Master the Skill of

Debugging CSS

Ahmad Shadeed

Tips and techniques on how to debug CSS the right way with easy and studied methods

Foreword by John Allsopp

Foreword

I've been using, and a big advocate for CSS since before it was even a standard, nearly 25 years.

It's hard to convey what a profound change it brought in developing for the web when introduced, although its widespread adoption took years of the technology maturing in browsers, and of advocacy, changing the long established use of tables for page layout, font elements, and other hacks web developers came up with to design for the Web.

Since then, CSS has matured in ways its originators and early adopters could barely imagine and brings developers incredible power. But this power and complexity come at a cost.

When developing with CSS, I sometimes think of the story of "The Sorcerer's Apprentice" (originally a poem by German Romantic poet Goethe, but made famous by Mickey Mouse in the Disney film Fantasia). The Sorcerer's Apprentice gains access to his wizard master's powers, but unable to wield them correctly, causes mayhem.

Which sounds like a lot of developing with CSS to me!

The Cascade, Specificity, Inheritance are all powerful features of CSS, but also cause many of the problems we associate with the language.

Which is why I'm surprised it's taken so long for someone to really address the significant challenges of debugging CSS. And why I'm excited for Ahmad's new book, which addresses this important topic in detail.

I really recommend this to any web developer, it's long over due!

John Allsopp — Web Directions

Table of Contents

Chapter 1

Introduction and Overview

Let's face it: The process of debugging CSS is not straightforward, because there is no direct or clear way to debug a CSS problem. In this book, you will learn how to sharpen your debugging CSS skills.

For traditional programming languages, such as Java, C, and PHP, the techniques of debugging have evolved over the years. That is not the case with CSS. Debugging CSS is not like debugging a programing language because you won't be alerted to errors at compilation time. You would get silent errors, which are not helpful.

Before debugging a CSS error, you need to spot it first. In some cases, you might receive a report from a colleague that there is a bug to be solved. Finding a CSS bug can be hard because there is no direct way to do it. Even for an experienced CSS developer, debugging and finding CSS issues can be hard and confusing.

This chapter will discuss:

- the history of debugging CSS,
- what has changed today,
- what debugging CSS means,
- the debugging mindset,
- why debugging needs time,
- an overview of this book's topics.

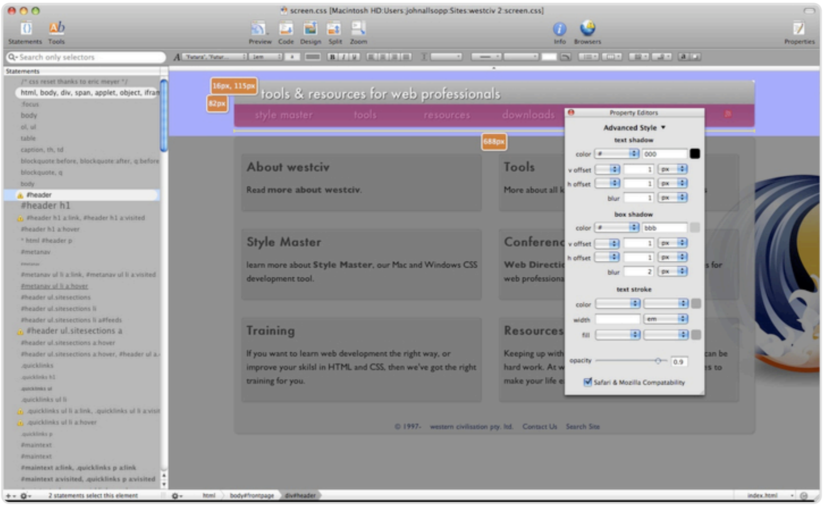
The History of Debugging CSS

Because this book is about debugging and finding CSS issues, you should be aware of a bit of the history of how debugging tools for CSS have developed over the years.

Style Master

You might be surprised to hear that the first CSS debugging tool was released in 1998 — 22 years ago! Its creators, John Allsopp and Maxine Sherrin, named it Style Master. As they described it:

> Style Master is the leading cross-platform CSS development tool. Much more than just a text editor, Style Master supports your workflow — including: creating style sheets based on your HTML; live CSS editing of PHP, ASP.NET, Ruby and other dynamically generated sites; editing CSS via ftp; and much, much more.



```
/* css reset thanks to eric meyer */
html, body, div, span, applet, object, ifram
:focus
body
ol, ul
table
caption, th, td
blockquote:before, blockquote:after, q:before
blockquote, q
body
#header
#header h1
#header h1 a:link, #header h1 a:visited
#header h1 a:hover
* html #header p
#metanav
#metanav
#metanav ul li a:link, #metanav ul li a:visited
#metanav ul li a:hover
#header ul.sitesections
#header ul.sitesections li
#header ul.sitesections li a#feeds
#header ul.sitesections a
#header ul.sitesections a:hover
#header ul.sitesections a:hover, #header ul a.
.quicklinks
.quicklinks h1
.quicklinks ul
.quicklinks ul li
.quicklinks ul li a:link, .quicklinks ul li a:visit
.quicklinks ul li a:hover
.quicklinks p
#maintext
#maintext
#maintext a:link, .quicklinks p a:link
#maintext a:visited, .quicklinks p a:visited
```

2 statements select this element

html body#frontpage div#header index.html

tools & resources for web professionals

style master | tools | resources | downloads

About westciv
Read more about westciv.

Tools
More about all k

Style Master
learn more about Style Master, our Mac and Windows CSS development tool.

Conferenc
Web Directi
web professiona

Training
If you want to learn web development the right way, or improve your skilsl in HTML and CSS, then we've got the right training for you.

Resources
Keeping up with
hard work. At w
make your life e

© 1997- western civilisation pty. ltd. Contact Us Search Site

text shadow: color # 000; v offset 1 px; h offset 1 px; blur 1 px
box shadow: color # bbb; v offset 1 px; h offset 1 px; blur 2 px
text stroke: color; width em; fill
opacity 0.9
[✓] Safari & Mozilla Compatability

The goal of Style Master was to make working with CSS more efficient, more productive, and more enjoyable. As you can see, then, debugging CSS has been a topic of interest to developers for a long time.

Chapter 3

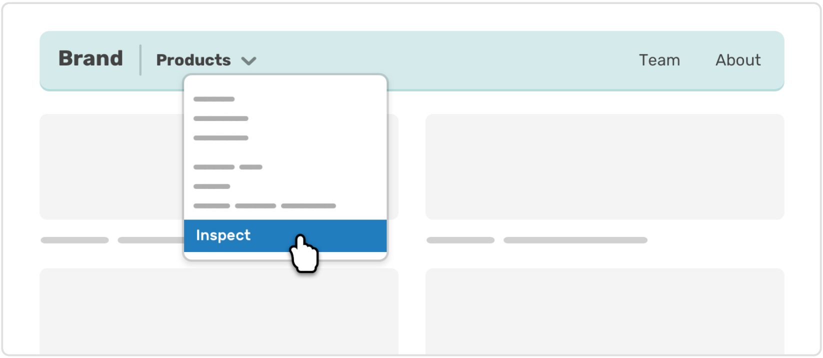
Debugging Environments and Tools

Every modern web browser has development tools, or DevTools, built in. In the history section, I explained a bit about the tools Style Master and Firebug. Browser DevTools are based on these projects. To open yours, right-click and select "Inspect element" from the menu. If you're a keyboard person, here are the shortcuts for each browser:

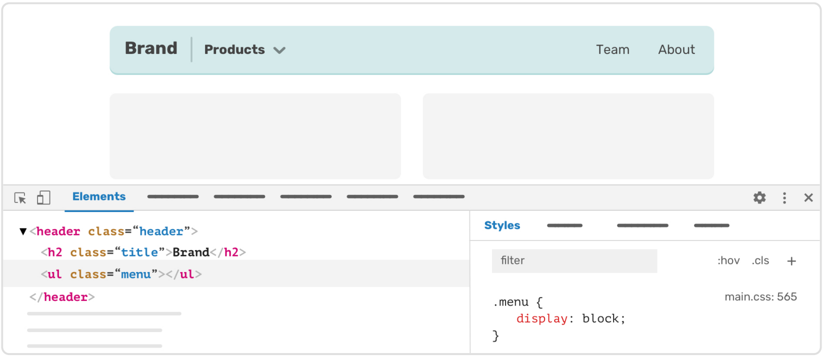
- Chrome: `⌥ + ⌘ + I` on a Mac, and `Ctrl + Shift + I` on Windows
- Firefox: `⌥ + ⌘ + C` on a Mac, and `Ctrl + Shift + I` on Windows
- Safari: `⌥ + ⌘ + I`
- Edge: `⌥ + ⌘ + I` on a Mac, and `Ctrl + Shift + I` on Windows

I will be using Google Chrome in this book, unless I mention another web browser.

You can inspect any element and toggle its CSS properties. To select an element, right-click and choose "Inspect" from the menu.



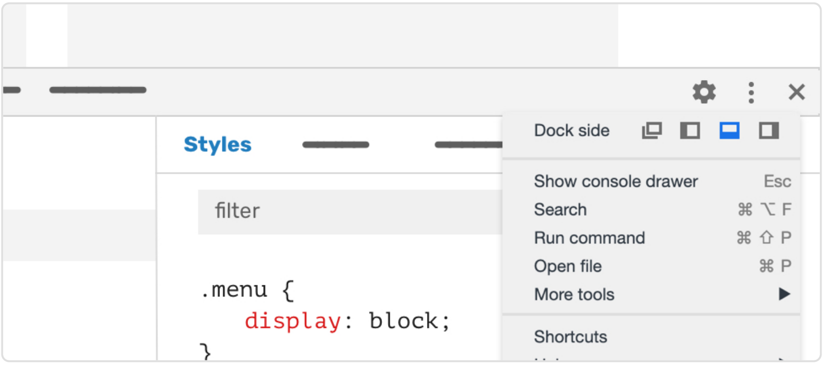
When you select "Inspect", the browser's DevTools will open at the bottom of the screen. That's the default position for it. You can pin it to the right or left side of the screen by clicking on the dots icon in the top right.



```
▼<header class="header">
    <h2 class="title">Brand</h2>
    <ul class="menu"></ul>
  </header>
```

```
.menu {
    display: block;
}
```

With the dots clicked, a little dropdown menu will open. You can choose where to pin the DevTools. There is no right place; choose based on your preference. However, you will need to dock it to the right when you are testing at mobile and tablet sizes. This is how it looks:



```
.menu {
    display: block;
}
```

Toggling a CSS Declaration

We’ve opened the DevTools and know how to access them. Let’s inspect an

Chapter 4

CSS Properties That Commonly Lead to Bugs

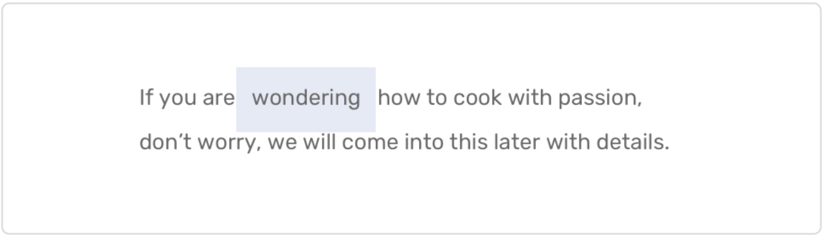
```
span {
    padding-top: 1rem;
    padding-bottom: 1rem;
}
```

This won't work. Vertical padding doesn't work for inline elements. You would have to change the element's `display` property to `inline-block` or `block`.

The same goes for `margin`:

```
span {
    margin-top: 1rem;
    margin-bottom: 1rem;
}
```

This margin won't have an effect. You would have to change the `display` type to `inline-block` or `block`.


If you are wondering how to cook with passion,
don't worry, we will come into this later with details.

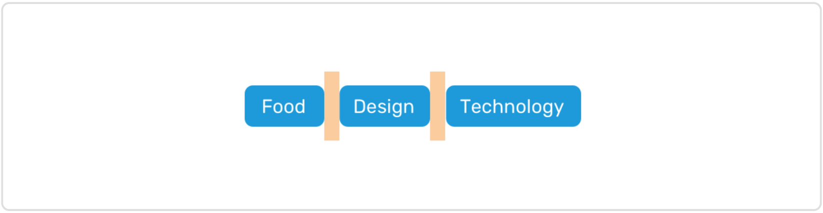
Spacing and Inline Elements

Each inline element is treated as a word. Take the following:

```
<span>Hello</span>
<span>World</span>
```

This will render `Hello World`. Notice the spacing between the two words. Where did this come from? Well, because an inline element is treated as a word, the browser automatically adds a space between words — just like there is a space between each word when you type a sentence.

This gets more interesting when we have a group of links:



The links are next to each other, with a space between them. Those spaces might cause confusion when you're dealing with `inline` or `inline-block` elements because they are not from the CSS — the spaces appear because the links are inline elements.

Suppose we have an inline list of category tags, and we want a space of 8 pixels between them.

```
<ul>
    <li class="tag"><a href="#">Food</a></li>
    <li class="tag"><a href="#">Technology</a></li>
    <li class="tag"><a href="#">Design</a></li>
</ul>
```

In the CSS, we would add the spacing like this:

```
.tag {
    display: inline-block;
    margin-right: 8px;
}
```

You would expect that the space between them would equal 8 pixels, right? This is not the case. The spacing would be 8 pixels **plus an additional 1 pixel from the character spacing** mentioned previously. Here is how to solve this issue:

```
ul {
    display: flex;
    flex-wrap: wrap;
}
```

By adding `display: flex` the the parent, the additional spacing will be gone.

Block Elements

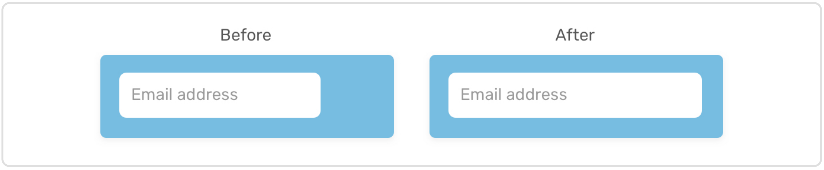
The `block` display type is the default for certain HTML elements, such as `div`, `p`, `section`, and `article`. In some cases, we might need to apply the `block` display type because an element is inline, such as:

- form labels and inputs,
- `span` and `a` elements.

When `display: block` is applied to `span` or `a`, it will work fine. However, when it's applied to an input, it won't affect the element as expected.

```
input[type="email"] {
    display: block; /* The element does not take up the full width. */
}
```

The reason is that form elements are **replaced elements**. What is a replaced element? It's an HTML element whose width and height are predefined, without CSS.



To override that behavior, we need to force a full width on the form element.

```
input[type="email"] {
    display: block;
    width: 100%;
}
```

There are replaced elements other than form inputs, including `video`, `img`, `iframe`, `br`, and `hr`. Here are some interesting facts about replaced elements:

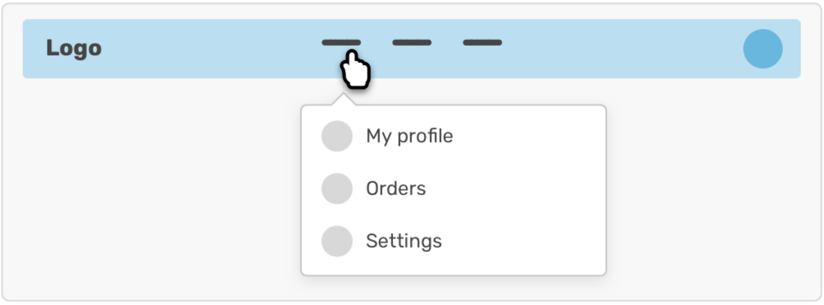
- It's not possible to use pseudo-elements with replaced elements. For example, adding an `:after` pseudo-element to an `input` is not possible.
- The default size of a replaced element is 300 by 150 pixels. If your page has an `img` or an `iframe` and it doesn't load for some reason, the browser will give it this default size.

Consider the following example:

`height: 0` is set on the element.

Transitioning Visibility and Display

Transitioning the `display` property of an element is not possible. However, we can combine the `visibility` and `opacity` properties to mimic hiding an element in an accessible way.



Here we have a menu that should be shown on mouse hover and keyboard focus. If we used only `opacity` to hide it, then the menu would still be there and its links clickable (though invisible). This behavior will inevitably lead to confusion. A better solution would be to use something like the following:

```
.menu {
    opacity: 0;
    visibility: hidden;
    transition: opacity 0.3s ease-out, visibility 0.3s ease-out;
}

.menu-wrapper:hover .menu {
    opacity: 1;
    visibility: visible;
}
```

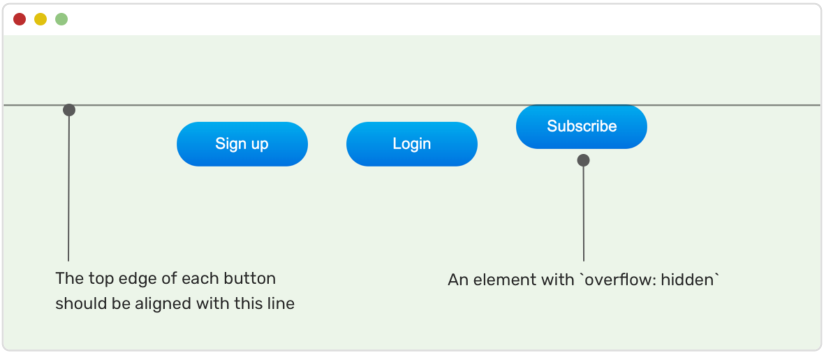
random bugs on Safari iOS.

Inline-Block Elements With `overflow: hidden`

According to the CSS specification:

> The baseline of an "inline-block" is the baseline of its last line box in the normal flow unless it has either no in-flow line boxes or if its "overflow" property has a computed value other than "visible", in which case the baseline is the bottom margin edge.

When an `inline-block` element has an `overflow` value other than `visible`, this will cause the bottom edge of the element to be aligned according to the text baseline of its siblings.


Sign up
Login
Subscribe
The top edge of each button should be aligned with this line
An element with `overflow: hidden`

To solve this, change the alignment of the button that has `overflow: hidden`.

```
.button {
    vertical-align: top;
}
```

The `d-block` class sets the element to display as a `block` type. Adding `!important` ensures it will be applied as expected.

Flexbox

The flexbox layout module provides us with a way to lay out a group of items either horizontally or vertically. There are many common issues with flexbox: Some are done mistakenly by the developer, and others are bugs in a browser's implementation.

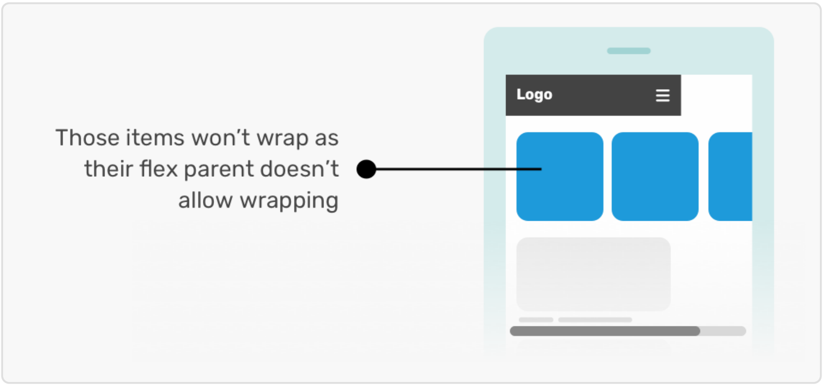
User-Made Bugs

Forgetting `flex-wrap`

When setting an element as a wrapper for flexbox items, it's easy to forget about how the items should wrap. Once you shrink the viewport, you notice horizontal scrolling. The reason is that flexbox doesn't wrap by default.

```
<div class="section">
    <div class="item"></div>
    <div class="item"></div>
    <div class="item"></div>
</div>
```

```
.section { display: flex; }
```

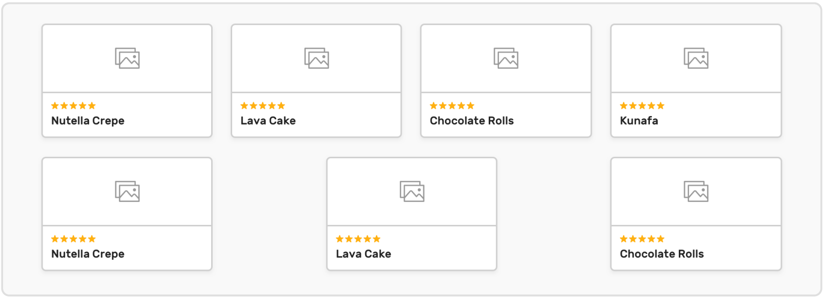

Those items won't wrap as their flex parent doesn't allow wrapping
Logo

Notice how the items aren't wrapping onto a new line, thus causing horizontal scrolling. That is not good. Always make sure to add `flex-wrap: wrap`.

```
.section {
    display: flex;
    flex-wrap: wrap;
}
```

Using `justify-content: space-between` for Spacing

When we use flexbox to make, say, a grid of cards, using `justify-content: space-between` can be tricky.



The grid of cards above is given `space-between`, but notice how the last row looks weird? Well, the designer assumed that the number of cards would always be a multiple of four (4, 8, 12, etc.).

CSS grid is recommended for such a purpose. However, If you don't have any option but to use flexbox to create a grid, here are some solutions you can use.

Using Padding and Negative Margin

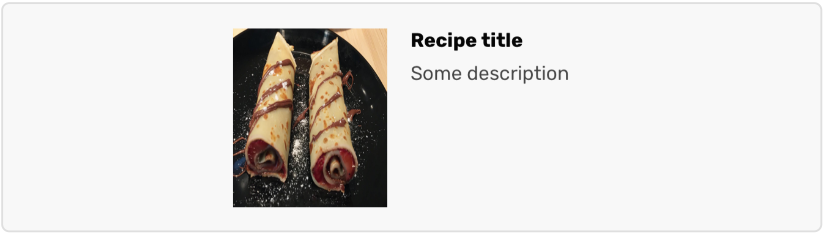
```
<div class="grid">
    <div class="grid-item">
        <div class="card"></div>
    </div>
    <!-- + 7 more cards -->
</div>
```

```
.recipe { display: flex; }

img { width: 50%; }
```

A simple online search reveals that this issue is common, and it has inconsistent browser behavior. The only browser that still stretches an image by default is Safari version 13. To fix it, we need to reset the alignment of the image itself.

```
.recipe img { align-self: flex-start; }
```



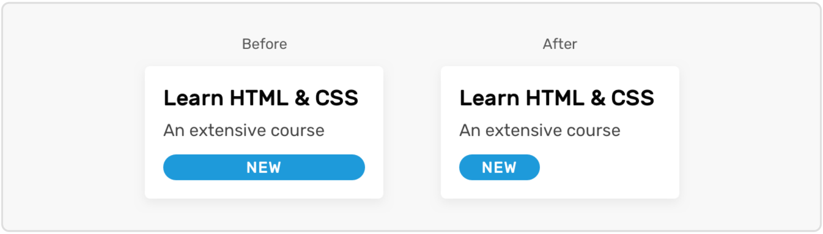
While Safari version 13 is the only one that has the inconsistent behavior of stretching the image, the `button` element is stretched in all browsers. The fix is the same (`align-self: flex-start`), but small details like this make you think about the weirdness of browsers.

We see a related problem when a flex wrapper has its direction set to `column`.

```
<div class="card">
    <h2 class="card__title"></h2>
    <p class="card__desc"></p>
    <span class="card__category"></span>
</div>
```

```
.card {
    display: flex;
    flex-direction: column;
}
```

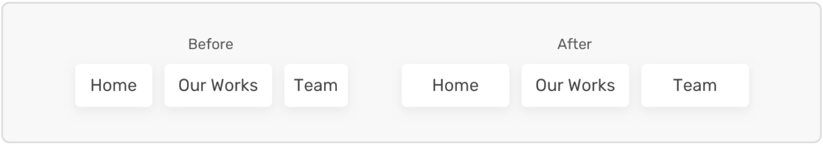
The `.card__category` element will stretch to take up the full width of its parent. If this behavior is not intended, then you'll need to use `align-self` to force the `span` element to be as wide as its content.



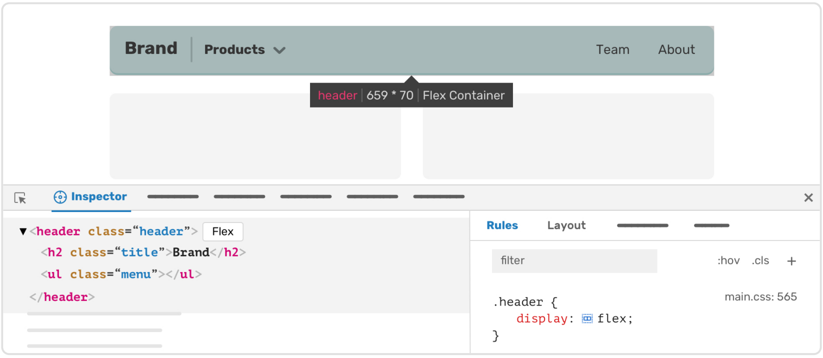
```
.card__category {
    align-self: flex-start;
}
```

Flexbox Child Items Are Not Equal in Width

A common struggle is getting flexbox child items to be equal in width.



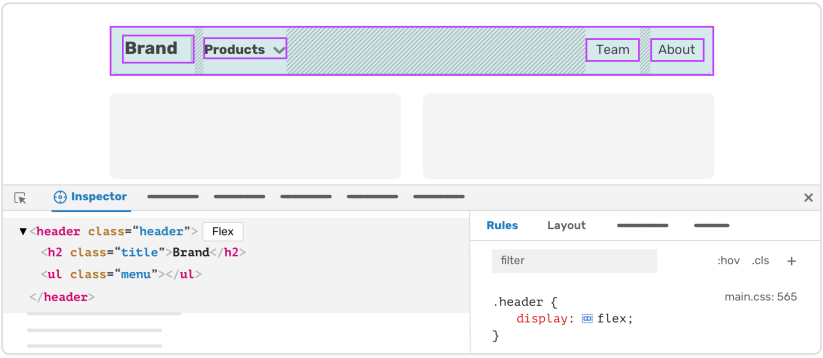
According to the specification:



```
▼<header class="header">
   <h2 class="title">Brand</h2>
   <ul class="menu"></ul>
 </header>
```

```
.header {
   display: flex;
}
```

The great thing is that the “flex” label is clickable. When it’s clicked, Firefox will highlight the flex layout items. It can also be accessed from the little flexbox icon beside the CSS declaration in the “Rules” panel.



```
▼<header class="header">
   <h2 class="title">Brand</h2>
   <ul class="menu"></ul>
 </header>
```

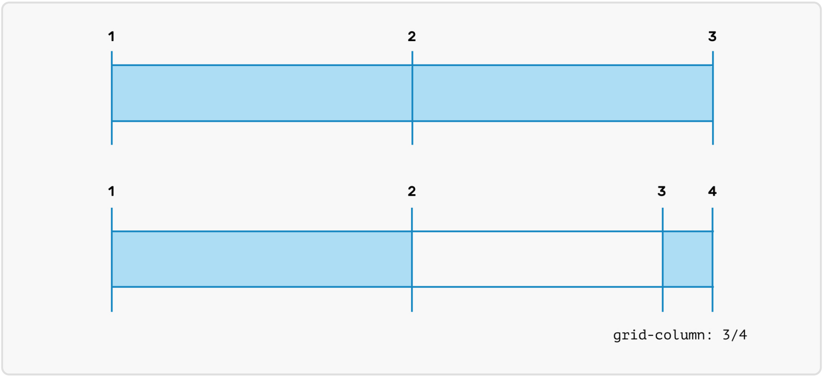
```
.header {
   display: flex;
}
```

The highlight is useful when you’re in doubt of how a flexbox layout works. Take advantage of these tools — they enable you to make sure that nothing weird is happening and clear up any confusion about a flexbox container.

CSS Grid

Unintentional Implicit Tracks

A common misstep with CSS grid is to create an additional grid track by placing an item outside of the grid's explicit boundaries. First, what's the difference between an **implicit** and **explicit** grid?



```
.wrapper {
    display: grid;
    grid-template-columns: 1fr 1fr;
}

.item-1 {
    grid-column: 1 / 2;
}

.item-2 {
    grid-column: 3 / 4;
}
```

The `.item-1` element has an **implicit** grid track, and it's placed within the grid's boundaries. The `.item-2` element has an **explicit** grid track, which places the element outside of the defined grid.

CSS grid allows this. The problem is when a developer is not aware that an implicit grid track has been created. Make sure to use the correct values for `grid-column` or `grid-row` when working with CSS grid.

A Column With `1fr` Computes to Zero

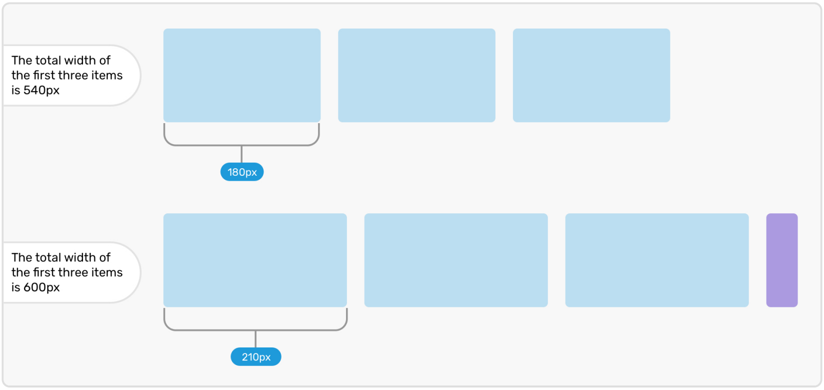
There is a case in which a column with `1fr` will compute to a width of `0`, which means it's invisible.

```
<div class="wrapper">
    <div class="item"></div>
    <div class="item"></div>
    <div class="item"></div>
    <div class="item"></div>
</div>
```

```
.wrapper {
    display: grid;
    grid-template-columns: repeat(3, minmax(50px, 200px)) 1fr;
    grid-template-rows: 200px;
    grid-gap: 20px;
}
```

We have three items with a minimum of 50 pixels and a maximum of 200 pixels. The last item should take the remaining space, `1fr`. If the sum of the widths of the first three items is less than 600 pixels, then the last column will be **invisible** if:

- it has no content at all,
- it has no border or padding.


The total width of the first three items is 540px
180px
The total width of the first three items is 600px
210px

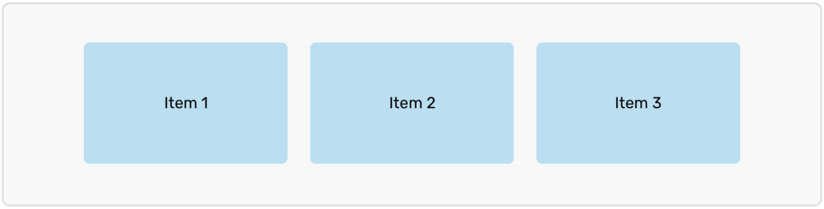
Keep that in mind when working with CSS grid. This issue might be confusing at first, but when you understand how it works, you'll be fine.

Equal `1fr` Columns

You might think that the CSS grid fraction unit, `fr`, works as a percentage. It doesn't.

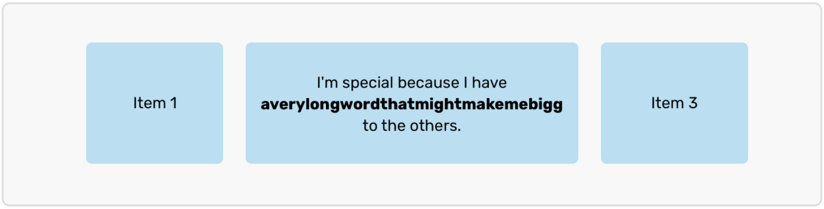
```
<div class="wrapper">
    <div class="item">Item 1</div>
    <div class="item">Item 2</div>
    <div class="item">Item 3</div>
</div>
```

```
.wrapper {
    display: grid;
    grid-template-columns: 1fr 1fr 1fr;
    grid-template-rows: 200px;
    grid-gap: 20px;
}
```



The items look equal. However, when one of them has a very long word, its width will expand.

```
<div class="wrapper">
    <div class="item">Item 1</div>
    <div class="item">I'm special because I have
averylongwordthatmightmakemebiggerthanmysiblings.</div>
    <div class="item">Item 3</div>
</div>
```


Item 1
I'm special because I have **averylongwordthatmightmakemebigg** to the others.
Item 3

Why does this happen? By default, CSS grid behaves in a way that gives the `1fr` unit a minimum size of `auto` (`minmax(auto, 1fr)`. We can override this and force all items to have equal width. The default behavior might be good for some cases, but it's not always what we want.

```
.wrapper {
    /* other styles */
    grid-template-columns: repeat(3, minmax(0, 1fr));
}
```

Beware that the above will cause horizontal scrolling. See the section on horizontal scrolling for ways to solve it.

Setting Percentage Values

The unique thing about CSS grid that it has a **fraction** unit, which can be used to divide columns and rows. Using percentages goes against how CSS grid works.

```
.wrapper {
  display: grid;
  grid-template-columns: 33% 33% 33%;
  grid-gap: 2%;
}
```

Using percentage values for `grid-template-columns` and `grid-gap` would cause horizontal scrolling. Instead, use the `fr` unit.

```
.wrapper {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-gap: 1rem;
}
```

account for another scenario, which is if we don't want the person's name to wrap onto a new line? In this case, `text-overflow` to the rescue.

```
.card-meta h3 {
    white-space: nowrap;
    text-overflow: ellipsis;
    overflow: hidden;
}
```

Wrapping Up

Now that we've reached the end of this chapter, I hope you're more comfortable with the most common CSS properties and their issues. Of course, I haven't mentioned every single property, but I've tried to include the things that you will be addressing in your daily work.

If you've gone through the first four chapters carefully, then you will be able to tackle any CSS issue from the start to finish using the techniques you've learned.

Acknowledgements

The book idea started as a note in April 2020. I asked Kholoud, my wife, what do you think about writing a book about debugging CSS? I told her that it will be a very short one (60 pages max). Seven months later, the book has 300 pages. Kholoud was the first person to support the book idea, and she insisted that I should move on with this, and here we are. Thank you, my dearest person!

The first person that encouraged me from the community is Mr. John Allsopp. He invited me to talk at Web Directions conference about the book topic and was one of the first supporters. Thank you very much!

I want to thank is Geoffrey Crofte. He was helpful and kind enough to proofread the whole book, and highlighting a lot of fixes. Thank you very much!

Finally, I would like to thank Bram Van Damme, who reviewed the very first draft of the book. He highlighted some important things that I should improve. Thank you, Bram!

I reduced the time I spend on debugging and fixing CSS bugs from hours to minutes. In this book, I will explain everything I learned about debugging and finding CSS issues.

About the author

A Digital Product Designer and Front-End Developer from Palestine. He enjoy working on large scale Product Design and Front End Projects which involves solving complex design problems. He writes extensively on CSS, Accessibility and RTL (right to left) text styling.

CSS is sadly an increasingly undervalued tool for front end developers, in no small part because developers find debugging CSS challenging. Yet, until now there's been little in depth published on debugging CSS. Ahmad Shadeed's book is long overdue, and I can't recommend it highly enough for any front end developer.

John Allsopp — Web Directions

Ahmad gathers in this practical book a bunch of CSS bugs encountered on a daily basis by all levels of developers, and how to solve all of them. A book that would have saved me hours of research in my early days, and well, still today! Well done!

Geoffrey Crofte — UX Designer