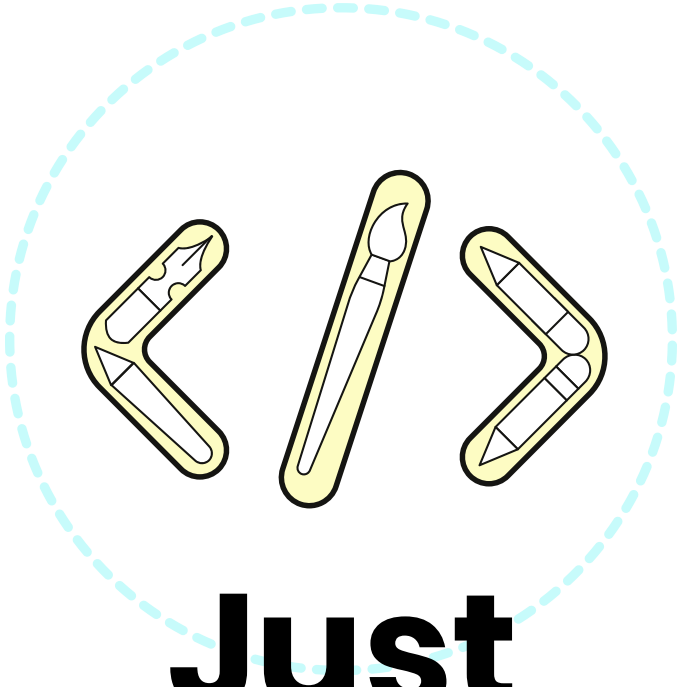


Just Enough Code.

Learn just enough code to be dangerous(ly creative).

HTML & CSS & DevTools

For designers and everyone who works on websites.

a free eBook from

mod&dot

Sponsored by https://mod-dot.com

Written by Jarrod Drysdale

<table/> of #contents

Chapter 1

HTML

On many teams, there is an indispensable person who bridges the gap between designers and developers. Someone who can step into tricky situations when a design is being built but it's not quite right yet.

That phase of projects is infamous for causing conflict. For a while, I was frustrated with developers who didn't want to build my designs the way I created them. I got so tired of it, I learned to code.

And something unexpected happened: as my coding knowledge increased I found it easier to work with developers. I still don't know if it was that devs took it as a gesture of mutual respect that I knew code or that we finally had a way to understand one another since I learned a bit of their language. (Pun intended!) But I saw a major shift in how developers responded to my requests from then on.

That was 15+ years ago, and I don't regret my decision to learn code one bit. To this day I take a lot of pride in my ability to sit down with developers, understand their perspective, and find a way forward together.

But is learning to code right for you? That old controversy about whether designers should code is impossible to settle. Coding can certainly aid your career growth as a designer, but there are many other ways to excel too.

However, for the more ambitious designer (are any of us not?), learning to code can unlock better opportunities for you. My ability to code gives me more control over the final details of my design. I love seeing my designs built 100% accurate. Coding them myself is the best way to see that happen. Even better, offering additional services like coding helps me to bring in bigger projects. This has proven so satisfying and worthwhile that I'm transitioning my entire design practice to focus on larger multi-disciplinary engagements.

If you're open to it, I'd like to take you along with me.

Learning to code is intimidating because everything you read assumes you want to become a career developer. But you don't need to know how to code a whole app from scratch or deploy production code to do your job as a designer.

When your goal is to manage the details of

implementation and smooth over collaboration with developers, which coding concepts you need to learn totally changes. Plus, new tools (cough) make it easier than ever to be effective as a product designer with only basic coding ability.

The rest of this ebook will teach you enough HTML/CSS to format simple text and prototype design edits in DevTools. You don't need to learn intimidating stuff like JavaScript, Git, or responsive design coding techniques yet. Instead, let's focus on building some fundamental coding knowledge from a design perspective. We'll start with the absolute basics.

My brilliant wife recently started learning to code, and seeing the learning through her eyes was a real eye opener. There is just so much knowledge and terminology. I wanted to try writing a different kind of coding lesson.

Even if you already know to code and this is too basic for you, this lesson will give you a little perspective on why this stuff is worth knowing.

HTML Sucks

The absolute worst thing about HTML is that when you start learning, it looks very simple, but when you see a real example outside of a tutorial it looks like an absolute nightmare:

```
<p>paragraph!</p>
```

Coding tutorial. It looks so easy!

```
<div id="orSomeReason"><div
class="alwaysAContainer"><p
class="headline" style="color:red;"><span
class="heavy">para</span><b
onclick="alert('BOO!')">graph</b></p></
div></div>
```

Real life. Ouch, my eeeeeyes!

That's what makes coding so hard at first. The basic tutorials are nothing like the real world.

HTML gets complex quickly because so does text. It's very rare that you'll have a passage of text that only needs basic formatting. You're a designer,

after all. You're all about managing those details. So internalize this now:

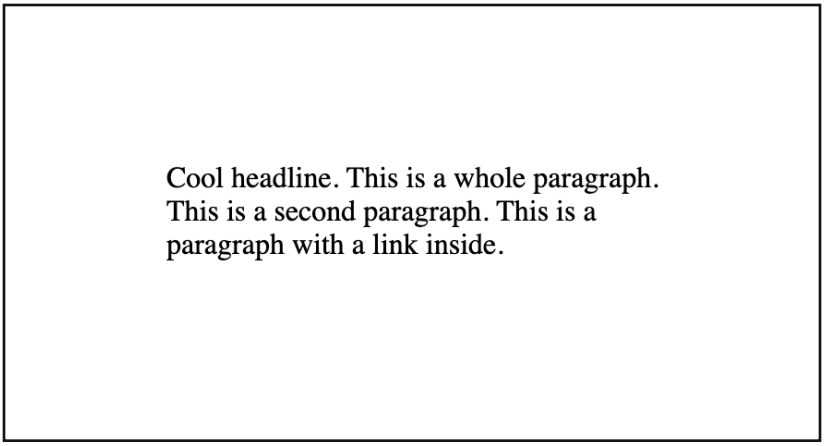
Every new design detail you create will require you to write a little bit more code.

I think that knowing this makes us designers just a little bit more respectful of developers and how hard they work to accommodate us. Anyway, on with HTML.

What is HTML, really?

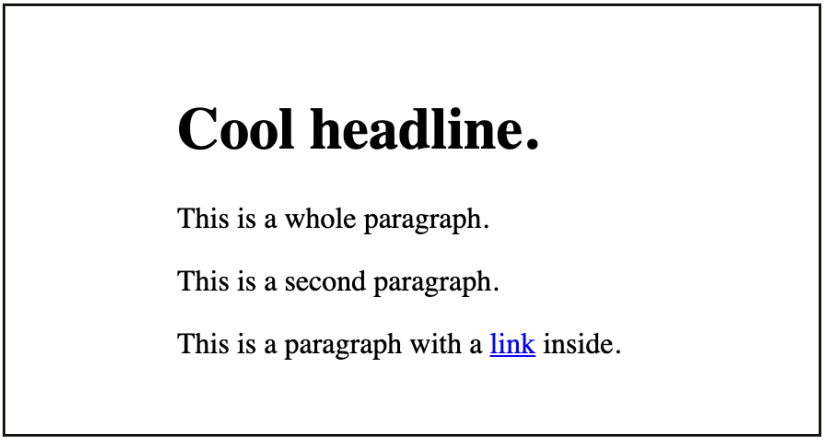
Maybe you've heard about 'tags' or seen HTML in a Wordpress text editor. Maybe you've even done some basic HTML tutorials. Most of them skip right to showing you an example to encourage you about how easy it is, but they don't stop to explain why you should learn it. Here's an example that shows why knowing HTML is worthwhile.

Without HTML, text on a website looks like this:



Well, that's confusing.

With HTML, it looks like this:



Still ugly, but much easier to understand.

HTML adds visual structure and interactivity to raw text. For example, in a design file you can create

two text boxes, one with a headline at 60px font size and the other with paragraph text at 16px font size. In HTML, you can similarly specify which part of the text is the headline and which part is the paragraph by adding your own invisible text containers right inside the text itself.

HTML doesn't let you set specific values like font sizes (you'll learn to do that too in the next article). Instead its purpose is to describe each part of the text. This is a headline. That is a paragraph. This part is a button, which can be intrinsically clicked and hovered over. And so on.

HTML is where text becomes interactive! That's actually pretty exciting!

Tags

HTML is built on the idea of adding extra text to describe the raw text you already have. Here's how it looks:

```
<p>this is a paragraph!</p>
```

Hey, that's the same crappy example from earlier!

A 'tag' is the part in between the '<' and the '>'. Tags are specific text the web browser understands. You can't make them up. They're built into every web browser, and you need to learn about the various tags that already exist so you can tell the browser how the text should appear.

Most tags surround text by providing an opening and closing tag. For a paragraph (shown above) the opening tag `<p>` says "the paragraph starts here". The closing `</p>` tag always has a slash inside, and it means "the paragraph ends here". Everything in between the opening and closing tags is a paragraph. You can put almost anything you want in between: words, spaces, sentences, etc. The same is true for a button `<button>`, a link `<a>`, a headline `<h1>`, and many other elements.

Your job as a coder is to think about which tags accurately describe the text you have. Think about the meaning of the text. Choose a tag that most accurately describes what a section of text is/does and how people should perceive it.

When you write tags, there is a small catch about

what you can put in between them. Certain text characters are reserved for the HTML syntax, and if you use them, it will break the HTML and confuse the browser. For example, notice how tags are written using angle brackets, '<' and '>'. What if your raw text has those characters?

```
<p> is 5 < or > 2?</p>
```

this won't work!

The browser will see the '<' and '>' and think you're trying to write another tag. But you're not. It will end up looking wrong.

So, if your text has certain special characters, you also need to replace them with special text to let the browser know it's not a tag. These are called HTML entities. Entities look like complete gibberish, which is not fun. Luckily, there are only a handful you need to worry about (you can see a list here).

An HTML entity always starts with an ampersand and ends with a semicolon. Here's how the above example looks using entities:

```
<p> is 5 &lt; or &gt; 2?</p>
```

Math is better than HTML anyway!

One more thing you need to know about writing tags. Some specific tags don't need a second closing tag, but instead are just one single self-closing tag. These tags are a bit special because they don't describe text but insert something into the text, such as a form input box or external asset like an image. Here's how a self closing tag looks:

```
<img />
```

Notice how the slash is inside the first and only tag. It closes itself. But because they don't have raw text inside, these self-closing tags need other information to work properly.

Attributes

Here's where HTML gets a little bit crazier, but stick with me because it's also the exciting part.

HTML tags can also provide additional information about the text they describe or the feature they

insert into the text. This additional information is called an 'attribute'. Here's how it looks:

```
<img src="smileyface.gif" />
```

The first self-closing tag you saw above wouldn't actually do anything. It inserted an image into the text without saying which image. Adding the src attribute to the image, which stands for 'source', lets you tell the browser which image to insert into that exact spot in the text.

Just like how you can only use specific tags every browser understands, there is also a list of specific attributes. It gets a little confusing at first, as certain attributes only work with certain tags. Don't expect to memorize all this. Just learn tags and attributes through practice, and you'll slowly expand your knowledge of them.

You can also use multiple attributes on the same tag, which is where it can start to look kind of crazy:

```
<a href="https://studiofellow.com"
title="the best website in the world">visit
my website</a>
```

In that example, the HTML creates a link to my website using the `<a>` opening and closing tags. The `href` attribute tells the browser where to navigate when the user clicks the link. The `title` attribute shows some extra title text when a user hovers over the link. A `title` can be placed on lots of tags, but `href` can only be used on links.

However there are a couple of attributes you can always use on any tag. Once you write a tag, HTML also lets you connect it to visual styles like font sizes and colors that you write in CSS. There are 2 special attributes for this: `id` and `class`.

```
<p id="summary">paragraph!</p>
```

An `id` gives a tag a unique name. For example, there might be a hundred paragraphs on a page, but only one is the 'summary' paragraph. If you give a tag the `id` of 'summary', you can't reuse that id on any other tag. This is a handy way of referencing that single tag on its own later in your CSS.

That said, modern convention is to avoid using ids unless absolutely necessary. This is a more advanced concern, but trust me for now that using

too many ids can lead you to having code that's more difficult to maintain. Usually, it's better to just use a class instead.

```
<p class="lead">paragraph!</p>
```

A `class` is a reusable identifier. You can put the same class on as many tags as you'd like. For example, you could make a paragraph, headline, and button all have the same class of "lead". Late in your CSS you'll write some visual styles under that class. Putting it into your HTML applies those styles to that element.

You don't need to understand ids and classes beyond this yet. For now, just realize that these are part of how HTML introduces more detailed visual design aspects into the text.

Nesting

The other thing that makes HTML look complex is nesting. Tags can be placed inside one another:

```
<p>this is a <a href=“google.com”>link</a>
inside a paragraph.</p>
```

In this example, a link is inside of a paragraph, which is obviously a very normal thing.

Nesting allows you to build up deeper structure within a web page. You can separate text into sections, forms, etc and add deeper meaning to the text by using multiple tags together.

You'll nest tags inside one another a lot! It can become difficult to track where closing tags belong when you've got a bunch of nested levels of tags. For this reason, use indents and new lines to stay organized:

```
<section>
  <p>
    this is a <a href=“google.com”>link</a>
inside a paragraph.
  </p>
</section>
```

Having opening and closing tags on new lines with

indents makes it much easier to stay organized so you don't forget to include a closing tag.

Specialized coding text editors like Atom, VS Code, and Sublime Text also make this easier by automatically adding closing tags and indents for you.

As you practice HTML, it's important to develop good habits with your indentation. Writing pretty, organized tags will help you avoid making mistakes with missing closing tags, which are a common gotcha for beginners and experts alike.

Learning Tags & Attributes: HTML reference

Over time you'll learn a bunch of tags and attributes as you practice. It's also fun to look at the source code of websites you like and see which ones they use (in a couple more lessons I'll teach you about DevTools).

But it's also extremely useful to have a reference. My favorite reference for HTML tags and attributes is the Mozilla Developer Docs because it provides huge searchable lists. Open the links below and Command+F away. When you're wondering if a certain tag exists, it's a great place to look. These lists also provide a brief description about how each item should be used and tell you which attributes can be used on which tags.

MDN Tags list

MDN Attributes list

HTML Is Awesome

Why should designers care about code? Because our designs can't exist without it. HTML turns our fancy design mockups into something real that people can use.

Understanding code allows you to make your designs real. It gives you total control over what users see and experience, right down to every detail. Which is exactly what we designers have

always wanted.

That control is waiting for you, right there in simple code. Anyone can learn it. And that's pretty exciting.

Chapter 2

CSS

Code scares off many designers because it's a deep complex topic that takes years to master. But not everyone who works on websites needs to master coding. You might not need to know how to build a whole website or app from scratch. Not everyone *should* master the intricacies and advanced topics of code.

But knowing just a little code helps you to work more closely with developers and gives you more control over how a design is built. Being able to make simple edits yourself goes a long way toward an accurate design implementation.

You just learned about basic HTML and how it can format raw text, but now it's time to learn how to apply visual design styles to that text using CSS.

For me, as a designer, CSS is the most enjoyable part of coding because it's where I get to work on the stuff that's most exciting to me—the visual stuff. Fine tuning every typographic detail and getting the margins and grid system perfect are extremely satisfying. And there's no one better to do those things than a designer who can code. Not to brag, but I'm just better at it than most developers. I have an eye for visual details, and being able to control

them directly is downright thrilling. I think you're going to love CSS because it's really a design tool at its heart.

What is CSS?

CSS stands for "Cascading Stylesheets", which is a difficult phrase for beginners to understand. The "Cascade" in CSS is a complex topic and I'm sure that initial definition alone has scared off countless people from learning it.

So, ignore that. You don't need to know what the Cascade is.

CSS is simple. Actually, stupid simple. Later on there are obviously more advanced aspects to learn, but the core of CSS is refreshing because:

CSS is just a simple way to write out visual styling information.

Paragraphs are red. Headlines use the Helvetica font at 62px size. That button should be purple.

Those are the kinds of things you can write in CSS.

And, it's not much different than writing it as a normal sentence. Here's how those styles written above might look in CSS:

```
p {
  color: red;
}
h1 {
  font-family: 'Helvetica';
  font-size: 62px;
}
button {
  background-color: purple;
}
```

That might look a little confusing, and we'll break it down in a sec. But for now, just realize that CSS is a simple way to write down visual details.

CSS requires two parts to apply a visual style: First, find part of the HTML, then second apply visual styles to that part.

Find some HTML with selectors

A selector in CSS is just some text that describes part of the HTML. Essentially, a selector says “find this part of the html”. You can write a selector that finds a bunch of HTML tags at once or that only finds one tag. Here’s what a selector looks like:

```
p {

}
```

That CSS chooses all the paragraphs in the HTML, or all of the `<p>` tags. Then, it’s followed by brackets, `{` and `}`. In between the brackets, you write the visual styles that apply to whatever that selector found (you’ll learn about writing those styles in the next section).

A selector always starts with text that matches part of the HTML, with brackets right after it. Here are more examples:

```
h1 {

}
button {

}
```

That CSS selects all `<h1>` tags, then all `<button>` tags in the HTML. The selector matches the HTML tag, with the angle brackets '<' and '>' removed.

Those selectors corresponded to HTML tags, which as you learned last time are specific tags the browser expects. But you can also write CSS to use the classes and IDs you learned how to write in your HTML.

```
.class {

}
#id {

}
```

To find a tag in the HTML by its class instead of by

the name of the tag, put a dot '.' in front of the name of the class. `.summary` selects every HTML tag with the summary class. That could include an `<h1 class="summary">`, `<p class="summary">`, and a bunch of other elements. Anything that has that `class` in the HTML will get selected.

To select IDs, use a `#` instead of a dot. An ID selector will only apply to one element because you're only allowed to use an ID once within your HTML. But, it's usually best to use a class instead of an ID. Trust me on that.

Selectors can also find a bunch of elements together so that you can apply the same styles to all of those elements even if they are different tags or don't have the same class. It's simple, just add a comma:

```
p, h1, button {

}
```

That selects all paragraphs, h1s, and buttons together. Just like writing a sentence.

The last thing about selectors is that they can

become more specific by selecting HTML tags with multiple aspects or based upon which tags they are nested inside.

You can make a more specific selector by typing the name of the HTML tag and then adding its class right after it *without* spaces. This example selects all paragraphs with the class of summary.

```
p.summary {


}
```

You can also select nested HTML tags *with* spaces like this:

```
header h1 {


}
```

That CSS selects only H1 tags that are inside header tags. The space between 'header' and 'h1' translates to 'select all h1 tags inside a header tag'. The space is important. If there is a space in a CSS selector, it's always selecting a tag within a tag.

You can even go deeper:

```
header h1 a {


}
```

That selects all `<a>` tags that are inside an `<h1>` that is itself inside a `<header>`. Notice how the last element in the selector is the one that gets picked. This selector would find the 'a' tag, not the h1 or header. The h1 and header are only referenced so we can find the 'a' tags inside them.

There are a few other methods for writing even fancier selectors you can learn later, but these are the important ones you should know for now.

By mixing and matching these techniques you can create very specific CSS that applies exactly how you want it to.

Apply visual styles with properties

Once you select some HTML, you're ready to apply visual styles to it! That's the fun part. A visual style

is called a 'property' in CSS parlance (some people call properties 'rules').

Here's how a property looks:

```
font-size: 16px;
```

A property starts with a name of the style you want, then a colon, its value, then a semicolon. Like this:

```
name: value;
```

CSS has a pre-written list of visual styles you can write, just like HTML has specific tags you can use. (Here's a list of the most common CSS properties.)

When you write a property, the name part is one of those pre-written ones the browser understands, and the value part is what you want to set it to.

Make the font size 16px and the text color red:

```
font-size: 16px;
color: red;
```

Make the font Helvetica and the margins 20px:

```
font-family: Helvetica;
margin: 20px;
```

Each property name has an expected type of value. Some accept colors, others expect a number with a unit such as a number of pixels.

To actually apply a CSS property, it must go inside a selector like this:

```
p {
  color: red;
}
```

Notice how the name and value of the property are on one line inside of the brackets of the selector. That example makes all paragraphs have a red text color. It's also best to indent the property, which makes your CSS easier to read.

You can also apply multiple properties within the same selector:

```
p {
  font-size: 16px;
  color: blue;
}
```

Now all paragraphs are blue and 16px font size.

You can write big selectors with lots of properties

too:

```
p, .summary, button, body {
  font-family: Helvetica;
  font-size: 16px;
  color: #FFFFFF;
  background-color: green;
  margin: 50px;
}
```

Browsers understand a bunch of plain text color names like `red` or `blue`, but you can also add hex color values like `#FFFFFF` or RGB values like `rgba(255,255,255,100)` (the last number being alpha transparency).

As for numbers, you can use pixel units like `24px`, percentages like `50%`, and many other advanced types of units (`em`, `rem`, `vh`, etc) to get lots of control. But at first, it's less confusing to use basic units like pixels since they translate directly to what you create in design software.

There are CSS properties for every visual style you would ever need to create. Don't expect to memorize them all. There are so many you'll probably be discovering new ones to try all the

time. I still am learning new ones after writing CSS for 15 years (just yesterday I discovered `text-decoration-color` and can't wait to try it!). And, the people who make browsers are creating new CSS properties for us all the time. It's fun to learn a new property because each one gives you a new dimension of control and a new way to explore the visual design.

How HTML & CSS Work Together

HTML provides entry points for adding visual styles. CSS adds those visual styles. Used together, they provide you with complete control over how a design is built and rendered in a browser.

Let's see an example of HTML and CSS working together.

Html:

```
<header>
  <h1>Welcome to my blog</h1>
  <p>by Jarrod</p>
</header>
<section>
  <p>My first post is about cat pics because the internet and cat pics just belong together.</p>
</section>
```

CSS:

```
h1 {
  font-size: 40px;
}
p {
 font-size: 20px;
}
h1, p {
  font-family: Hevletica;
  color: #333333;
}
```

My CSS sets the text in the HTML page to Helvetica and a dark gray color. I also set the font sizes.

Notice how I didn't need to mention the `<header>` or `<section>` tags in my CSS. I only wrote selectors for the tags I wanted to apply styles to.

Also, if I wrote CSS selectors and properties for tags that aren't in my HTML, they wouldn't do anything. That might seem obvious, but it happens a lot more often than you'd expect!

CSS can overwrite itself

CSS can overwrite itself using more specific rules and the order rules are written. Consider this HTML and CSS:

```
<p class="welcome">hi!</p>
<p>nice to meet ya.</p>
```

```
p {
  font-size: 14px;
}
.welcome {
  font-size: 20px;
}
```

Here I've written CSS that conflicts. The first paragraph tag has the class of 'welcome', and my CSS applies one font size to all paragraphs and a different font size to anything with the 'welcome' class.

So which font size will the first paragraph with the class of 'welcome' be? The answer is 20px.

CSS will apply the last rule written or the most specific rule written. In this case, browsers consider a class selector to be more specific than a tag name selector. The `.welcome` CSS was also written second. So the welcome class styles overwrite the paragraph styles.

This might seem confusing, but it's incredibly useful. You'd never want to take all the time

needed to write a separate CSS selector for every paragraph on a long page. It would take way too long. But, if some of the paragraphs are different from the rest, what do you do?

The answer is to write generic CSS that applies to all the paragraphs, then make a couple of them different by writing more specific selectors to overwrite those defaults. Here's an example. We have 100 paragraphs that all need the same font size except just 1 that needs to be bigger. I could write a separate CSS selector for 100 different paragraphs (and want to give up coding forever) or, I could just do this:

```
<p>first paragraph</p>
<p class="highlighted">second paragraph</p>
<p>third paragraph</p>
...
<p>one hundredth paragraph</p>
```

```
p {
  font-size: 16px;
}
p.highlighted {
  font-size: 20px;
```

```
}
```

The first CSS selector applies font sizes to all paragraphs, then we override that size on just the one paragraph that needs to be bigger by writing a more specific selector for it.

Writing default rules and overriding them allows you to write less CSS. This saves you tons of time and also makes the CSS easier to manage and edit over time.

How do you connect your HTML to your CSS?

So far you've only seen HTML and CSS written separately but I haven't showed you how to actually connect them. The connection happens in your HTML, and there are 2 ways.

First, remember that an HTML file is just a plain `.txt` text file re-saved with the `.html` file extension. You

can double click an HTML file on your computer to open it inside a web browser. A CSS file is a text file with the `.css` file extension.

Write CSS Inside Your HTML

For a browser to understand that .html file, the HTML code inside the file needs a few required tags: doctype, `<html>`, `<head>`, and `<body>`. Every HTML file must have these.

Doctype always comes first, and don't worry about it for now. Just always include it at the beginning of every HTML file. Next is the `<html>` tag, with the `<head>` and `<body>` inside it. Here's a basic requirements for a new HTML file:

```
<!DOCTYPE html>
<html>
  <head>

  </head>
  <body>

  </body>
</html>
```

The body represents all the content on the page. Up until now all the HTML you've seen would go inside the body tag.

The head includes information about the page the browser might need to know, like its title and if there is any CSS to use.

Inside the head tag you can create a tag that holds your CSS, called a style tag:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      p {
        color: red;
      }
    </style>
  </head>
  <body>
    <p>here's where all my content goes</p>
  </body>
</html>
```

Using this method, any CSS you write inside the style tag will apply to any HTML you write inside the

body tag automatically. This is commonly called ‘inline CSS’ because the CSS is contained within the HTML.

Use separate HTML and CSS files

However as your page gets longer and you end up with lots of HTML and CSS, the first method can get very long and cumbersome. So, most sites end up putting HTML and CSS in separate files.

You can link them together using a special tag inside the head element of the HTML file, called a ‘link’ tag. Here’s how it looks:

```
<!DOCTYPE html>
<html>
  <head>
    <link rel=“stylesheet” href=“style.
css”>
  </head>
  <body>
    <p>here’s where all my content goes</p>
  </body>
</html>
```

That’s much shorter! This version does the exact

same as the first method, but allows you to keep HTML and CSS organized in separate files. The link tag loads the CSS file and applies it to all the HTML tags automatically. Ignore the link's 'rel' attribute for now. And remember that the 'href' attribute needs to contain the file name of the CSS file.

(If you try this in a text editor, just make sure your HTML file and CSS file are in the same folder on your computer! If you put them in different folders, you'll need to learn about absolute and relative paths, which can be kinda confusing at first. So just remember to put them in the same folder for now.)

CSS Is Like Writing a Design

We designers love our menus and panels. Design software is just a bunch of panels and menus, dropdowns and input boxes.

But here's something crazy: did you know that all those swanky panels in Sketch and many of the

newer design apps are built with HTML and CSS?

That's right, most of those settings you use in design software correspond directly to HTML and CSS. The font dropdown in Sketch is the CSS `font-family` property. The width of a rectangle in sketch is the `width` property in CSS.

You already know way more code than you might have realized.

Writing code is just using that same knowledge you already have from design software and typing it out instead of clicking some menus. It can really be that simple.

Exciting right? HTML and CSS were created on the idea of making it easy to write a design. The format is simple because it's supposed to be doable for anyone. You don't need to know math. You don't have to understand all that crazy hacker stuff from Mr. Robot.

Code is design written. Any designer can do it. And now, so can you.

Chapter 3

DevTools

You can use the HTML & CSS you just learned to start doing exciting design work right now. Without learning all the other advanced coding stuff designers and non-technical web folks are often pressured into. I'm not going to teach you how to build a whole webpage or use version control. Only what you need to know to retain more control over the final result.

You can use code to to work on a live website using a magical design tool. It's free. It gives you the ability to redesign and edit anything on the web with only basic HTML & CSS knowledge—which you just learned.

Put on your hacker hoodie and queue up some techno. Let's hack a website in DevTools.

What DevTools Isn't

If you've ever opened DevTools accidentally you might have regretted it or thought you broke something. DevTools in any browser is an extremely dense UI with tons of features. It's

intimidating. And, it's actually called 'developer tools'. That name alone makes it seem like it should be off limits to anyone who's not a coding expert.

But DevTools, while it masquerades as a thing only meant for developers, is one of the most useful and powerful tools designers and writers have. It's also much simpler to learn than it seems at first glance. I know it's hard to look past the dense UI, but it's worthwhile, trust me.

With DevTools, you can bypass all the complex developer tooling used to create modern sites and applications. A modern SaaS app for example might use a library like React, which requires some effort to learn, not to mention the prerequisites of HTML, CSS, and JavaScript. And some sort of back end like Node. And Git for version control and likely many other coding tools. As a freelance designer I work on projects that require a huge variety of coding libraries and tools and it's impossible for me to learn all of them. Even if I know React and Node, what if a client comes along whose product is built on Angular and Laravel? Now I need to learn those too, just to do my work as a designer? It's just too much.

DevTools is my shortcut. I can work directly on the code aspects I care about—the visual stuff—without having to learn or even understand everything happening under the hood.

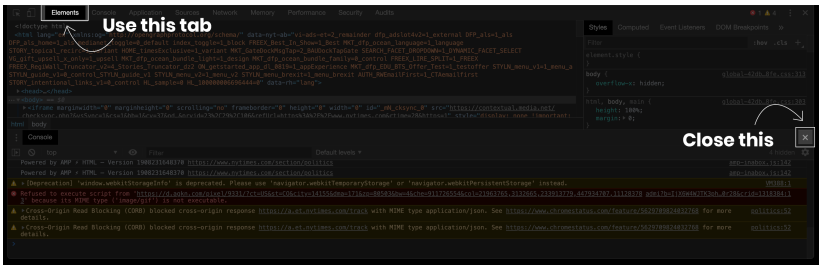
DevTools is not only a developer tool. It is a design tool, a writing tool, and even a collaboration tool. You can use it to do much, much more than debug and optimize code.

Oh, and while I mentioned hacking, DevTools isn't intended for bypassing security measures on websites or anything nefarious. It does, however, let you modify any website, see how it works, and try things out.

Yes, DevTools will let you break a website, but only for yourself. One quick page refresh and anything you did to the site disappears (even if you don't want it to, but I'll show you a way to prevent that soon). So feel free to experiment and break stuff in DevTools. It's a fun way to learn code, and an even better way to work with developers as they build your design.

The only part of DevTools you need to use

Open your browser, then open DevTools by pressing `command + option + i`. A tray will open in your browser. In the thin tab bar across the top of the DevTools area, on the left, you'll see a tab called 'elements' if you're using Chrome or 'inspector' if you're using Firefox. If it's not already open, click it. (If you have a DevTools window broken into two large rows with one labelled console, close the console by clicking the X on the right in the same row. You don't need it.)



This is the only panel you really need to know in DevTools. You might never venture outside this panel—I rarely do even as an experienced

developer.

(Note: for now, I'm going to refer to this panel using the Chrome browser's name 'elements' for simplicity.)

The elements panel shows all the HTML and CSS code for the page you're viewing. That's right, you can see the source code of every website on the internet. The ability to see how any website works is a pretty special thing because it means you have open access to every design on the web. Now, they're all yours to revise, tinker with, hack, and learn from.

The left side of the elements panel shows the page's HTML, while the right side shows the CSS for any HTML tag you select. Click any part of the HTML and you'll see the right side update and show new CSS that applies to whatever tag you clicked.

Now that you understand what you're looking at,

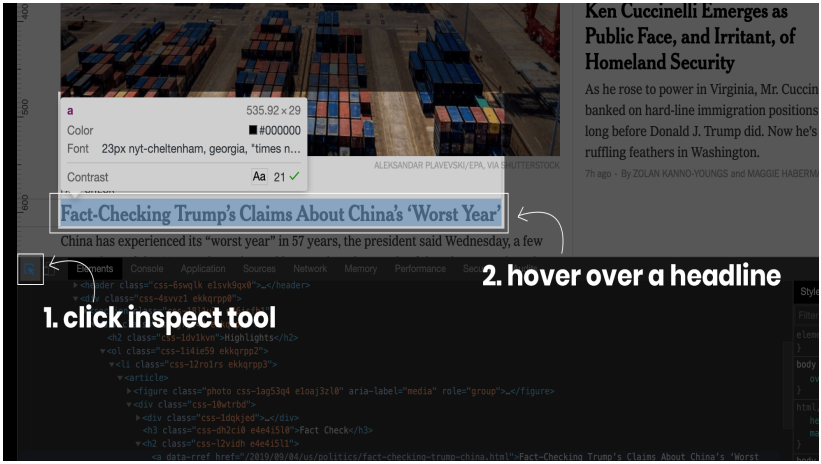
let's try making some edits. Everything you see is editable.

Prototyping in the browser

Exercise #1

Let's start by prototyping a simple design change right in DevTools. Your goal: make fake news. (Gasp!)

Visit your favorite news site and crack open DevTools (`command + option + i`). Then, click the little arrow in the top left corner.



That arrow is called the 'inspect' tool. Move your mouse over part of the page and you'll see sections become highlighted. Click the main news headline on the page, and it will be selected in the DevTools panel, showing the HTML on the left and all the CSS that applies to it on the right.

Making fake news is easy. In the HTML area in DevTools, double click the text of the headline, and write a new one. When you're done, click anywhere outside the text you just edited, and you'll see the change reflected right on the website.

Make a few changes to the text on the page and craft some fake news! Here's what I came up with:

In DevTools, anything can happen.

Jokes aside, DevTools can be used for all kinds of more practical design tasks. Editing text is just the tip of the iceberg. Let's look at a more, well, useful use case.

Exercise #2

Pretend you were just hired as a product designer at Stripe. Go to stripe.com and open up DevTools. Your team is brainstorming new updates to the design system—leadership wants to move toward a more flat design style—but you can't start over from scratch.

Usually these kinds of projects are endlessly frustrating. You often have to open up design software and basically redraw an existing design so that you have up-to-date source files and can actually start your real work. Or you slice up screenshots and make a patchwork design with half of it flattened and uneditable. Both routes are tedious to say the least.

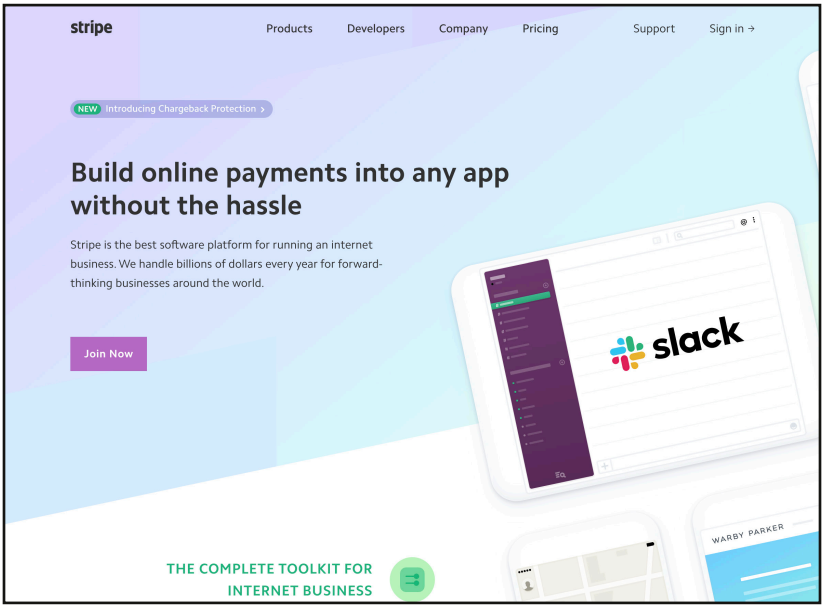
But with DevTools and basic coding knowhow, you don't have to update old mockups or slice screenshots. You can work right on top of the real, live design.

Try redesigning the main button style on Stripe's homepage. Click the inspect tool and then click the green button.

Change the text inside the button in the HTML on

the left in DevTools. Then, scroll through the CSS on the right and change some properties to redesign the button. To edit a property, just click the value you want to change, type in a new value, and hit enter.

Next, write a new main headline and change the font styles. Edit some other elements on the page as you see fit.



Here's what I came up with.

If you were on the Stripe design team, you could take a screenshot and share your concept with your team in all of 15 minutes' time. Instead of the

hours it would take to rebuild a design mockup. I told you that learning code would pay off!

DevTools is magic

Using DevTools to make basic edits, you can prototype any kind of design or content direction right on top of a live site. There is no limit. It's an open door into the inner workings of the web. It's powerful. It's fun. It's creative. And it's definitely not just for developers.

There's a science fiction cliche about how when technology becomes sufficiently advanced, it seems like magic. Maybe DevTools seemed that way to you before. But with the mystery of DevTools dispelled by just a little coding knowledge, a different kind of magic happens.

You finally have a way to control the details of a live product and communicate with developers about how to make the design better in a way they understand. After all the years we've spent arguing with developers and trying to get our requests

answered, using this humble little tool gives us all a simple way to work together. It really is kind of magical.

Chapter 4

??!!

There's one last thing you need to know: how do you actually fit all of those skills into your normal process?

Most designers and teams don't have a great workflow for handling long-term optimization or the implementation phase of feature releases. But it is critical to have a solid process for these because you'll face them even more as long-term optimization and product design gain traction in more businesses. I can help you create a more reliable way of handling that frustrating phase of projects.

You can use DevTools to edit live sites and share those edits with developers to improve the design and content while they're building, so that every detail is perfect. You can also use DevTools to prototype concepts to share with your team. This goes a long way toward filling in that missing process.

Everyone seems to use DevTools to fill in the gaps of the process where there isn't a tool to bridge between disciplines.

DevTools is amazing, but the more you use it like

that, the more you realize it's missing a few key features. This is the wishlist I've had for years:

- Track all HTML & CSS changes from DevTools
- Save & reapply changes if you refresh or close the page (I've lost so many edits by accidentally clicking links)
- Take before & after screenshots showing your edits (Instead of doing it bt hand)
- View a diff clearly showing which HTML & CSS changed
- View original line numbers from CSS preprocessors (when you edit CSS in the browser, it's handy to know which source file that CSS came from when it comes time to build your changes into the live site)
- A way to share your edits and discuss with your team without having to draw red lines on screenshots to show what you did

DevTools doesn't fit into the usual workflow quite as well as it could. Many people respond to that by digging deeper into coding and learning more advanced techniques so they can just work on the

production code base. But you shouldn't have to become a full-fledged developer just to do your job as a non-technical team member.

So the last thing for you to learn is a new tool that turns DevTools into a design and collaboration tool that does exactly what you need—without needing to learn any more coding than you already have.

How to save & share your DevTools edits

Mod&Dot is a new tool to enhance the DevTools workflow for designers, created by yours truly. I took that wishlist above and built it. The whole thing. Here's a quick demo of how it works:

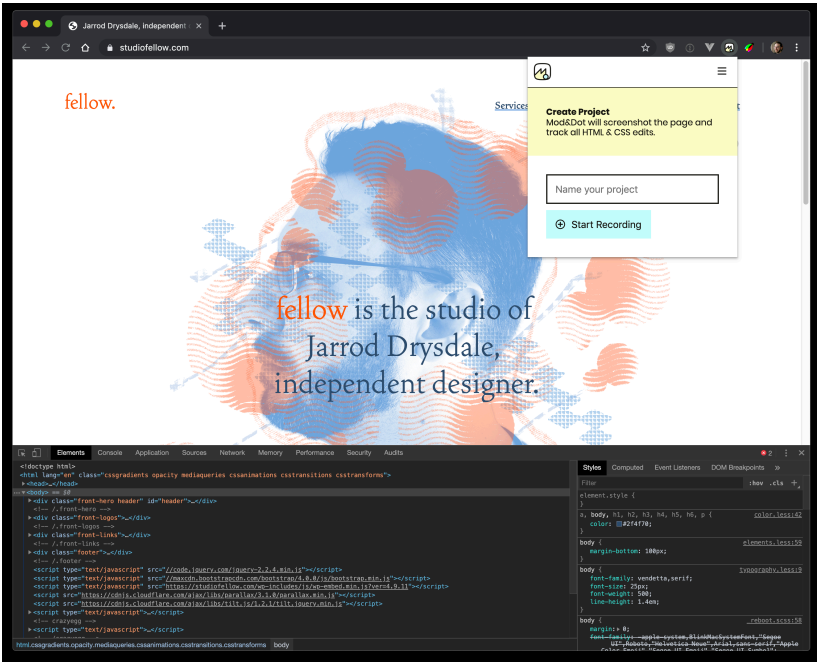
Click here to watch on YouTube

Mod&Dot allows you to save and share edits you make in DevTools. Use a browser extension to record edits, then share it like a mix of Invision and Github.

First, create your account at Mod&Dot, then install

the browser extension.

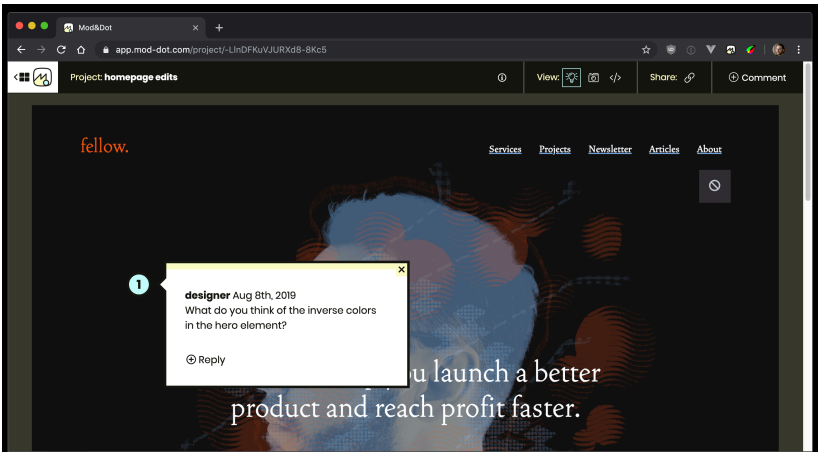
When you're ready to edit a site, just open the extension, log in, and click 'Start Recording'.



Make your edits in DevTools like you normally would. You can even make edits in other extensions like VisBug, which helps you to edit a webpage just like design software!

When you're done, open the extension again and click 'Finish & Publish'. You'll get a link where you can see before & after screenshots of your edits plus a code diff showing everything you changed.

You can share that link with your team and add comments on top of the screenshots to discuss.



Designers can use it to prototype & share design directions or revisions right in the browser without updating mockups or learning advanced coding.

Writers can use it to edit text and demonstrate content strategy in the real context of the live website. And to deliver copy edits in a more useful format.

Coders can use a familiar tool to unleash your creative side—code! Your dev environment will force you to think about code quality. Prototyping in DevTools is liberating.

Web Professionals can use it to contribute to the team without learning fancy design or coding

tools. Finally a way to get your hands dirty!

Instead of taking screenshots of your DevTools edits, drawing red lines to point out what you changed, and recreating all the code edits from memory, you can use Mod&Dot to capture all of that automatically and then package it up in a convenient share link.

Try Mod&Dot Free for 2 weeks

Learn More & Try Free →

If you need help or have questions, you can reach out here.

Just Enough Code

First Edition

Sponsored by Mod&Dot

Written and designed by Jarrod Drysdale

© Studio Fellow LLC. All rights reserved. No part of this book may be reproduced in any form or by any electronic or mechanical means, including information storage and retrieval systems, without permission in writing, except for brief excerpts in reviews or analysis.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the author was aware of the claim, the designations have been marked with ® symbols.

While every precaution has been taken in the preparation of this book, the author assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.