striking face for display use, but a bit dull for text. Bagatela (top right) has a beautifully balanced texture. Give Basic Sans, Edita, and Bagatela each a touch of letterspacing.

Letter- and word-spacing adjustments affect typographic color, so stick with the typeface's default spacing for the most part, or adjust with extreme sensitivity. Also, make sure you're looking at type set with a rag edge—in other words, left- or right-aligned rather than justified. That way, you'll know the word spaces you're seeing are part of the font itself and not the result of justification spacing.

Although we're looking for evenness here—the lightness or darkness of the color doesn't matter as much—finding a type family with several weight variations that are good for body text can be helpful. Later on, when we're balancing the composition, it might come in handy to be able to slightly lighten or darken the typographic color of body text by adjusting its weight.

Active texture

In addition to having color, text blocks also have *texture*. Think of the strokes and spaces in a text block like the fibers and gaps in a piece of fabric. Just as there are different weaves and patterns, different typefaces yield different textures.

To evaluate typographic texture, sharply observe the text block. Feel for the horizontal rhythm of black and white. Observe any patterns of movement in the glyphs. Looking over these shapes, you should feel they are lively and balanced—neither too dull, nor too active (**FIG 3.18**).

If this kind of evaluation seems frustratingly abstract, don't be discouraged. The kind of sensitive evaluation required for judging texture is indeed abstract. It's the same kind of abstract visual analysis that great type designers engage in themselves as they're designing typefaces. Beef up your skills by practicing, and practice by staring at type for a long time. Keep notes.

Appropriate type for body text

For most people, chairs are part of their everyday life. They use them throughout the day, from their kitchen counter stool, to the desk chair, to the restaurant booth, to the lounge chair in front of the TV. And as they go through the day sitting in various kinds of furniture, they don't think twice about it unless something feels wrong. They know immediately when the chair isn't right for its purpose.

—Stephen Coles, “A Typeface is a Chair” (<http://bkaprt.com/ft/03-08/>)

Who doesn't love a nice, comfy recliner? You lean back, put your feet up, and relax. But sprawling out in a recliner isn't very appropriate when you're entertaining guests, and you certainly wouldn't slide your recliner over to the dining room table to eat from. Finding the appropriate chair is as important as identifying a good chair. Similarly, finding the appropriate typeface is as important as identifying a good typeface. Over time, I've developed a process for determining whether a given typeface

is appropriate for the task at hand. Follow along; perhaps you'll find it useful too.

Read the text

At this point in the process, you need to start thinking about the context of the project for which you're choosing type. Project Gutenberg text isn't going to help you decide whether a typeface is appropriate for a real project.

So first, read. You *must* have a close relationship with the content, voice, and tone of the text for which you're trying to decide whether the type is appropriate. As we discussed in Chapter 2, the better you know and understand the text, the more successful you will be.

Examine what the type offers

Next, study the typeface in detail. Explore its variations. Survey the language support of its glyph set to be sure it works for your text.

Review the typeface's available OpenType features—for body text, you'll want *proportional oldstyle figures* (lowercase numerals designed for text, not tables). You'll also want small caps, "caps to small caps" (a separate feature that turns capital letters into small caps), and standard ligatures. Note that ligatures exist to solve spacing problems, and not every typeface requires them.

Try the type at different sizes, and with different amounts of text. Make sure the family provides a means of emphasis (italic, bold) unless you plan to use a different typeface for emphasis (which is possible, but not common).

Prefer typefaces that were *designed* for text sizes. We just spent a few pages identifying qualities that show whether type is good for body-text use; well, some typefaces were drawn and produced with those precise qualities in mind (**FIG 3.19**).

Type made for use at text sizes generally has lower contrast, thicker features, and more generous spacing than type made for use at larger sizes. Personal computers have made it easy to scale type up or down—just change the font size!—but every font has



Roger Roger
Roger Roger

FIG 3.19: Left: Tobias Frere-Jones's Mallory and Mallory MicroPlus. Right: Robert Slimbach's Minion Pro optical sizes for display and text.

size limits, beyond which it begins to look out of place. Type designed for text use makes those limits clearer. To learn more about size-specific adjustments to type designs, I recommend the aptly titled *Size-specific Adjustments to Type Designs* by Tim Ahrens and Shoko Mugikura (<http://bkaprt.com/ft/03-09/>).

Consider variable fonts, too. A *variable font* is, as John Hudson has written, “a single font file that behaves like multiple fonts” (<http://bkaprt.com/ft/03-10/>). Jointly developed by Microsoft, Google, Apple, and Adobe, variable fonts are officially known as OpenType Font Variations. The CSS specification for working with variable fonts is in its infancy, but browser and operating system support is progressing quickly. Get comfortable with variable fonts, because they’ll soon be the new normal (**FIG 3.20**).

Variable fonts are to traditional fonts as websites are to print typography. They will enable us to typeset flexible compositions with elegance and subtlety, by stretching and squeezing our typefaces in nuanced ways. The adjustable qualities of a variable font are available in what are called *axes* of the *design space*.

Think of the design space as all of the possibilities inherent in a variable font, and axes as possibilities of a specific kind. Weight is an axis (light, bold, and everything in between). Width is an axis (condensed, extended, and everything in between). There are *registered* axes, which are common and predictable

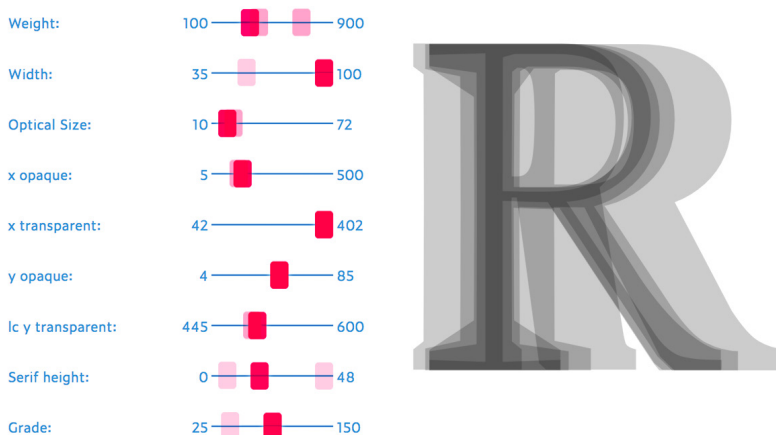


FIG 3.20: This screenshot from Nick Sherman’s v-fonts.com shows a single variable font—David Berlow’s Amstelvar—with its settings tweaked (<http://bkaprt.com/ft/03-11/>).

(weight and width fall into this category) as well as *custom*, or *foundry-defined* axes, which are wide open to experimentation and can later be registered and added to the OpenType specification (<http://bkaprt.com/ft/03-12/>).

Some examples of early custom axes are “Worm” (Decovar, by David Berlow), “Splatter!” (Jam, by Erik van Blokland), and “Cookies” (Buffalo Gal, by Tom Rickner). Those all sound wild, and in fact they do some pretty wild things! But custom axes are also being used to explore subtle, sophisticated effects that will help produce smarter defaults. For example, David Berlow’s Amstelvar arrives at optical sizes by compounding several very specific axes (<http://bkaprt.com/ft/03-13/>). You can check these and other variable fonts out by visiting Axis-Praxis, a website for playing with variable fonts (<http://bkaprt.com/ft/03-14/>).

Major benefits of variable fonts will become clearer in Chapter 5, but I wanted to mention them now to get you thinking about what their existence means for the practice of typesetting.

See how the type looks

Spend time looking at the type. Preview a type specimen in a variety of different operating systems and different browsers, as well as on different screens and in different mediums. Make sure the typeface looks good and works well in every context that matters to you and your potential readers. If you're not sure where to start, head over to your local library and look at the type on one of their computers. Find an older mobile device and see how the type looks on that—try a brand you don't regularly use. When you're ready for more, consider loading the type specimen at CrossBrowserTesting.com, which provides a wide range of screenshots to study.

Identify distracting glyphs that might disqualify the typeface. If your text has many proper nouns, for example, a typeface with prominent capital letters may disturb readers. Review glyph combinations that show up often in your project's text, too—if the words “giggle” and “egg” are used many times, for instance, consider type with a simpler *g* that doesn't attract much attention.

Aside from critical inspection, try to get a general feel for “whether the letterforms represent the personality of the voice you're trying to evoke,” as Aura Seltzer once put it (<http://bkaprt.com/ft/03-15/>).

Describe the type

Your type choice should possess the qualities, and evoke the atmosphere, that you're trying to convey. Consider using word association as a way of articulating those qualities, of capturing that atmosphere.

Remember that your feelings about a typeface are valid, no matter how little experience you have. You may not be able to justify your feelings with a cogent argument right away; that comes with time and practice. Even so, being able to clearly describe type is a valuable skill that improves your ability to make tasteful choices. Trust your instincts, keep an open mind, and practice finding the right words.

Although you probably won't want to talk to clients or stakeholders about how type makes you feel, you *will* eventually need to convince other people that your design solution is successful by focusing on a project's business goals (<http://bkaprt.com/ft/03-16/>). And for that, you need persuasive language and cogent arguments at the ready.

Consult testimonials and examples

To get to know a typeface better (and therefore be able to describe it better), read about why it was made. Investigate the designer's motivations; explore the typeface's historical associations and cultural influences. Find out what people think of it. Read typeface reviews at Typographica (<http://bkaprt.com/ft/03-17/>). Study how the typeface has been used. Look at Fonts In Use (<http://bkaprt.com/ft/03-18/>). If there's a designer, agency, or company whose work you really enjoy, check to see which typefaces they're using; you can find bookmarklets to help with this at Typography Supply (<http://bkaprt.com/ft/03-19/>). Think about what people were trying to convey by using those typefaces. Or ask them.

Relate to the brand, if necessary

Your text choice may need to be compatible with a particular style of lettering, or a logotype. Because much lettering, and many logotypes, are based on the same written forms and geometric ideas as type, treat this relationship like you would any other in your type palette. If you're working with brand guidelines that already include a text face, reread this chapter with that typeface in mind—make sure it's both good and appropriate for body text.

Be different, but not too different

Compare the typeface you're considering with the usual kind of type found in the genre of text you're typesetting. Try to strike a balance between familiarity and novelty so that readers get

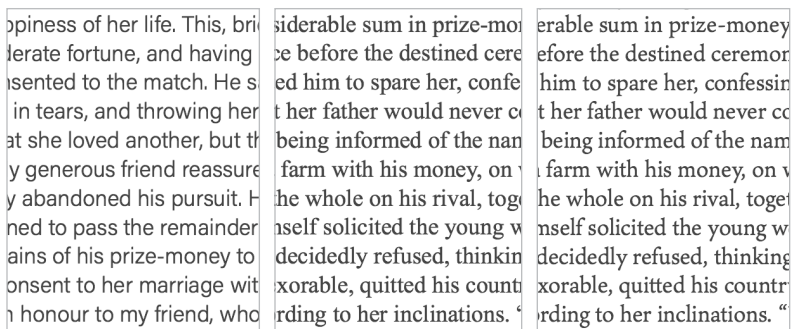


FIG 3.21: Left: Acumin is a readable sans serif, but very different from what people normally see in this context. Center: Times New Roman, a common typeface for novels. Right: Ten Oldstyle feels both comfortable and refreshing.

what they expect but also don't get bored. For example, novels traditionally use serified type. Using a face like Times New Roman might seem boring because it's so familiar, whereas a sans serif might feel out of place. Try a serified face that reads well and feels fresh (**FIG 3.21**).

Consider compositional contrast

Body-text type should be understated relative to other visual elements in a composition. That way, elements whose job it is to stand out don't have to work as hard to grab a reader's attention.

We talked about evenness of color, and activity in texture, in body text. Watch out for color that is heavy or dark, and texture that is overly active, because these things make it harder to establish compositional contrast; details that should stand out end up needing to work harder for attention.

Save your work

Identifying body-text typefaces that perform well as anchors is a great investment—not only for the project you're working on now, but also for your future self.

If a typeface is objectively good for body text, that never changes. Not every typeface will be appropriate for every project, but once you've determined that a typeface is good for body text, you can consider it for any project you do in the future. So keep a running list of good typefaces, and write down your thoughts about why they are good.

While determining whether type is *appropriate* is a difficult task, it's extremely satisfying. Record your reasoning and bookmark good resources. Every step you take builds your confidence and creates opportunities for future study.

BUILD A TYPE PALETTE

Once you have a body-text typeface picked out to anchor the composition, you'll need to decide which typefaces to try for all the other text blocks in the inventory. There are a variety of strategies for doing this, but let me show you the process I use—and get us back into our St. Remy example project. Schedule some time for this activity, and set a timer so you know when to quit (two or three hours is usually perfect).

Make loose choices

First, look for typefaces that complement your anchor. Make loose choices, gathering six to eight options. You should have a specific text block in mind to style with one of your choices. Consider variations that are part of your anchor typeface family, too—concentrating on a single family's weights, widths, and more expressive styles (if they exist) is a great way to begin exploring a palette.

Below are six typefaces I might potentially use for St. Remy's big display type (FIG 3.22). I found these typefaces by browsing around, but I also enjoy visiting *Fonts In Use* to see how my anchor typeface has been used in the past—and which, if any, other typefaces have been combined with it (<http://bkaprt.com/ft/03-20/>).

These were chosen quickly, without much consideration, and you can probably tell that some of the combinations don't work very well. But that's fine, because making loose choices at

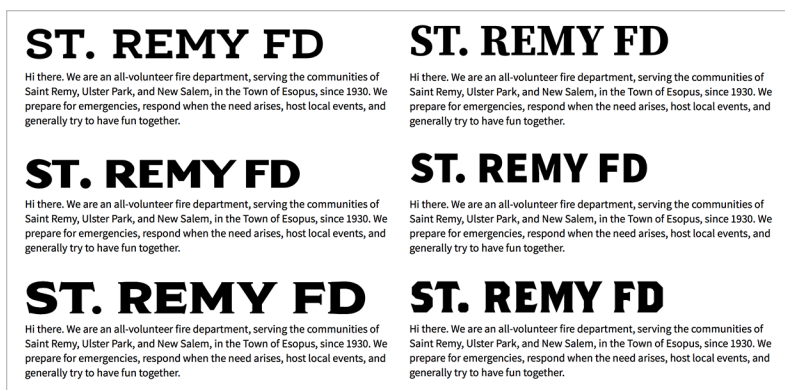


FIG 3.22: Six potential additions to my type palette. Clockwise from top left: FF Ernestine, Source Serif, Source Sans (a bolder weight of our anchor face), Brothers, Modesto, and Dogma.

this point in the process leaves me open to delightful surprises. It fosters a richness and depth in my type palette that I couldn't achieve by being too rational early in the process.

Do a quick tech check

Go over the technical details (language support, features, glyphs) of every typeface choice you just made. Also, look at how each typeface renders in different environments. (Flip back to the “Examine what the type offers” and “See how the type looks” sections from earlier in this chapter if you need to refresh your memory about these technical tasks.)

Be quick about it. Don't dive into research the way we did for the anchor typeface, for a couple of reasons. First, that's a time sink. Your task here is to make a choice so you can start using the type. Second, you're already invested in your anchor face. Researching more typefaces before you try them can psychologically lock you into a combination before you're happy with it *visually*. That can leave you fighting to make it work, pitting a research-based rationale against your aesthetic sensibilities.

If a typeface passes your tech check, congratulations! Read on.

If no typefaces from the initial set look like viable candidates, make a few more loose choices and try again. Don't give up. Even if your preliminary exploration didn't pan out, you found a handful of interesting typefaces. Write down not only what you liked about them, but also why they didn't work, for future reference.

Seek graphic harmony

Now we're ready to begin a thorough evaluation by putting different typefaces together in one space and studying what we see. Our goal here is to decide whether these typefaces together produce a harmonious visual experience.

Gather the candidates for your type palette, and get them all onto the same surface. Do this in whatever design software you prefer, or carefully position browser windows so that you can see how the different typefaces look together (**FIG 3.23**). (I often use browser windows, because I like to see how the type renders in a browser.)

Arrange the previews of your anchor typeface and a potential palette addition so they reflect how the type will be used in your composition. Using real text from a project helps, but you can make do with placeholder text. Sit back and judge the previews from a short distance.

Study texture

Make sure the potential palette addition is doing its job. Squint at the previews, or back up a little so you stop seeing words and see gray masses instead. If the new typeface is being used as display text, it should stand out. If it's for an interface, it should be a little clearer than the rest of the text even as you squint. If it's for supplementary text, it should recede or have about the same color and texture as the anchor face.

Now, without squinting, come closer to the text. Examine the visual activity generated by the combination of typefaces. Look at the angles and curves that characterize the body text. Is there a hint of that activity in the other typeface? That usually helps two faces work well together (**FIG 3.24**).

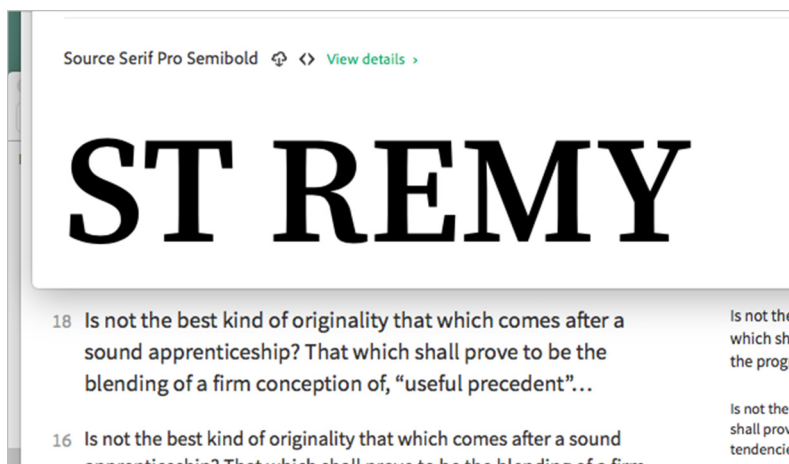


FIG 3.23: Positioning browser windows is a fine way to compare typefaces and consider palette combinations.

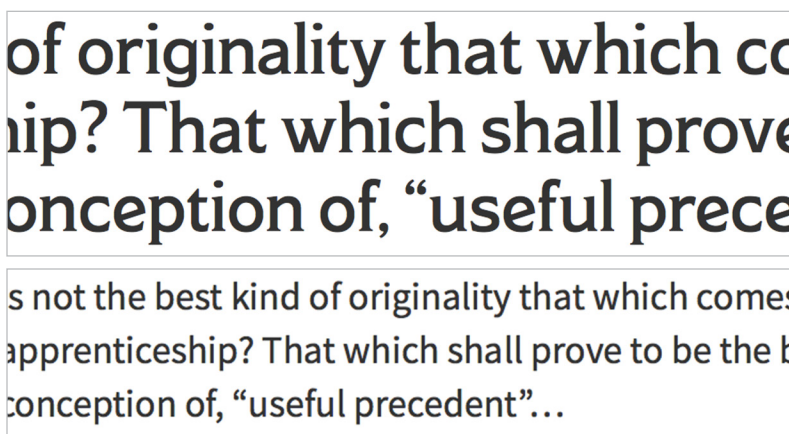


FIG 3.24: The lowercase *t* in Source Sans (bottom) has a slight taper, which gives it an upward energy; that same energy is exaggerated in the lowercase *t* of Modesto Text (top). Look at the liveliness in the white space of the letter sequence *use* in both typefaces. Look at the attitude in the question marks. Modesto has a very happy, casual vibe that really comes out in its lowercase. Although I only plan to use Modesto Expanded, which does not contain lowercase glyphs, I find it helpful at this stage to study members of palette candidates’ typographic families that do have lowercase glyphs. It helps me understand the type designer’s thought process, and allows me to compare the typefaces more directly.

Compare the energy of the typefaces. How does the potential palette addition look when compared with the anchor typeface? Is it blocky? Angular? Fluid? Is it tight or loose? Is it calm or vibrant? Jumpy? Relaxed? Some of these adjectives will feel right for the project you're working on; some won't. Recall the descriptive words I chose for St. Remy back in Chapter 2: personal, welcoming, informational, encouraging, brief, clear.

Study rhythm

Set up new previews. Then set one line of each typeface, all at a medium size. Look at the white shapes within letters, and between letter combinations. With your eyes, loosely measure the volume they consume, and how regularly those volumes repeat themselves in a series of glyphs. Survey the black shapes of the glyphs, as well, for their regularity. Notice the balance between black shapes and white spaces. Well-made typefaces will have a black-and-white rhythm that feels steady and organized (**FIG 3.25**).

The rhythm of the potential palette addition should have a relationship to the rhythm of the text face that reflects how they'll be used together. If the palette addition is supposed to stand out, a contrasting (but still orderly) rhythm helps. If it's supposed to blend in, a similar rhythm will be critical.

Study proportion

Next, get closer to the letterforms. Look at specific letters—the same ones from each of the typefaces you're evaluating. You're trying to find compatible proportions. How much taller are the capitals than the lowercase letters? How wide are some glyphs in relation to their height? How high, or how deep, do extenders reach (**FIG 3.26**)?

Shared proportions throughout a typeface palette can work wonders in a composition, making it feel complete and tranquil in a way that's hard for people to describe without taking a very close look at the type. Contrasting proportions in the palette, on the other hand, can make a composition look agitated and



FIG 3.25: From top to bottom: Modesto, Source Sans, and Brothers. These typefaces each have an attractive, steady procession of vertical positive and negative spaces. Each, on its own, has a pleasant balance of black and white, but Modesto feels more like Source Sans than Brothers does. Shared rhythm in a palette produces harmony in the composition; for this project, we want that harmony.

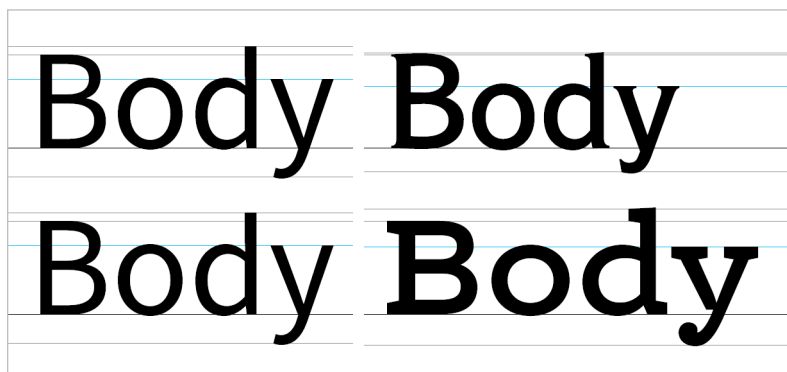


FIG 3.26: Modesto (top right) and FF Ernestine (bottom right) both have proportions that I find compatible with those of Source Sans (left). Modesto is a closer match—its proportions are tighter all around, as befits a display face, so it's our winner for this particular project. But Ernestine would also be a fine choice. Its vertical proportions are extremely similar to those of Source Sans, and the two typefaces' contrasting widths could end up making a powerful statement.



FIG 3.27: Going strictly by their outlines, these are very different styles of type—but they are quite alike in terms of contrast (thicks and thins) and structure. Compare the leg of the *R*, the aperture of the *c*, and the way the diagonal stroke of the *k* connects to the stem. And the slope and curve of the lowercase *a*'s bowl make for a similar shape in both examples. These likenesses help make the two typefaces feel compatible.

discordant. Use proportional relationships to achieve the effects your composition needs.

Study shape

Look even closer at the type now, enlarging it so that it's easier to concentrate on the abstract shapes that make up the glyphs. Do the shapes seem to be based on geometry, or perhaps on the movement of a writing implement? How different are the thicks and thins? Are the shapes angular? Smooth? What words would you use to describe the contours you see?

When you're trying to articulate your thoughts about this, it helps to know a little about typeface anatomy. For that, you can find plenty of resources online, or take a look at a book like *The Anatomy of Type*, by Stephen Coles (<http://bkaprt.com/ft/03-21/>).

Shapes don't have to be similar to work well together. For instance, a square and a circle are very different, but they're

both geometric—they feel more compatible with one another than either would alongside a more calligraphic shape (FIG 3.27).

In studying shape, what you're trying to do is identify cues that make the typefaces feel related. Those cues will help hold the palette together, no matter how you use the type.

Forge ahead with a rich, full palette

You now have a palette of typefaces, anchored by a body-text face that is both objectively good and appropriate for the project. Nice work!

The best way to feel satisfied about these choices is to try them out—as we did with our CodePen examples—in design software, or at least with the type-tester previews available on some websites. Later, if you see that a particular combination doesn't work the way you had hoped, you may return to this selection process to edit your palette. That's to be expected.

You may also be tempted to reduce your palette—doing more text-block jobs with your body-text typeface in order to load fewer font files, or subsetting webfonts to help them load faster—but try not to give in to that temptation. Never underestimate the power and usefulness of a big type palette. And don't make any decisions about cutting back until you've begun crafting a composition and can really evaluate it, which we'll do together in the coming chapters.

It's time to roll up our sleeves and really start getting our hands dirty. There are many ways to begin crafting a composition, but I like to start by shaping text blocks. You'll soon see why.

4 SHAPING TEXT BLOCKS

NOW YOU HAVE A TEXT that you understand and care about, and have chosen some typefaces that you intend to use. You also have some compositional sketches from your text-block inventory. This bird's-eye view will guide you as you focus on shaping individual text blocks (**FIG 4.1**).

At this point in the process, people often reach for a shortcut. They look for a system, a script library, a framework—some preexisting code they hope will make it easier to manage a site's typography or save development time by handling all of the details, so that they can focus on other, more “important” stuff.

Being efficient is great, but you should understand the decisions a typographic system makes on your behalf. Especially because, as you'll see in this chapter, the flexible, related decisions we need to make about typography are not something any system or tool currently supports.

Let's look at all of the decisions that need to be made in shaping text blocks, so that you can either make them from scratch or properly vet future systems that make them for you. Fuel up and focus. This chapter is where everything comes together.

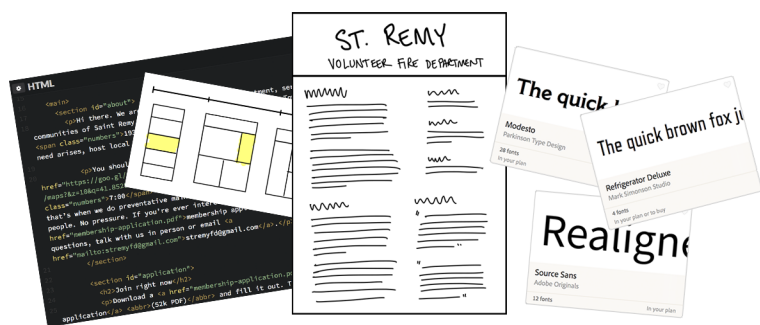


FIG 4.1: Gather all of the pieces you need to begin shaping text blocks.

SET THE BODY TEXT TYPE

We'll begin by styling all of the text in our St. Remy example project as if it were body text (specifically, paragraphs). Because body text usually represents the majority of text in a project, it makes sense to style everything this way first, and then later style the elements that need to stand out.

We'll also begin mobile first. You've heard this approach recommended before—Luke Wroblewski's book of the same name works well for typesetting (<http://bkaprt.com/ft/04-01/>). With a mobile-first approach, we start by addressing very limited layouts at the narrow end of our compositional continuum. The typesetting conventions we establish in this constrained context will help later, when we move to roomier layouts.

Initially styling everything as body text, and mobile first, may differ from your usual approach to designing for the web. That's okay. I'm not asking you to change your process. Just stay with me, and I'll show you the reasons for some common pressures you run into in layouts. Learning about this stuff will help you in future projects, no matter where you start.

Going back to our example project, let's turn what we see on the left into what we see on the right (FIG 4.2). This may not look like a significant visual change, but remember: we're

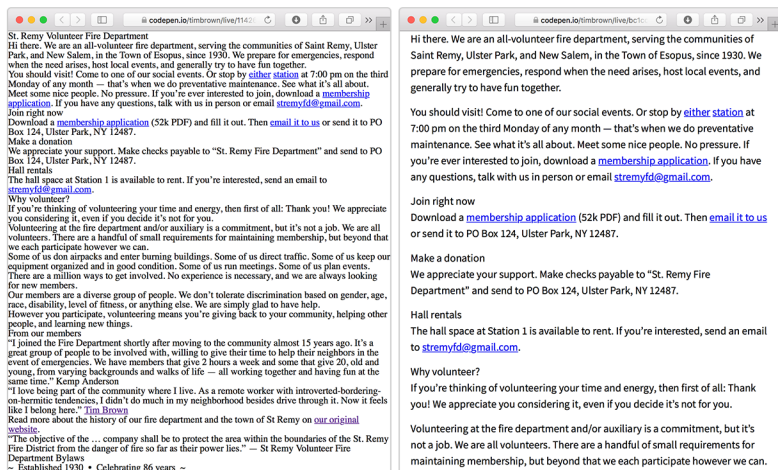


FIG 4.2: Left: St. Remy’s cleaned-up, marked-up, unstyled body text as we left it in Chapter 2. Right: the same St. Remy text, styled with the typeface we selected in Chapter 3—plus a few other rules we’re about to discuss.

laying a foundation for flexible typography that looks good and performs well in many contexts.

To do this, we’ll add a few declarations that override the basic decisions in the reset stylesheet from Chapter 2. Open up that Pen and paste this into the CSS pane:

```
:root {
  font-size: 100%;
}

body {
  font-family: sans-serif;
  line-height: 1.5;
}

main {
  width: 40em;
}
```

Code example: <http://bkaprt.com/ft/02-14/>

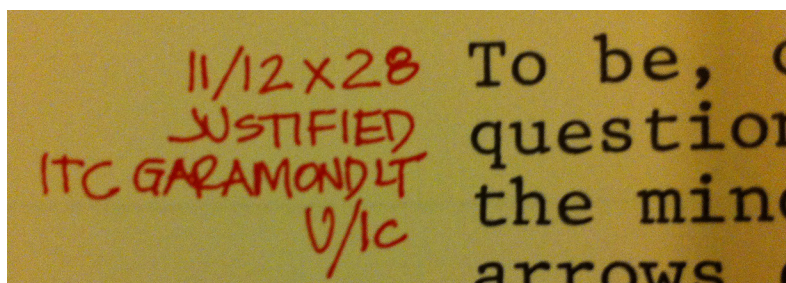


FIG 4.3: This photo from my copy of Alex White's *How to Spec Type* shows a typographer's instructions to set text in 11pt ITC Garamond LT, with 12pt line spacing (1pt of leading between lines) and a measure of 28pt.

The values here aren't final. We'll decide on the appropriate values as we go along. The CSS here represents the essential properties of a text block—the typeface it uses (**font-family**), as well as the font size, line spacing (**line-height**), and measure (**width**). Producing a smooth text block by subtly adjusting the values of these four properties is what typographers have been doing for centuries.

Look, this traditional formula has the same four properties as our CSS (**FIG 4.4**)! Graphic designers used to mark up manuscripts like this, sending instructions to print shops, which would execute those instructions—much as we send instructions to browsers and devices today. Fortunately, we now have a much quicker turnaround time; we can see the results of our adjustments immediately, which means we can instantly evaluate them and keep tweaking.

Tweaking is important. Finding an effective balance among these four properties is critical because *they are all related*. If you change one property, you almost certainly need to change the others. See for yourself—making changes to this text block gives rise to the need for additional adjustments (**FIG 4.4-7**).

If you're wondering how this jibes with the flexibility of responsive layouts, then you're on the right track. You're starting to sense the pressure we talked about in Chapter 1. We'll begin talking about pressure in more detail in the next section.

Clerval had never sympathized in my tastes for natural science; and his literary pursuits differed wholly from those which had occupied me. He came to the university with the design of making himself complete master of the oriental languages, and thus he should open a field for the plan of life he had marked out for himself. Resolved to pursue no inglorious career, he turned his eyes toward the East, as affording scope for his spirit of enterprise. The Persian, Arabic, and Sanskrit languages engaged his attention, and I was easily induced to enter on the same studies. Idleness had ever been irksome to me, and now that I wished to fly from reflection, and hated my former studies, I felt great relief in being

Clerval had never sympathized in my tastes for natural science; and his literary pursuits differed wholly from those which had occupied me. He came to the university with the design of making himself complete master of the oriental languages, and thus he should open a field for the plan of life he had marked out for himself. Resolved to pursue no inglorious career, he turned his eyes toward the East, as affording scope for his spirit of enterprise. The Persian, Arabic, and Sanskrit languages engaged his attention, and I was easily induced to enter on the same studies. Idleness had ever been irksome to me, and now that I wished to fly from reflection, and hated my former studies, I felt great relief in being the fellow-pupil with my friend, and found not only instruction but consolation in the works of

FIG 4.4: Left: An example text block set in Open Sans. Right: the same text set in Kepler. Only the font has changed, yet the text now feels very different. To regain the original feel, font size needs to increase and line spacing may need adjustment.

Clerval had never sympathized in my tastes for natural science; and his literary pursuits differed wholly from those which had occupied me. He came to the university with the design of making himself complete master of the oriental languages, and thus he should open a field for the plan of life he had marked out for himself. Resolved to pursue no inglorious career, he turned his eyes toward the East, as affording scope for his spirit of enterprise. The Persian, Arabic, and Sanskrit languages engaged his attention, and I was easily induced to enter on the same studies. Idleness had ever been irksome to me, and now that I wished to fly from reflection, and hated my former studies, I felt great relief in being

Clerval had never sympathized in my tastes for natural science; and his literary pursuits differed wholly from those which had occupied me. He came to the university with the design of making himself complete master of the oriental languages, and thus he should open a field for the plan of life he had marked out for himself. Resolved to pursue no inglorious career, he turned his eyes toward the East, as affording scope for his spirit of enterprise. The Persian, Arabic, and

FIG 4.5: Left: the same example text block as before. Right: the font size has increased. Tighter line spacing or a wider measure would help the text block regain balance.

Clerval had never sympathized in my tastes for natural science; and his literary pursuits differed wholly from those which had occupied me. He came to the university with the design of making himself complete master of the oriental languages, and thus he should open a field for the plan of life he had marked out for himself. Resolved to pursue no inglorious career, he turned his eyes toward the East, as affording scope for his spirit of enterprise. The Persian, Arabic, and Sanskrit languages engaged his attention, and I was easily induced to enter on the same studies. Idleness had ever been irksome to me, and now that I wished to fly from reflection, and hated my former studies, I felt great relief in being

Clerval had never sympathized in my tastes for natural science; and his literary pursuits differed wholly from those which had occupied me. He came to the university with the design of making himself complete master of the oriental languages, and thus he should open a field for the plan of life he had marked out for himself. Resolved to pursue no inglorious career, he turned his eyes toward the East, as affording scope for his spirit of enterprise. The Persian, Arabic, and Sanskrit languages engaged his attention, and I was easily induced to enter on

FIG 4.6: Left: the same example text block. Right: a narrower measure. To rebalance the text block, a smaller font size or tighter line spacing would do the trick.

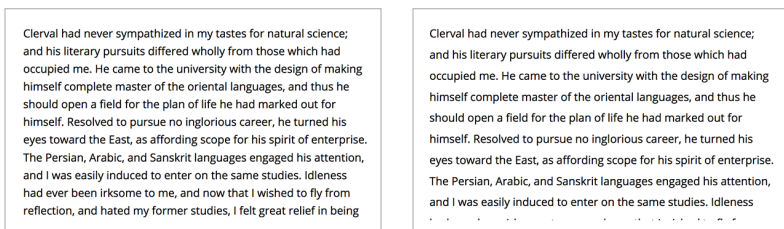


FIG 4.7: Left: the same example text block. Right: looser line. To regain balance, this text block needs a wider measure, a larger font size, or both.

Typeface, then size, then measure, then leading

You might also be wondering: If the properties of a text block are all related, and changing one means adjusting others, then which properties matter most? Which ones should we change first, and which ones should follow and react?

Choice of typeface always comes first. That's why changing the typeface in an existing project or system creates such a disruption—because many other decisions are affected by it.

The order in which we need to make further adjustments depends on the kind of typography we're practicing. Because we're setting type, we adjust font size, followed by measure, and then line spacing. (For other kinds of typography, like fitting or arranging type, the sequence is different.) Let's adjust the values of our four properties in that order: typeface, font size, measure, and line spacing. As we go, we'll confront the various pressures that arise.

SPECIFY A TYPEFACE

First up is the `font-family` property. For its value, let's replace `serif` with the anchor typeface we chose in Chapter 3, Source Sans Pro:

```

:root {
  font-size: 100%;
}
body {
  font-family: "Source Sans Pro";
  line-height: 1.5;
}
main {
  width: 40em;
}

```

This looks simple, but as we discussed in Chapter 2, there’s a lot going on logistically that makes putting the font’s name here possible. If you run into a glitch and you don’t see the font in your browser, reread the “Tackle Webfont Logistics” section from Chapter 2. Also, don’t worry about fallback fonts (alternative typefaces in the `font-family` property’s value) right now. We’ll talk about those at the end of this chapter.

Specify a variable font instance

If you’re using a variable font, now is the time to choose a specific *instance* of that font by adjusting its axis values. Think of this instance as an anchor point in the variable font’s design space. You should then refine and finalize that anchor point when choosing a font-size.

CHOOSE A SIZE THAT IS BOTH FAMILIAR AND GOOD

Next, we’ll choose a font size. This is a special part of typesetting; many other typography and layout decisions rest on this one. While other aspects of our text block (like measure and line spacing) will flex, the font size should be locked down. At least until the composition gets wider—but we’re not going to worry about that just yet. At the moment, we’re confining ourselves to styling text in a mobile-first, linear layout.

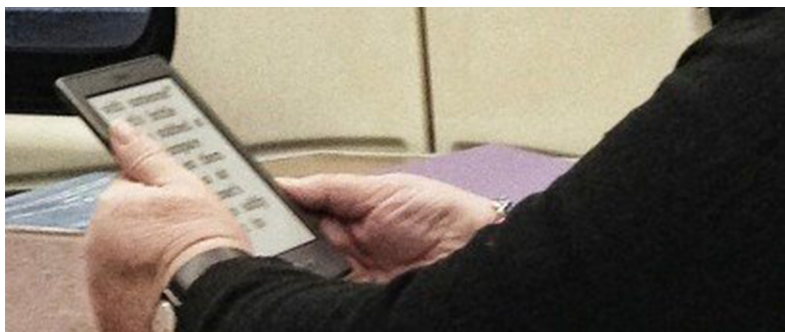


FIG 4.8: If someone prefers an extreme default font size, that's not wrong; it is absolutely right for that person. Image courtesy of Jon Tan (<http://bkaprt.com/ft/04-02/>).

Let's use a font-size of **1.15rem**:

```
:root {  
  font-size: 1.15rem;  
}  
body {  
  font-family: "Source Sans Pro";  
  line-height: 1.5;  
}  
main {  
  width: 40em;  
}
```

Notice that our root font size has changed. It now uses **rem** units—which, as we discussed when we framed the composition in Chapter 2, are a good way to size type relative to any given reader's familiar, default font-size.

It's true: sizing fonts this way means that our type could end up being literally *any size* (**FIG 4.8**). Some readers beef up their default font size quite a bit. If this makes you uncomfortable, please reread Chapter 1 for a reminder that readers now participate in typesetting. What's most important here is not that we, all-knowing designers, are prescribing a font-size—but that our font-size decision feels right to each individual reader.

As for the specific numeric value of the body-text font size, you can arrive at that either by going with your gut, or by getting extremely nerdy with font metrics and math.

Go with your gut

Look at specimens of your anchor typeface and try out a few different sizes. Play around and see what looks good to you. The size you're happy with will probably work great, though your chances of success with this approach will improve with experience. Whatever size you choose, convert it to ems. Ethan Marcotte wrote a handy explanation for this conversion (<http://bkaprt.com/ft/04-03/>). For a refresher on font size, revisit Chapter 2.

Sizing type doesn't have to be hard. Most likely, the size that looks good to you will be larger if the font is small for its size and smaller if the font is big for its size. (As we learned in Chapter 1, fonts with the same specified size are different actual sizes.) For something like Open Sans to look right, we need to set it a little smaller. Compare that to Kepler, which we'd set somewhat larger (**FIG 4.4**).

This is the basic idea when we're making gut decisions about font size. As an alternative, we can take a more scientific approach.

Use font metrics to find a familiar size

For our example project, I chose a numeric value—1.15—that I'm confident will feel familiar to each reader. How can I be so confident? Well, I measured the x-heights of the most popular fonts on the web over the past fifteen years, at the most common sizes. This is anecdotal data on my part, but I really have been paying close attention to this stuff for that long. I also made sure our font's x-height falls within that common range by setting its size accordingly (**FIG 4.9**). I'll explain how to measure x-height in a moment.

Putting the x-height in this range is what makes the font size feel familiar. Most lowercase letters operate within the x-height range; in body text, their mass produces the strong horizontal



FIG 4.9: A familiar x-height range, based on Times New Roman (left), Verdana (center), and some anecdotal data. Source Sans Pro (right) can be sized so that its x-height falls within this familiar range.

shape along which readers run their eyes. That massive band of letterforms needs to end up being a familiar size, regardless of how far extenders and capital letters protrude, and regardless of the actual `font-size` value in our CSS.

But what makes certain font sizes familiar?

Believe it or not, there was a time before webfonts existed. Back in the twenty-aughts, as browser support for CSS layout went mainstream, but before browser support for `@font-face` really took off, body text could only be styled using fonts that were present on a reader's device. Not everyone had the same fonts installed, so typographers' options were extremely limited. Most often, we stuck with Microsoft's "core fonts for the web" (<http://bkaprt.com/ft/04-04/>), which were actually pretty great. They were solid typefaces in a variety of styles; some, notably Matthew Carter's Georgia and Verdana, worked exceptionally well on screen—and still do, at small sizes and at coarse resolutions (**FIG 4.10**).

For more than a decade, these core fonts held our attention as designers, so nearly everyone reading anything on the web would have encountered them. As a result, they feel extremely familiar. And the sizes at which we would set them also feel familiar. Because screens at the time were very coarse, we gravitated toward specific fonts sizes that looked better to us. Of the core fonts most often used for body text, Times New Roman is the smallest for its size—when using it, `12px` is about as small as any reasonable designer normally ventured. Similarly, Verdana is the largest body-text-worthy core font for its



FIG 4.10: Some of Microsoft’s core fonts for the web: Times New Roman at 10px and 12px; Arial at 10px and 12px; Georgia at 10px and 12px; Verdana at 11px and 13px.

size; 14px is about as large as any reasonable designer would have set Verdana in the early days of the web (and even that was considered a bit large).

Since that time, we have begun seeing larger font sizes for text. I attribute this to increases in devices’ screen resolutions, advances in rendering-engine technology, and the availability of webfonts (which, until around 2014, often looked crummy at small sizes, so designers would set them larger). Because of all that, I bumped each of these numbers up by three pixels, to 15px and 17px. *Science!*

Then—and here’s where it gets interesting—I used these numbers to generate minimum and maximum familiar x-height values for body text. Here’s how:

- First, I measured the x-heights of Times New Roman and Verdana with a handy “Get x-height” measurement tool I made on CodePen (<http://bkaprt.com/ft/04-06/>). Those x-heights turned out to be .447 and .545, respectively.
- Next, I calculated the x-heights of Times New Roman at 15px and Verdana at 17px:
- 15px * .447 = 6.705px
- 17px * .545 = 9.265px

- Finally, I converted these values to em units using Ethan Marcotte’s cogent formula:
- $6.705\text{px} / 16\text{px} = .4190625\text{em}$
- $9.265\text{px} / 16\text{px} = .5790625\text{em}$

An x-height between $.419\text{em}$ and $.579\text{em}$ is what I recommend for body text, no matter what typeface you’re using—assuming it’s a good typeface for body text. Take your body-text font, measure its x-height, and be sure that when you determine font size, the x-height falls within this value range. Let’s do that for the St. Remy anchor typeface, Source Sans Pro; we don’t want it to be too big or too small.

- First, measure the x-height of Source Sans Pro using the “Get x-height” Pen.
- Next, divide 1 by that x-height to get its reciprocal: $1/.486 = 2.0576131687$
- Finally, calculate a size range for your font by multiplying the reciprocal of its x-height by the x-height limits we established earlier:
- $.419\text{em} \times 2.0576131687 = .862\text{em}$
- $.579\text{em} \times 2.0576131687 = 1.191\text{em}$

We now know that a font-size between $.862\text{em}$ and 1.191em should work nicely. Sure enough, choosing a number from the center of this range gives us an appealing font size: $(1.191\text{em} + .862\text{em})/2 = 1.0265\text{em}$. For St. Remy, I went with my gut and chose a slightly larger size, 1.15em , which also falls within this range. And then I switched it to rem units: 1.15rem . Because we’re styling the root font size itself, rems and ems behave exactly the same. I chose rems here in the interest of consistency, since we’ll be using rems for all of St. Remy’s other font sizes.

I used to believe that designers should look at type specimens to find a typeface’s “sweet spot”—the size at which that particular typeface looks its best. That was a smart thing to do when coarse screens were more common, because even a tiny size adjustment could make a typeface feel totally different. But screens come in much sharper resolutions now, and

type-rendering engines are better at making text look good in even low-resolution environments.

No exact, pixel-perfect font-size we choose is as important as sizing type relative to each reader's default settings and making that specific value feel as familiar as possible. Because no matter who you are, if you want to read *and text rises to meet you*, that's pretty sweet.

Adjust your anchor point

If you're using a variable font, now is the time to make any subtle adjustments to its anchor point—the way you want it to look by default, when it is not responding to any particular conditions in the browser. Then, having decided on font size, you might consider refining weight, width, or any other axis:

- A slight adjustment to the weight of your typeface may lighten the text block's typographic color, granting you greater flexibility in managing compositional contrast. We'll talk more about that in Chapter 5.
- Condensing slightly along the width axis may make for more efficient *copyfitting*. Copyfitting refers to the amount of horizontal space a running text consumes; line after line, a tiny nudge can add up to significant savings.
- Adjusting the *grade* of your typeface or the amount of *bracketing* serifs have—if your variable font contains such sophisticated axes—can make your text block exude a different vibe while maintaining its overall shape. Grades affect letterform thickness without changing copyfit; bracketing refers to the slope with which serifs transition into stems (the upright parts of letters).

These are just a handful of ideas for adjusting the way a variable font looks by default. Axes like these can also be tweaked to relieve pressure. We'll talk more about pressure in Chapter 6.

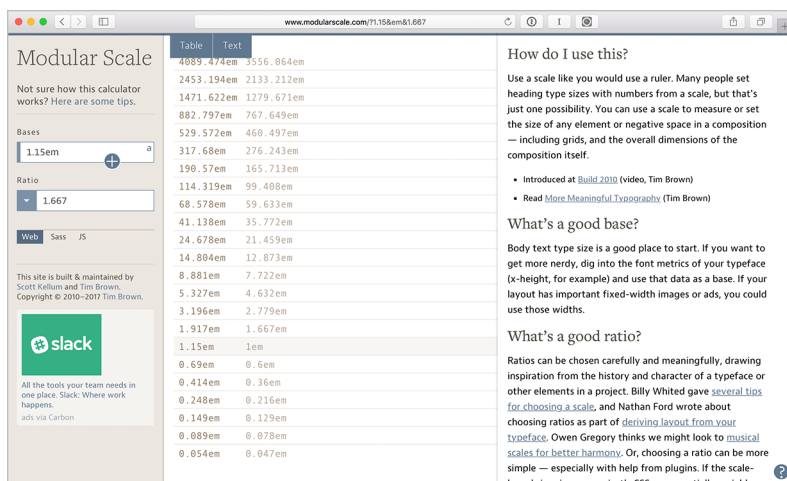


FIG 4.11: This modular scale calculator features a table view and a help panel with more information (<http://bkaprt.com/ft/04-07/>).

Make a modular scale

Now is also the time to generate a *modular scale*. A modular scale is like a ruler. It's a system of measurement you can refer to as you decide on sizes and spaces throughout your composition. But instead of inches or centimeters, the numbers of a modular scale are tailored to your project, because you base them on your body text. Head over to the calculator that Scott Kellum and I made (<http://bkaprt.com/ft/04-06/>). Enter the St. Remy body text size (**1.15em**) as a base number, choose “3:5—major sixth” for the ratio, and you're done (**FIG 4.11**).

The underlying idea here is that numbers related by a ratio feel harmonious. Compare the following examples. Study them enough, and you'll see that the shapes related by a ratio are more compatible (**FIG 4.12–14**).

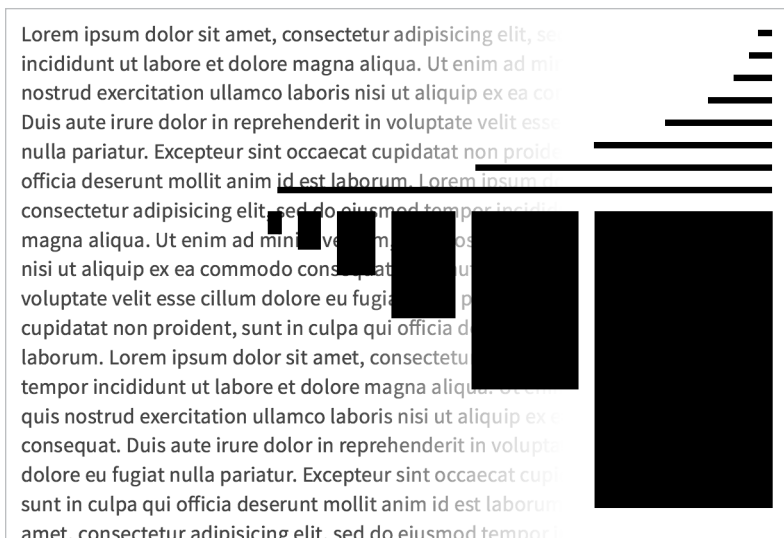


FIG 4.12: Source Sans Pro with shapes sized and proportioned in relation to our project's ratio of 3:5.

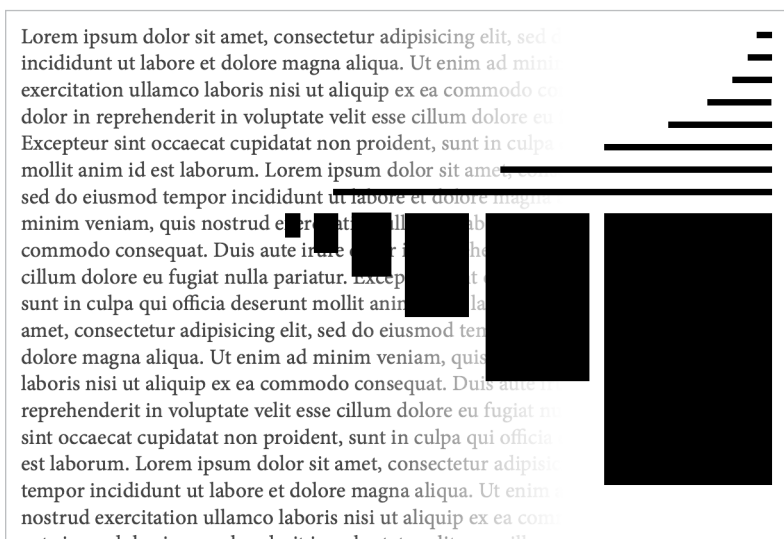


FIG 4.13: Minion Pro with shapes sized and proportioned using a ratio of 1:1.618.

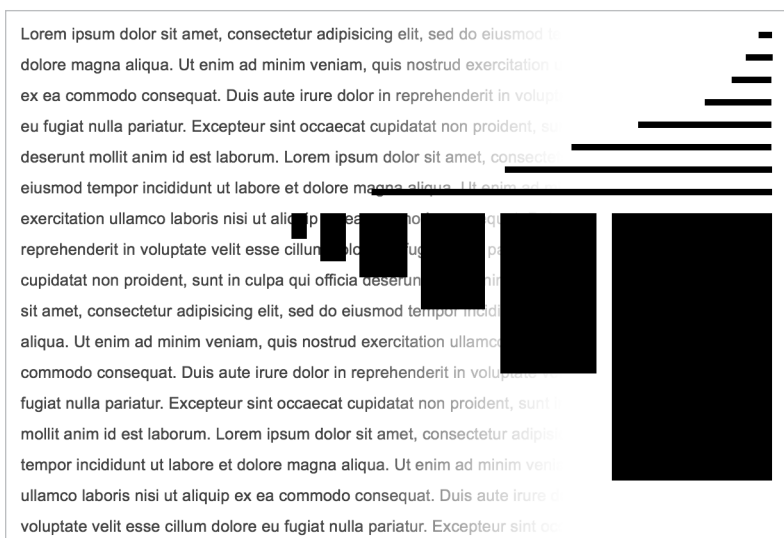


FIG 4.14: Arial with shapes sized and proportioned arbitrarily. I'm laying it on thick here with badly typeset Arial, but study these shapes and you'll see my point. They do not feel great.

Different ratios feel different, so you'll need to experiment to find the right one for your project. Consider ratios that make sense for the content you're working with, or for the typeface you're using. Bringhurst's chapter on harmony and counterpoint in *The Elements of Typographic Style* explores the historical, geometric, and musical bases of many ratios. It's a good place to start if you want to consider your choice of ratio more deeply.

I chose 3:5 (or 1:1.667) as the ratio for our example project because it's a stout, standard proportion. The notecards in my pocket are 3" × 5", and I often take notes on them when I'm learning to fight fires and respond to emergencies. Note that these reasons wouldn't matter if the resulting numbers in my scale were impractical. (I don't think I've ever used the "15:16—minor second" ratio.)

I'm sure you've seen that golden-ratio spiral on the internet, ironically (or not) overlaid on a design to demonstrate why it looks so perfect. Or perhaps you've seen the book-canon dia-

grams for geometrically laying out printed pages (<http://bkaprt.com/ft/04-08/>). These things don't account for the flexibility of the web, but our modular scale does, because it's based on a body-text font size rooted in each reader's default font size.

Remember, just as when using a ruler, you can be very exact or you can eyeball measurements. There's nothing wrong with trying something to see how it looks, and as you gain experience you won't need scales as much because you'll just make decisions that feel good to you—that you're confident about—without any help. But, especially when you're starting out, and in general as a useful tool to have around, scales can be incredibly helpful.

Now we have our typeface and font size nailed down. These factors should not dramatically change as body text flexes on the web. The next thing to decide on is the text block's line length, or measure.

DECIDE ON LINE-LENGTH LIMITS

It's possible that you rarely think about line length, because most layout templates take care of it for you. You pick a typeface and a font size, and your text takes up however much horizontal room the layout has provided. Think about this from the perspective of the text block, though: what the layout container is really doing is constraining the text at some arbitrary width.

Or maybe you have some experience with typography, and you already carefully decide on the measure of a text block. You compose your layouts around your text blocks, making sure the type looks great and the composition looks great *around it*.

What we want to do with our example project—which still uses a simple, linear layout—is determine not only the measure of text blocks, but the measure *limits*. Our text block will flex on the web, so we need to determine how much it should flex. How narrow can the text block be and still look good? How wide can it be before it becomes difficult to read? Finding these limits is very important. We'll change the `width` property we started with into `min-width` and `max-width` properties.

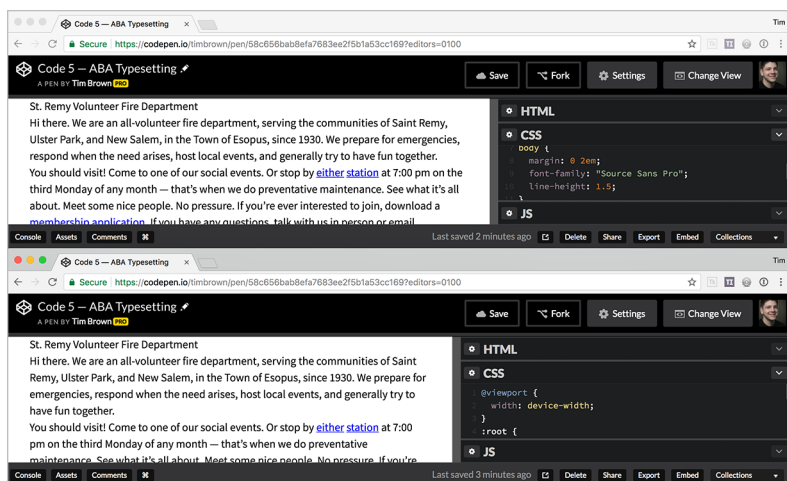


FIG 4.15: Determine line length limits with some trial and error.

But what's the best way to decide on the actual line length values? What if you have no idea what looks good? Some people will tell you to count characters and shoot for a particular minimum and maximum number of characters per line. That's a good way to narrow things down a bit, but taking this advice prescriptively produces layouts that feel stiff—text can often flex more and look fine. For similar reasons, ratio-based measurements and grid systems are tempting to rely on, but don't choose these numbers without studying the results visually. Measure is a visual decision.

"We read with our eyes, not with rulers," writes Tobias Frere-Jones, "so the eye should win every time" (<http://bkaprt.com/ft/04-09/>). Your own intuition, an instinct about what feels comfortable to read, is just as valid an approach as a mathematical one. Hone your instincts.

Let's choose measure limits for our example project. Follow along with me.

Code example: <http://bkaprt.com/ft/04-10/>

- First, delete the `width: 40em;` declaration so the text will behave in a completely fluid way, with its line length stretching as wide as the CodePen preview pane.
- Next, grab CodePen’s vertical divider and resize the preview area’s width to find a good maximum measure—the widest you would want this text to be.
- Then, open a second window with the same example, and do that again.
- After that, with the two windows side by side, look at them for a while and try to choose your favorite (FIG 4.15).
- If you can’t see them well enough because the text is jammed up against the sides of each window, temporarily add horizontal margin space to the body element—put some on each side, like this: `margin: 0 2em;`
- Then, pick your favorite of the two and resize the width of the other one again. Compare it to your original favorite and see which one you like better.
- Do this a few more times.
- Then do the same thing for determining a good minimum measure.
- When you’re done, use a little trial and error to figure out the exact maximum and minimum line lengths you liked. If you added two ems of horizontal margin on the body element, remove them now. Then, put the `width: 40em;` declaration back in your CSS and start changing its value until the paragraph looks the way it looked at your favorite minimum and maximum widths. Note those measurements. Just for kicks, compare these measurements to numbers in our modular scale. If you’re feeling really ambitious, try numbers from the modular scale as minimum and maximum measures and see how they look.
- Remove the `width` declaration when you’re done.

The measurements I liked were `21em` and `35em`. Let’s put those in our code as a comment about ideal `min-width` and an

actual `max-width` value. We don't want to set a minimum width value, because that would prevent the text from flexing to fit very narrow viewports. Also, we're using ems here, not rems, because—as a best practice—we want these measurements to relate to the text block's font size, not the root font size.

```
:root {  
  font-size: 1.15rem;  
}  
body {  
  font-family: "Source Sans Pro";  
  line-height: 1.5;  
}  
main {  
  /* Ideal min-width: 21em */  
  max-width: 35em;  
}
```

Adjusting measure is fun. When I'm messing around, resizing my text narrow and wide, figuring out the line-length limits, I'm channeling my inner Bob Ross. Join me. We're looking for happy little paragraphs. You can get the same happy little paragraphs whether you're resizing a browser window by hand or testing out measurements from a scale or grid to see how they look.

More often than not, what you're typesetting—the length of paragraphs, the kinds of words, the frequency of glyphs, and the message's tone—will determine how good the line length feels. Immense paragraphs can withstand wide measures, whereas brief bits of news, or a series of blurbs, look much better with a narrower measure.

Try this technique from designer, teacher, and typographer Juliette Cezzar: read out loud, breathing only at the end of each line of text. If you're gasping for air, the measure is too long; if you're hyperventilating, the measure is too short.

ADJUST LINE SPACING ACCORDINGLY

Turning our attention to the final essential property of our text block, let's set line spacing in a way that works for the typeface we're using, at the size we're using it, and within the line-length limits we've determined are appropriate. We'll go about this in the same kind of loose, experimental way we did when we were finding measure limits.

Resize the CodePen preview pane, move the text block to an extreme width—this time start with the narrow measure limit—and then try out some different `line-height` values (FIG 4.16).

As you adjust line spacing, pay attention to the size of spaces between words. You want the white space between lines to be somewhat taller than word spaces are wide, because this helps people read comfortably.

Tight text is distracting because of what Gestalt theory calls the *law of proximity*. If two objects are closer to one another than they are to other objects, the two will seem related. Similarly, if line spacing is too tight, words will seem closer to the text on lines above and below than to text on the left and right. When this happens, reading becomes uncomfortable (FIG 4.17). Let this concept sink in—it will come up repeatedly as we continue working out spatial relationships.

My favorite strategy for arriving at a good minimum line-height is to make the value so tight that I know it's wrong. Then I loosen it little by little, until it feels okay again.

You'll want to do the same kind of exercise for your text block at its widest (FIG 4.18). Go ahead and try some different line-spacing values. Notice that looser line heights feel better when the text is wide. Loosen line spacing up so much that it feels wrong—much too loose—and then tighten it little by little until it feels okay again. Happy little paragraphs.

I ended up setting the `line-height` at 1.5 for our example project. Because the text block will flex to be both narrow and wide, it's better to set a loose line height by default. Text becomes very difficult to read if line spacing is too tight when the measure is wide, so we want to avoid that. This is a shame,

<p>If you're thinking of volunteering your time and energy, then first of all: Thank you considering it, even if you don't.</p> <p>Volunteering at the fire department auxiliary is a commitment, but it's not a job. We are all volunteers. There are a handful of small requirements for maintaining membership, but beyond that we each participate however we can.</p> <p>Some of us don airpacs and enter burning buildings. Some of us direct traffic. Some of us keep our equipment organized and in good condition. Some of us run meetings. Some of us plan events. There are a million ways to get involved. No experience is necessary, and we are always looking for new members.</p> <p>Our members are a diverse group of people.</p>	<p>If you're thinking of volunteering your time and energy, then first of all: Thank you! We appreciate you considering it, even if you decide it's not for you.</p> <p>Volunteering at the fire department auxiliary is a commitment, but it's not a job. We are all volunteers. There are a handful of small requirements for maintaining membership, but beyond that we each participate however we can.</p> <p>Some of us don airpacs and enter burning buildings. Some of us direct traffic. Some of us keep our equipment organized and in good condition. Some of us run meetings. Some of us plan events. There are a million ways to get involved. No experience is necessary, and we are always looking for new members.</p> <p>Our members are a diverse group of people.</p>	<p>If you're thinking of volunteering your time and energy, then first of all: Thank you! We appreciate you considering it, even if you decide it's not for you.</p> <p>Volunteering at the fire department auxiliary is a commitment, but it's not a job. We are all volunteers. There are a handful of small requirements for maintaining membership, but beyond that we each participate however we can.</p> <p>Some of us don airpacs and enter burning buildings. Some of us direct traffic. Some of us keep our equipment organized and in good condition. Some of us run meetings. Some of us plan events. There are a million ways to get involved. No experience is necessary, and we are always looking for new members.</p> <p>Our members are a diverse group of people.</p>
---	--	--

FIG 4.16: Scrunch up the line height so that it's too tight, and then loosen it little by little until it looks right to you.

Some of us don airpacs and enter burning buildings. Some of us direct traffic. Some of us keep our equipment organized and in good condition. Some of us run meetings. Some of us plan events. There are a million ways to get involved. No experience is necessary, and we are always looking for new members. Our members are a diverse group of people.

FIG 4.17: Here, line spacing is similar in volume to the spaces at the ends of sentences. This line spacing is too tight.

though, because line spacing that feels too loose when the measure is narrow is not great. Ideally, I would set a **line-height** of **1.3** when the text block is at its narrowest. If only there were some way to use dynamic line spacing!

We appreciate you considering Volunteering at the fire depart job. We are all volunteers. The membership, but beyond that Some of us don airparks and Some of us keep our equipme meetings. Some of us plan eve experience is necessary, and v Our members are a diverse gr on gender, age, race, disability to have help. However you participate, volu community, helping other pec From our members "I joined the Fire Department ago. It's a great group of peop their neighbors in the event o	If you're thinking of volunteering your time and energy, then first of all: Thank you! We appreciate you considering it, even if you decide it's not for you. Volunteering at the fire department and/or auxiliary is a commitment, but it's not a job. We are all volunteers. There are a handful of small requirements for maintaining membership, but beyond that we each participate however we can. Some of us don airparks and enter burning buildings. Some of us direct traffic. Some of us keep our equipment organized and in good condition. Some of us run meetings. Some of us plan events. There are a million ways to get involved. No experience is necessary, and we are always looking for new members. Our members are a diverse group of people. We don't tolerate discrimination based
---	--

FIG 4.18: Loosen the line height so that it's exaggeratedly generous, and then tighten it up little by little.

USE DYNAMIC LINE SPACING

Ask any experienced typographer or type designer, and they'll tell you that the right line-spacing value depends on the font in use, the font size, and—especially—the measure. Given a specific font at a specific size, if the measure is narrow, the line spacing should be on the tight side; if the measure is wide, the line spacing should be on the loose side. We saw this when we went through the previous section about adjusting line spacing.

We also know that because web text can be both narrow and wide, no single line-spacing value will work perfectly. So we need to set our CSS `line-height` as a range. That way, it remains appropriate for our flexible text at all times—tight when the measure is narrow, loose when the measure is wide.

What we can do is use a CSS “lock” to establish a relationship between our line-length limits and the corresponding line-spacing values.

In canal and river navigation, a *lock* is a device used for raising and lowering vessels between stretches of water that are at different levels (**FIG 4.19**). That's exactly what we want to do here with line spacing (**FIG 4.20**).

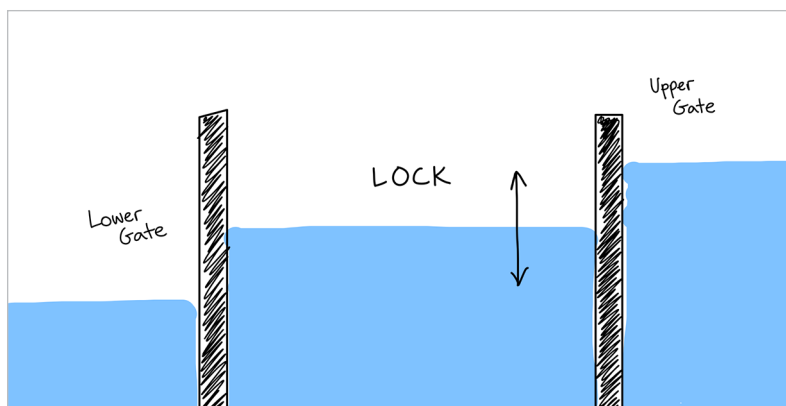


FIG 4.19: Waterway locks allow water levels to rise and fall between watertight doors called gates.

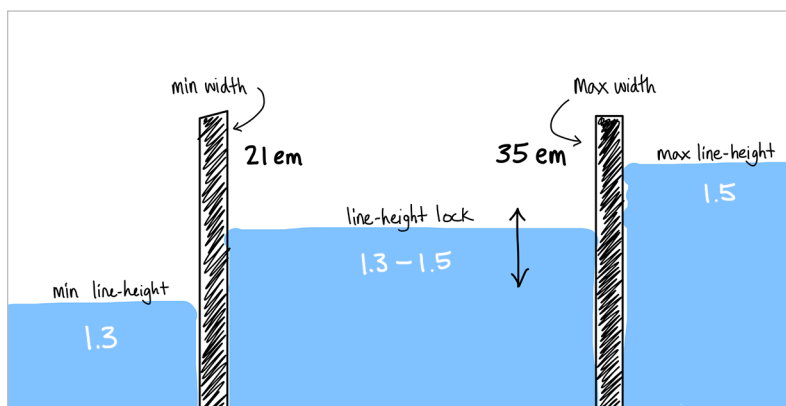


FIG 4.20: CSS locks allow the value of a property like `line-height` to flex within a specific range, relative to a different set of values—like the minimum and maximum widths of a paragraph.

Below the lock (with the measure at its narrowest), we want our `line-height` to be `1.3` (the value I liked for our text block’s narrowest measure). Above the lock, we want our `line-height` to be `1.5`. The “gates” are our paragraph’s minimum and maxi-

mum widths, and the water level is our line height. The magic happens between the gates—as our paragraph’s width flexes, our line height flexes dynamically to maintain an appropriate value. Note that line-height values are purposely unitless to avoid inheritance issues (<http://bkaprt.com/ft/04-11/>).

If you’re interested in this method, I explain the formula on the Typekit blog (<http://bkaprt.com/ft/04-12/>). This approach to line spacing feels natural for the web, although it is complicated to implement right now—especially in multicolumn layouts. Ideally, we would use container queries (rather than media queries) to establish a lock’s gates, and we would set this range as a native part of the CSS `line-height` syntax. For our St. Remy project example, we’ll stick with a static `line-height` of 1.5.

Did you know that line spacing is just one kind of white space in our text blocks?

CONSIDER NEARBY WHITE SPACE PART OF THE TEXT BLOCK

White space in typography means anything that is not glyph shapes. As far as our text block is concerned, there are many kinds of white space. Counter spaces, letter spaces, and word spaces are all white space. Line spacing is white space. And the margins that surround our text block are white space too (**FIG 4.21**).

All of this white space is related. The margins that surround our text blocks are *part of our text blocks*. So let’s make decisions about margins in the same way we’ve made all of our decisions about shaping the text block. By doing so, we can learn more about the amount of room our text needs before it becomes pressured.

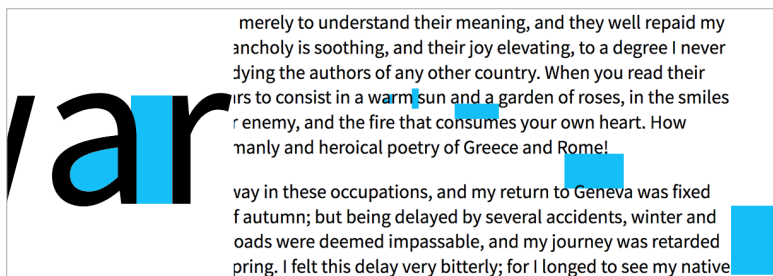


FIG 4.21: From left to right: counter space (enlarged), letter space (enlarged), word space, line space, vertical margin, and horizontal margin.

Vertical margins

The vertical margin sizes of paragraphs depend on which elements come before and after the paragraphs. If you're using a margin *between* paragraphs, try a **margin-bottom** equivalent to half of the line height, or try a small number from your modular scale. You can eyeball it too. Just like choosing a font size, line height, and measure, see what feels good to you.

Whichever value you choose, use **em** units so that vertical margins relate to the body-text font size. And set vertical margins in a single direction on every element. Harry Roberts has explained the benefits of single-direction margins (<http://bkaprt.com/ft/04-13/>); I've found this strategy to be very effective.

Indents

If you're using indents instead of vertical space to separate paragraphs (uncommon on the web, but not a bad practice), consider the size of the indent like you would any decision in the text block. The indent is a kind of white space. Try using the line-height value, or the font-size value. Try half of the line height. Try a small number from your modular scale. Eyeball it.

Only indent paragraphs that follow other paragraphs, like so:


```
p+p {  
  text-indent: 0.75em;  
}
```

Avoid using both indents *and* inter-paragraph margins: the combination is unconventional, which means it will attract unwanted attention in a reading experience. It also introduces an amount and shape of white space that generally feels uncomfortable.

Horizontal margins

Horizontal margin sizes can fluctuate a lot, from very tight to very loose, and still look okay. But make deliberate decisions about these, too. At the body text's narrowest, or when a small screen pressures the text to be *even narrower*, what amount of marginal, horizontal space should surround the text? Get a feel for this by trying the break-it-and-fix-it method I described in the sections on line-length limits and adjusting line spacing: take it to an extreme, and then ease back until it feels right. You can also try using numbers from your modular scale for horizontal margin spaces. Or try making the horizontal margins the same as the vertical margins, or bigger, or smaller, and see what those combinations are like.

Then, when the viewport is wide, decide how spacious the horizontal margins should be before the text looks abandoned—floating in the middle of nowhere. At their most generous, horizontal margins should still feel like part of the composition.

A big part of sizing horizontal margins is deciding how much breathing room it feels like the text should have. Tighter margins feel busier, and newsier, while looser margins feel quieter and more bookish. What kind of margins feel right for your project's content, and with the typeface you're using? Spend time deciding this by studying different options.

Sizing horizontal margins also depends on what's happening around and adjacent to the text block in more complex layouts. (We'll talk more about that in Chapter 5.) The point of making these decisions in isolation—before we get to more complex layouts—is to take the perspective of the text, and to consider

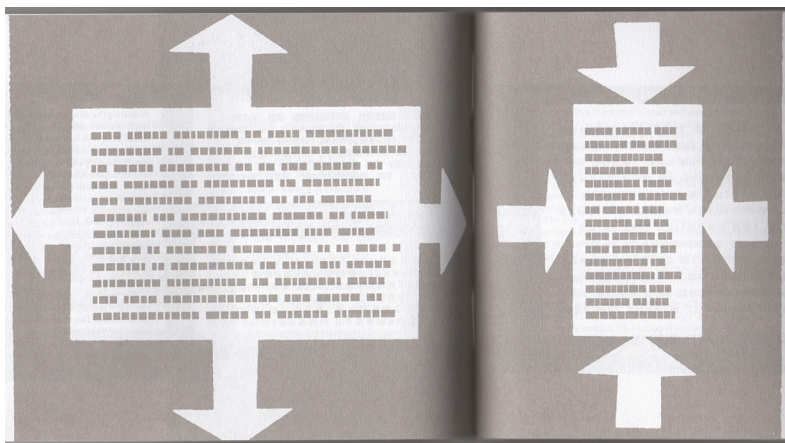


FIG 4.22: Cyrus Highsmith’s illustration of text block tempo, from my copy of *Inside Paragraphs*.

how its natural boundaries and limitations are pressured in more complex situations.

Tempo

But how does white space relate to flexibility? If dynamic line spacing is what our text needs, and our margins are related to our line spacing and the other white space in our text block, then shouldn’t our margins, word spaces, and glyph spaces also be dynamic as the text block flexes?

Yes. Cyrus Highsmith calls this simultaneous white-space adjustment a matter of *tempo* (FIG 4.22). As a text block becomes narrower, all of its white space should condense—the text block should have a faster tempo. But as it widens, all of the white space should loosen. In a wide text block, the tempo should be slower.

We can accomplish the kinds of adjustments Highsmith discusses with CSS—by using properties like [letter-spacing](#), [word-spacing](#), and [margin](#). And variable fonts add another layer of potential here, enabling us to act on the counter spaces

within letters themselves. However, none of this is automatic or intelligent yet. In CSS today, there exists no holistic way of defining relationships among these properties. Fortunately, though, one of the great powers of the web is that we can brainstorm solutions and invent the things we need. Together, I'm sure that we can make dynamic white space a reality—let's experiment with it and advocate for it!

For now, here's how we'll set the margins on our example project's text blocks:

```
:root {
  font-size: 1.15rem;
}
body {
  font-family: "Source Sans Pro";
  line-height: 1.5;
}
main {
  /* Ideal min-width: 21em */
  max-width: 35em;
  padding: 0 1em;
  margin: 0 auto;
}
p {
  margin: 0 0 .75em;
}

@media screen and (min-width: 52em) {
  /* At this point, horizontal margins should stop
  growing. */
}
```

Code example: <http://bkaprt.com/ft/o4-14/>

As you can see, there's a lot happening here. Believe it or not, it's all margin-related—in the sense of white space around the text block. (Some of it is padding.) Let's go over it line by line. First of all, the `main` element now has a left and right `padding`

of `1em`. This is the minimum width I felt was acceptable for the horizontal “margins” that surround our text block.

Next, the `main` element has a left and right `margin` of `auto`. This will center it horizontally; as the viewport widens, the margins on each side of the text block will grow at the same rate. And we’ll allow them to grow—until we want them to stop, at which point we’ll invoke a media query. We still need to figure out what happens once the media query gets involved, and we’ll do that in Chapter 5, where we’ll also talk more about the media query’s em-based `min-width` value.

Finally, we have a `.75em` bottom margin on paragraphs to manage vertical white space within the text block.

CONSIDER FALLBACK FONTS

So far, we’ve styled our St. Remy example project with the body-text typeface we selected in Chapter 3; then we meticulously sized that type while considering its x-height (a metric that varies from font to font). We carefully set line-length limits. We also determined line spacing and other white-space values with sensitivity and a Gestalt sense of balance.

But everything we’ve done in this chapter assumes we know exactly which typeface the reader is seeing. Of course, as we discussed in Chapter 1, there’s a good chance our readers will not see the typeface we intend; instead, they’ll see a fallback font. In fact, *many* readers will see a fallback font while our webfont loads. Loading time will vary; sometimes it may be imperceptible, but other times it will be extremely obvious. At those times, our typography risks looking pretty bad, because the typesetting decisions we’ve made depend on our ideal font’s presence.

Given this reality—that many readers will see a fallback font *and then* the webfont we intend—it’s very important that the transition between those two things is not jarring. It has to be as smooth as possible so that people aren’t interrupted if the transition occurs while they’re browsing or reading (FIG 4.23).

To avoid any jarring shifts during this transition, we need to do a few things.

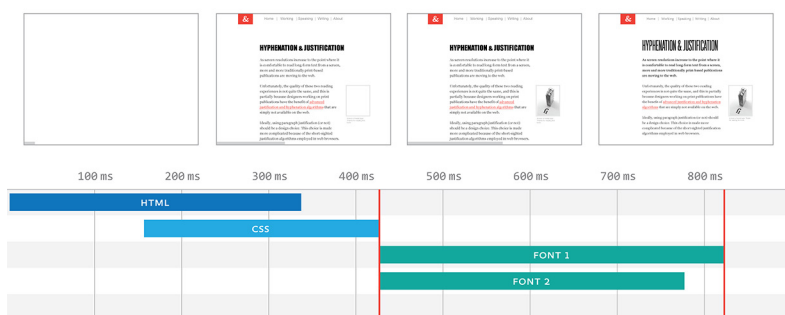


FIG 4.23: In his *Webfont Handbook*, Bram Stein details font-loading behavior and explains how in the future we'll be able to control it with CSS.

First, we need to identify fallback fonts that are a good fit for our ideal typeface. While we're at it, we should fill out our font stack by choosing additional fallbacks that are common and generic, to ensure that—at the very least—most people will see a typographic style that is roughly appropriate.

Next, we'll need to scope the styles we've already written so that they only apply when our intended font is present (which makes sense, because we chose them for this typeface specifically). Our default styles, in contrast, should assume that the reader is seeing a fallback font.

Finally, we'll take some measurements. We'll gather the font metrics of our intended font and our most likely fallback. We can match them up in a few different ways using a little proportional math, to minimize the visible effects of switching from one to the other.

It's tempting to avoid all of this work by hiding text until webfonts load. Resist the temptation! Fortify yourself by rereading Chapter 1.

Identify fallback fonts

The first thing to do is add a fallback font stack, which is a prioritized list of font-family names. Browsers will look at the names we list and style our text with the first available family.

Let's put this font stack into our example project's `font-family` declaration:

```
body {  
  font-family:  
    "Source Sans Pro", /* Ideal */  
    "Lucida Sans", "Lucida Grande", "Lucida Sans  
Unicode", /* Fit */  
    "Open Sans", "Droid Sans", /* Common */  
    sans-serif; /* Generic */  
  line-height: 1.5;  
}
```

What we've essentially done here is choose what I like to think of as *understudies* to our ideal webfont. Just like theatrical understudies, these fonts need to perform the same lines and choreography as the main typeface. Good fallback fonts can do a very capable job. I like Nathan Ford's strategy for building font stacks (<http://bkaprt.com/ft/04-15/>). It goes like this: after your ideal typeface, choose one that fits well, then something common, and finally list a generic style.

We decided on our ideal choice, Source Sans Pro, back in Chapter 3. For the others, let's work backward—starting from generic.

The generic style at the end of your font stack should be one of the five CSS-defined generic font-family values (<http://bkaprt.com/ft/04-16/>). Browsers will look at this value and choose an available font that matches the style. For our case-study site, we'll use `sans-serif` as the generic font family. To get an idea of what readers using different operating systems will see, let's plug this choice into Zach Leatherman's fontfamily.io (<http://bkaprt.com/ft/04-17/>).

What a dog's breakfast (**FIG 4.24**). But remember that this value is our font stack's last resort, so very few readers will see these typefaces. Most likely, they'll see the next font up: the "common" choice.

Think of a common fallback as a more predictable generic. Rather than leave the browser to decide on a sans serif, *you* can suggest Verdana, Helvetica, or another common sans serif. Put more than one in your stack, if you care. To learn which







OS		✓ Supported	✓ Aliased	✗ OS Default
 Snow Leopard 10.6.8	✓ Helvetica	ALIASED AS SANS-SERIF		
 Lion 10.7.5	✓ Helvetica	ALIASED AS SANS-SERIF		
 Mountain Lion 10.8.5	✓ Helvetica	ALIASED AS SANS-SERIF		
 Mavericks 10.9.5	✓ Helvetica	ALIASED AS SANS-SERIF		
 XP (NT 5.1 SP3)	✓ Arial	ALIASED AS SANS-SERIF		
 XP (NT 5.2)	✓ Arial	ALIASED AS SANS-SERIF		
 7 (NT 6.1)	✓ Arial	ALIASED AS SANS-SERIF		
 8 (NT 6.2)	✓ Arial	ALIASED AS SANS-SERIF		
 8.1 (NT 6.3)	✓ Arial	ALIASED AS SANS-SERIF		
 5.1	✗ Times			
 6.0	✓ Helvetica	ALIASED AS SANS-SERIF		

FIG 4.24: When we specify a generic **font-family** value, browsers choose a font that’s available on the operating system. The results are usually not pretty.

fonts are available where, click the operating system names at fontfamily.io. Contribute to Leatherman’s GitHub project to help keep this resource up-to-date.

By the way, if you ever wondered what typography on the web was like before webfonts, now you know. It was exactly this easy to figure out which fonts were “available”—and those were the only fonts we had, unless we were going to typeset our text using Flash or render it as an image.

But why *would* we decide on a more predictable generic? Why do we care about suggesting one common fallback font instead of another, when none is ideal? One reason is aesthetic. Perhaps Verdana looks more like our webfont than Helvetica does. Another reason—an even better one—is that Verdana *fits* better.

Fit is the most important quality in a fallback font. Here, *fit* refers to how well the fallback’s metrics—like its x-height, cap height, and glyph widths—match the proportions of our ideal webfont. In most cases, people will only see a fallback font temporarily, and its main job is to shape the text in a way that closely matches the intended font—to mitigate a jarring shift when the webfont loads.

Lorem ipsum dolor sit, amet amet
consectetur adipisicing elit. lusto
eveniet voluptatem, sunt aliquid
aliquam repellendus esse nisi cum
animi fugit ea praesentium. um
voluptates at neque, accusamus et
dignissimos possimus harum.
accusamus et dignissimos
possimus harum.

Lorem ipsum dolor sit, amet met
consectetur adipisicing elit. lusto
eveniet voluptatem, sunt aliquid nt
aliquam repellendus esse nisi cum
animi fugit ea praesentium. it ea
voluptates at neque, accusamus et
dignissimos possimus harum.
dignissimos possimus harum.

FIG 4.25: Source Sans Pro in black, with fallback choices in gray. Lucida Grande, at right, fits better than Verdana.

The “fit” part of your fallback stack isn’t really a new font choice, but rather a way of ordering the common fallbacks you’ve already chosen so that the one that fits best appears highest in the stack. Because all fonts are different, however, there will be no automatic, perfect fit. For the best results, we’ll need to make specific typesetting decisions about our fallback—changing its size and other attributes to match our ideal font as closely as possible (**FIG 4.25**).

Scope custom styles

What we need to do is set fallback styles by default, and scope customized styles—the specific, careful, typographic measurements we made in this chapter—so that they apply only when we know for sure the intended webfont is present. Here’s how to do that, in a nutshell:

```
body {  
  /* Default styles that fallback fonts will use. */  
}  
  
.wf-active body {  
  /* Scoped styles that apply only when our intended  
  webfont is present. */  
}
```


Because our example project uses Typekit, font events provide this `.wf-active` class (<http://bkaprt.com/ft/04-18/>). The styles scoped by this class will only apply if the webfont is active. You can accomplish the same thing with the open-source Font Face Observer that Bram Stein maintains (<http://bkaprt.com/ft/04-19/>), and soon with the CSS font-loading API (<http://bkaprt.com/ft/04-20/>). In fact, Stein explains how to use the font-loading API for exactly this in his *Webfont Handbook* (<http://bkaprt.com/ft/04-21/>).

Now that we have a means of identifying when our webfont has loaded, we can separate default styles from customized ones. Let's reorganize some of the careful decisions we've made in this chapter:

```
:root {
  /* Fallback font-size */
}
:root.wf-active {
  font-size: 1.15rem;
}
body {
  font-family: "source-sans-pro", "Lucida Sans",
  "Lucida Grande", "Lucida Sans Unicode", "Open
  Sans", "Droid Sans", sans-serif;
  /* Fallback line-height */
}
.wf-active body {
  line-height: 1.5;
}
main {
  /* Fallback min-width */
  /* Fallback max-width */
  /* Fallback horizontal padding */
  margin: 0 auto;
}
```

```
.wf-active main {
  /* Ideal min-width: 21em */
  max-width: 35em;
  padding: 0 1em;
}
p {
  margin: 0 0 .75em;
}
```

Code example: <http://bkaprt.com/ft/04-22/>

Now that we've scoped these customized measurements, we can add some fallback styles and try to match everything up.

Take measurements

Let's do this the easy way. Open the previous code example ; then, in a new tab, open another code example in which I've disabled the webfont and added fallback styles: <http://bkaprt.com/ft/04-23/>. As you can see, I adjusted the font size, line height, and measure limits of the fallback styles and tried to match them up with the webfont—I even added a smidgeon of letter spacing.

Of course, the fallback font you see as you compare these Pens may differ from the one I see. And it's beyond unreasonable to set different fallback measurements depending on which fallback font actually ends up being active. Just do your best with the fallback you think most people will see. (This may require typesetting fallbacks with a device that is not part of your regular setup.)

What you want is for the x-heights to *sort of* match, and for the running text to occupy roughly the same amount of space. It's okay if the cap heights and extenders are a bit different. It's okay if not everything is exact (**FIG 4.26**). Maybe your customized measurements will even work for the fallbacks, but don't assume they will. Look at them and make sure.

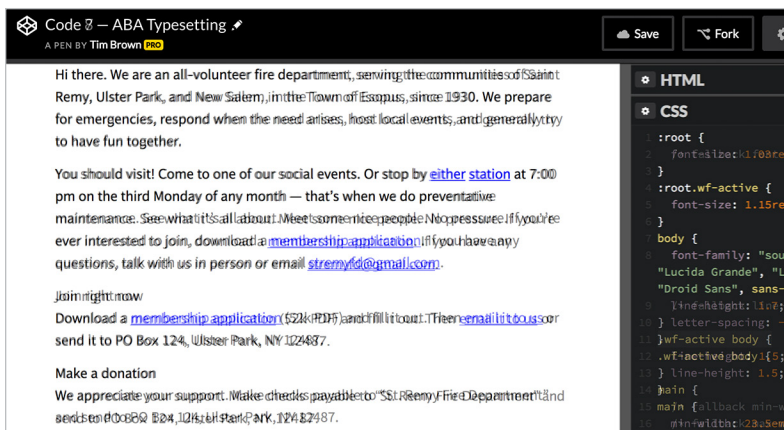


FIG 4.26: The fallback typesetting overlaid on our ideal typesetting. Not a perfect match, but decent.

When you have a decent match, congratulations! You just lessened the jarring shift that many people see when browsers transition from fallback fonts to webfonts. That's some important typesetting.

This will all become easier when CSS `font-size-adjust` is more widely supported. As of this writing, it only works in Firefox. If you don't have Firefox, go get it and open this CodePen example: <http://bkaprt.com/ft/04-24/>. Try putting this just below the root font-size:

```
:root {
  font-size: 3rem;
  font-size-adjust: .486;
}
```

Sorcery. Keen-eyed readers will observe that the `font-size-adjust` value here, `.486`, is Source Sans Pro's x-height measurement. When this property is supported by more browsers, sizing fallbacks will be that much simpler.

On the subject of fallback fonts, I'll leave you with a deeper level of nerdiness to explore if you so choose. You may have noticed that we can get *close* to a match between a fallback font and a webfont, but most of the time we can't get it exactly right. We have to compromise in one way or another.

One approach is to match up the fonts' x-heights and let copyfit differ. Whether we do this by manually adjusting `font-size` and `line-height` or use `font-size-adjust`, this can cause a jarring shift as text reflows when the webfont loads—especially if readers have scrolled while reading in a fallback font. Another way to compromise is to match up the fonts' copyfit and let x-height differ. This could place a fallback font's x-height outside of the range we established as being familiar, and messes with the balance among the four essential properties of our text block when the fallback is active. Both of these solutions are problematic—but a new day is dawning.

Imagine our fallback font is a variable font that includes axes for x-height and width. First we match up its x-height, and then we adjust its width so the copyfit also matches. Voilà! No jarring shift. And because Apple, Google, and Microsoft want variable fonts for their operating systems, the broad availability of variable fallback fonts is all but guaranteed.

Still, there is a real need for resources and tools. Choosing similar common fallback fonts, matching them up based on font metrics, and potentially fine-tuning variable fallback fonts entails work that most people don't want to spend time doing. Doing it, though, provides a tangible benefit to readers, and putting readers first is what professional typesetting demands of us.

So far, this chapter has focused on body text. Next, we'll branch out to nearby text blocks that do related jobs; after that, we'll go over a variety of typesetting details that pertain to both body text and analogous text blocks. As we move forward, continue to think about fallback fonts. For every decision you make about how text is styled, you'll need to decide whether to scope that decision to your intended font or make it a default that your fallbacks also use.

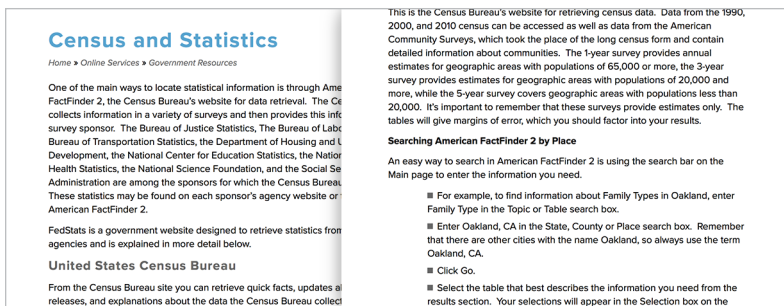


FIG 4.27: Many texts require multiple levels of subheadings, so be sure the way they're styled works for a text's entire hierarchy of information. These headings need to be stronger; we'll look at compositional contrast in Chapter 5.

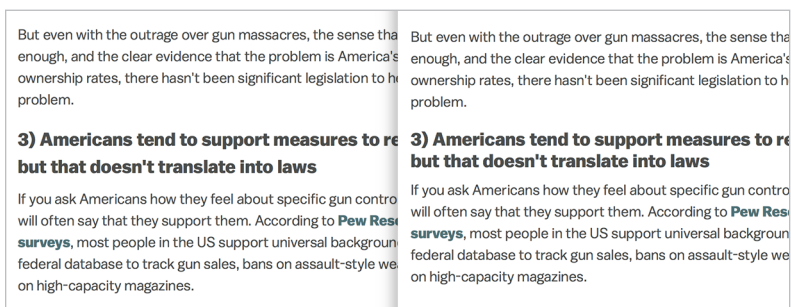


FIG 4.28: Left: subhead and paragraphs have the same line spacing. Right: subhead line spacing is tighter. Subheads often need a different line-height than paragraphs; otherwise, they look too loose.

SHAPE ANALOGOUS TEXT BLOCKS

Thinking back to our wheel of typographic jobs from Chapter 3, let's briefly review a few more text blocks, borrowing another term from the study of color: *analogous*.

Analogous colors sit side by side on the color wheel; some typographic jobs are analogous, too. Subheadings, lists, and quotations aren't exactly body text. But they often live and flow

<p>Typo Day – Mumbai, IN March 1–3, 2018</p> <p>Robothon – The Hague, NL March 8–9, 2018</p> <p>Type Drives Culture – New York City, US March 23, 2018</p> <p>Tipos Latinos April 2018</p> <p>TYPO Labs – Berlin, DE April 12–14, 2018</p> <p>Fontstand Typography Conference – Zagreb, HR April 21, 2018</p> <p>Typotage – Leipzig, DE April 27–29, 2018</p> <p>The Design Conference – Brisbane, AU May 09–11, 2018</p>	<p>I’m always quick to caution that one person’s path isn’t necessarily a pattern for someone else to follow. And on a more personal note, I’ve been independent for some time, and still don’t really know what I’m going to be when I grow up.</p> <p>But alongside those caveats, I <i>do</i> have some generic advice I give to new developers and designers. Until recently, there was only one item on the list; and now, more recently, there are two:</p> <ol style="list-style-type: none"> 1. Set up a blog somewhere, anywhere, and write as much as you can. If I’m in a position to hire you, I don’t just want to see the quality of your final mockup, your finished set of templates: I want to learn how you got there. I want to read what worked, what didn’t, and the decisions you made along the way. 2. Read Ursula Franklin’s <i>The Real World of Technology</i>, or listen to recordings of the original lectures. (Or both!) And then, once you’ve done that, consider returning to it every so often. <p>The matter is now settled. (At least, I guess, until such a time that it isn’t.)</p>
---	---

FIG 4.29: List items with small amounts of text generally look better if their line spacing is tightened up a bit. For list items with lots of text, manage white space to make sure that no two list items are conflated, and that the list is visibly separate from any surrounding paragraph text. Consider the line spacing, margins between list items, and margins that surround the whole list.

with body text—they are analogous to it—so it’s a good idea to think about these elements in a coordinated way.

Subheadings

Also known as subheads (**FIGS 4.27–28**). These attract a moderate amount of attention in order to help readers organize their thoughts or scan a text according to their interests. Subheads are a good place to expand the typographic palette, but a stronger style of the body-text typeface also works well. Subheads can be bolder, bigger, or otherwise a bit louder than paragraph text.

Lists

How you handle lists will vary with your content and taste (**FIGS 4.29–30**). Lists can be treated very much like paragraph text, or set in a different typeface from the palette, but generally the item indicators (bullets, numbers, letters, and so forth) and spacing are enough to make lists stand out.

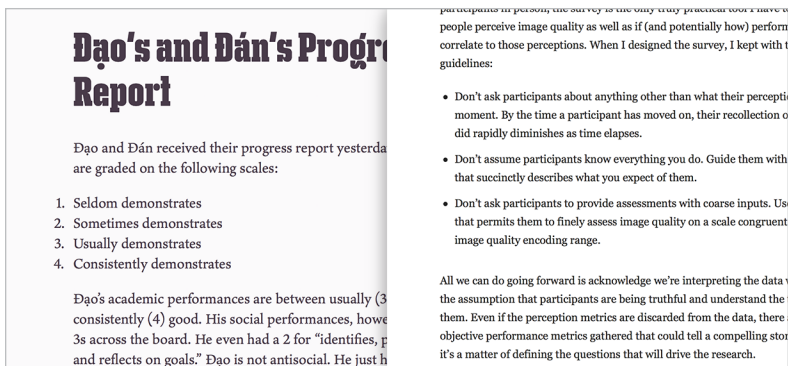


FIG 4.30: Should lists be indented, or should list-item markers “hang” in the margin? Neither style is incorrect. Shoot for consistency. For ordered lists, use **list-style-position: outside** so that list-item markers align on the decimal. Be sure that markers hung in the margins have enough room.

Quotations

From subtle indentation to splashy expressive type, a wide range of styles can be suitable for quotations. On our wheel of jobs, these sit squarely between body text and type for display. Deciding on an appropriate style for your project has a lot to do with compositional goals, which we’ll address in Chapter 5. For now, though, just make sure that block quotations are clearly differentiated from running text and that expressive pull quotes remain legible (**FIG 4.31**).

More analogous text blocks

Other kinds of text blocks can also be analogous to body text. The W3C’s HTML Element Sampler (<http://bkaprt.com/ft/04-25/>) is great for making sure you’ve covered your bases in terms of styling elements—those that are part of the project now, and those that may be in the future. (I used to maintain websites for Vassar College, and you’d be surprised at how often people add content for which the designer had never written any styles.)

In addition to this ordinary variety of HTML elements, a project may use content in specific ways, yielding special kinds

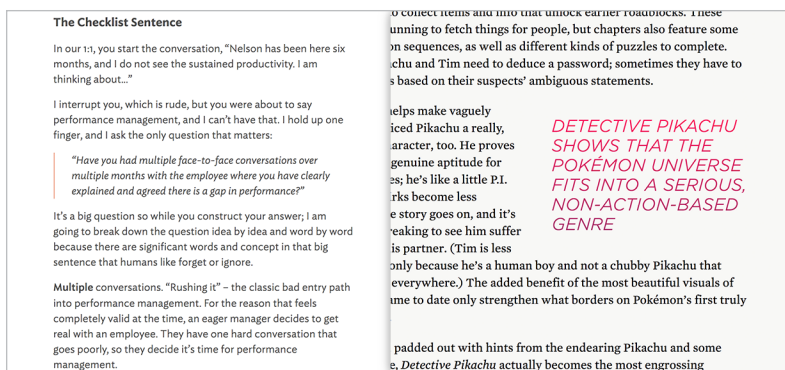


FIG 4.31: Left: this block quote is clearly differentiated from the paragraphs, but the stylistic change is also subtle—very nice. Right: with expressive pull quotes like this, pay extra attention to color contrast, line spacing, and margins.

of text blocks that need to live alongside body text but that require distinct treatment. An example of this would be running dialogue, where participants' voices are styled differently (as in interviews).

As you style text blocks analogous to body text, remember to pay careful attention to their spacing and how much or how little they differ from body text paragraphs. Also, just as we did with body text, we'll need to craft fallback measurements for each analogous text block.

Now, as the final step in shaping all of our text blocks, let's fine-tune them a bit.

TEND TO THE DETAILS

Our body text looks pretty good! We've handled those four essential properties—typeface, font size, measure, and line spacing—and our text block feels balanced. We're satisfied that we have a fallback plan in place, and analogous text blocks are quietly doing their own jobs.

Now we'll turn our attention to a number of important details that, though subtle, add up to the difference between

an amateurish experience and a professional one. We'll go over a variety of things, and then at the end of this section I'll give you an exercise: tending to the details of our St. Remy example project.

Many books and articles address the handling of small details like the ones we're about to review—and much of the advice they offer remains relevant. I'm going to mention a few topics here, specifically calling attention to challenges related to the web, but I encourage you to check out the titles in the Resources section for further reading.

Kerning, ligatures, and other essential OpenType features

The OpenType specification comprises more than a hundred OpenType features. You don't have to worry about all of them all the time, but there are a few that should *always* be enabled—like kerning, which resolves spacing issues between pairs of glyphs; ligatures, which resolve spacing issues and avoid collisions between glyphs; and contextual ligatures and alternates, which also resolve spacing issues and improve flow (**FIG 4.32**). Cumulatively, enabling these features makes a noticeable improvement in body text.

We enable these OpenType features in CSS like so:

```
:root {  
  font-size: 1.03rem;  
  font-kerning: normal;  
  font-variant-ligatures: common-ligatures  
    contextual;  
  font-feature-settings: "kern", "liga", "clig",  
    "calt";  
}
```

We're looking at both *high-* and *low-level properties* here. The high-level properties are `font-kerning` and `font-variant` (which includes subproperties like `font-variant-ligatures`); they're nice because their property names are descriptive and use natural-language values, like “common-ligatures” and “con-



FIG 4.32: Left and center, Nina Stössinger’s FF Ernestine; right, Robert Slimbach’s Caflisch Script. In the top examples, kerning, standard ligatures, and contextual alternates have not been enabled. The bottom examples show improved text flow when kerning, ligatures, and contextual alternates have been turned on.

textual.” The W3C prefers that we use high-level properties for working with OpenType features, and browser support is getting better, but we still need low-level properties in some contexts (<http://bkaprt.com/ft/04-26/>). The problem with low-level properties—well, there are a few problems.

Low-level properties require us to use four-character OpenType feature tags, like `kern` (kerning), `liga` (standard ligatures), `clig` (contextual ligatures), and `calt` (contextual alternates). It can get tedious to have to look those up in the registry (<http://bkaprt.com/ft/04-27/>); on top of that, feature tags need to be specified, comma-delimited, in a single property—`font-feature-settings`—which can be confusing to manage (especially because the high-level property values are space-delimited rather than comma-delimited).

Another slight problem—more of an annoyance, really—is that because a low-level property is intended “for special cases where its use is the only way of accessing a particular infrequently used font feature,” it will override high-level `font-variant` declarations (<http://bkaprt.com/ft/04-28/>). Practically, this means that low-level properties nullify our high-level

declarations—but we should still write those, because it’s the right thing to do, and because they are the future of OpenType feature syntax. Enter CSS `@supports`:

```
:root {
  font-size: 1.03rem;
  font-feature-settings: "kern", "liga", "clig",
    "calt";
}
@supports (font-kerning: normal) and (font-variant
  ligatures: common-ligatures contextual) {
  :root {
    font-kerning: normal;
    font-variant-ligatures: common-ligatures
    contextual;
    font-feature-settings: normal;
  }
}
```

With `@supports`, high-level declarations are only used in browsers that support them. In those cases, we return `font-feature-settings` to its default state so that it won’t override the high-level declarations.

But there’s one more annoying problem to consider: inheritance. Sticking with our example, let’s enable small caps—but only on abbreviations. Here’s how to do it:

```
:root {
  font-size: 1.03rem;
  font-feature-settings: "kern", "liga", "clig",
    "calt";
}
@supports (font-kerning: normal) and (font-variant
  ligatures: common-ligatures contextual) {
  :root {
    font-kerning: normal;
    font-variant-ligatures: common-ligatures
    contextual;
  }
}
```

```

        font-feature-settings: normal;
    }
}
abbr {
    font-feature-settings: "c2sc", "smcp";
}
@supports (font-variant-caps: all-small-caps) {
    abbr {
        font-variant-caps: all-small-caps;
        font-feature-settings: normal;
    }
}

```

Here, the values in the `:root` selector's `font-feature-settings` declaration are overridden. The `font-kertering` and `font-variant-ligatures` properties are inherited, but we would run into the same inheritance problems that the low-level properties experience if we were to enable small caps with the `font-variant` shorthand syntax rather than the `font-variant-caps` subproperty. (The `all-small-caps` value would override the `common-ligatures` and `contextual` values.)

To resolve inheritance issues like this, we need to (ugh) respecify all of the property values that are not inherited:

```

abbr {
    font-feature-settings: "kern", "liga", "clig",
        "calt", "c2sc", "smcp";
}

```

You can find more information about OpenType features in the Adobe Typekit help documentation I wrote (<http://bkaprt.com/ft/04-28/>). OpenType features are worth these minor hassles.

Besides enabling critical features like the ones we just looked at, we can enable features that matter specifically for body text—like small caps for abbreviations and proportional oldstyle figures for numerals. Let's look at these next.

Abbreviations

Remember that we used `abbr` to mark up abbreviations in Chapter 2. In Chapter 3, we learned about the importance of small caps in a body-text typeface. And just a moment ago we brought these concepts together by using small caps to style abbreviations in body text. Small caps are designed to reduce the size of imposing capital letters in a balanced way, so that they don't disrupt running text.

All it took was a little OpenType-feature-syntax fun, and we've got small caps (**FIG 4.33**). Note that I've used the high-level value `all-small-caps` and included the low-level value `c2sc`; these enable the Small Capitals From Capitals feature, so that uppercase letters—in addition to lowercase letters—will be transformed into small caps.

Letterspace small caps vary subtly, using ems so the spacing scales with font size. A little will do, like `letter-spacing: .01em`. The extra space helps small caps breathe, and better matches the black-and-white balance of body text.

Numerals

For body text, we also want to use proportional oldstyle figures (also known as *lowercase numerals*):

Lowercase numerals act like lowercase letters, some with ascenders (6, 8), some with descenders (3, 4, 5, 7, 9), and some that reside at x-height (0, 1, 2), allowing them to blend in with text and cause less visual disruption than uppercase numerals.

—Jason Santa Maria, *On Web Typography*

Because fonts' default figure styles are unpredictable (even typefaces designed for text sometimes use lining, tabular figures by default), we need to explicitly enable proportional oldstyle figures with CSS (**FIG 4.34**).

<p>hepkov, in collabor physical Institute, p apparatus, RAPID, ca ft within 3 km of a ed their first mass RUS-1 and RUS-2 F</p>	<p>hepkov, in collabo physical Institute, p apparatus, RAPID, ca ft within 3 km of a ed their first mass RUS-1 and RUS-2]</p>
--	---

FIG 4.33: Left: abbreviations styled with normal capitals. Right: the same abbreviations set in letterspaced small caps.

<p>l, only 607 Redut st g the war. The first l entered into servic ore than 230 Gneiss end of 1944. The Fr r, featured continu</p>	<p>l, only 607 Redut st g the war. The first l entered into servic ore than 230 Gneiss end of 1944. The Fr r, featured continu</p>
---	---

FIG 4.34: Left: numbers styled with default (lining tabular) figures. Right: the same numbers set in letterspaced, proportional, oldstyle figures.

```

:root {
  font-size: 1.03rem;
  font-feature-settings: "kern", "liga", "clig",
    "calt", "onum", "pnum";
}
@supports (font-kerning: normal) and (font-variant
  ligatures: common-ligatures contextual) and (font-
  variant-numeric: oldstyle-nums proportional-nums)
{
  :root {
    font-kerning: normal;
    font-variant-ligatures: common-ligatures
    contextual;
    font-variant-numeric: oldstyle-nums
    proportional-nums;
    font-feature-settings: normal;
  }
}
.numbers {
  letter-spacing: 0.01em;
}

```

Like small caps, these are OpenType features. Here we're using the high-level property `font-variant-numeric`, as well as the low-level values `onum` and `pnum`. It often makes sense to set these features on `:root`, or on the container that will house most body text, so that paragraphs, lists, and other elements inherit this style.

Back in Chapter 2, we talked about wrapping long sequences of numbers in span tags. Now we can apply `letter-spacing` to those as we did for small caps, and for the same reason.

Emphasis

Emphasized text can help readers scan and can call attention to phrases in a variety of ways, ranging from subtle to strong (FIG 4.35).

Bold type can make text easier to scan, but it can also be distracting. Use it sparingly. Great italics blend in nicely with their

protective equipment can be categorized by the area of hazard, and by the type of garment or accessory. A *single multiple forms of protection*: a steel toe cap and safety shoes to protect from crushing or puncture injuries, impervious rubber boots to protect from chemicals, **high reflectivity and heat resistance** for protection from electrical resistivity for protection from electric shock. The selection of equipment must be compared with the hazards. **More breathable types of personal protective equipment**

FIG 4.35: Bold type stands out, while italics and obliques blend in. (What we see here is an oblique, not an italic. An oblique is a slanted version of a typeface's roman style; an italic is a separate design from the roman.)

roman counterparts, but clearly signify a change in tone. Watch out for italics that stand out either too much or not enough. Save underlining for linked text; we'll talk about that next.

You need not stick with a type family's bold weight for bold emphasis. A semibold or medium might do (if the family includes them), depending on the typeface and the weight you're using for text. For example, I've found that a typeface's semibold usually works well for emphasis if the text weight is on the lighter side (**FIG 4.36**).

If you're using a variable font, you might consider adjusting its weight axis (a variable font made for body text most likely has one) as a way to find the amount of emphasis you're after.

Speaking of variable fonts, it's harder for true italics to be included in the design space of a variable font in the same way that a weight or width axis is included, because italic styles of a typeface usually have very different shapes than the upright styles. For performance reasons, it can be tempting to jettison a typeface's italic styles if they're not part of the same variable font as, say, the regular and bold styles. But consider the subtlety and formatting conventions associated with italics before you decide to drop them. Many style guides, like *The Chicago Manual of Style*, for example, recommend setting book titles in italics.

<p>categorized by the area of the body protected, by the ment or accessory. A single item, for example boots, on: a steel toe cap and steel insoles for protection of uries, impervious rubber and lining for protection from and heat resistance for protection from radiant heat, ction from electric shock. The protective attributes of pared with the hazards expected to be found in the ersonal protective equipment may not lead to more user satisfaction.</p> <p>astes for natural science; and his literary pursuits occupied me. He came to the university with the design the oriental languages, and thus he should open a field or himself. Resolved to pursue no inglorious career, he ording scope for his spirit of enterprise. The Persian, ed his attention, and I was easily induced to enter on en irksome to me, and now that I wished to fly from s, I felt great relief in being the fellow-pupil with my</p>	<p> categorized by the area of the body protected, by the rment or accessory. A single item, for example boots, on: a steel toe cap and steel insoles for protection of uries, impervious rubber and lining for protection from y and heat resistance for protection from radiant protection from electric shock. The protective must be compared with the hazards expected to be ble types of personal protective equipment may not sult in greater user satisfaction.</p> <p>tastes for natural science; and his literary pursuits occupied me. He came to the university with the aster of the oriental languages, and thus he should marked out for himself. Resolved to pursue no oward the East, as affording scope for his spirit of anskrit languages engaged his attention, and I was tudies. Idleness had ever been irksome to me, and now d hated my former studies, I felt great relief in being</p>
--	--

FIG 4-36: Left: Acumin Light with Acumin Semibold. Right: Acumin Regular with Acumin Bold.

Links

Because linked text can be styled in so many different ways, different levels of emphasis are possible. You may want links to stand out like bold text, or you may want them to recede so that they don't interrupt reading.

Take care not to use link styles that confuse readers by being too similar to emphasized text—or too similar to normal, unlinked text. Denoting links by making them bold or italic is confusing. Denoting links simply by linking the text without changing its style is also confusing—and flagrantly inaccessible, since this makes the links undetectable.

Changing only the color of a link can be just as bad for people with impaired vision. In fact, it constitutes a *Web Content Accessibility Guidelines* (WCAG) failure (<http://bkaprt.com/ft/04-29/>). With color alone, it's extremely difficult and limiting to establish the right amount of contrast between link color and background color, as well as between link color and body text color.

Links should be underlined. It's a convention with which readers are familiar, and an accessibility boon. This is worth discussing more, as Ray Schwartz urges (<http://bkaprt.com/ft/04-30/>). For better or worse aesthetically, underlined links have been a familiar convention on the web since its inception. And while you may not like the look of standard underlines

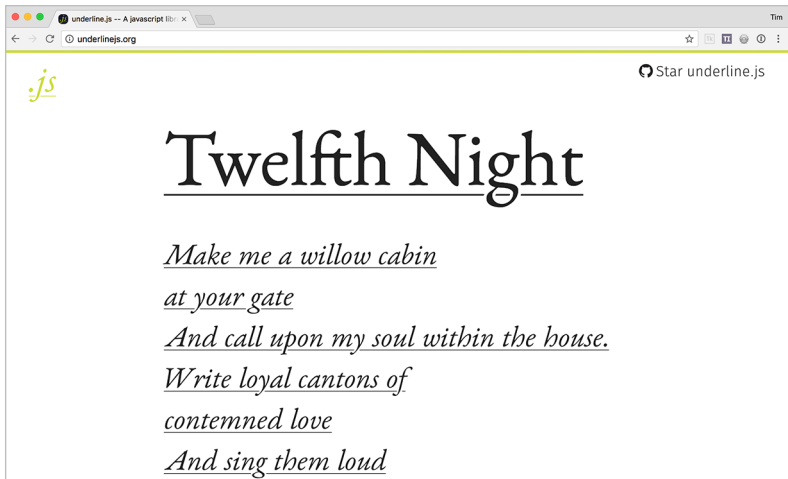


FIG 4.37: With underline.js, Wenting Zhang demonstrates better text underlining (<http://bkaprt.com/ft/04-31/>).

(I don't), it's worth taking stock of current options, as well as features that are in development.

There are a few different ways to make lines of one kind or another show up below our text's baseline, but they are not all created equal (4.37). Good old `text-decoration: underline` works, but most of the time the results are rather disappointing: the underlines are too thick and too close to the text's baseline, they intersect and obscure descenders, and they are necessarily the same color as the linked text.

Using something like `border-bottom: 1px #000 solid` allows us to control color and thickness. But borders on inline elements are hard to position and often end up too far below the baseline, depending on a typeface's vertical metrics (FIG 4.38).

The most attractive underlines are ones whose color, thickness, and position we can style as we like. Ideally, an underline will actively avoid clashing with descending letters by skipping them. Marcin Wichary explained how to do this with background-image gradients (<http://bkaprt.com/ft/04-32/>). Then Wenting Zhang made it happen with JavaScript (<http://bkaprt.com/ft/04-31/>).

The image shows two examples of the word 'all' with horizontal underlines. On the left, the word 'all' is in a standard serif font, and its underline is positioned just below the baseline. On the right, the word 'all' is in a cursive script font, and its underline is positioned significantly lower, creating a large gap between the text and the line. This illustrates the difference in vertical metrics between the two typefaces.

all *all fall*

FIG 4.38: Right: Adobe Caslon’s vertical metrics are extremely generous, so trying to use bottom borders as underlines looks ridiculous. Left: even with its tighter vertical metrics, Acumin hovers far above the line.

[com/ft/04-31/](http://bkaprt.com/ft/04-31/)), and proposed new CSS properties on her project’s GitHub page (<http://bkaprt.com/ft/04-33/>). Now, the CSS working group has added subproperties for controlling underline thickness and position to the CSS Text Decoration Module Level 4 specification (<http://bkaprt.com/ft/04-34/>). Beautiful underlines are just around the corner.

John Jameson broke down a variety of underline techniques in an article for CSS Tricks (<http://bkaprt.com/ft/04-35/>).

Hyphenation and justification

The final aspect of shaping text blocks that this book will cover is a very important one. Of the topics in this chapter, it has the most bearing on the actual shape—the outer edges—of our text blocks, but it also affects white-space relationships inside text blocks. I’m talking, of course, about *H&J*—*hyphenation* (breaking words at the ends of lines by inserting a hyphen) and *justification* (the alignment of a text block). These concepts go hand in hand.

You have seen alignment options in word processors before—*left-aligned* (also known as left-justified, flush-left, or rag-right), *right-aligned* (also known as right-justified, flush-right, or rag-left), and *centered* text. Maybe you’ve also seen

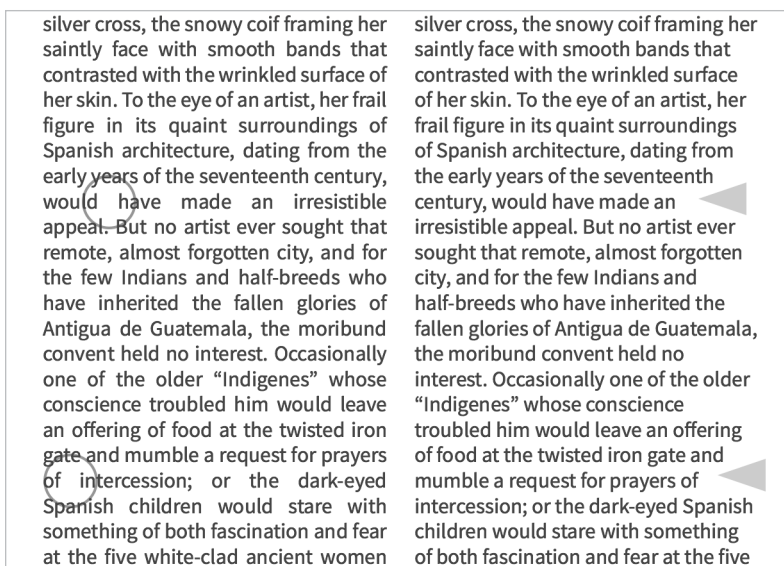


FIG 4.39: Left: full-justified text with visible gaps. Right: left-justified text with a rough rag. Neither example uses hyphenation, which is one reason why they exhibit these problems.

full-justified text as an option, where each line touches both the left and right sides of a text block's container. Without hyphenation, word spaces in full-justified text can grow to uncomfortable sizes, and flush-left text can end up with a rough *rag*—a term for a text block's uneven edge (**FIG 4.39**).

Hyphenation helps considerably in these situations. In full-justified text, it reduces the frequency of very large and very small word spaces, and in flush-left text it produces a smoother rag (**FIG 4.40**).

Even with hyphenation enabled, though, what we're seeing here is not good enough. In the flush-left example, consecutive lines that end in hyphens can be distracting. This is just one state of the flexible measure the paragraph uses; at other widths, the problem is even more prevalent.

And in the full-justified example, word spaces are often way too generous or too tight. We paid close attention to word spac-

silver cross, the snowy coif framing her saintly face with smooth bands that contrasted with the wrinkled surface of her skin. To the eye of an artist, her frail figure in its quaint surroundings of Spanish architecture, dating from the early years of the seventeenth century, would have made an irresistible appeal. But no artist ever sought that remote, almost forgotten city, and for the few Indians and half-breeds who have inherited the fallen glories of Antigua de Guatemala, the moribund convent held no interest. Occasionally one of the older “Indigenes” whose conscience troubled him would leave an offering of food at the twisted iron gate and mumble a request for prayers of intercession; or the dark-eyed Spanish children would stare with something of both fascination and fear at the five white-clad ancient women who, morn-

silver cross, the snowy coif framing her saintly face with smooth bands that contrasted with the wrinkled surface of her skin. To the eye of an artist, her frail figure in its quaint surroundings of Spanish architecture, dating from the early years of the seventeenth century, would have made an irresistible appeal. But no artist ever sought that remote, almost forgotten city, and for the few Indians and half-breeds who have inherited the fallen glories of Antigua de Guatemala, the moribund convent held no interest. Occasionally one of the older “Indigenes” whose conscience troubled him would leave an offering of food at the twisted iron gate and mumble a request for prayers of intercession; or the dark-eyed Spanish children would stare with something of both fascination and fear at the five white-clad ancient women

FIG 4.40: The same full-justified and flush-left text blocks, this time with hyphenation enabled. Better, but still not perfect.

ing when determining our line-height, remember? This throws a wrench in our plans for careful text-block balance; the text block’s tempo is very uneven.

Desktop applications like InDesign have extensive libraries and options that let you configure hyphenation thresholds to avoid undesirable results, like multiple sequential lines that end in hyphens. But on the web, we don’t have anything that approaches that flexibility yet.

—Jason Santa Maria, On Web Typography

Sophisticated H&J algorithms give InDesign users a lot of control over things like hyphenation frequency, hyphenation dictionaries (especially helpful when working with different languages), word-breaking rules, and word-space thresholds. The algorithms even subtly stretch and squeeze glyphs to better serve the overall balance of a text block. We badly need this

kind of sophistication in flexible layouts. Unfortunately, there isn't much we can do to get it until browsers incorporate better line-breaking algorithms.

Bram Stein wrote all about H&J algorithms in an article for *8 Faces* called “Justification & Hyphenation on the Web.” He concludes that piece by saying: “If we want to present longform text on the web that is comparable in quality to the text in printed books and magazines, we’ll need better hyphenation and justification algorithms in all browsers.” As a proof of concept, Stein used JavaScript to implement one of the best algorithms, the Knuth-Plass line-breaking algorithm, in his Typeset project (<http://bkaprt.com/ft/04-36/>).

One of these days, some smart engineer is going to build outstanding H&J technology into a browser that surpasses even that of InDesign, and the world will be a better place. Until then, our best bet (our only acceptable option, really) is to set text flush-left and enable basic hyphenation via the CSS `hyphens` property:

```
main {  
  hyphens: auto;  
}
```

The finer points of St. Remy

Now, take all of what we looked at in this section and apply it to our St. Remy example project. Open up this CodePen example (<http://bkaprt.com/ft/04-37/>) and enable kerning, ligatures, small caps, and oldstyle proportional figures. Add emphasis, link styles, and hyphens. After you’ve given it a shot, check your work by comparing it to another CodePen example (<http://bkaprt.com/ft/04-38/>), where I added all that stuff already.

All done? Great.

Just like that, we’ve shaped our text block (**FIG 4.41**). Take a deep breath and think about what we’ve accomplished in this chapter. It’s truly remarkable. Although there are imperfections with the web’s technology right now, the decisions we’ve made here—the limitations and relationships we’ve begun embedding into typeset text itself—are incredibly smart and thoughtful. We

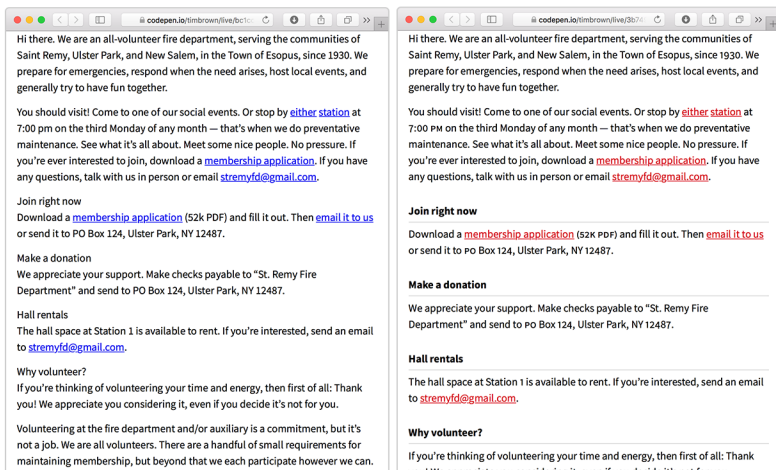


FIG 4.41: Our text block, shaped.

are preparing text to go, beautifully, where no text has gone before in the history of typography: everywhere.

Now, rather than spill these carefully shaped text blocks into containers, let's craft flexible compositions around them.

5 CRAFTING COMPOSITIONS

WE SPENT CHAPTER 4 shaping text blocks in a simple, linear layout. Now, like those cooking shows where someone suddenly takes a finished meal out of the oven, I happen to have here a slightly more complex layout of our St. Remy site. In this chapter, I'll explain how I went from the simpler layout to the more complex one (**FIG 5.1**).

Traditionally, the dimensions of a composition have been extremely important in typesetting. And they're still critical when typesetting static media, because text needs to fit into the space at hand. Quantity of text, font size, line length, spacing—all must be reconciled with the permanent outer boundaries of the work.

With dynamic media, we can't know those boundaries. But by working outward from text blocks that we've carefully shaped, instead of inward from the edges of a composition, we can manage any amount of space and maintain balance.

Let's start by roughing in layouts and hierarchy. Next, I'll walk you through my method of tailoring layouts to text. Then we'll strengthen our composition with contrast. Finally, we'll

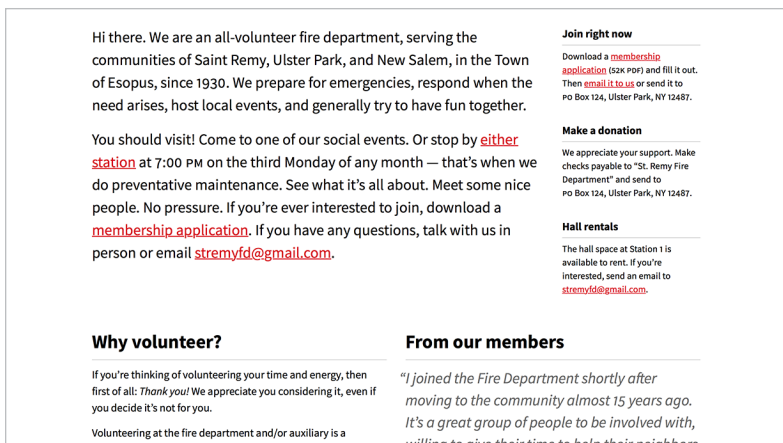


FIG 5.1: Our example project has grown from a simple, linear layout into this more complex one—but these different layouts are part of one flexible composition.

talk about codifying tailored measurements and hierarchical adaptations by baking them into our systems and grids.

The goal of this chapter is to put some decent rules in place. That doesn't mean they'll be final, or even that you won't notice problems immediately. Push through that. The *next* chapter, Chapter 6, is about polishing everything and making tweaks.

ROUGH IN LAYOUTS AND HIERARCHY

Looking back on the compositional sketches and breakpoint graphs from our text-block inventory in Chapter 3, we want to roughly match these with CSS layout. There is *so much interesting stuff* happening these days in CSS layout. With tools like CSS Grid, flexbox, and multicolumn layout, we're able to go much further than we could with floats—and in ways that feel in tune with the web.

For our purposes in this book, it doesn't matter which kind of CSS layout model you're using. What matters is the measurements you use, and the pressures those measurements put

<p>Hi there. We are an all-volunteer fire department, serving the communities of Saint Remy, Ulster Park, and New Salem, in the Town of Esopus, since 1930. We prepare for emergencies, respond when the need arises, host local events, and generally try to have fun together.</p> <p>You should visit! Come to one of our social events. Or stop by either station at 7:00 PM on the third Monday of any month — that's when we do preventative maintenance. See what it's all about. Meet some nice people. No pressure. If you're ever interested to join, download a membership application. If you have any questions, talk with us in person or email stremfyd@gmail.com.</p>	<p>Join right now</p> <p>Download a membership application (52K PDF) and fill it out. Then email it to us or send it to PO Box 124, Ulster Park, NY 12487.</p> <p>Make a donation</p> <p>We appreciate your support. Make checks payable to "St. Remy Fire Department" and send to PO Box 124, Ulster Park, NY 12487.</p> <p>Hall rentals</p> <p>The hall space at Station 1 is available to rent. If you're interested, send an email to stremfyd@gmail.com.</p>
<p>Why volunteer?</p> <p>If you're thinking of volunteering your time and energy, then first of all: Thank you! We appreciate you considering it, even if you decide it's not for you.</p> <p>Volunteering at the fire department and/or auxiliary is a commitment, but it's not a job. We are all volunteers. There</p>	<p>From our members</p> <p>"I joined the Fire Department shortly after moving to the community almost 15 years ago. It's a great group of people to be involved with, willing to give their time to help their neighbors in the event of emergencies. We have members that give 2 hours a week and some that give 20, old and young... from various backgrounds and walks of life — all</p>

FIG 5.2: This is what we want. All of the text chunks are roughly where they belong.

on text blocks. I'll kick things off for us. Using Grid for layout, I quickly chose measurements and put in some breakpoints. After a little trial and error, the layouts started looking roughly like my sketches (**FIG 5.2**).

Code example: <http://bkaprt.com/ft/05-01/>

A WORD ABOUT CHANGING FONT SIZE

Way back in Chapter 2, we talked about setting a base font size—a fundamental step in typesetting flexible compositions. Then, in Chapter 4, we carefully chose a font size for body text, finding a size that looked great for our anchor typeface and declaring that size *relative to the base font size*. We spent a lot of time nailing that down, making sure it felt familiar and comfortable to readers.

Looking at the composition more broadly now, at these more spacious roughed-in layouts, we can see there is room for modifications—maybe different font sizes, different widths, or different spacing decisions. We'll get to that stuff in a minute.

But first, I want to take a moment to single out one kind of alteration—changing font size—that has major implications for readers and for the structure of our composition. Approach this kind of alteration with care.

Changing the body-text font size as the composition widens or narrows can sacrifice reader comfort and familiarity if it moves outside of the range we reviewed in Chapter 4, and can change the basic structural unit we use for measurement in confusing ways, especially if calculations with viewport units are involved.

Type that grows larger or smaller may help fill compositional space, but it moves readers away from their personalized, comfortable, familiar font size for reading. If type grows larger or smaller to normalize its size for reading distance, but this growth is based on the dimensions of the viewport, that’s an assumption—the actual reading distance is unknown. Don’t try to “fix” things like this. Let readers be.

If type grows larger or smaller, then media queries may need to be recalculated, and so may any calculations bounded by those media-query values, because media queries (even em-based media queries) don’t pay attention to font-size changes. I get it—sometimes you want to beef up the font size (or reduce it). Just be sure you know what’s happening to readers and measurements when you do. As we’ll discuss, there are many different alterations to consider—in addition to font-size changes—that affect body text. And there are font-size changes that can be made for good compositional reasons. Onward!

TAILOR LAYOUTS TO THE TEXT

When we create CSS layouts, we draw boxes. That’s how CSS works. Everything is a box. If you dig into the mechanics of CSS, you’ll even discover something called *the box model*. No matter what kind of CSS layout you’re doing, it boils down to boxes nested within other boxes (FIG 5.3).

CSS layout boxes are fine, as long as they respect the balance in our carefully shaped text blocks. We can make sure this balance is respected by deciding on meaningful measurements for our containers, and by being aware of the moments when

Hi there. We are an all-volunteer fire department, serving the communities of Saint Remy, Ulster Park, and New Salem, in the Town of Esopus, since 1930. We prepare for emergencies, respond when the need arises, host local events, and generally try to have fun together.	Join right now
You should visit! Come to one of our social events. Or stop by either station at 7:00 PM on the third Monday of any month — that's when we do preventative maintenance. See what it's all about. Meet some nice people. No pressure. If you're ever interested to join, download a membership application . If you have any questions, talk with us in person or email stremyfd@gmail.com .	Download a membership application (52K PDF) and fill it out. Then email it to us or send it to PO Box 124, Ulster Park, NY 12487.
	Make a donation
	We appreciate your support. Make checks payable to "St. Remy Fire Department" and send to PO Box 124, Ulster Park, NY 12487.
	Hall rentals
	The hall space at Station 1 is available to rent. If you're interested, send an email to stremyfd@gmail.com .
Why volunteer?	From our members
If you're thinking of volunteering your time and energy, then first of all: Thank you! We appreciate you considering it, even if you decide it's not for you.	"I joined the Fire Department shortly after moving to the community almost 15 years ago. It's a great group of people to be involved with, willing to give their time to help their neighbors in the event of emergencies. We have members that give 2 hours a week and some that give 20, old and young, from varying backgrounds and walks of life — all
Volunteering at the fire department and/or auxiliary is a commitment, but it's not a job. We are all volunteers. There	

FIG 5.3: Our CSS layout is made of boxes that have no inherent connection to the typographic balance that we spent Chapter 4 perfecting. Only by measuring carefully and responding to pressure attentively can we maintain that balance.

our text blocks are pressured to go beyond their natural limits. In those moments, we invoke breakpoints.

This is what I call tailoring layout to text. We watch for pressure as we move from a limited view (like a narrow, linear layout) to a more complex view, and then we carefully consider the balance in and among text blocks as we decide how to respond.

Now, look at the paragraphs in St. Remy's linear layout when the composition is just slightly narrower than the first roughed-in media query breakpoint (**FIG 5.4**).

This looks okay, but it's not our only option. Should we invoke the breakpoint sooner, before the text block gets this wide? How soon? How do we decide when exactly to invoke the breakpoint?

Let's consider a few approaches for deciding on a breakpoint, and then we'll explore ways of relieving pressure by making alterations—strategies that don't require a breakpoint, but that help our work look incredible when combined with tasteful breakpoints.

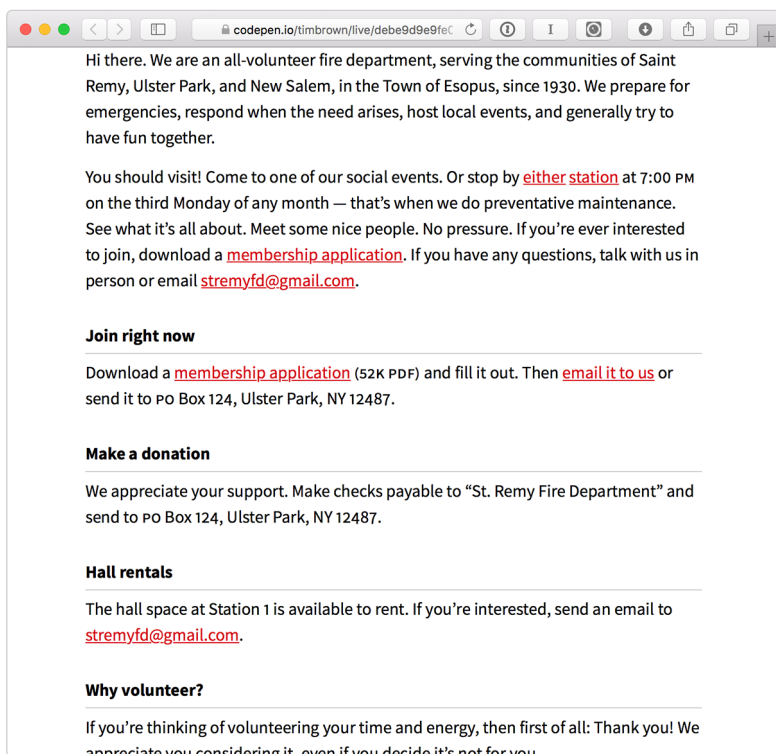


FIG 5.4: The maximum width of our linear layout, just before the roughed-in media query switches to a more complex layout.

Decide on a breakpoint using text-block limits

When we shaped our text blocks back in Chapter 4, we decided on an acceptable maximum line length. We determined that the measure should be *at most* 35em. We also worked out that our line height should be *its loosest* at this measure and, because of that, our horizontal margins should use their loosest value of approximately 5.1em each. We could calculate the total width of our widest text block and most generous margins, and use that as our breakpoint. This is what I roughed in—a breakpoint at

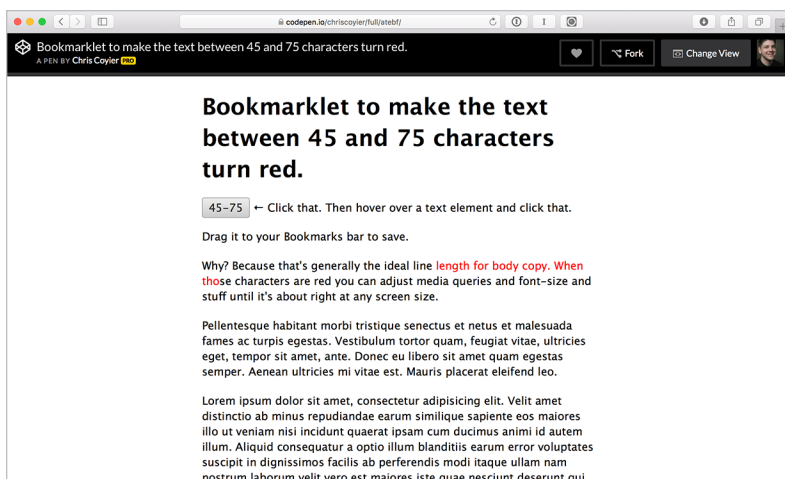


FIG 5.5: Chris Coyier made a bookmarklet that counts characters. Edit it on CodePen if you want a different range than the 45-75 character default (<http://bkaprt.com/ft/05-02/>).

52em. (35 + 5.1 + 5.1 does not add up to 52, but we'll talk about breakpoint units and values in a bit.)

Although it's good to remember these text-block limits, we don't necessarily need to use them in our code. Using a measure value from anywhere within the acceptable limits we decided on also works great.

Decide on a breakpoint by counting characters

Some common advice about choosing a line length is to count characters and shoot for a particular range. But as we discussed in Chapter 4, a text block's measure should primarily be a visual decision you make within the context of a given project. So if you've made such a decision, and you happen to figure out how many characters that amounts to, then counting characters is fine. Because then you are actively deciding how many characters to count, rather than taking general advice about measurement and then attempting to apply it to your particular project (**FIG 5.5**).

Decide on a breakpoint using a modular scale

Numbers from a modular scale can also help determine breakpoints. Remember the modular scale we made for St. Remy back in Chapter 4? We could use these numbers in a variety of ways to arrive at a breakpoint—for example, setting measure and margins individually with numbers from the scale, or choosing a larger number from the scale for the breakpoint without changing measure or margins (**FIG 5.6**).

There is no single, “correct” way to use a modular scale. We can set our paragraph’s measure and horizontal margins independently or cumulatively. We can combine smaller values from a scale, too, for more options. Or we can improvise. Have a scale, but ignore it at will.

Using modular scales this way can lend your work a subtle, underlying feeling of completeness, but remember that a scale is a tool—not a formula for instant success. Use scales as a starting point, or to double-check your visual decision-making.

Relieve pressure by making alterations

Let’s pause for a moment. We’ve just considered a few approaches for deciding on a breakpoint. But these decisions don’t necessarily require invoking a media query. Instead, they require us to think about making a change. What we’ve really just done is examine ways of identifying moments when something must change—moments when pressure is imminent for our text block.

By making alterations at such moments, we can relieve pressure. And, as you may recall from my introduction of the concept of pressure in Chapter 1, relieving pressure is critical for successful typesetting. It’s raining now, so it’s time to get our text a raincoat.

Here are some alterations we can consider—starting with none at all.

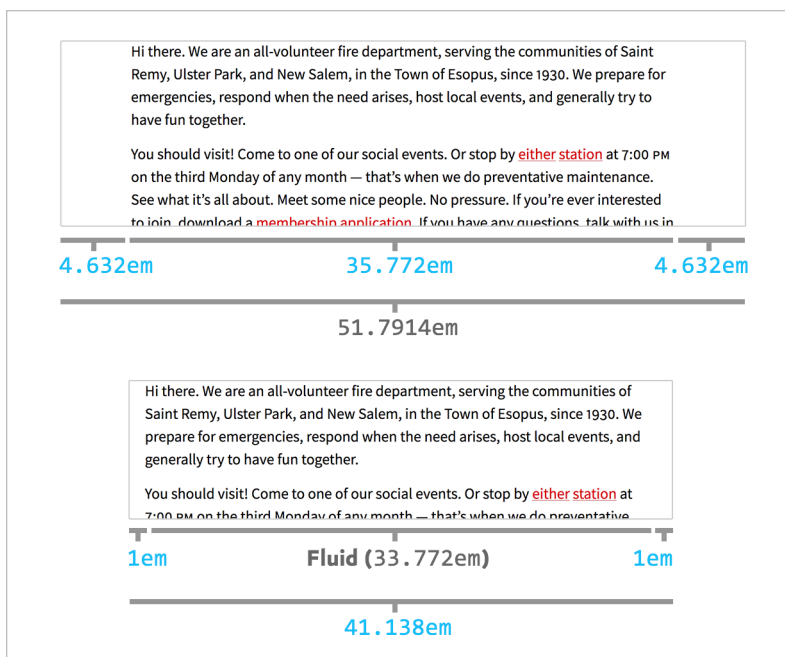


FIG 5.6: Numbers from our modular scale (35.772, 4.632, 41.138, and 1) appear in blue. Top: measure and margins set individually with numbers from the scale. Bottom: a larger number from the scale used as a breakpoint. (Again, these numbers don't seem to add up, but don't worry. We'll talk about breakpoints and units in a moment.)

Do nothing

We can do nothing at all and allow the measure to widen beyond the maximum limit we established for our text blocks in Chapter 4. This doesn't feel good in principle, but sometimes we need to be practical. This is a *choice we can make*. And if we make it consciously, aware that other options exist and aware of the pressure being exerted on the text, that's respectable.

Increase horizontal margins

We can let the white space grow beyond its own limits. We determined maximum horizontal margin sizes, right? We could allow those to increase, just as we considered allowing the text block's measure to widen. There are similar downsides (the margins may feel too big), but there are also ways to mitigate the discomfort of loose margins. For example, subtle visual buffers can help (FIG 5.7).

Increase font size

We can increase the font size of our body text. This doesn't feel good, either, for the reasons I explained in the section about changing font size—but it, too, is a respectable choice if the decision is made with careful consideration (FIG 5.8).

Slow the tempo

We can slow the tempo of the text block, too. Loosening the letter and word spacing *ever so slightly* makes a longer measure feel more acceptable and allows us to increase the line spacing a bit, which in turn affords larger horizontal margins. This can add up to quite a bit of breathing room.

Variable fonts may help in a similar way. By widening glyphs themselves just a tad, we can achieve the same kind of subtle relief we get by using white space to slow a text block's tempo—we can even combine the two techniques (5.9). Keep in mind that wider glyphs usually have wider letter spaces and word spaces; in effect, by making glyphs wider, we also slow tempo somewhat (even without explicitly modifying letter or word spacing).

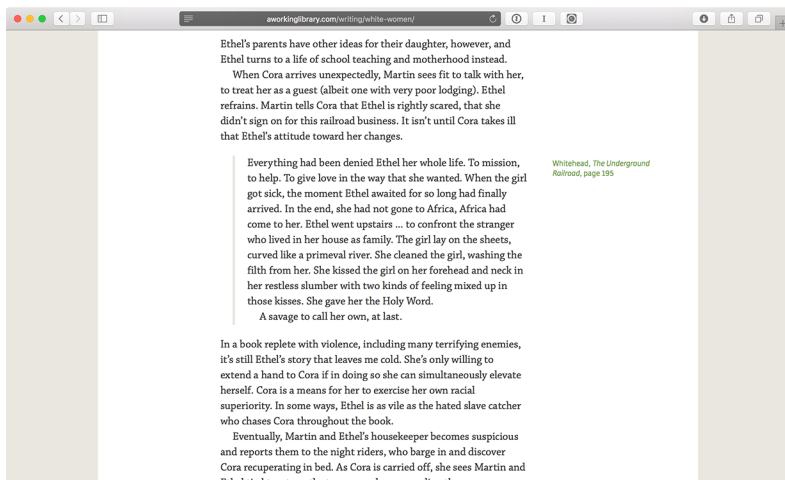


FIG 5.7: Mandy Brown's *A Working Library* uses a warm gray background to break up horizontal space when the composition is wide.

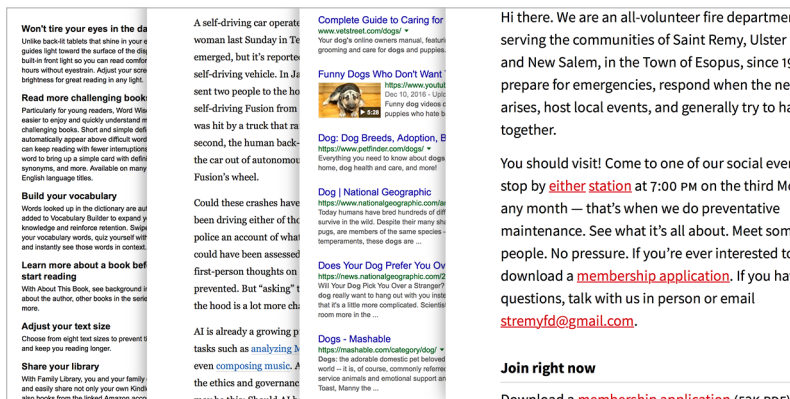


FIG 5.8: By increasing font size, our text could end up being much larger than the other text a person might usually encounter.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi nisl nisl, porttitor in feugiat id, accumsan eu mi. Sed et est tempor, elementum erat nec, congue nulla. Aliquam cursus ex eu nunc posuere, vitae volutpat purus rhoncus. Phasellus interdum tristique elementum. Curabitur nec tellus vitae elit gravida iaculis vitae vel odio. Integer porttitor enim id vestibulum molestie. Curabitur porttitor nisi est, eu pulvinar lorem congue eget. Proin orci augue, convallis vel ex vehicula, pretium vehicula magna. Morbi sit amet bibendum libero, vel malesuada dolor. Morbi

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi nisl nisl, porttitor in feugiat id, accumsan eu mi. Sed et est tempor, elementum erat nec, congue nulla. Aliquam cursus ex eu nunc posuere, vitae volutpat purus rhoncus. Phasellus interdum tristique elementum. Curabitur nec tellus vitae elit gravida iaculis vitae vel odio. Integer porttitor enim id vestibulum molestie. Curabitur porttitor nisi est, eu pulvinar lorem congue eget. Proin orci augue, convallis vel ex vehicula, pretium vehicula magna. Morbi sit amet bibendum libero, vel malesuada dolor. Morbi

FIG 5.9: Grade, a variable font by Adam Twardoch on Axis-Praxis (<http://bkaprt.com/ft/05-03/>). Here, the tempo slows because glyphs widen and white spaces grow.

Change the layout

Finally, another option is to lay the text out in a different way. There are a couple of approaches we can take here.

We can use this opportunity—this moment of relieving a pressure our text block faces—to transition our whole composition to a new layout. If we do this, it's important to think ahead. We've been focused on the current layout at its widest; what will the new layout be like at its narrowest? What pressures will text blocks face in that new context, and how might we avoid or resolve those (**FIG 5.10**)?

Another approach is to keep the composition as it is, linear, but reflow text chunks into multiple columns (**FIG 5.11**). This shortens the measure in places, makes margins more manageable, and leaves the font size unchanged! But tread carefully, because readers are still not very familiar with columns of text on the web, whereas scrolling has come to seem natural. Plus, spreading a text block over a wider area could make it uncomfortably shallow.

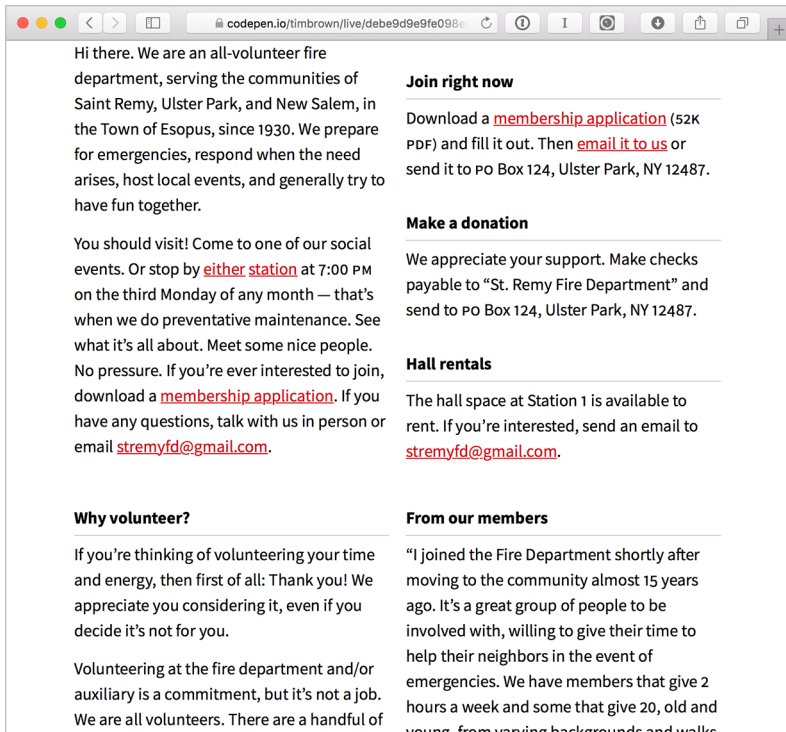


FIG 5.10: Our complex layout in its narrowest state—the viewport is just wider than our roughed-in breakpoint.

Alterations to St. Remy

For our project example site, I decided to let the paragraph grow just a bit wider than the maximum measure we determined in Chapter 4, and to allow the horizontal margins to widen until they each reached about **4.6em**, before transitioning the whole composition to a new layout:

```
main {
  /* Fallback min-width: 23.5em */
  max-width: 40em; /* Fallback max-width */
  padding: 0 1.1em; /* Fallback horizontal padding
  */
```

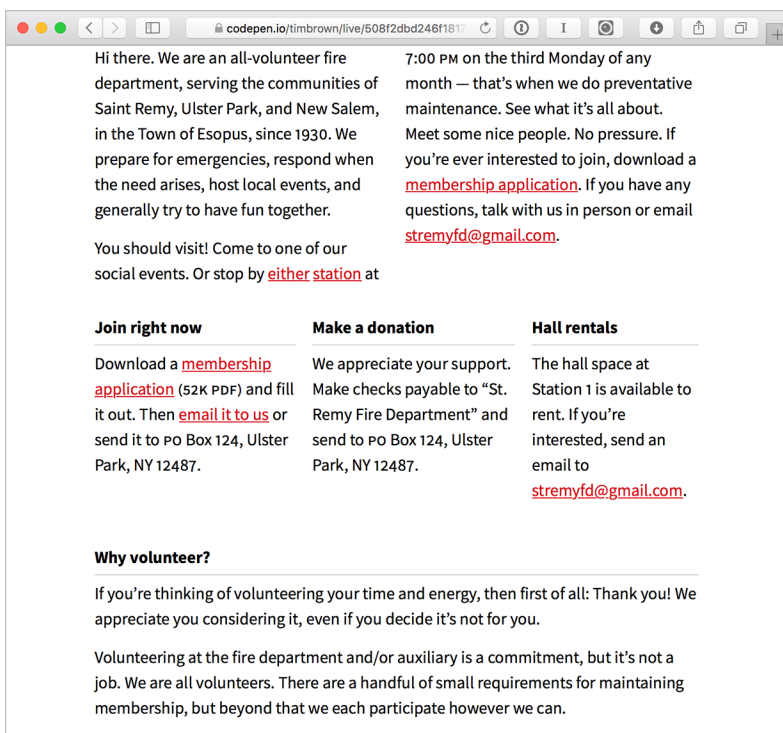


FIG 5.11: Applying CSS Multi-column Layout (<http://bkaprt.com/ft/05-04/>) or flexbox within sections of our linear layout solves some problems, but isn't quite ideal.

```
margin: 0 auto;
hyphens: auto;
}
.wf-active main {
  /* Ideal min-width: 21em */
  max-width: 36em;
  padding: 0 1em;
}
```

Code example: <http://bkaprt.com/ft/05-05/>

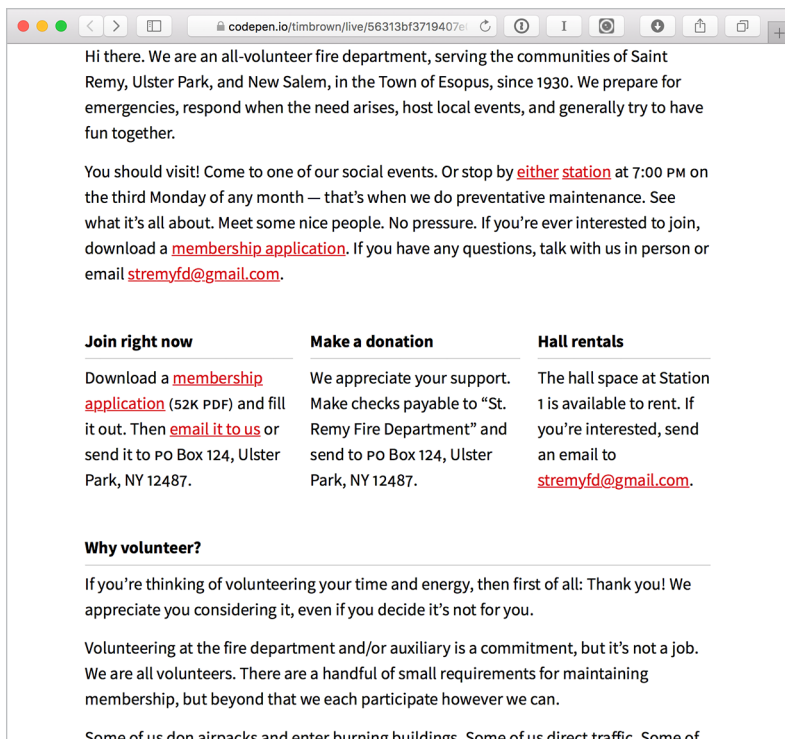


FIG 5.12: This looks good. I also added a "minor" (less important) breakpoint and used flexbox to lay out the section on joining, donating, and renting the hall.

Let's review. First, I widened the text block's `max-width` to `36em`, up from `35em`. Then I checked on horizontal margins. The container (`main`) incorporates minimal horizontal margins via the `padding` property, so the text won't sit flush against the edge of narrow viewports. And thanks to `margin: 0 auto`, horizontal margin space will expand when the container reaches its maximum width. Next, I widened the composition until those margins felt uncomfortably loose to me. At that moment, I invoked a media-query breakpoint of `52em`—our roughed-in breakpoint was perfect. I'm happy with the way the composi-

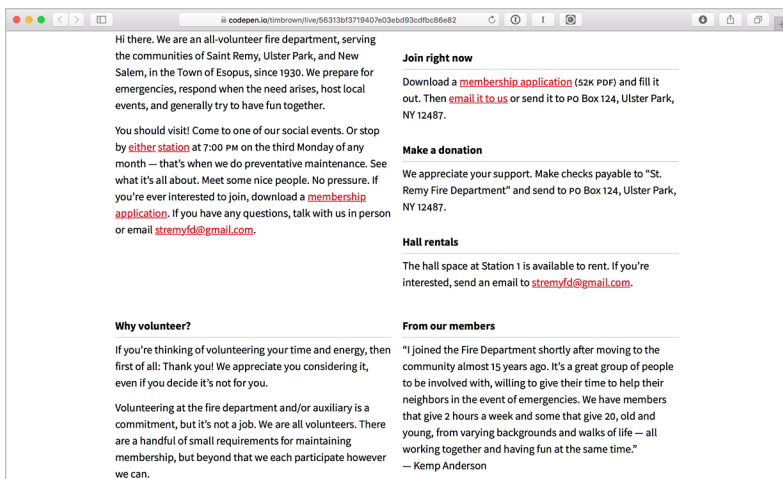


FIG 5.13: Think of this as several linear layouts, close to each other. Shape these text blocks the way we did in Chapter 4.

tion looks immediately before the main breakpoint (FIG 5.12). Immediately after this breakpoint, though, it needs some work.

Do it all again for the following layout(s)

Now, look at the paragraphs in St. Remy’s more complex layout when the composition is just wider than our new, tailored media-query breakpoint (FIG 5.10). Also, when it is considerably wider (FIG 5.13).

This situation is very similar to the one we just resolved in the narrow context, and our options here are the same. We’ll need to make alterations, again and again, in as many situations as we find pressure. For now, treat each column of this layout as if it were its own isolated, narrow, linear layout.

Don’t stress about this tailoring too much. There are no “correct” answers, and there are many additional factors to consider in complex layouts. Later in this chapter, we’ll talk more about breakpoints and grids, and all of Chapter 6 discusses

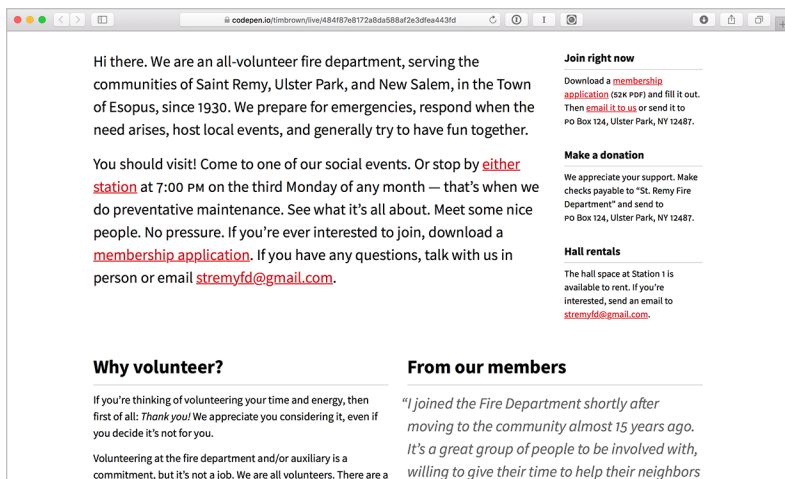


FIG 5.14: Our complex St. Remy layout with stronger contrast. Now we can see what's most important. All we're doing here is shaping text blocks!

identifying and resolving pressure. But before we get to any of that, we need to talk about contrast.

ADAPT HIERARCHY TO LAYOUTS

Even when layout measurements are in good shape, and text blocks don't seem to be facing specific pressures, a composition can feel flat, as though it has an uncomfortable stillness. The reason is often that it lacks sufficient contrast.

Contrast, in this case, means that some parts of a composition noticeably stand apart from others. Great compositional contrast results from using those differences to accomplish the goal of the design—establishing hierarchy by guiding viewers' eyes and compartmentalizing information, as well as facilitating the desired graphic experience (**FIG 5.14**).

Code example: <http://bkaprt.com/ft/05-06/>

Let's study how this updated version of our composition's wide layout uses contrast to successfully establish hierarchy. Then we'll talk about when contrast is most necessary across the compositional continuum.

Recognize hierarchy

Look at our example site. The introductory text is now larger than the rest of the body text. The "Why volunteer" heading is larger. The quotations from members use a different style. And the smaller bits of information (membership application, details about donations and hall rentals) occupy a narrower space. Chunking the composition into pieces this way can guide readers, create emphasis, and help people digest information (**FIG 5.15**).

Squint at this composition and you'll get a sense of its depth. Bigger, bolder chunks feel closer, while other parts recede. When I look at this composition, I'm first drawn to the intro text, then to the "Why volunteer" heading, and then to the "From our members" section. Lastly, I notice useful information tucked away on the right.

St. Remy achieves this compositional depth and hierarchy by affecting contrast in a variety of ways, which we'll look at in turn.

Create hierarchy by affecting contrast

Larger things feel closer to us, and darker chunks of text draw our attention. Here, the introductory paragraph uses a larger font size, which yields a darker typographic color. This adjustment creates substantial depth in the layout (**FIG 5.16**).

Increasing font size, bolding text, quickening a text block's tempo by tightening up its white space, and other subtle adjustments can influence a text block's color. All can be effective methods for generating depth in a composition.

Changes in contrast work best within constraints, because constraints provide a baseline from which some elements can stand out. For this reason, it's important that we tailor our layout structure to the body text before adjusting contrast.



FIG 5.15: Squinting at or blurring in-progress compositions is a great way to judge whether there is sufficient contrast.

“Why volunteer?” is another important part of this composition, and a natural next step in the narrative of the content (FIG 5.17). The intro basically says, “Hello, you should join.” It makes sense that most people’s next question would be: “Why?”

Join right now

You should visit! Come to one of our social events. Or stop by [either station](#) at 7:00 PM on the third Monday of any month — that's when we do preventative maintenance. See what it's all about. Meet some nice people. No pressure. If you're ever interested to join, download a [membership application](#). If you have any questions, talk with us in person or email stremmyd@gmail.com.

"I joined the Fire Department shortly after moving to the community almost 15 years. It's a great group of people to be involved

Why volunteer?

"I joined
moving
It's a gr
willing

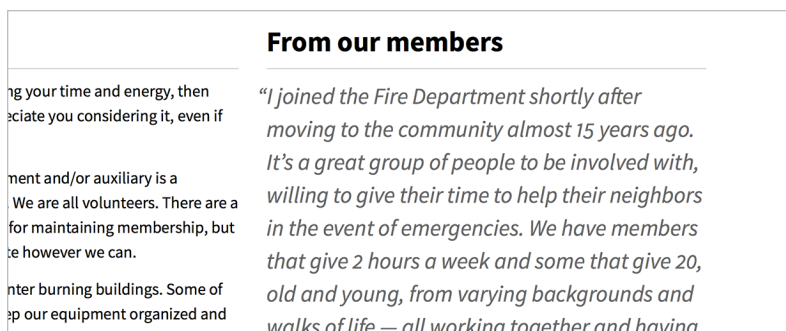


FIG 5.18: Even though this text uses a larger font size, it recedes in the composition. Note the hanging open double-quote marks; a nice touch, and easy to do with a little negative [text-indent](#).

I bumped this heading size up to **1.667rem**, a number from the modular scale I created for this project, and I’m allowing the heading’s measure to be determined by the body text underneath—the heading serves the text block.

This heading is still an **h2** semantically, but I’ve made it look stronger in this context. In more complex projects, I often do this by using what Harry Roberts calls a “double-stranded heading hierarchy” (<http://bkaprt.com/ft/05-07/>). He picked the technique up from Nicole Sullivan, who encourages its use as a way to manage both semantics and style (<http://bkaprt.com/ft/05-08/>).

Quotations in the “From our members” section are italicized, with a larger font size and tighter line spacing (**FIG 5.18**). They also use a lighter gray than the main text. All of these adjustments contribute to the section’s typographic color, so tweaking any one of them will affect the whole. Because of the decisions I made here, this part of the composition recedes a bit, which helps the other parts stand out.

Be careful when you modify *actual* color to affect compositional contrast, because that can make text hard to read for lots of people. Here, I’m using a light gray that just passes WCAG 2.0 level AAA guidelines for minimum contrast ratio in large-scale text (<http://bkaprt.com/ft/05-09/>). Lea Verou made a great

tool for checking text's color-contrast ratio. As you can see, our quotations have a ratio of 4.5:1 (<http://bkaprt.com/ft/05-10/>).

Finally, the supplementary information on the right recedes as well, because it's smaller in size and its measure is narrower, and because the main hierarchy in the composition directs attention away from this area.

Shaping smaller text blocks like this is very similar to setting body text, with two exceptions: you're shooting for a smaller size, and you want it to look good alongside the body text. If your body-text typeface isn't working well at such a small size, try subtly increasing its letter spacing and line spacing. Check to see if your chosen typeface includes optical sizes for caption or "micro" use; if not, consider a different typeface for the palette's small-text needs. Typefaces and optical-size variations made for small text are designed with looser spacing and higher x-heights.

This is all just the tip of the iceberg. Carl Dair's book *Design with Type* devotes a chapter each to contrasts of size, weight, structure, form, color, direction, and texture, as well as a short chapter about "multiplying the contrasts."

Dair stresses that size is the most fundamental way to affect contrast. I agree, but I think it's equally fundamental to understand the effects that different typefaces and variations have on contrast.

The style of a typeface, as well as adjustments to its weight, width, and other qualities, can have strong effects on the typographic color of a text block—which can profoundly affect compositional contrast. It's especially important to remember this with variable fonts, because making adjustments to the shapes and spaces of type itself may offer subtle, effective solutions for managing contrast that do not require layout gymnastics or the loading of additional font files.

Decrease contrast as the composition gains focus

In the previous section, we used compositional contrast to establish hierarchy in a wide layout. Contrast plays a huge role when the composition is wide and complex because so much text is visible at once. Faced with a busy layout, readers need

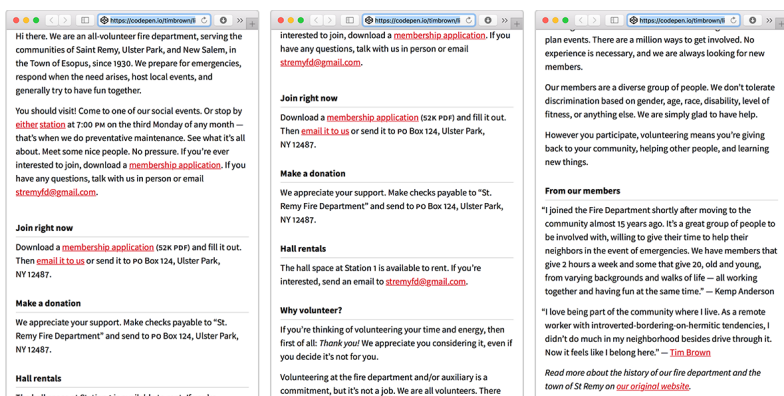


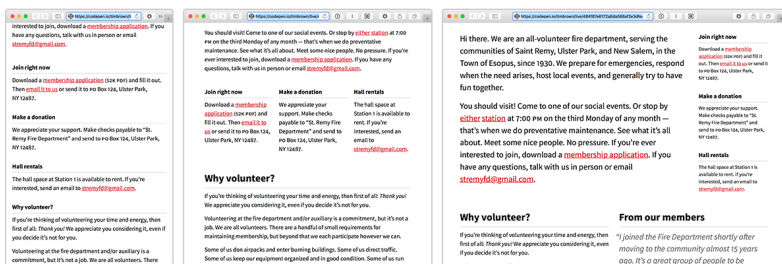
FIG 5.19: Readers usually only see one screen like this at a time. A little contrast goes a long way in a focused layout.

help finding their way, making sense of the composition, and understanding how parts are related.

Contrast is less important in narrow layouts, where compositions are very focused (**FIG 5.19**). Readers can't see much at any one time. Scrolling or paging through text is a natural behavior with this much focus, so the sequence of information provides the primary sense of depth.

As a general rule, decrease contrast when the composition is more focused (**FIG 5.20**). As a composition condenses—from a wider, taller layout like the one we've been looking at in this chapter to a narrow, short layout like the mobile-first, linear layout we started with in the early chapters—continually reduce its contrast.

It helps to work backward a bit. Start by introducing compositional contrast in the most complex, least focused layout and work downward, adapting the hierarchy to each layout in the compositional continuum. Reduce heavy contrast down to the very subtle hierarchical cues that simpler, more focused layouts need. Another approach is to figure out how both extremes should look (wide layout, narrow layout), and then try to interpolate the steps in between. In-between layouts need in-between contrast.



More focus

More contrast

FIG 5.20: A focused, narrow layout? Basic contrast. A wider, more complex layout? Strong contrast. Note that the layout in the middle has contrast that is neither too weak nor too strong.

INFORM GRIDS AND SYSTEMS

Now that we know a lot more about how our flexible composition will look, it helps to put systems in place for managing all of the decisions we’ve made across the compositional continuum. We’ll do that by declaring variables, keeping breakpoints tidy with comments, and measuring grids meaningfully.

Some of the tips in this section involve Sass, a CSS preprocessor that makes writing styles more efficient via variables, nesting, and functions. If you’re not familiar with Sass, I highly recommend Dan Cederholm’s Sass for Web Designers (<http://bkaprt.com/ft/05-11/>).

Also, CSS custom properties will offer some of the same conveniences that Sass does, and they’re just around the corner in terms of browser support. Review the logic here to get an idea of how variables and custom properties can help with typesetting.

Variables

Having our system of measurement handy for reference is a huge time-saver. Let’s put our modular scale’s URL in a comment, and turn numbers from the scale into variables. By allow-

ing us to name all of the different values we decide on, variables help us keep track of the flexible limits we determine. You'll recognize many of the variable names and values below from this book's earlier chapters.

```
/* Variables
-----
Scale: http://www.modularscale.com/?1.15&em&1.667
----- */

$text-size-fallback: 1.03rem;
$text-size: 1.15rem;
$text-size-small: 0.839rem;
$text-size-large: 1.398rem;

$text-measure-min-fallback: 23.5em;
$text-measure-max-fallback: 40em;
$text-measure-min: 21em;
$text-measure-max: 36em;

$text-line-min: 1.3;
$text-line-max: 1.5;
```

Variables can also help us keep font stacks simple and reusable:

```
$font-source-sans-pro: "source-sans-pro", "Lucida
  Sans", "Lucida Grande", "Lucida Sans Unicode",
  "Open Sans", "Droid Sans", sans-serif;
```

Because we're using variables for our measurements and typefaces, it makes sense to use them for our breakpoint measurements too. We can list those along with all of our other variables:

```
$width-medium: 39em;
$width-wide: 52em;
```

Give breakpoint variables names that describe the layout changes they each handle, or the pressures they resolve—with-

out mentioning specific numbers or specific devices. Check out Chris Coyier’s CSS-Tricks post about naming media queries (<http://bkaprt.com/ft/05-12/>). Using named breakpoint variables in combination with a Sass mixin makes the code we write much clearer to read, and easier to understand:

```
p {  
  .wf-active main {  
    max-width: 36em;  
  }  
  @include at-least(width-wide) {  
    .wf-active main {  
      max-width: 59.633em;  
    }  
  }  
}
```

Now, let’s talk about the em-based values we’ve been using to set breakpoints. They aren’t quite the same as the ems we’ve been using for `max-width` values. Let me explain.

Breakpoint units and values

Designers often decide on breakpoints visually, either invoking them as they observe their flexible compositions or choosing specific pixel values to approximate popular devices. There’s nothing wrong with the former practice, but there are major problems with the latter. Devices change constantly, so we can’t rely on any particular pixel resolution. And pixels, as a unit of measurement, are fraught with trouble.

In Chapter 2, we learned that type sized with pixels does not respect a user’s default font-size preference (which is why we use em-based font sizes). Similarly, pixel-based breakpoints do not respect a user’s default font-size preference. If we use pixel-based values in our media queries, people who have adjusted their font-size preferences will receive a broken layout (<http://bkaprt.com/ft/05-13/>).

So we'll use em-based breakpoint values. Besides being more accessible, they offer us the chance to be more purposeful with our measurements. We understand the value of careful balance in our text blocks; we should express the same care when we measure our breakpoints.

But there's a catch: we need to do some math.

It seems logical that if the maximum measure for a text block is, say 35em, and the maximum horizontal margins on either side are each 5.1em, then I should be able to define a breakpoint at 45.2em (instead of the 52em we ended up with back in Chapter 5). However, em-based media queries aren't calculated this way.

Remember St. Remy's font-size situation? We set a :root font size of 1.15rem. The main element inherits this font size, treating 1.15rem as its 1em. When we give the main element a max-width of 36em, that's actually 41.4em from the root's perspective. Here's the rub: em-based media queries always refer to the default root font size, which is essentially 100% or 1em (<http://bkaprt.com/ft/05-14/>). They ignore any changes we make to the root font-size value.

If this all seems confusing and frustrating, that's because it is. The problem stems from the fact that we're making design decisions about specific text blocks, but basing breakpoints on something much broader—the viewport. This is why a handful of smart people have been advocating for the specification of *container queries*.

Why can't we apply styles based on the space available to the module we're designing, rather than looking at the shape of the viewport?

This is, in a nutshell, the disconnect that container queries are trying to address. As our designs become more modular and pattern-driven, the value of media queries has decreased. That's not to say media queries are, like, bad—not at all. After all, they're the best tool we've got right now. But there's a divide growing between what our responsive designs need to do, and the tools CSS gives us to meet those needs. We're making design decisions at smaller and smaller levels, but our code asks us to bind those decisions to a larger, often-irrelevant abstraction of a "page."

That’s Ethan Marcotte on container queries (<http://bkaprt.com/ft/05-15/>). Querying our text blocks’ containers, rather than the entire viewport, makes a lot of sense. It could transform that logical 45.2em scenario from a few paragraphs ago into a reality, and allow for more meaningful breakpoint calculations. We’ll get there. In the meantime, I encourage you to keep very detailed comments near your measurements.

Grids

This isn’t a book about grids, so I’ll keep my foray into them short and sweet. Typesetting without a grid is like pouring yourself a glass of water without the glass: unless you’re doing some kind of magic trick, it’s a mess. Get very good at making grids. Rachel Andrew’s *The New CSS Layout*, is an excellent introduction to CSS layout, including the new CSS Grid Layout specification (<http://bkaprt.com/ft/05-16/>).

Whether you’re using Grid, flexbox, a combination of the two, or an older layout method, my advice to you is straightforward: consider proportional harmony and watch for pressure as the composition flexes.

Proportional harmony happens when broad divisions in a layout—the widths of columns, the aspect ratios of media, and so on—are guided by the same modular scale, or the same ratio, that informs how the type is sized and spaced.

For example, if a modular scale uses a ratio of 3:4, numbers from that scale could be used for font sizes and margins (1.333em, 2.369em), as well as the maximum width of a grid container (55.927em). Inside the container, grid-template columns and rows can use fraction units from the scale (1fr, 1.777fr, 5.61fr); grid gaps and other small spaces can too (FIG 5.21). Of course, modular scales aren’t the only way to measure grids. You could opt for simpler fractional divisions (1fr, 2fr), or eyeball your measurements.

However you decide to measure grids, though, be sure to resolve any pressure your layout puts on text blocks. Pay special attention to the flexible aspects of your grid—properties like `align-content` and `justify-content`, as well as the `minmax()` function and `auto-fit/auto-fill` keywords. These factors

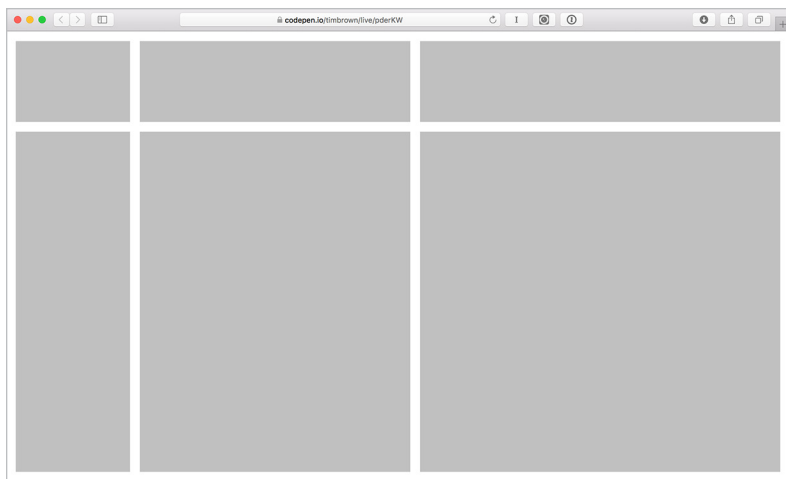


FIG 5.21: Modular scales are a great way to measure grids.

can cause pressure by introducing unpredictable amounts of white space.

Still with me? If you’ve made it this far in the book, then you’ll probably like the next section. It’s about baseline grids and how I think they need to flex.

Baseline grids

A baseline grid is a series of rows spun out of the spacing between baselines in a text, or the invisible line that letters sit on. It’s a means to horizontally align all the type on the page, including captions, headlines, and running text.

—Jason Santa Maria, *On Web Typography*

Toward the end of *On Web Typography*, Jason Santa Maria explains why baseline grids matter in print typesetting: they “back up” lines of text on translucent paper and provide a consistent rhythm down the page. Santa Maria also emphasizes that baseline grids are difficult to use on the web today: “CSS isn’t optimized to achieve this level of control yet.” He adds that

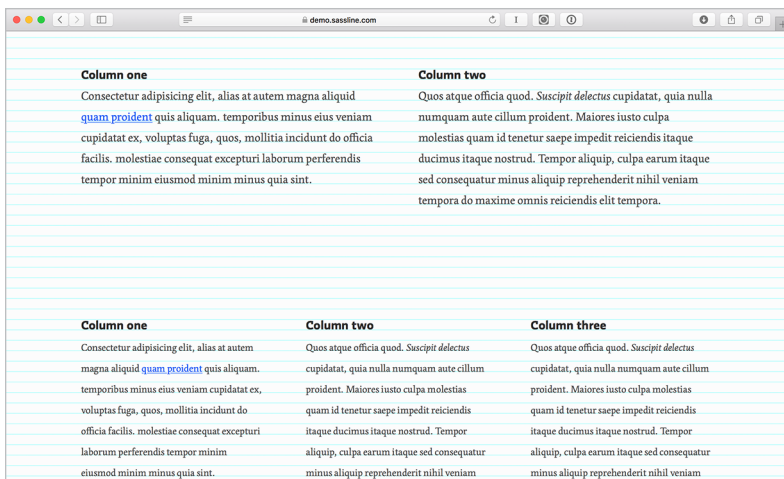


FIG 5.22: Jake Giltsoff’s Sassline makes creating baseline grids easier, and offers a host of other interesting features, but we need a standard CSS solution for flexible baseline grids.

because they interfere with flexibility, baseline grids are “more trouble than they’re worth.”

I agree, but I also think that baseline grids could significantly improve complex layouts featuring lots of text (like news sites), and even simpler multicolumn layouts. If baseline grids could be specified in CSS in a straightforward way, that would be a very big deal. And, wouldn’t you know it, things are beginning to take shape. In 2017, Jen Simmons linked to the CSS Rhythmic Sizing specification (<http://bkaprt.com/ft/05-17/>), along with a demo that shows how using a property like `line-height-step` might affect layout (<http://bkaprt.com/ft/05-18/>).

It’s great to see this happening. My hope is that we will be able to use *flexible* baseline grids someday.

In this book, we’ve talked about the balance inherent in text blocks, and the fact that this balance is disrupted when text blocks face pressure. Baseline grids usually cause pressure in flexible compositions because they constrain line spacing; they

essentially make rigid a key property of our text block—and it’s precisely the property we depend on to be most flexible. As we learned in Chapter 4, for body text to feel balanced, line-spacing values should react to the typeface, font size, and measure.

I believe the solution lies in defining white-space-value ranges and conditions. Dynamic line spacing (the example I used in CSS locks) directly influences the integrity of each text block, and yet this is incompatible with an inflexible baseline grid. If baseline-grid CSS syntax were to flex all white space simultaneously, in a coordinated way—including margins, line spacing, and text-block tempo—then we would finally achieve in web layouts the kind of vertical rhythm baseline grids have always provided in print.

I know how difficult that sounds. But nowadays we design and build responsive websites—and *that* would have sounded incredibly difficult ten years ago. The conditional flexibility that Ethan Marcotte introduced in *Responsive Web Design* was an important first step, and his more recent focus on patterns and principles is very smart. Breaking layouts into isolated components makes it easier to maintain each component’s integrity. We’ve seen how important that is in typesetting, where preserving the integrity of each text block is critical. The next step is articulating how components relate to one another—defining spatial relationships among flexible elements.

In the final chapter, we’ll learn how to evaluate the quality of flexible design work by carefully examining it from a variety of angles, and bringing all of what we’ve learned in this book to bear as we identify and relieve pressure.

6 RELIEVING PRESSURE

“*All of typography is based upon a reflexive relationship: the small things inform the big things, and the big things inform the small things.*”

—JASON SANTA MARIA, ON WEB TYPOGRAPHY

OUR EXAMPLE PROJECT has guided us through this book. I’ve shown you my process for preparing text and code, selecting typefaces, shaping text blocks, and crafting compositions. But our site still has some problems.

In this final chapter, we’re going to address those lingering problems. I’ll show you how to evaluate the composition and relieve the pressures we notice. We’ll continue to assess our project example site, refine it, and make it look great.

You can also start here if you have a finished site that you want to improve. This chapter is about evaluating something that exists—and then going back through the process of past chapters to resolve problems reflexively. We’ll return to work we’ve already done, but now with more information and more context, and we’ll make revisions.

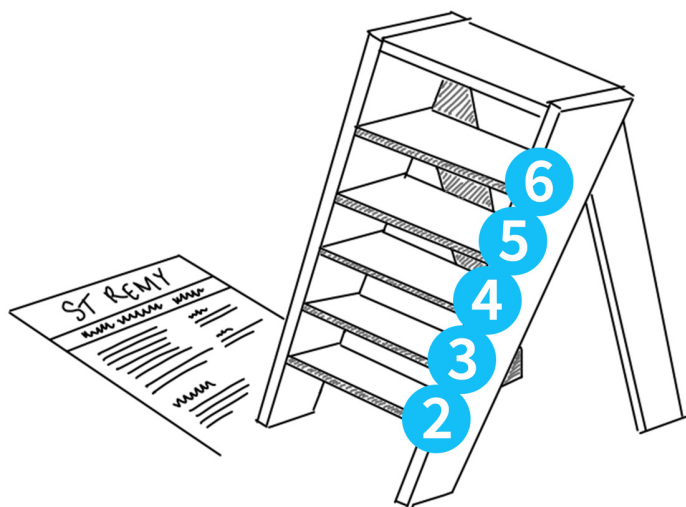


FIG 6.1: The typographer's ladder. Keep it handy.

Think of this approach to evaluating and revising compositions as a ladder, where the second through fifth chapters of this book are the rungs (**FIG 6.1**). We're standing at the top of the ladder now, looking down. From up here, we'll evaluate the composition. Then we'll step down to relieve pressure in our text blocks, step back up to relieve compositional pressure, and climb back to the top for further evaluation.

EVALUATE THE COMPOSITION

Look at the composition in a browser window on the largest screen you have access to. Use CodePen's live view for this (<http://bkaprt.com/ft/06-01/>). Maximize the browser window, open the browser's inspector, and dock the inspector to the right. Then, make the viewport very narrow. We want to evaluate this composition at its narrowest extreme. I like Google Chrome for this (**FIG 6.2**).

Now open a blank spreadsheet. We're going to flip back and forth between this spreadsheet and the composition we're inspecting. In the first row's second cell, write "Body text". In the first column's second cell, write "<150px". And in the second row's second cell, write down any problems you see with the body text in this very narrow setting.

Right off the bat, I can see that the body-text line lengths are way too short—in some cases, a single word per line! And the line height feels too loose (probably because the measure is so narrow). Note what you see (**FIG 6.3**).

You won't be able to see the entire composition in this narrow context—it's too tall—so scroll down a bit. Take notes and, if it helps, screenshots. Make a new column in the spreadsheet for every text block you evaluate.

Now slowly widen the composition by resizing the inspector pane. Watch how the text reflows and changes. Scroll up and down and resize the width repeatedly so you can see how the whole composition flexes. Stop when you notice something new to write about, title a new row using the width from the inspector, and keep taking notes. Do this continually, making the composition wider and wider. If you notice that certain pressures have gone away, write that down too.

Sooner or later, some or all of the composition will likely change significantly when a breakpoint is introduced. Study the composition carefully, before and after breakpoints. I find it helps to indicate breakpoints using a border that spans the entire width of my evaluation spreadsheet (**FIG 6.4**).

Use pressure to find the seams

[Look] at the smallest version of a piece of content, then [expand] that element until its seams begin to show and it starts to lose its shape. Once that happens, that's an opportunity to make a change—to introduce a breakpoint that reshapes the element and preserves its integrity. But first, we need a method of finding an element's seams, and understanding how it loses its shape.

—Ethan Marcotte, *Responsive Design: Patterns & Principles*

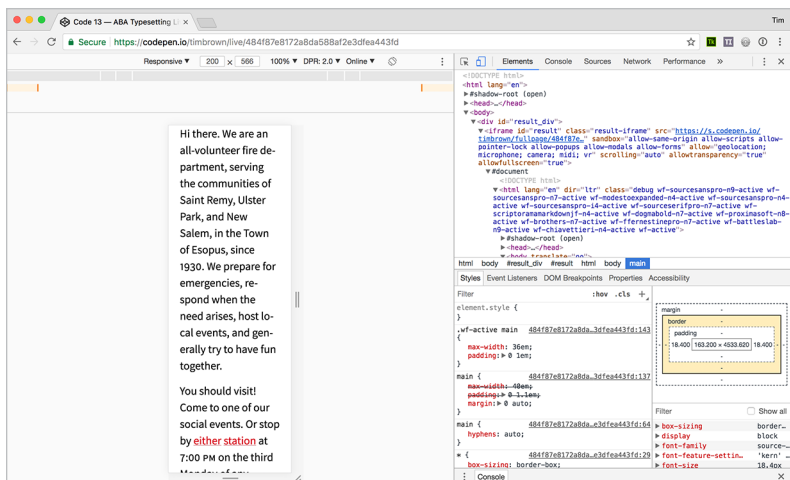


FIG 6.2: Google’s Chrome browser has a feature called “Toggle device toolbar” that makes evaluating compositions easy. Importantly, this feature allows you to make the simulated viewport narrower than the browser window would go.

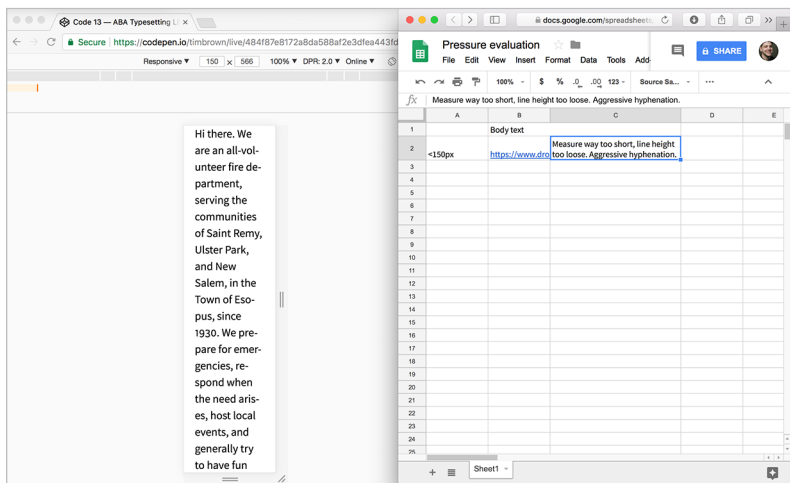


FIG 6.3: My pressure evaluation spreadsheet in Google Drive. I find it helpful to merge my header cells and use two columns for each chunk I’m evaluating—a column for a screenshot URL, and a column for my comments.

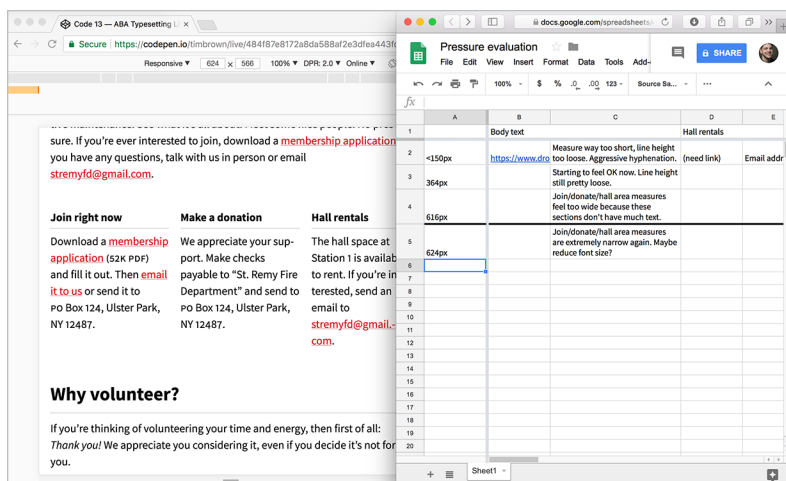


FIG 6.4: Broad borders indicate media-query breakpoints. I can imagine single-cell borders being useful for container-query breakpoints.

By doing this evaluation exercise, we’re looking for what Ethan Marcotte calls seams. When he talks about seams, about an element losing its shape, this is in many cases directly analogous to pressures that typeset text blocks face as they are pushed beyond their natural limits.

So far, we have only evaluated pressures related to width. We began Chapter 5 with a narrow, linear layout and moved to wider layouts. Width is one axis along which our composition flexes, so it’s one axis along which seams can be exposed.

But we need to study and take notes about other ways in which the composition flexes, too—additional axes along which seams get exposed. These require different ways of looking, but we can continue with our note-taking method simply by adding rows.

Axes of evaluation

Evaluate the composition in a variety of dimensions. Here are some ideas to get you started:

- **Width.** Change the width of the composition, as we did.
- **Height.** Change the height of the composition. You can usually also dock web inspectors to the bottom of the browser window.
- **Aspect ratio.** Change both the width and height of the composition to arrive at a specific aspect ratio. Change the screen's orientation, too, by swapping the dimensions. The Safari, Firefox, and Chrome browsers all offer a “responsive design mode” as part of their developer tools, which may help.
- **Closeness.** Zoom the composition to simulate a close-up view. Better yet, use a variety of real devices meant for use at different distances (phones, tablets, laptops, desktop monitors, televisions). In each of these contexts, zoom the composition and use settings and preferences to increase and decrease font size. Keep notes about any unfamiliar steps you took, so you can recreate them later. Take screenshots or photos.
- **Coarseness.** This is another good axis to evaluate with real devices, if possible. Look at your composition at a variety of screen resolutions, and from the perspective of devices that use different rendering engines and antialiasing settings (<http://bkaprt.com/ft/06-02/>).
- **Network presence and quality.** Evaluate the composition without a network connection, and with various levels of network throttling. Some browsers offer a network timeline view, where you can disable your cache and throttle the connection speed (<http://bkaprt.com/ft/06-03/>). There are also dedicated tools for this, like Charles Proxy (<http://bkaprt.com/ft/06-04/>).

Ways of looking

Some ways of looking require us to manipulate the composition itself, or our relationship to it:

- **Resizing.** We did this already, as we resized the width of a composition to evaluate width-related pressures. We can resize a composition horizontally, vertically, or both.

- **Reorienting.** We can also flip a composition around and look at it from a different angle. Nathan Ford’s Flippant bookmarklet is helpful for this (<http://bkaprt.com/ft/06-05/>). We can also physically move our screens or our bodies to find the right angle.
- **Comparing.** Opening up a second copy of the composition allows us to keep it in a browser tab, manipulate it in a slightly different way than we did before, and toggle back and forth. Alternatively, we can juxtapose them and look at them side by side.

Other ways of looking can be used in combination with these, independently of how we’ve manipulated the site itself.

- **Squinting.** Squinting, or blurring the composition slightly to simulate squinting, can help us see things we might not see otherwise. This is particularly useful for evaluating compositional pressures. Nathan Ford has another helpful bookmark, MIN, that includes a blurring feature (<http://bkaprt.com/ft/06-06/>).
- **Waiting.** Spend some time away from the work. Distancing yourself, even for fifteen minutes, can make a big difference.
- **Getting feedback.** Ask friends or colleagues to evaluate the composition with you. They may notice things you don’t. Remind them that this is a work in progress. Take any criticism with a grain of salt.

If you feel the urge to consider these axes and look in the ways I’ve described here as you typeset compositions in the first place, so much the better! There’s always a need for this kind of reflexive problem-solving—climbing up and down that ladder—whether you’re starting from scratch or evaluating an existing composition.

Save that spreadsheet

The rest of this chapter will focus on relieving the kinds of pressures we found during evaluation. By tackling issues one by one, our spreadsheet can serve as a handy to-do list—but it

may prove even more useful as a reminder and a map. Because evaluation isn't something we do just once.

For every pressure we alleviate, we risk introducing new pressures. Evaluation is part of a continual, cyclical process of refinement. With experience, you may not need to represent this part of the process with an actual artifact like a spreadsheet, but it helps to show where you've been and gives context to any new issues that arise.

In that spirit of ongoing improvement, we'll now step away from the St. Remy example project. You may catch a glimpse of it again as we finish up here, but there will always be new pressures to face; consider it a work-in-progress, and check on it anytime at stremyfd.com.

RELIEVE PRESSURE

Now let's step down from our imaginary ladder and fix some things. What this really means is that for each pressure we've noticed, we need to identify the source of the problem, make a different decision, and reevaluate every compound decision after that problematic one.

The chapters in this book are designed to help with this process. If a pressure is related to compositional balance, flip back to Chapter 5. If the cause of pressure is deeper, specific to a text block, first revisit Chapter 4 to make the appropriate change, and then double-check every decision that comes after it in the book. Or maybe the real problem is the anchor typeface. If so, turn to Chapter 3, choose a new typeface, and then proceed through Chapters 4 and 5. (This is what makes choosing new type for a project so challenging, and so vital: many other decisions depend on it.)

What follows is a pattern language of typesetting pressures. Note that for any specific pressure, the solutions are limitless. But I hope these patterns will help illustrate the interdependencies among text, type, text blocks, and layouts, so that it's easier to remember how every element affects other elements.

Keep an eye on your budget

Consider your performance budget as you make decisions about how to resolve pressure. In cases where using an additional typeface will relieve pressure, think about the effect this will have on how many total fonts are in use. In fact, this part of the process is when I usually begin to think about pruning my palette—reducing the number of fonts in use overall—especially because many pressures can be resolved in ways that don’t require more font files.

Now, let’s get to the patterns.

Pressure related to shape

These patterns illustrate pressures that affect the balance in a text block’s typographic relationships—typeface, font size, line spacing, and measure.

Text feels too tight

Text usually feels too tight because its line spacing is too small for its measure. This makes it difficult, while reading, to find the beginning of the next line as your eyes move backward from right to left (**FIG 6.5**).

When we loosen the line spacing, the horizontal margins feel better, too. Their generosity contributed to the original feeling of tightness in the text block—because they consisted of large volumes of white space, they made the white space within the text block seem smaller.

Another approach is to reduce the font size, which makes the same amount of line spacing feel more acceptable (**FIG 6.6**). But with this approach, it’s easy to wind up with type that is too small. If the type needs to be small but just doesn’t look good that way, we can try switching to a typeface specifically made for use at diminutive sizes—or, if the body-text typeface we’re already using happens to have a caption-sized counterpart, we could try that.

Intellectual history includes the history of thought in many disciplines, such as the history of philosophy, and the history of economic thought. Analytical concepts – such as the nature of paradigms and the causes of paradigm shifts – have been borrowed from the study of other disciplines, exemplified by the use of the ideas of Thomas Kuhn about The Structure of Scientific Revolutions to explain revolutions in thought in economics and other disciplines.

Intellectual history includes the history of thought in many disciplines, such as the history of philosophy, and the history of economic thought. Analytical concepts – such as the nature of paradigms and the causes of paradigm shifts – have been borrowed from the study of other disciplines, exemplified by the use of the ideas of Thomas Kuhn about The Structure of Scientific Revolutions to explain revolutions in thought in economics and other disciplines.

FIG 6.5: Mark Simonson's Proxima Nova feels too tight (top). With a bit more line spacing (bottom), the text block's overall balance improves.

Intellectual history includes the history of thought in many disciplines, such as the history of philosophy, and the history of economic thought. Analytical concepts – such as the nature of paradigms and the causes of paradigm shifts – have been borrowed from the study of other disciplines, exemplified by the use of the ideas of Thomas Kuhn about The Structure of Scientific Revolutions to explain revolutions in thought in economics and other disciplines.

Intellectual history includes the history of thought in many disciplines, such as the history of philosophy, and the history of economic thought. Analytical concepts – such as the nature of paradigms and the causes of paradigm shifts – have been borrowed from the study of other disciplines, exemplified by the use of the ideas of Thomas Kuhn about The Structure of Scientific Revolutions to explain revolutions in thought in economics and other disciplines.

FIG 6.6: Smaller type balances this text block too (top), but keep an eye on its x-height; if downsized type feels too small, ease off a bit and find a happy medium (bottom).

Code example: <http://bkaprt.com/ft/06-07/>

Text feels too loose

Text can feel too loose when a font's default letter spacing is tight (because the word spacing feels loose by comparison) or when the text block's line spacing is too large for its measure (**FIG 6.7**).

Loosening letter spacing, combined with tightening vertical white spaces like line height and margin, improves this text block. We could also try increasing the font size a little, or using a heavier weight of Helvetica Neue. Even with such adjustments, though, a multipurpose typeface pales in comparison with one that has been explicitly designed for text. Consider selecting a better typeface for the job (**6.8**).

Text feels unwieldy

Sometimes, type feels too large (**FIG 6.9**). There are a couple of reasons why. First, the font size is just a bit too big for this typeface. Georgia can feel horsey—exaggerated—at large sizes because it was designed for use at smaller ones. Recall from Chapter 3 that every font has size limits, beyond which it begins to look out of place.

The other issue with this example can be explained by Gestalt theory's law of proximity—I mentioned that in Chapter 4 when we shaped our text block's line spacing and margins. Because the margins that surround this particular text block are so tight, words in the text seem like they're being pulled apart. Ampler margins resolve this issue (**FIG 6.10**).

Text feels weak or dull

Text can feel dull if it has been set in a dull typeface. For reading, choose type that has an active texture (**FIG 6.11**).

Popping the font size up, choosing a different font weight, or choosing a different font can all have positive effects if you're faced with a dull text block (**FIG 6.12**).

The words in a text can also produce dullness. Some texts contain many consecutive words of similar length; like marching music, they beat beat beat. Try different values for measure and line spacing to see if you can imbue this sort of text with more energy; minor adjustments sometimes help.

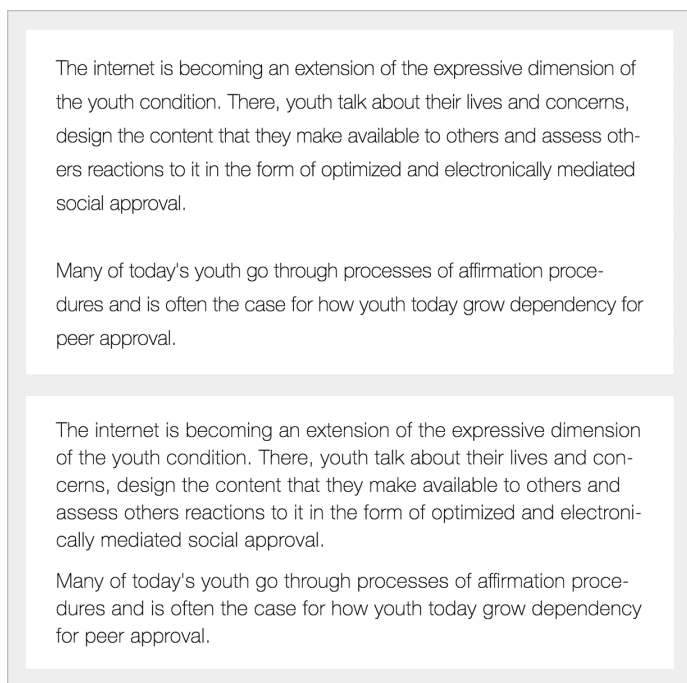


FIG 6.7: Top: tight letter spacing and large line spacing with Helvetica Neue Light. Bottom: loosened letter spacing, tighter line spacing, and smaller vertical margins.

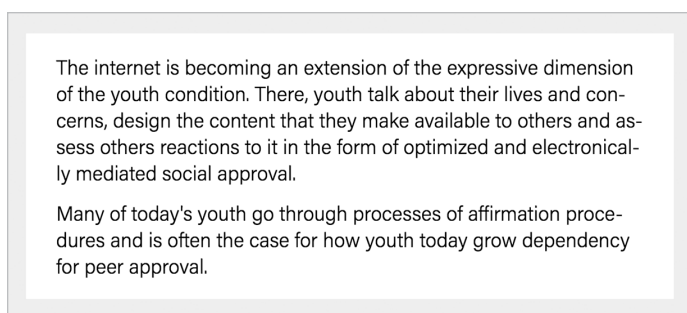


FIG 6.8: Acumin is the same general style of typeface as Helvetica, but was designed for text use; its Light weight is much more readable and balanced than Helvetica's.

Code example: <http://bkaprt.com/ft/06-08/>

In studying such transformations it is always necessary to distinguish between the material transformation of the economic conditions of production, which can be determined with the precision of natural science, and the legal, political, religious, artistic or philosophic — in short, ideological forms in which men become conscious of this conflict and fight it out.

In studying such transformations it is always necessary to distinguish between the material transformation of the economic conditions of production, which can be determined with the precision of natural science, and the legal, political, religious, artistic or philosophic — in short, ideological forms in which men become conscious of this conflict and fight it out.

FIG 6.9: Georgia, a typeface designed for small sizes on coarse-resolution screens, looks a bit unwieldy at larger text sizes. The tight margins here also throw the text block out of balance.

In studying such transformations it is always necessary to distinguish between the material transformation of the economic conditions of production, which can be determined with the precision of natural science, and the legal, political, religious, artistic or philosophic — in short, ideological forms in which men become conscious of this conflict and fight it out.

FIG 6.10: Top: Georgia at a smaller size with more appropriate margins. Bottom: Kepler, a similar-looking typeface that works better at the original, larger size.

Code example: <http://bkaprt.com/ft/06-09/>

One of the main problems when generating hypotheses on the formation and evolution of objects in the Solar System is the lack of samples that can be analysed in the laboratory, where a large suite of tools are available and the full body of knowledge derived from terrestrial geology can be brought to bear. Fortunately, direct samples from the Moon, asteroids and Mars are present on Earth, removed from their parent bodies and delivered as meteorites.

FIG 6.11: Museo Sans is a versatile type family with sturdy shapes and even color. Because its texture is not active, though, it can seem rather dull when used for body text. This example feels flat.

One of the main problems when generating hypotheses on the formation and evolution of objects in the Solar System is the lack of samples that can be analysed in the laboratory, where a large suite of tools are available and the full body of knowledge derived from terrestrial geology can be brought to bear. Fortunately, direct samples from the Moon, asteroids and Mars are present on Earth, removed from their parent bodies and delivered as meteorites.

One of the main problems when generating hypotheses on the formation and evolution of objects in the Solar System is the lack of samples that can be analysed in the laboratory, where a large suite of tools are available and the full body of knowledge derived from terrestrial geology can be brought to bear. Fortunately, direct samples from the Moon, asteroids and Mars are present on Earth, removed from their parent bodies and delivered as meteorites.

FIG 6.12: Azo Sans has a much better texture for reading. Its activity is even more pronounced at larger text sizes. Note that when size increases but container width stays the same, the text block's measure is effectively shortened—so line spacing needs to be tightened up a bit.

Code example: <http://bkaprt.com/ft/06-10/>

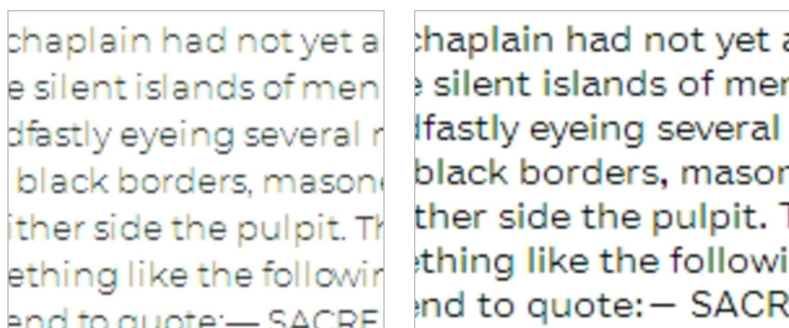


FIG 6.13: Left: Montserrat. Right: Benton Modern RE. Both of these screenshots were taken in Windows XP on a virtual machine, so the actual rendering results may differ; however, this illustrates the kind of improvement possible with a typeface designed for harsh environments.

Pressure related to type rendering

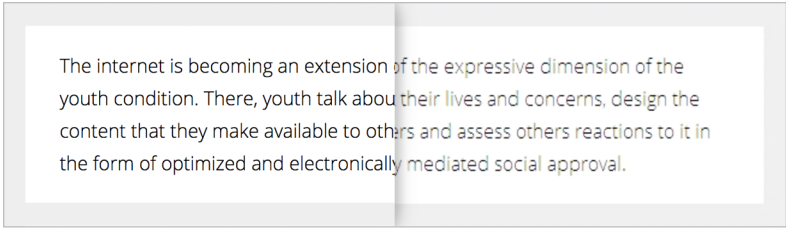
These patterns illustrate pressures caused by screens that are too coarse or too fine for the font in use. Most of the solutions involve choosing a more appropriate typeface. That means revisiting Chapter 3 and moving forward from there.

For an overview of type-rendering technologies, check out the series of blog posts I wrote for Typekit (<http://bkaprt.com/ft/06-11/>). Also see Bram Stein's *Webfont Handbook* (<http://bkaprt.com/ft/01-03/>), which has more up-to-date information.

Type renders poorly

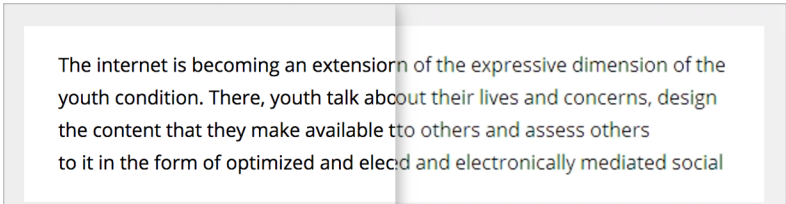
Some type renders poorly because it was neither designed nor produced for use on coarse screens in unforgiving rendering environments. Consider instead a typeface specifically made for low-resolution environments (**FIG 6.13**).

Type Network, Frere-Jones Type, Adobe, and Hoefler & Co. all offer body-text typefaces with styles specifically designed for small sizes on coarse screens—Reading Edge (RE), MicroPlus, optical sizes for captions, and ScreenSmart styles, respectively. Of course, typefaces themselves are only one part of how text gets rendered on screens.

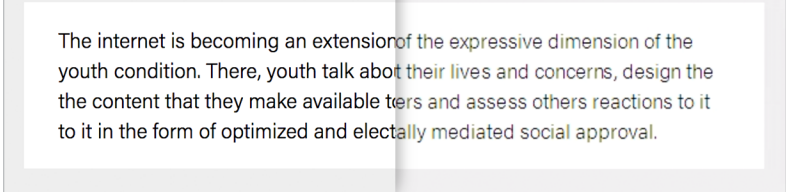


The internet is becoming an extension of the expressive dimension of the youth condition. There, youth talk about their lives and concerns, design the content that they make available to others and assess others reactions to it in the form of optimized and electronically mediated social approval.

FIG 6.14: Open Sans Light feels too light for body text. In some rendering environments, parts of letters even begin to disappear.



The internet is becoming an extension of the expressive dimension of the youth condition. There, youth talk about their lives and concerns, design the content that they make available to others and assess others reactions to it in the form of optimized and electronically mediated social approval.



The internet is becoming an extension of the expressive dimension of the youth condition. There, youth talk about their lives and concerns, design the content that they make available to others and assess others reactions to it in the form of optimized and electronically mediated social approval.

FIG 6.15: Using a heavier weight of Open Sans (top) or the light weight of a different typeface (Acumin, bottom), resolves the issue. Watch for text reflow if you try one of these adjustments.

Code example: <http://bkaprt.com/ft/06-12/>

Type feels too light

Occasionally, you may find that the typeface you’re using feels much too thin, especially in particular rendering environments. Try a heavier weight of the face, or a different (thicker) face (**FIG 6.14–15**). Increasing font size sometimes also helps—but usually if a typeface feels too light, it is too light.

This problem often occurs when display or all-purpose typefaces are used too small. Choose a good text typeface instead. If you need to refresh your memory about what constitutes a good typeface for body text, revisit Chapter 3.

Pressure related to tempo and alignment

These patterns illustrate minor compositional pressures that can be resolved by tightening up the composition a bit, or making sure elements align.

The layout feels stretched

Sometimes, even if text-block measurements and spacing are within the limits we've determined are acceptable, a layout can feel stretched—like it's a little too light on content, or a little too wide for the content it has (**FIG 6.16**).

A variety of adjustments can help. I find modular scales to be enlightening in scenarios like this—see how the layout feels using different numbers from a scale. Strengthen the contrast, if necessary, by emphasizing important content (as discussed in Chapter 5). But to start, try limiting the overall width of the layout, bumping up the font size, and adjusting white spaces like line height and margins (**FIG 6.17**). For each adjustment you make, return to its part earlier in this book—step down the metaphorical ladder—and consider every dependent decision.

Rag is rough

Sometimes a rag can look really rough (**FIG 6.18**). If that's the case, make sure hyphenation is enabled. It's not supported in every browser, but it makes a big difference when it is. Another useful tactic is to try widening the measure and giving the element a little more room in the composition (**FIG 6.19**).

There are plenty of alternative tactics, too. Try tightening the tempo (with subtle adjustments to letter and word spacing), reducing the font size, using a more condensed variation, or even switching up the typeface. Variable body-text typefaces with width, weight, and x-height (or better yet, optical size) axes

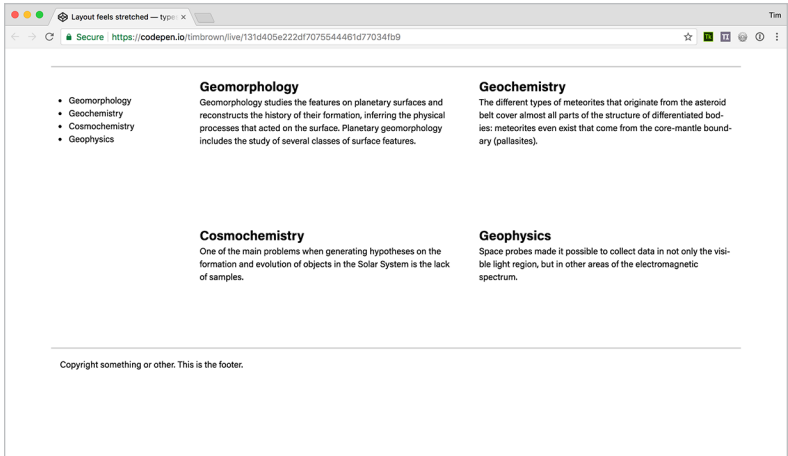


FIG 6.16: Have you ever have a layout end up looking stretched like this? I sure have.

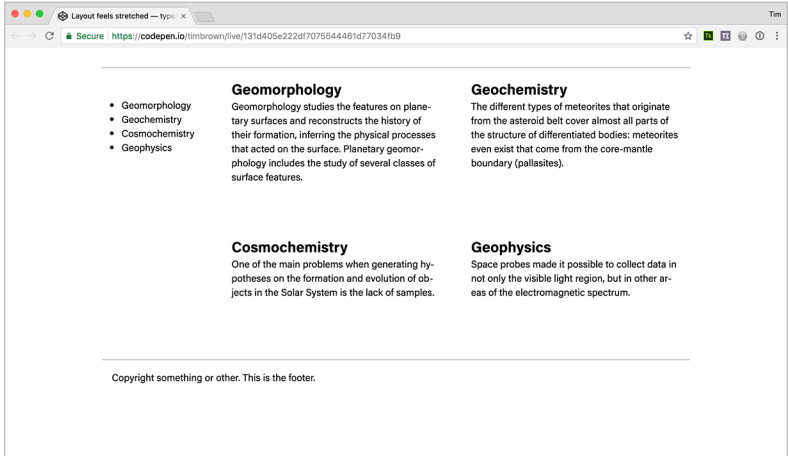


FIG 6.17: This feels a little better, but it required a trip down the ladder. (It could use a few more trips up and down, in fact.)

could help significantly here, enabling valuable refinements without requiring major changes.

The Organisation for Economic Co-operation and Development (OECD; French: Organisation de coopération et de développement économiques, OECD) is an intergovernmental economic organisation with 35 member countries, founded in 1961 to stimulate economic progress and world trade. It is a forum of countries describing themselves as committed to democracy and the market economy, providing a platform to compare policy experiences, seeking answers to common problems, identify good practices and coordinate domestic and international policies of its members.

FIG 6.18: Narrow measures, and even medium-width measures, risk rough rags if the text contains long words—especially in wider typefaces like Mallory MicroPlus.

The Organisation for Economic Co-operation and Development (OECD; French: Organisation de coopération et de développement économiques, OECD) is an intergovernmental economic organisation with 35 member countries, founded in 1961 to stimulate economic progress and world trade. It is a forum of countries describing themselves as committed to democracy and the market economy, providing a platform to compare policy experiences, seeking answers to common problems, identify good practices and coordinate domestic and international policies of its members.

The Organisation for Economic Co-operation and Development (OECD; French: Organisation de coopération et de développement économiques, OECD) is an intergovernmental economic organisation with 35 member countries, founded in 1961 to stimulate economic progress and world trade. It is a forum of countries describing themselves as committed to democracy and the market economy, providing a platform to compare policy experiences, seeking answers to common problems, identify good practices and coordinate domestic and international policies of its members.

FIG 6.19: Top: the same text with hyphenation enabled. Bottom: the same text with hyphenation enabled, as well as a reduced font size, wider measure, looser line spacing, and broader horizontal margins—it often takes some work, and some difficult decision making, to smooth out a rough rag.

Code example: <http://bkaprt.com/ft/06-13/>

The Organisation for Economic Co-operation and Development (OECD; French: Organisation de coopération et de développement économiques, OECD) is an intergovernmental economic organisation with 35 member countries, founded in 1961 to stimulate economic progress and world trade. It is a forum of countries describing themselves as committed to democracy and the market economy, providing a platform to compare policy experiences, seeking answers to common problems, identify good practices and coordinate domestic and international policies of its members.

FIG 6.20: Without any kind of hyphenation, full-justified text can cause all sorts of uncomfortable situations in a text block, from absurdly large word spaces, to congregations of large word spaces (see “*It is a*”), to what typographers call rivers—obvious word spaces that happen to align, appearing to make a vertical path through the text.

Gappy text

Full-justified text is usually rife with uncomfortable gaps, especially if hyphenation isn’t enabled (**FIG 6.20**). Set text flush-left if you can; if you must fully justify it, however, try the same tips that helped with a rough rag: tighten the tempo, reduce the font size, or use a condensed variation (**FIG 6.21**).

Misalignment of text-block areas

Grid layout spacing can sometimes leave text blocks slightly misaligned, so watch out for that (**FIG 6.22**).

A similar problem can happen to a sequence of text blocks in which some have a background color field and others don’t (**FIG 6.23**). Make sure the text blocks themselves line up.

The Organisation for Economic Co-operation and Development (OECD; French: Organisation de coopération et de développement économiques, OECD) is an intergovernmental economic organisation with 35 member countries, founded in 1961 to stimulate economic progress and world trade. It is a forum of countries describing themselves as committed to democracy and the market economy, providing a platform to compare policy experiences, seeking answers to common problems, identify good practices and coordinate domestic and international policies of its members.

The Organisation for Economic Co-operation and Development (OECD; French: Organisation de coopération et de développement économiques, OECD) is an intergovernmental economic organisation with 35 member countries, founded in 1961 to stimulate economic progress and world trade. It is a forum of countries describing themselves as committed to democracy and the market economy, providing a platform to compare policy experiences, seeking answers to common problems, identify good practices and coordinate domestic and international policies of its members.

FIG 6.21: Top: Acumin, full-justified and hyphenated. Bottom: the same text block, but set in Acumin Semicondensed—one of several narrow widths in the Acumin family. If you squint, you'll notice that the Semicondensed paragraph produces a lighter color, but also has more evenness than the regular-width Acumin. Note, too, that some of the words in both of these paragraphs are hyphenated at awkward spots; this is a shortcoming of hyphenation algorithms on the web today.

Code example: <http://bkaprt.com/ft/06-14/>

Misalignment of headings and text

Headings often need to be outdented slightly for optical alignment with the text that follows them, especially in high-contrast layouts with larger headings (**FIG 6.24**). Use a small amount of negative `text-indent` to get this right. Don't precisely align edges or stems of letters in the heading with the left edge of the text block—just eyeball it. You'll usually find a sweet spot somewhere between a precise stem alignment and the default (metric) alignment.

<p>job market success than children whose takes place in schools where resources a</p> <p>For example, children who attend school pupil–teacher ratio and a better educate</p> <p>staff appear to earn higher wages as adu</p> <p>dren who attend poorer schools.</p>	<p>job market success than children whose takes place in schools where resources a</p> <p>For example, children who attend school pupil–teacher ratio and a better educate</p> <p>staff appear to earn higher wages as adu</p> <p>dren who attend poorer schools.</p>
<p>Smaller classes are widely believed to pupils because of individual attention f</p> <p>and low-attaining pupils are seen to be</p> <p>the secondary school level, where the</p> <p>is more challenging. Pupils in large cla</p> <p>task because of too much instruction f</p>	<p>Smaller classes are widely believed to be pupils because of individual attention fro</p> <p>and low-attaining pupils are seen to bene</p> <p>the secondary school level, where the co</p> <p>more challenging. Pupils in large classes</p> <p>because of too much instruction from the</p>

FIG 6.22: Left: misaligned text blocks. Right: alignment corrected. See the left edges of the text blocks? Whatever else is happening in your layout, try to make text align like this. It makes everything look orderly rather than sloppy.

<p>a class results in a diverse field of stud</p> <p>varying degrees of learning ability. Con</p> <p>ly, the class will spend time for less ac</p> <p>students to assimilate the information, v</p> <p>that time could be better spent progres</p> <p>through the curriculum. In this way, stu</p> <p>teacher ratios are compelling argument</p> <p>vanced or honors classes.</p>	<p>ing to education. Also, too many stud</p> <p>results in a diverse field of students, wi</p> <p>degrees of learning ability. Consequen</p> <p>will spend time for less academic stude</p> <p>similate the information, when that time</p> <p>better spent progressing through the c</p> <p>this way, student–teacher ratios are co</p> <p>guments for advanced or honors classe</p>
<p>Many analysts have found that extra schoo</p> <p>play a negligible role in improving student</p> <p>ment while children are in school. Yet many</p> <p>mists have gathered data showing that stu</p> <p>attend well-endowed schools grow up to e</p> <p>job market success than children whose ec</p>	<p>Many analysts have found that extra sc</p> <p>play a negligible role in improving stud</p> <p>ment while children are in school. Yet n</p> <p>mists have gathered data showing that</p> <p>attend well-endowed schools grow up</p> <p>job market success than children whos</p>

FIG 6.23: Left: the text block in the gray box is not aligned with the text block below it. Right: alignment corrected. This fix simply takes some negative margins, but remember two things: adjust the width of the text in the gray box—it should *grow* left, not *move* left; also, make sure your layout’s margins and nearby content can handle the outdenting of boxes like this.

Code example: <http://bkaprt.com/ft/06-15/>

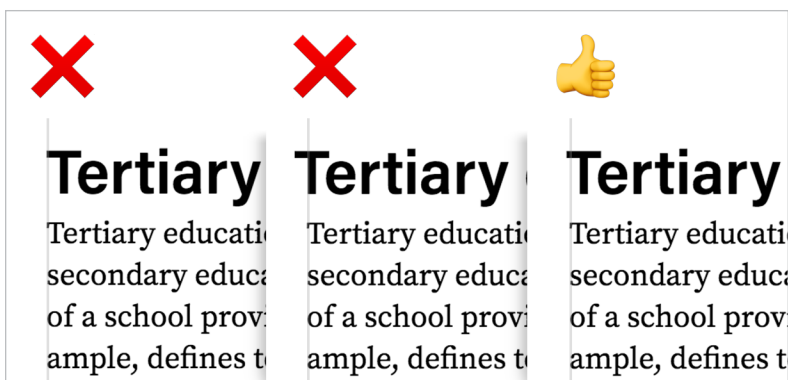


FIG 6.24: From left to right: default alignment (not good), stem alignment (not good), and optical alignment (good).

Code example: <http://bkaprt.com/ft/06-16/>

Pressure related to compositional space

These patterns illustrate major compositional pressures that often require significant adjustments to resolve—and some of those adjustments fall outside the scope of typesetting. Sometimes the graphic direction of a layout applies pressure that type can’t handle without bigger-picture changes. In these cases, typesetting adjustments are more about mitigating pressure than resolving it.

Layout feels empty

Simpler things like blogs, as well as longer texts on larger sites, are often in situations where a single column of text needs to be presented for reading. Because there’s little else in the composition, big empty spaces dominate (**FIG 6.25**).

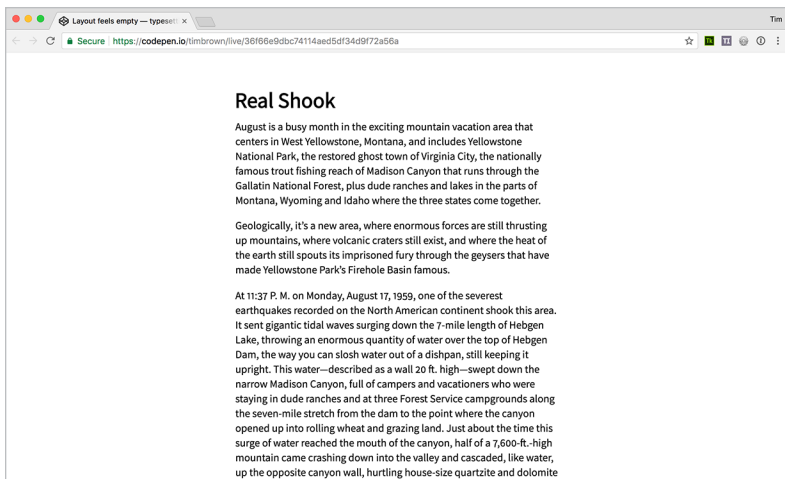


FIG 6.25: Pressure occurs in this layout because the text was designed for reading, but the viewport is a wide-open space. This little text block is all alone in the void.

In some ways, this emptiness is good for the reading experience—fewer distractions, right? But it can also feel *too* empty. Bumping up the body-text font size is one way to fill some of that empty space but, as we discussed in Chapter 5, that’s not ideal. Try widening text blocks, slowing their tempo, or using wider variations of your text typeface (**FIG 6.26**). If you’re using a variable font with a width axis, try increasing its width a bit. Consider multiple columns. Try a slight **font-size** adjustment, up or down, in combination with these other potential remedies.

Graphic adjustments like borders and background colors can help make the space feel comfortable and quiet, rather than empty. Their presence divides large empty spaces into smaller ones; measured with sensitivity, these graphic elements provide a connection between the text block’s immediate margins and overall compositional white space (**FIG 6.27**).

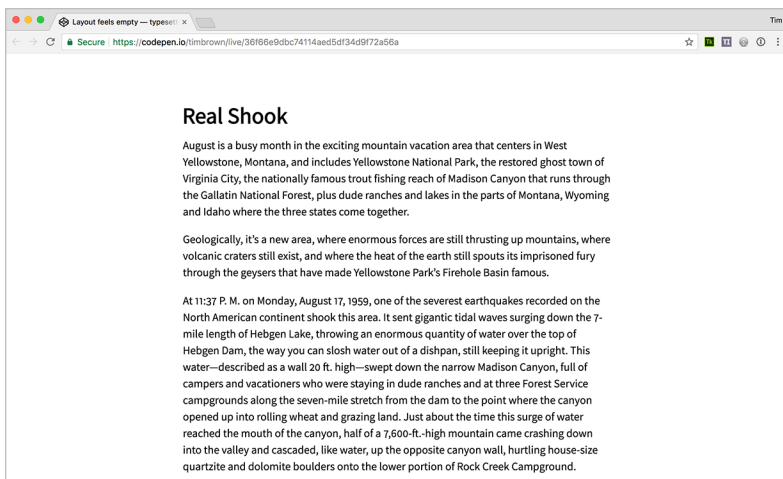


FIG 6.26: Slight adjustments can help the text block better fit the viewport space. Here, I've increased font-size by a small amount, widened the measure, and loosened the line spacing.

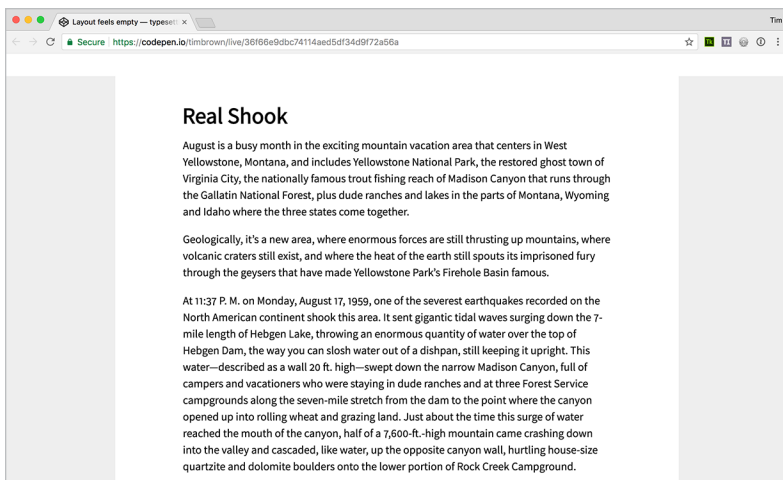


FIG 6.27: Incorporating a background-color or a border can cut down on large amounts of white space, separating the marginal area of the text block from the peripheral background area.

Importance	Physical origin	Importance	Physical
Magnetosphere		Magnetosphere	
Time dependence	Earth's core and	Time dependence	
Measurement and analysis	Numerical models	Measurement and analysis	Earth's core
see also	magnetohydrodynamic	See also	Numerical models
Geomagnetic jerk	Simulating the geomagnetic	Geomagnetic jerk	the magnetohydrodynamic
Geomagnetic latitude	ear partial differential equation	Geomagnetic latitude	Simulating the geomagnetic
History of geomagnetism	(MHD) of the Earth's core	History of geomagnetism	linear partial differential equation
Magnetic field of	formed on a 3D grid	Magnetic field of	(MHD) of the Earth's core
			performed on a

FIG 6.28: Left: an oppressive layout. Right: salvation! Many of the lessons in this book have been brought to bear in this before-and-after. The type palette has changed, from Abril Fatface to Abril Display; the font weights have been adjusted; sizes and spacing have been tweaked; and white space has been managed better.

Code example (before): <http://bkaprt.com/ft/06-17/>

Code example (after): <http://bkaprt.com/ft/06-18/>

Layout feels crowded or oppressive

Too many elements, packed too tightly, with unclear signals about hierarchy add up to a recipe for reader discomfort (**FIG 6.28**).

If you can, spread text blocks out and loosen up their spacing. Otherwise, try to cope with noisy, cramped quarters by strengthening contrast and aiming for a faster tempo. Situations like these often call for a reassessment of the typographic palette—a more consistent heading hierarchy could make an enormous difference.

Composition feels jammed up or top-heavy

Sometimes there's too much hierarchy in too little space. When a text has many levels of headings close together, you want just the right amount of contrast (**FIG 6.29-30**).

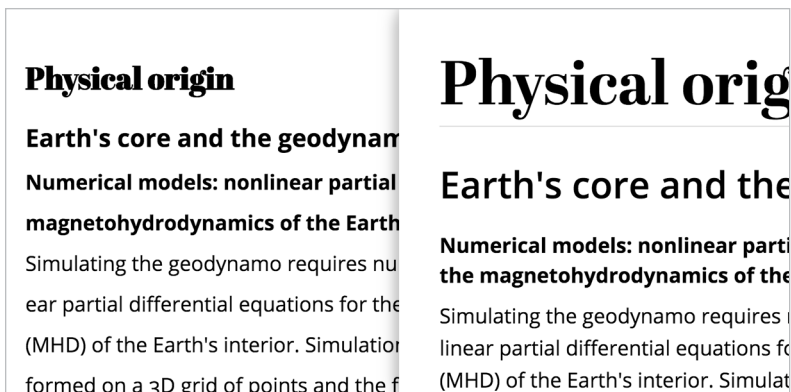


FIG 6.29: Left: not enough contrast. Right: better contrast. Line spacing plays a big role in this improvement, and so do clear differences in subhead sizes.

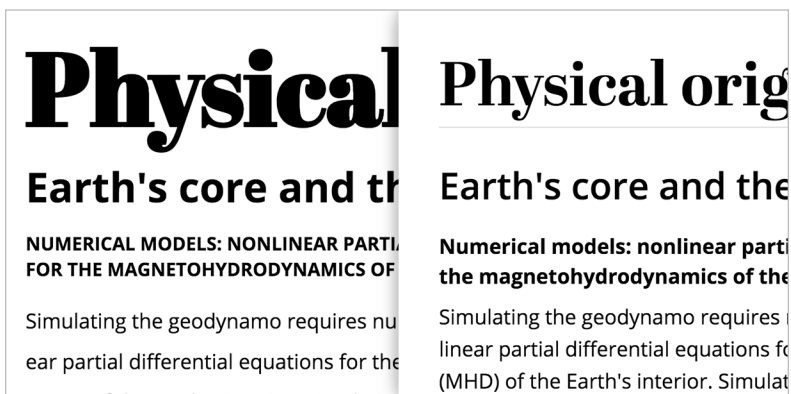


FIG 6.30: Left: too much contrast. Right: better contrast. Overpowering headings are tough to handle with subheads nearby; the main heading is so dominant that even strong subheads feel weak. Beefing up subheads in response leads to what we see on the left—a hierarchy that is just too loud.

Getting contrast and hierarchy right takes patience and practice. If you're not sure where to begin, work upward from the smallest subhead. Make that level the same font size as the text, but bold, and with tighter line spacing. Then, move up another heading level and change as few styles as possible—see how little will work, and do just enough. Try sizing the type with numbers from a modular scale.

Distracting forms

Chunks of text and other pieces of content can end up close to one another in unexpected ways, as layouts flex throughout a compositional continuum. Watch out for strange empty spots, interior text-block spacing that is too generous, and distracting tightness among text blocks. Nathan Ford wrote more about these specific problems in his *A List Apart* article “Content-out Layout” (<http://bkaprt.com/ft/06-19/>).

Pay special attention to the relationships between interior and exterior white spaces. In a busy layout, generous vertical margins with tight horizontal margins can cause confusion. And watch out for loud supplementary text or graphic elements that interfere with the reading experience by competing with the text's hierarchy (**FIG 6.31–32**).

Clashing textures

Sometimes after massaging a composition for a while—subtly adjusting text blocks to find the right levels of contrast and hierarchy—you can end up with textures that don't feel right together (**FIG 6.33**). If text blocks seem to clash, reassess the typographic palette (see Chapter 3). Consider any text block adjustments (see Chapter 4) and hierarchical tweaks (see Chapter 5) that follow from new type choices.

Small text that is too active can create pockets of liveliness that distract readers (**FIG 6.34**). Consider suppressing the activity by slowing tempo, increasing margins a bit, or using color to make distracting text blocks recede. Make sure such text blocks use a less active typeface.

geodynamo

linear partial differential equations for the

of the Earth's interior

requires numerically solving a set of nonlin-

equations for the magnetohydrodynamics

Simulation of the MHD equations is per-

formance and the fineness of the grid, which in

Announcements

Intellectual history includes the history of thought in many disciplines, such as the history of philosophy, and the history of economic thought.

Van Allen radiation belts

A zone of energetic charged particles, most of which originate from the solar wind.

Request info

Intellectual history includes the history of thought in many disciplines, such as the his-

FIG 6.31: The script subheads in this sidebar are distracting for a few reasons: they and the paragraphs that follow them are tightly spaced, they are extremely close to the main text, and the strength of the subheads conflicts with the main text's subheads.

origin

and the geodynamo

linear partial differential equations for
mics of the Earth's interior

no requires numerically solving a set of non-

equations for the magnetohydrodynamics

rior. Simulation of the MHD equations is

of points and the fineness of the grid, which

Announcements

Intellectual history includes the history of thought in many disciplines, such as the history of philosophy, and the history of economic thought.

Van Allen radiation belts

A zone of energetic charged particles, most of which originate from the solar wind.

Request info

FIG 6.32: Here, the sidebar has been adjusted. The script subheads are weaker, the measure is narrower, and the spacing is better overall. (It doesn't hurt that the main text's heading hierarchy has also improved.)

<p>If an accurate es</p> <p>needed, the me</p> <p>model used to r</p>	<p>If an accurate e</p> <p>time is needed</p> <p>el and the mod</p>	<p>If an accurate</p> <p>needed, the m</p> <p>model used to</p>
<p>Electric currents</p> <p>(ionospheric dy</p> <p>where the atmo</p> <p>that can deflect</p>	<p>Electric currents</p> <p>(ionospheric dy</p> <p>where the atmo</p> <p>that can deflect</p>	<p>Electric currents</p> <p>(ionospheric dy</p> <p>where the atmo</p> <p>that can deflect</p>

FIG 6.33: Three different subhead styles, with Open Sans for text. Left: Azo Sans may have looked promising in the palette at one point, but here we see it doesn’t work well—the combined textures of the two typefaces feel clunky. Center: Open Sans Bold is an acceptable choice, though it feels a bit boring. Right: Abril Text is a more interesting direction to pursue.

<p>Magnetosphere of Jupiter</p> <p>Magnetotellurics</p>	<p>puter</p> <p>ic dyn</p> <p>vance</p>	<p>Magnetosphere of Jupiter</p> <p>Magnetotellurics</p>	<p>cor</p> <p>kin</p> <p>che</p>
<p>Each measurement of the magnetic field is at a particular place and time.</p>	<p>namo</p> <p>and te</p> <p>first s</p> <p>id mo</p>	<p>Each measurement of the magnetic field is at a particular place and time.</p>	<p>Kir</p> <p>flo</p> <p>a d</p> <p>de</p> <p>vel</p>

FIG 6.34: Left: this sidenote is distracting because it’s too large and too tightly spaced, and it abuts the vertical rule on the right. Right: the font size has been reduced, the line spacing loosened, and a sufficient right margin added.

Becoming fluent in this language

Typesetting pressures abound. The patterns in this chapter are merely a beginning, but I hope they've shown you how to address a few key concerns and, more importantly, how to frame your responses to any problems you come up against in the future. Start by describing the pressures you see, and then try to relieve them in various ways while keeping in mind that every typesetting decision affects other decisions. Explain your successes. Share your experiences.

Together, we'll build the language we need, help others become fluent in typesetting, and foster a deeper understanding of typography in this brave new era.

CONCLUSION

I hope this book has inspired you to make typesetting a distinct part of your design process. I also hope I've helped you understand not only how the web has fundamentally changed typography, but how typography can help us understand the web. This is a very exciting moment.

It's also a very complicated moment. We can't see what we're doing, because we don't yet have the right tools. We can't describe what we're doing, because we don't yet have all the words we need. And while we're busy dealing with that, typographic traditions are fading from our collective memories.

Having gone through this book with me, you understand and can better articulate the subtle complexities in flexible design. As you use CSS layout systems like Grid and flexbox, you'll know why you should choose certain measurements. And, thanks to typesetting, you have a solid foundation for expressive work.

But beyond the practical, I hope this book inspires you. You can shape the words we use. You can influence design tools and do original work. You can return to tradition and map treasured, old advice onto this new world.

Now what?

Do some work. Try applying the guidance in this book. Let me know what doesn't go quite as planned. And don't hesitate to tell me if there's something I didn't think of.

Read some books. Bring that tradition forward. Typekit's Practice Library is a good place to start (<http://bkaprt.com/ft/07-01/>). History matters. Learn about where we came from. Read Lawson's *Anatomy of a Typeface*, Bringhurst's *A Short History of the Printed Word*, Tracy's *Letters of Credit*, and everything by Robin Kinross.

Read more books. Learn about the parts and purposes of type. Read *The Anatomy of Type* by Stephen Coles and *Why Fonts Matter* by Sarah Hyndman. Learn about how type is designed and try designing a typeface yourself. Read *How to Create Typefaces* by Cristóbal Henestrosa, Laura Meseguer, and José Scaglione. Learn about balance in text from authors like Cyrus Highsmith and Jan Tschichold. Dig further into web typography with Jason Santa Maria, Bram Stein, Richard Rutter, and Jason Pamental. Attend conferences on type and typography.

Try out new tools, like Axis-Praxis (<http://bkaprt.com/ft/07-02/>) and Typeshift (<http://bkaprt.com/ft/07-03/>). Experiment with making your own tools. That's how modularscale.com and many of the other resources mentioned in this book got started.

Talk to other people about this book. Talk to people who work on web browsers, and people who work on design software. Talk to type designers.

Join me. Typography is ours to shape.

ACKNOWLEDGEMENTS

Since A Book Apart opened its doors for business, I have admired these books. It's an honor and a privilege to have written one. My thanks to Jeffrey Zeldman for setting himself apart—for striking a helpful, caring tone in his writings; for fiercely advocating for openness, collaboration, and community on the web; and for building businesses that cement these values. Thanks to Jason Santa Maria for his humility, his visual work, and writings—which have always inspired me—and for designing this beautiful set of books. Thanks to Mandy Brown for her leadership and her encouragement, and for showing me the power of a great editor.

Thanks to Katel LeDû for her contagious enthusiasm, her patience, and her diligent work running A Book Apart. Thanks to Caren Litherland for editing me. Caren's kind words kept me going, and her critical eye brought clarity to this text. I could not have accomplished this without her support and guidance. Thanks to Ray Schwartz and Juliette Cezzar for their incredible expertise as technical editors; their empathy for readers improved this book tenfold. Thanks to Tina Lee for shepherding the earliest versions of this book. Thanks to Bram Stein for writing his *Webfont Handbook*; years ago, he and I strategized about how these two texts could harmonize, and it's exciting to see our plan in action. Thanks to Jessica Hische for writing my foreword. Jessica is well known for her beautiful lettering, but she's an equally gifted writer—helpful, straightforward, and kind. Thanks to the production crew at A Book Apart for making this book real. Thanks to Mike Monteiro and David Demaree for their advice about publishing.

Thanks to Greg Veen for our many private discussions about the nature of flexible typesetting, and for his evergreen support. Thanks additionally to the other managers in my career—Matthew Rechs, Bryan Mason, Carolyn Guyer, Megg Brown, and James Roy—for their leadership, and for shielding me from nonsense so the ideas in this book could take root. Thanks to

Jeff Veen, Ryan Carver, Andy McMillan, Robert Eerhart, Toby Malina, Marci Eversole, Vitaly Friedman, Gavin Elliott, and Jared Spool for having faith in me and giving me a platform. Thanks to the Typekit and Type teams, and to all of my friends at Adobe, for your profound care toward people who make type and practice typography.

Thanks to George Laws for opening my mind to the abstraction, balance, and systems in graphic design and type, for his mentoring, and for our breakfasts. Thanks to Arthur Hoener for introducing me to typography and giving me work opportunities that catalyzed my career. Thanks to Anne Galperin and Clif Meador for helping me think critically and find confidence. Thanks to Mr. Mahon for making me feel at home in art class. Thanks to the elementary school teachers who shaped and cared for me, including Mr. Weiss, Mr. Cafon, Ms. Oliver, Mr. Schmidt, and Ms. Hart.

Thanks to Ray Schwartz for geeking out with me about web development. Thanks to Chris Silverman for his perpetually fresh eyes and articulate feedback. Thanks to Donny Truong for his openness and hustle. Thanks to Ethan Marcotte for the invaluable discussion and his unparalleled optimism. Thanks to Bram Stein, Scott Kellum, Elliot Jay Stocks, Jake Giltsoff, Wenting Zhang, Weston Thayer, Indra Kupferschmid, Chris Coyier, Dave Rupert, Andrew Johnson, Ivan Bettger, and Tobias Frere-Jones for exchanges and collaborations that bolstered ideas in this book. Thanks to Kevin Davis and Joe Juriga; rest in peace.

Thanks to my friends in the Nice Web Type Slack for continually reminding me how much there is to learn about typography. Thanks to friends who gave me feedback on early drafts of this book: Greg Veen, Ethan Marcotte, Donny Truong, Robin Rendle, Bram Stein, and Frank Griesshammer. Thanks to writers who have inspired me over the years, especially Dan Cederholm and John Gruber.

Thanks to my friends at the St. Remy Volunteer Fire Department for their patience and understanding, and for their service to our community.

Thanks to my family for their love and patience as I concentrated on this book. Thanks Mom, for encouraging and supporting me throughout my life. Thanks Dad, for teaching me to think deeply. Thanks Mom S., for feeding and housing me while I learned to make websites. Thanks Dad S. for repeatedly asking when I would write a book (this is it). Thanks to my brothers, Greg and Ken, for being awesome-o. Shout-out to my sisters- and brothers-in-law, and my nieces and nephews. Love you all.

Thanks to my three beautiful, gifted, funny, kind, powerful, appreciative children, Kate, Julie, and Lori. Daddy and Mommy worked hard on this book, and you can work hard on things too. Make time to listen, speak, read, and write. I love you.

It is an immeasurable privilege to share my life and work with such wonderful people, and I profoundly appreciate each of you.

But my deepest appreciation, my most humble thanks, and all of my love, go to my wife Eileen. She alone truly understands what it meant for me to write this book. For years, she has tolerated burdens that would have exhausted my patience in a week if our roles had been reversed. Her loving support, encouragement, and selflessness deserve a book of their own. I will treasure her example for the rest of my life.

RESOURCES

This book will continue to grow at flexibletypesetting.com.

Here's a handful of typographic resources that will help you develop a deeper understanding of the craft. Keep practicing!

On text

- *Nicely Said*, Nicole Fenton and Kate Kiefer Lee (<http://bkaprt.com/ft/08-01/>)
- *The Elements of Content Strategy*, Erin Kissane (<http://bkaprt.com/ft/08-02/>)—especially Chapter 2.
- *Going Responsive*, Karen McGrane (<http://bkaprt.com/ft/08-03/>)—especially Chapter 4.

On code

- *HTML and CSS Web Standards Solutions*, Christopher Murphy and Nicklas Persson (<http://bkaprt.com/ft/08-04/>)
- *HTML for Web Designers*, Jeremy Keith (<http://bkaprt.com/ft/08-05/>)
- *Webfont Handbook*, Bram Stein (<http://bkaprt.com/ft/08-06/>)
- *A Tale of Two Viewports*, Peter-Paul Koch (<http://bkaprt.com/ft/08-07/>)
- *The Lengths of CSS*, Chris Coyier (<http://bkaprt.com/ft/08-08/>)

On type

- *The Anatomy of Type*, Stephen Coles (<http://bkaprt.com/ft/08-09/>)
- *Anatomy of a Typeface*, Alexander Lawson (<http://bkaprt.com/ft/08-10/>)
- *Size-specific adjustments to type designs*, Tim Ahrens and Shoko Mugikura (<http://bkaprt.com/ft/08-11/>)

On typography

- *On Web Typography*, Jason Santa Maria (<http://bkaprt.com/ft/08-12/>)

- *Inside Paragraphs*, Cyrus Highsmith (<http://bkaprt.com/ft/o8-13/>)
- *The Elements of Typographic Style*, Robert Bringhurst (<http://bkaprt.com/ft/o8-14/>)
- *The Form of the Book*, Jan Tschichold (<http://bkaprt.com/ft/o8-15/>)
- *Typekit Practice*, Adobe (<http://bkaprt.com/ft/o8-16/>)

On responsive and flexible composition

- *Responsive Design, Patterns & Principles*, Ethan Marcotte (<http://bkaprt.com/ft/o8-17/>)—especially Chapter 5.
- *Responsible Responsive Design*, Scott Jehl (<http://bkaprt.com/ft/o8-18/>)—especially Chapters 1 & 2
- *The New CSS Layout*, Rachel Andrew (<http://bkaprt.com/ft/o8-19/>)

Tools

- Chrome Developer Tools (<http://bkaprt.com/ft/o8-20/>)
- Firefox Developer Tools (<http://bkaprt.com/ft/o8-21/>)
- Modular Scale (<http://bkaprt.com/ft/o8-22/>)
- Typography.supply (<http://bkaprt.com/ft/o8-23/>)

Conferences

- Ampersand (<http://bkaprt.com/ft/o8-24/>)
- ATypI (<http://bkaprt.com/ft/o8-25/>)
- Kerning (<http://bkaprt.com/ft/o8-26/>)
- Robothon (<http://bkaprt.com/ft/o8-27/>)
- Typographics (<http://bkaprt.com/ft/o8-28/>)
- TypeCon (<http://bkaprt.com/ft/o8-29/>)

Type design education programs

- Cooper Type (<http://bkaprt.com/ft/o8-30/>)
- ÉSAD Amiens (<http://bkaprt.com/ft/o8-31/>)
- SVA Type Lab (<http://bkaprt.com/ft/o8-32/>)
- TypeMedia (<http://bkaprt.com/ft/o8-33/>)
- University of Reading (<http://bkaprt.com/ft/o8-34/>)

REFERENCES

Shortened URLs are numbered sequentially; the related long URLs are listed below for reference.

Introduction

00-01 <https://codepen.io/collection/XgmEyV/>

00-02 <https://www.google.com/chrome/>

Chapter 1

01-01 <http://www.scientificamerican.com/article/long-live-the-web/>

01-02 <http://www.fourthdimensionapp.com>

01-03 <https://abookapart.com/products/webfont-handbook>

01-04 <https://abookapart.com/products/responsible-responsive-design>

01-05 <https://www.webstandards.org>

01-06 <https://aneventapart.com/news/post/jeff-veen-live-at-an-event-apart-how-the-web-works-video>

01-07 <http://nicewebtype.com/notes/2012/02/03/molten-leading-or-fluid-line-height/>

01-08 <https://css-tricks.com/molten-leading-css/>

Chapter 2

02-01 <https://codepen.io/timbrown/pen/BJmdgj?editors=1000>

02-02 <http://www.nicelysaid.co>

02-03 https://w3techs.com/technologies/history_overview/character_encoding

02-04 <https://codepen.io/timbrown/pen/EowGZb/?editors=1000>

02-05 <http://www.joelonsoftware.com/articles/Unicode.html>

02-06 <http://charcod.es>

02-07 <https://blog.codepen.io/2016/12/19/regex-find-replace-codepen>

02-08 <https://regexone.com>

02-09 <https://github.com/adamaveray/PHPTypography>

02-10 <https://blot.im/typeset/>

02-11 <https://codepen.io/timbrown/pen/dJZZvN/?editors=1100>

02-12 <https://necolas.github.io/normalize.css>

02-13 <http://meyerweb.com/eric/tools/css/reset/>

02-14 <https://codepen.io/timbrown/pen/opopOJ/?editors=1100>

- 02-15 <https://quirksmode.org/mobile/viewports2.html>
- 02-16 <https://developer.apple.com/library/content/documentation/AppleApplications/Reference/SafariWebContent/UsingtheViewport/UsingtheViewport.html>
- 02-17 <https://www.w3.org/TR/css-device-adapt-1/>
- 02-18 <https://caniuse.com/#feat=css-deviceadaptation>
- 02-19 <https://twitter.com/nicewebtype/status/773174521268465665>
- 02-20 <https://www.w3.org/TR/css3-values/#viewport-relative-lengths>
- 02-21 <https://twitter.com/nicewebtype/status/773176544634232832>
- 02-22 http://www.456bereastreet.com/archive/201011/beware_of_-webkit-text-size-adjustnone/
- 02-23 <https://typekit.com/>
- 02-24 <https://fontstand.com/>
- 02-25 <https://deardesignstudent.com/paying-for-type-7d77a5c18c97>
- 02-26 <http://trentwalton.com/2012/06/08/jiro-sushi-web-type/>
- 02-27 <https://typekit.com>
- 02-28 <https://frerejones.com/>
- 02-29 <https://www.w3.org/TR/css-fonts-3/#font-face-rule>
- 02-30 <http://stateofwebtype.com/#WOFF2>
- 02-31 <https://css-tricks.com/snippets/css/using-font-face/>
- 02-32 <https://helpx.adobe.com/typekit/using/browser-os-support.html>
- 02-33 <https://alistapart.com/article/using-webfonts>
- 02-34 <https://practice.typekit.com/lesson/caring-about-opentype-features/>

Chapter 3

- 03-01 <https://www.responsivedesignworkflow.com>
- 03-02 <http://codepen.io/KingKabir/full/grXLyG/>
- 03-03 <https://github.com/stubbornella/type-o-matic>
- 03-04 <http://blog.typekit.com/2014/10/22/creating-a-type-style-guide/>
- 03-05 <http://gutenberg.org>
- 03-06 <http://insideparagraphs.com>
- 03-07 <https://practice.typekit.com/lesson/selecting-typefaces-for-body-text/>
- 03-08 <http://www.typtotalks.com/videos/stephen-coles-2/>
- 03-09 <https://justanotherfoundry.com/size-specific-adjustments-to-type-designs>
- 03-10 <https://medium.com/@tiro/https-medium-com-tiro-introducing-open-type-variable-fonts-12ba6cd2369>
- 03-11 <https://v-fonts.com/>
- 03-12 <https://docs.microsoft.com/en-us/typography/opentype/spec/>

- 03-13 <https://www.typenetwork.com/brochure/opentype-variable-fonts-moving-right-along>
- 03-14 <https://axis-praxis.org>
- 03-15 <http://practice.typekit.com/lesson/getting-the-most-out-of-type-specimens/>
- 03-16 <https://medium.com/@monteiro/13-ways-designers-screw-up-client-presentations-51aaee11e28c#.yf3z5lvrh>
- 03-17 <http://typographica.org/category/typeface-reviews/>
- 03-18 <http://fontsinuse.com>
- 03-19 <http://typography.supply>
- 03-20 <https://fontsinuse.com/typefaces/10819/source-sans>
- 03-21 <http://typeanatomy.com/>

Chapter 4

- 04-01 <https://abookapart.com/products/mobile-first>
- 04-02 <https://twitter.com/jontangerine/status/298722082957705216>
- 04-03 <https://blog.typekit.com/2011/11/09/type-study-sizing-the-legible-letter/>
- 04-04 <https://web.archive.org/web/20020124085641/http://www.microsoft.com/typography/fontpack/default.htm>
- 04-05 <https://codepen.io/timbrown/pen/VrbWNx/?editors=0100>
- 04-06 <http://modularscale.com>
- 04-07 <http://www.modularscale.com/?1.15&em&1.667>
- 04-08 https://en.wikipedia.org/wiki/Canons_of_page_construction
- 04-09 <https://frerejones.com/blog/typeface-mechanics-001>
- 04-10 <https://codepen.io/timbrown/pen/LdjegB/?editors=0100>
- 04-11 <http://meyerweb.com/eric/thoughts/2006/02/08/unitless-line-heights>
- 04-12 <https://blog.typekit.com/2016/08/17/flexible-typography-with-css-locks/>
- 04-13 <https://csswizardry.com/2012/06/single-direction-margin-declarations/>
- 04-14 <https://codepen.io/timbrown/pen/xppXGz/?editors=0100>
- 04-15 <http://artequalswork.com/posts/better-css-font-stacks/>
- 04-16 <https://www.w3.org/TR/css-fonts-3/#generic-font-families>
- 04-17 <http://fontfamily.io/sans-serif>
- 04-18 <https://helpx.adobe.com/typekit/using/font-events.html>
- 04-19 <https://fontfaceobserver.com>
- 04-20 <http://caniuse.com/#feat=font-loading>
- 04-21 <https://abookapart.com/products/webfont-handbook>
- 04-22 <https://codepen.io/timbrown/pen/JMMpGB?editors=0100>
- 04-23 <https://codepen.io/timbrown/pen/EEvOWx?editors=0100>

- 04-24 <https://codepen.io/timbrown/pen/LdjXBb?editors=0100>
- 04-25 <https://www.w3.org/StyleSheets/Core/preview.html>
- 04-26 <https://caniuse.com/#search=font-variant>
- 04-27 <https://www.microsoft.com/typography/otspec/featurelist.htm>
- 04-28 <https://helpx.adobe.com/typekit/using/use-open-type-features.html>
- 04-29 <https://www.w3.org/TR/WCAG20/>
- 04-30 <https://medium.com/vassar-design/links-with-underlines-as-a-best-practice-fe1ba0d6ba71>
- 04-31 <http://underlinejs.org/>
- 04-32 <https://medium.design/crafting-link-underlines-on-medium-7c03a9274f9>
- 04-33 <https://github.com/wentin/underlineJS>
- 04-34 <https://drafts.csswg.org/css-text-decor-4/>
- 04-35 <https://css-tricks.com/styling-underlines-web/>
- 04-36 <https://github.com/bramstein/typeset/>
- 04-37 <https://codepen.io/timbrown/pen/dmzwNd/?editors=0100>
- 04-38 <https://codepen.io/timbrown/pen/vRjvZa?editors=0100>

Chapter 5

- 05-01 <https://codepen.io/timbrown/pen/qoXvMa?editors=0100>
- 05-02 <https://css-tricks.com/bookmarklet-colorize-text-45-75-characters-line-length-testing/>
- 05-03 <https://www.axis-praxis.org/specimens/grade>
- 05-04 <https://www.w3.org/TR/css3-multicol/>
- 05-05 <https://codepen.io/timbrown/pen/EEWYPz?editors=0100>
- 05-06 <https://codepen.io/timbrown/pen/ZxXLaz?editors=0100>
- 05-07 <https://csswizardry.com/2012/02/pragmatic-practical-font-sizing-in-css/>
- 05-08 <http://www.stubbornella.org/content/2011/09/06/style-headings-using-html5-sections/>
- 05-09 <https://www.w3.org/TR/2008/REC-WCAG20-20081211/#visual-audio-contrast-contrast>
- 05-10 <http://leaverou.github.io/contrast-ratio/#%23777-on-white>
- 05-11 <https://abookapart.com/products/sass-for-web-designers>
- 05-12 <https://css-tricks.com/naming-media-queries/>
- 05-13 <https://codepen.io/nwalton3/pen/xvnHy>
- 05-14 <https://cloudfour.com/thinks/the-ems-have-it-proportional-media-queries-ftw/#comment-706>
- 05-15 <https://ethanmarcotte.com/wrote/on-container-queries/>
- 05-16 <https://abookapart.com/products/the-new-css-layout>

- 05-17 <https://drafts.csswg.org/css-rhythm/>
- 05-18 <https://twitter.com/jensimmons/status/851874977439784961>

Chapter 6

- 06-01 <https://codepen.io/timbrown/live/ZxXLaZ>
- 06-02 <https://blog.typekit.com/2010/10/15/type-rendering-operating-systems/>
- 06-03 <https://css-tricks.com/throttling-the-network/>
- 06-04 <https://www.charlesproxy.com>
- 06-05 <http://flippant.artequalswork.com/>
- 06-06 <http://min.artequalswork.com/>
- 06-07 <https://codepen.io/timbrown/pen/wmyQrM?editors=1100>
- 06-08 <https://codepen.io/timbrown/pen/dmdLQp?editors=1100>
- 06-09 <https://codepen.io/timbrown/pen/pLaXjE?editors=1100>
- 06-10 <https://codepen.io/timbrown/pen/aYYOOd?editors=1100>
- 06-11 <https://blog.typekit.com/2010/10/05/type-rendering-on-the-web/>
- 06-12 <https://codepen.io/timbrown/pen/dmmGZY?editors=1100>
- 06-13 <https://codepen.io/timbrown/pen/wmmGQZ?editors=1100>
- 06-14 <https://codepen.io/timbrown/pen/jzzrEb?editors=1100>
- 06-15 <https://codepen.io/timbrown/pen/QmmEXg?editors=1100>
- 06-16 <https://codepen.io/timbrown/pen/GxxjMr?editors=1100>
- 06-17 <https://codepen.io/timbrown/pen/PRRbdx?editors=1100>
- 06-18 <https://codepen.io/timbrown/pen/MVVOrm?editors=1100>
- 06-19 <https://alistapart.com/article/content-out-layout>

Conclusion

- 07-01 <https://practice.typekit.com/library/>
- 07-02 <https://www.axis-praxis.org>
- 07-03 <http://typeshiftapp.com>

Resources

- 08-01 <http://www.nicelysaid.co>
- 08-02 <https://abookapart.com/products/the-elements-of-content-strategy>
- 08-03 <https://abookapart.com/products/going-responsive>
- 08-04 <https://www.worldcat.org/title/html-and-css-web-standards-solutions-a-web-standardistas-approach/oclc/929009843>

- 08-05 <https://html5forwebdesigners.com>
- 08-06 <https://abookapart.com/products/webfont-handbook>
- 08-07 <https://quirksmode.org/mobile/viewports2.html>
- 08-08 <https://css-tricks.com/the-lengths-of-css/>
- 08-09 <http://typeanatomy.com/>
- 08-10 <https://www.worldcat.org/title/anatomy-of-a-typeface/oclc/552996220>
- 08-11 <https://justanotherfoundry.com/size-specific-adjustments-to-type-designs>
- 08-12 <https://abookapart.com/products/on-web-typography>
- 08-13 <http://insideparagraphs.com>
- 08-14 <https://www.worldcat.org/title/elements-of-typographic-style/oclc/967670384>
- 08-15 <https://www.worldcat.org/title/form-of-the-book-essays-on-the-morality-of-good-design/oclc/813259460>
- 08-16 <http://practice.typekit.com/>
- 08-17 <https://abookapart.com/products/responsive-design-patterns-principles>
- 08-18 <https://abookapart.com/products/responsible-responsive-design>
- 08-19 <https://abookapart.com/products/the-new-css-layout>
- 08-20 <https://developer.chrome.com/devtools>
- 08-21 <https://developer.mozilla.org/en-US/docs/Tools>
- 08-22 <http://modularscale.com>
- 08-23 <http://typography.supply>
- 08-24 <https://ampersandconf.com>
- 08-25 <https://www.atypi.org>
- 08-26 <http://kerning.it>
- 08-27 <http://typemedia.org/robothon>
- 08-28 <http://typographics.com>
- 08-29 <http://www.typecon.com>
- 08-30 <http://postdiplome.esad-amiens.fr/>
- 08-31 <http://coopertype.org>
- 08-32 <http://typography.sva.edu>
- 08-33 <http://typemedia.org/>
- 08-34 <http://typefacedesign.net>

INDEX

A

Ahrens, Tim 73
alignment 190-195
anchor typeface 64
Andrew, Rachel 170
arranging 8
automation 36
axes
 foundry-defined 74
 registered 73

B

Beier, Sofie 67
Berlow, David 74
Berners-Lee, Tim 15
box model 146
breakpoints 148-150, 168-170
Bringhurst, Robert 33-34, 101
Brown, Mandy 153

C

calibrating 8
Caneso, Mark 7
Carter, Matthew 95
Cederholm, Dan 166
Cezzar, Juliette 105
characters 29-30
character string 29
Coles, Stephen 71, 84
composition 12, 175-181
container queries 169
Content Delivery Network (CDN) 50
contrast 164-165
copyfitting 98
CSS Device Adaptation 42
CSS Rhythmic Sizing 172
custom styles 119-121

D

Dair, Carl 164
Dowding, Geoffrey 34

E

em box, 12
ems
encoding 30
End User License Agreement (EULA)
 50

F

fallback fonts 115-123
Fenton, Nicole 26
font metrics 94-98
font size
 base 43
 changing 145-146
 CSS-specified 12
 pixel-based 45
Ford, Nathan 117, 180, 201
Frere-Jones, Tobias 73, 103

G

glyph 12-13
Green, KC 22
Grid 145, 170
grids 170-173

H

Hay, Stephen 57
Hendel, Richard 55
hierarchy 159-165
Highsmith, Cyrus 67, 113
Hochuli, Jost 33
HTML Element Sampler 126
Hudson, John 73
Hunt, Paul D. 65
hyphenation 138-141

J

Jameson, John 138
Jarociński, Michał 66
Jehl, Scott 21
justification 138–141

K

Kiefer Lee, Kate 26
Knuth-Plass line-breaking algorithm
141
Koch, Peter-Paul 42

L

layout 146–153
Leatherman, Zach 117
line-length limits 102–105
line spacing 106–109

M

Marcotte, Ethan 43, 97, 170, 173, 176
margins 111–112
measure 12
measurement 121–123
Meyer, Eric 38
modular scale 99
Mugikura, Shoko 73
multidimensional 16–17

O

OpenType 35, 54, 72
features 128–131

P

percentages 46
performance 20–21
budget 182
presentational 19
pressure 5, 13–15, 176–179
Project Gutenberg 67
proportional oldstyle figures 72
Proxy, Charles 179

R

regular expressions (regex, grep) 36
rems 45–46
reset stylesheet 38
Rickner, Tom 74
Roberts, Harry 163
Robertson, Susan 64
Rutter, Richard 43

S

Santa Maria, Jason 132, 140, 171, 174
Schwartz, Ray 136
Seltzer, Aura 75
setting type 8
Simmons, Jen 172
Simonson, Mark 183
Slimbach, Robert 69, 129
special strings 34–35
Spiekermann, Erik 69
Stein, Bram 21, 120, 141, 188
Stössinger, Nina 129
structural pseudo-class 43
structure 20, 27
Sullivan, Nicole 64, 163

T

tempo 190–195
text block 12
text-size-adjust 46
texture 71
Tschichold, Jan 34
Twardoch, Adam 154
type designer 10
typeface 91–92
Typekit's Practice Library 205
type palette 78–84
type rendering 188–189
typographer 10
typographic relationship 10–12

U

unicode 30
user-interface (UI) 60

V

van Blokland, Erik 74
variable font 73, 123
variables 166-168
Veen, Jeff 23
Verou, Lea 163-164
viewport 13
 settings 42
 units 45

W

Walton, Trent 48
Web Content Accessibility Guidelines
 (WCAG) 136
 level AAA guidelines 163-164
webfonts 47-48
web standards 22-23
White, Alex 89
white space 110-115
Wichary, Marcin 137
WOFF, WOFF2 50
Wroblewski, Luke 87

Z

Zhang, Wenting 137

ABOUT A BOOK APART

We cover the emerging and essential topics in web design and development with style, clarity, and above all, brevity—because working designer-developers can’t afford to waste time.

COLOPHON

The text is set in FF Yoga and its companion, FF Yoga Sans, both by Xavier Dupré. Headlines and cover are set in Titling Gothic by David Berlow.

ABOUT THE AUTHOR



Tim Brown is a designer, writer, speaker, and tool-maker who focuses on typography. As head of typography at Adobe, he thinks about product direction, designs software, advises integration partners, and helps novices and experts alike hone their typographic skills.

Tim speaks about the evolving craft of typography—and the immense business value of great typography—at conferences both worldwide and online. He lives and works in New York State’s beautiful Hudson Valley with his wife and college sweetheart, Eileen, their three daughters, and their dogs.