



HTML Whitespace is Broken

September 2, 2024

Recently, I was working on a project which required a deeper understanding of how whitespace works in HTML. I was never a fan of HTML's whitespace behavior before as I've been burned by it a few times. But as I dug into it more deeply, I found myself discovering complex design issues that I wanted to explore in a blog post. This is partially to write down my knowledge in this space for future reference and partially to vent about how unnecessarily complicated it all is.

So let's discuss:

1. [How whitespace actually works.](#)
2. [Why it works that way.](#)
3. [The problems many HTML tools have.](#)
4. [How it *should* work.](#)
5. [What we can do about it.](#)

How HTML Whitespace Works

[MDN has a great article](#) explaining whitespace in HTML but I'll try to break it down here. Let's start with `in line` elements.

Inline Elements

HTML is whitespace-sensitive in that elements will render differently based on whether or not whitespace exists between them. In these two examples, if a space is present between the two links, then that space is rendered in the output.

Example 1: No whitespace.

```
<a href="#">First</a><a href="#">Second</a>
```

[FirstSecond](#)

Example 2: Single space.

```
<a href="#">First</a> <a href="#">Second</a>
```

[First](#) [Second](#)

This makes some amount of sense to me. The difference between 1. and 2. is clear in the code, obviously the developer intentionally put a space in 2. and it follows that 1. would not have any space between its links.

Beyond single spaces, HTML is subject to whitespace "collapsing", where multiple spaces are "collapsed" into a single space. This means adding additional spaces has no effect. It's exactly the same as having one space.

Example 3: Lots of spaces.

```
<a href="#">First</a>      <a href="#">Second</a>
```

[First](#) [Second](#)

Newlines and tabs are also treated identically and collapsed into spaces.

Example 4: Newline.

```
<a href="#">First</a>  
<a href="#">Second</a>
```

[First](#) [Second](#)

The space between the tags is rendered as an independent text node, meaning it does not inherit the styles for either of the `<a>` tags. It does not include an underscore and clicking the whitespace does not trigger either link. Given that the space in HTML is not within either `<a>` tag, that seems pretty reasonable to me.

But let's keep experimenting. If we put this inside a `` tag, then any leading and trailing whitespace is *not* preserved.

Example 5: Indented.

```
<span>  
    <a href="#">First</a>  
    <a href="#">Second</a>  
</span>
```

First Second

Even though there are many spaces before the first `<a>`, they do not render to the user. This is because whitespace at the start of a rendering context (basically whitespace before the first line of a block) is removed completely.

Typically, spaces which are visible to the user are referred to as *significant*, while spaces which are not rendered are considered *insignificant*. For the above example, the newline and indentation between the links are significant because they will be collapsed to a single space and rendered. The indentation before the first link and trailing newline after the second link are insignificant and not displayed to the user.

This also applies to whitespace inside a tag. Consider these examples:

Example 6: Basic link.

```
Hello, <a href="#">World</a>!
```

Hello, [World](#)!

Example 7: Spaced link.

```
Hello, <a href="#"> World </a>!
```

Hello, [World](#)!

Example 8. Very spacey link.

```
Hello, <a href="#">           Wor ld           </a>!
```

Hello, [World](#)!

In 7., we can see that the space between "World" and "!" is preserved and the space is underscored with the rest of the link. If you click that space precisely enough you'll actually follow the link.

In fact 7., shows even more collapsing. There are actually *two* spaces between "Hello," and "World", one outside the `<a>` tag and the other inside it. You might expect this to render two spaces, the latter of which is underlined because it is a part of the link.

However the browser actually collapses both spaces together and only displays one. This begs the question: Which space is preserved? How does the browser decide?

Let's test this out by swapping the ordering:

Example 9: Link before text.

```
<a href="#">Hello, </a> World!
```

[Hello, World!](#)

Here we see that the space *is* underscored, so the space was again given to the preceding text node.

This also highlights the most common foot gun I've seen with HTML whitespace: links with extra spaces. Consider this example:

Example 10: Long link text.

```
Hello, <a href="#">  
    here is some long link text that goes on its  
</a> please take a look at it!
```

Hello, [here is some long link text that goes on its own line](#) please take a look at it!

Since the link is on its own line, the text ends with a newline character before the ``. This means the rendered output places the space within the link itself, so the underscore trails one character further than you might expect.

There's also a space after the ``, meaning this falls into the case where there are multiple spaces across two elements. As a result, the space goes to the preceding node which is the link in this case. The solution here is to remove the newline at the end of the `<a>` tag by reformatting it, line length limits be damned.

Example 11: Single-line link.

```
Hello,  
<a href="#">here is some long link text that goes  
please take a look at it!
```

Hello, [here is some long link text that goes on its own line](#) please take a look at it!

Your formatter might not like this solution, but we'll [get to that later](#).

Block Elements

All of the above applies to *inline* HTML elements.

Block elements are similar, but preserve a little less whitespace. As mentioned earlier about inline elements, any spaces at the start or end of a line are dropped. Block elements work similarly. Any whitespace in a [block formatting context](#) becomes its own block, except that whitespace-only blocks are then dropped entirely. Practically speaking, this means that any spaces around blocks are effectively ignored.

Example 12: Block elements without whitespace.

```
<div>Hello</div><div>World</div>
```

Hello
World

Example 13: Block elements with whitespace.

```
<div>Hello</div>    <div>World</div>
```

Hello
World

Example 14: Block elements with newline.

```
<div>Hello</div>  
<div>World</div>
```

Hello
World

Recall that for inline elements, a space between them is significant and gets included in the former element.

For block elements, there's actually no difference between these examples because all the whitespace differences are ignored and newlines are placed between the blocks. Example 12. does not actually contain *any* whitespace, yet the two `<div>` tags are presented as if there is a newline between them.

CSS

Now that you understand how block and inline elements behave, let's do a quick pop quiz, how do you expect the following to render?

Example 15: <aside> without whitespace.

```
<aside>Hello</aside><aside>World</aside>
```

You might intuitively think, "Well <aside> is a block element, so it should follow the same rules as <div>. Therefore this will render exactly like example 12., and there's a newline between them."

That's very well-reasoned of you, and you are correct... *most* of the time... This is actually a trick question. <aside> is natively a block element, but it doesn't have to be. Therefore you can actually render different spacing based on how you *style* the element.

Example 16: Block <aside>.

```
<style>aside { display: 'block'; }</style>  
<aside>Hello</aside><aside>World</aside>
```

Hello
World

Example 17: Inline <aside>.

```
<style>aside { display: 'inline'; }</style>  
<aside>Hello</aside><aside>World</aside>
```

HelloWorld

The same HTML can actually lead to different whitespace behavior. That might not sound too bad. After all, this is exactly the layout difference between `block` and `inline`. However it actually *changes* the fundamental text content displayed to the user. Example 16. displays two strings, "Hello" and "World". While 17. displays a single string "HelloWorld". There's a semantic difference between those two options, not just a styling distinction.

You can actually observe this distinction in JavaScript through [textContent](#) and [innerText](#) on a parent element. The former joins the strings together with no spacing in both cases.

```
> parentOfBlockAsides.textContent  
'HelloWorld'  
> parentOfInlineAsides.textContent  
'HelloWorld'
```

However `innerText` adds a newline for the `block` elements *only*.

```
> parentOfBlockAsides.innerHTML
'Hello\nWorld'
> parentOfInlineAsides.innerHTML
'HelloWorld'
```

[Per MDN](#), `innerHTML` is "aware of styling" so it can tell the difference between these two examples in a way `textContent` cannot.

You can even hear the difference with text-to-speech tools! Windows Narrator on Chrome treats block elements as different text fields while inline elements are joined into a single word.

Here I put the word "Refrigerator" split across multiple `<aside>` tags. The first attempt uses the default `display: block;` while the second is `display: inline;`.

Narrator treats the first attempt as four different words. It implicitly converts "fri" to "Friday" (assuming certain semantics on the text) and can't pronounce "ger" at all, choosing to spell it out instead as "g-e-r".

The second attempt correctly speaks "Refrigerator" even though it has identical DOM structure to the first attempt. The only difference is the `display` property. This tells us two things:

First, whitespace handling of HTML content can be controlled purely through CSS. This is also interesting because it means that whitespace handling is not done by the HTML parser. The parser must retain all spaces because it's actually the CSS layer which decides whether or not those spaces are significant.

Second, the actual *content* of a web page can be manipulated by CSS. The page should contain either "Refrigerator" or "Re-fri-ger-ator", regardless of the CSS applied. The presentation layer of CSS should not get to decide which of those two interpretations is correct, that's HTML's job. This also implies that search engines may index different textual content based whether they process a page's CSS styling, which really bulldozes any idea of separation of concerns between HTML and CSS.

Preformatted Text

"But wait!" I hear you say. "If you don't like HTML spacing, just use the `<pre>` tag!"

Yes, that is a valid point. HTML does have a `<pre>` tag for "preformatted" text which automatically preserves *all* whitespace.

Example 18: Preformatted text.

```
<pre>
Hello world
I am preformatted      text, which is interest.
    This line is indented more than the rest!
</pre>
```

```
Hello world
I am preformatted      text, which is interest
    This line is indented more than the rest!
```

No whitespace collapsing occurs and all of it is considered significant... except when it isn't, thanks HTML! There are actually two insignificant spaces here:

1. The first newline immediately following the initial `<pre>` (`<pre>\nHello...`).
2. The last newline immediately preceding the final `</pre>` (`the rest!\n</pre>`).

Neither of these newlines are rendered. There's no blank line at the start or end of the rendered result. Surprisingly, if we check `textContent` we don't see the first newline, but we *do* see the second newline, even though it's not rendered.

```
> preElement.textContent
'Hello world\n...than the rest!\n'
```

This detail hints at even more nuanced behavior, as whitespace at the start of a `<pre>` tag is treated *differently* from whitespace at the end.

Example 19: Preformatted text with leading / trailing whitespace.

```
<pre>
```

```
Hello world
```

```
</pre>
```

```
Hello world
```

Writing this out directly, the source text looks like:

```
\n\nHello world\n\n
```

Here, you'll notice the two newline characters at the start are dropped and not rendered at all. However, only one newline character at the end is dropped, and we end up rendering an extra line. So all newlines at the start of a `<pre>` tag are dropped, but only the *last* newline at the end is dropped.

If we add spaces between the `<pre>` and `</pre>` tags and their nearest newlines... (I'm using an underscore to make this visible).

Example 20: Preformatted text with leading/trailing spaces.

```
<pre>_  
Hello world  
_</pre>
```

```
Hello world
```

Now we get the empty lines at the start and end of the block. This means `<pre>` tags treat newlines distinctly from other spaces, and those newlines *must* be the first or last characters of the text.

I can kind of see where the spec authors were going here. Usually if you're putting a `<pre>` tag on the page, you're probably going to put a newline before the content like so:

```
<pre>
```

```
Some preformatted text  
which is multiple lines.
```

```
</pre>
```

You're probably not going to do:

```
<pre>Some preformatted text  
which is multiple lines.</pre>
```

even though that would be more accurate when it comes to whitespace handling. The main takeaway is that even `<pre>` isn't as straightforward as you might have expected, despite that being kind of the whole point of the tag. Not only that, but up until now spaces and newlines have used the same collapsing behavior. This shows us that newlines and spaces can sometimes be treated distinctly from each other.

`<pre>` is also just generally unergonomic. While the whitespace behavior is definitely more intuitive, you can't indent it at all without affecting its content. Compare these two examples:

Example 21: Indented <pre> tag.

```
<pre>  
    Hello, World!  
</pre>
```

Hello, World!

Example 22: Double indented <pre> tag.

```
<div>  
    <pre>  
        Hello, World!  
    </pre>  
</div>
```

Hello, World!

These both feel like they should contain the same text "Hello, World!" and that's clearly the developer's intent. However 21. is preceded by 4 spaces while 22. is preceded by 8 spaces and trailed with a newline and another 4 spaces. This is especially tricky because the indentation causes spaces to come *after* the trailing newline, so the `</pre>` collapsing of a final newline doesn't apply because it's not the last character. Therefore, we end up with an extra blank line in the output.

One other challenge is that `<pre>` applies its whitespace rules to the entire element, which can be much broader than really necessary. Like the links with trailing spaces earlier, sometimes the problem is just a single space. If you fix that problem by using a `<pre>` tag, you're likely just creating more problems for all the other spaces in the text which were previously rendering just fine.

So while `<pre>` does solve a number of whitespace issues I've described, it causes a whole separate set of developer experience (DX) issues which make it harder and less ergonomic to use correctly.

white - space

[The `white-space` CSS property](#) adds even more complexity to this. It supports `white-space: pre;` which can basically opt any element into the `<pre>` tag's parsing rules. It also supports `pre-line`, `pre-wrap` and a few other possible options to further configure the behavior for specific use cases.

As I mentioned earlier, whitespace processing is defined by CSS rules, not the HTML parser so the [standard which specifies this behavior](#) is actually maintained by the CSS working group and primarily focuses on the behavior of the white-space property.

Between block elements, inline elements, and `<pre>` tags I think I've done a decent job justifying why HTML whitespace is confusing. However if you still don't believe me, I'll also mention that flexbox [forces block rendering on its children](#) and inline-block elements are also [sensitive to minor whitespace changes](#).

** **

So if this is all so complicated and `<pre>` isn't the right solution, what other options are there?

We've all been there. Two elements are right next to each other and you need just a little spacing between them. What do you do?

** **

If you've never understood what this actually is, ` ` is an [HTML entity](#) representing a ["no-break space"](#). Specifically, it's a space which the browser will never line wrap on.

He l l o w o r l d will never be split up into He l l o and w o r l d on different lines, no matter how narrow the viewport gets.

This is a useful, but frequently misused tool. If you need a space between two elements, especially elements in an inline text context where devs frequently reach for ` `, you probably don't want the non-breaking behavior. If the entity appears outside of a text context, such as between two blocks, then the non-breaking behavior doesn't even apply and has no effect.

Instead, I suspect what devs really *want* from ` ` is its non-collapsible behavior. Multiple ` ` characters are not merged together and they can start or end lines.

Example 23: ` `.

```
<div>Hello, &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;World</div>  
<div>&nbsp;</div>  
<div>&nbsp;&nbsp;&nbsp;&nbsp;test</div>
```

Hello, World

test

` ` is not considered whitespace like newlines, tabs, or general spaces so collapsing rules treat it like any other text.

This sounds nice but comes with additional baggage. Beyond the non-breaking behavior, ` ` *always* takes up exactly one space-worth of width and this is never reduced. Except there is one extremely common scenario where spaces are eliminated, and that's line wrapping.

 is a non-breaking space, so you'd think it never line wraps. Except as I mentioned, the "non-breaking" behavior has no effect when used outside of a text context. So if you use to space out two blocks, you've created a line wrapping problem. Check out this example of two red boxes with a blue between them:

The ideal behavior would only show the space between the two squares when they are adjacent to each other, but is not able to support that and *always* takes up space, even if the boxes are already separated due to line wrapping. I suspect most usages of actually have line wrapping bugs which developers don't notice because they don't go out of their way to test this particular behavior.

Unfortunately there's no great alternative to ` `; which would line wrap in the correct fashion. ` ` sounds like the right entity, but that [gets collapsed](#) like a regular space, so it doesn't help here either.

So what's the correct solution for spacing out two elements? Well it's probably best to do this in CSS with `margin`, `padding`, or any of the other thousand properties which introduce spacing and fail to center elements. If you really need to, a `<pre>` tag or the `white-space` property is probably the best way to have maximum control over your spacing behavior. However, I honestly can't blame you if you still end up reaching for ` `. I don't have a great alternative beyond "do it in CSS", which just isn't a drop-in solution.

How Did We Get Here?

Why exactly does HTML work this way? Why did we make this language so complicated?

I think the core problem here is that all whitespace in HTML is *ambiguous*. Specifically, it is ambiguous with regard to the developer's intent. For any given space, did the developer mean for it to be displayed to the user or did they just want to keep their code under the line length limit? It's impossible for the browser to know.

To address this, the designers of HTML tried to come up with a set of rules which would roughly map the HTML code they wanted to write to the rendered output they wanted to create. So you, as a developer, have a UI in your head and write out the HTML to display it, and usually the whitespace "just works". Honestly, I'm kind of amazed the browser is as consistently correct as it is.

But even that isn't 100% correct. Sometimes the developer doesn't expect whitespace to behave the way it does and leads to complicated, hard to understand bugs. `<pre>` tags simplify things and are intended for use cases where whitespace is significant and needs to be retained, but it makes authoring those strings in HTML really awkward and overly precise.

Developers are forced to choose between the default syntax that *usually* works and is convenient to write or a `<pre>` syntax which can very precisely express the spacing they want but is incredibly awkward and inconvenient to work with.

Contrast this with basically any other programming language where user visible strings are syntactically distinct from general whitespace:

```
function sayHello(): string {
    return 'Hello, World!' +
        ' I\'m some text without a newline.' +
        ' and I\'m some text with a trailing newline'
}
```

All the spaces and newlines are explicit. Everything within the quotes is intended for the end-user and everything outside the quotes is intended for the developer. This format has its own problems, multiline text can get very awkward for example. But it's at least unambiguous whether any given space is significant to the end user or insignificant and intended only for the developer.

Just imagine working in a language where text didn't need to be quoted:

```
function sayHello(): string {  
    return Hello, World! +  
        I'm some text without a newline. +  
        and I'm some text with a trailing newline.\n}
```

That sounds awful and I have no idea how it would work. Except you don't need to imagine such a situation, because you already write HTML which works exactly like this, and despite writing a whole blog post on the matter, I have no idea how that works either!

Also, if you believe HTML's syntax is justified because it is intended to represent documents and occasionally authored by non-developers, I just want to say: [No it isn't.](#)

HTML Tooling

Let's zoom out from the browser to talk about the developer writing HTML and the tools which help them succeed. Any tool which processes HTML needs to understand these whitespace semantics, so let's look at just three: [automated formatting](#), [Content Management Systems](#), and [minification](#).

Automated Formatting

For as long as we've had code, we've had arguments about the best way to write it. Everyone's got an opinion and all of them except mine are wrong. So we use tools which automatically format everyone's code into a single, consistent, agreed-upon format. Sounds great.

The problem with this is that formatting regularly changes whitespace. A long element can be broken up into multiple lines:

```
<!-- Before -->
<div class="cool colorful bright awesome">Here is so

<!-- After -->
<div class="cool colorful bright awesome">
  Here is some long text.
</div>
```

Anything affecting indenting can also cause line breaks. For example, consider adding a wrapper `<div>`.

```
<!-- Before -->
<div>Here is some long text saying important stuff.</div>

<!-- After -->
<div class="wrapper">
  <!-- Now it's over the length limit and gets wrapped -->
  <div>
    Here is some long text saying important stuff.
  </div>
</div>
```

These formatting changes are intended to be no-ops. They make my life easier as a developer, but should never change significant whitespace for the user. Except they do change significant whitespace because they introduce leading and trailing spaces.

Consider:

Example 24. One-line link.

```
Check out my <a href="#">web site</a> and read my
```

Check out my [web site](#) and read my blog!

In this case we have text with a link in the middle. But if this exceeds the line length limit, formatting the text can introduce line breaks like:

25. Link with overextended underline.

```
Check out my
<a href="#">
  web site
</a>
and read my blog!
```

Check out my [web site](#) and read my blog!

Now we have that overextended underline again, all because of a single formatting change!

[Prettier](#) actually has an option for this called [--html-whitespace-sensitivity](#). Setting this to `ignore` will allow the above change, so the formatted code looks great, but may break your UI. `strict` will avoid introducing a significant whitespace change so your UI is safe, but leads to truly "pretty" HTML code like:

```
Check out my
<a href="#"
  >web site</a
>
and read my blog!
```

In strict mode, Prettier can't introduce a newline between `<a>` and `` because that could change the rendered output. Instead, it has to put the newline *inside* the `<a>` start tag since that's the only location it can add insignificant whitespace. Same for the `` and the following `and`.

Prettier also has an `--html-whitespace-sensitivity css` option which tells it to "Respect the default value of CSS `display` property." Hopefully after reading this post you should know what that means! Given a `<div>` tag, it can format with the `ignore` behavior because leading and trailing whitespace aren't significant in block rendering contexts. `` tags will use `strict` behavior because the leading and trailing whitespace *is* significant. This makes a lot of sense as a useful middle ground between `ignore` and `strict`.

But after reading this post you should also know that's not entirely accurate and breaks if you do `div { display: inline; }`. Prettier doesn't know anything about your CSS so it can only infer the default `display` for a given tag, it can't know the actual `display` value used at runtime.

No shame on Prettier here by the way. I can't blame the tool for formatting HTML like this. The `css` mode is actually a very useful middle ground for getting the nicer `ignore` formatting in at least most of the cases where it won't have any negative effects while still using `strict` where it's likely to be important.

Again, the real problem here is that HTML whitespace is ambiguous and can't tell that Prettier wants to insert whitespace for the developer's benefit without affecting the rendered output. The fact that you can't know the actual whitespace behavior without knowing the CSS `display` and `white-space` properties is just the cherry on top that cements HTML formatting as a fundamentally unsolvable problem.

Content Management Systems

Many hands are involved in developing and shipping a web page from scratch to its end users. Web developers, designers, product managers, and so much more participate in various parts of the process. As a result, the idea of a single individual writing an entire web page from scratch in a `*.html` file is relatively rare in the year 2024.

A very common scenario is for a web developer to focus on the "HTML parts" of the page while the marketing, product, or localization teams are focused on the raw text content inside a separate content management system (CMS). An e-commerce site selling shoes probably does not want to file a bug with the developers every time they need to update the marketing copy for how the shoes will make you a faster runner in a not-at-all-legally-binding manner. As a result, modern HTML code looks less like `Fast shoes.` and more like:

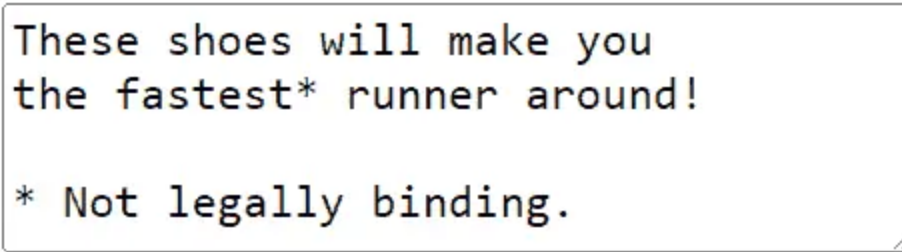
```
<span>${cms.getString('shoes.description')}</span>
```

That sounds straightforward, but pulls on a whole host of whitespace issues. The content is displayed in an HTML context, meaning HTML's whitespace rules still apply, however the author of the text likely doesn't know that.

For example, if the text starts or ends with whitespace, that whitespace will be naturally placed into the `` and affect spacing with adjacent elements.

Another possibility is if the marketing team is led by one of those people who insists on all sentences ending with two spaces after the period. Those two spaces will be collapsed into a single space, and lead to a very upset marketing team which directly leads to a mildly annoyed development team.

Those use cases are possible, but not exactly common and likely something most developers can just ignore. However you have to imagine that the marketing team is looking at a text field like this:

Description: 

Marketing is interpreting the content as plain text, but that's [not how HTML works](#). As a result, they end up looking at a completely borked disclaimer:

These shoes will make you the fastest* runner around! * Not legally binding.

You'd think the CMS should be able to solve this problem, but it really can't. There's two newlines between around! and * Not, but there's no trivial replacement which will prevent them from being collapsed into a single space. The only viable option here is to render all text as `<pre>`, which is a very heavy-handed solution and forces *all* whitespace to be retained when likely only some of it actually mattered. For example, the newline between you and the is likely just the author avoiding a horizontal scrollbar in the text area, not a hard line break which end users should see.

Realistically, both the CMS and the individual writing this copy need some understanding of HTML's whitespace collapsing rules which will be applied to it.

One other negative effect is that it becomes difficult to reuse text content between multiple frontends. For example, if the same shoes are displayed in a native Android or iOS app which don't have the same whitespace collapsing behavior, they will render the same text differently. This makes it very difficult to ensure that the same text is actually rendered consistently and implicitly couples any text rendered to HTML to the whitespace collapsing rules of that environment, even though the raw text itself should be independently usable in any frontend.

Minification

The same problems exist for HTML minifiers: tools which remove unnecessary content from web pages to reduce file size. Multiple spaces are equivalent to single spaces, so they can pretty easily reduce

```
<div>  
  Hello      World  
</div>
```

down to just

```
<div> Hello World </div>
```

But they need to retain those leading and trailing spaces for all the same reasons as mentioned above. HTML minifiers can't know for certain what elements will render in which kind of context and exactly what the whitespace behavior will be. Therefore they need to retain a bunch of spaces which likely don't matter to make sure they retain the one space which does matter and accept a larger output file size, penalizing all users of the page.

How Could we Fix This?

Since this is all so complicated and involved, it would be great to fix HTML so all these whitespace problems go away. Is that possible? What would it look like?

I don't have a perfect solution in mind, but I do have some thoughts. As I mentioned earlier, the root problem is that whitespace in HTML is ambiguous: does a space exist to support the developer authoring their HTML code, or to display something reasonable to the end user? That question is unanswerable and is the core problem we should fix.

The best way to do that is to change HTML syntax such that significant whitespace is syntactically distinct from insignificant whitespace. The obvious approach is to do this just like every other programming language, quote your strings!

```
<div>  
    "Hello, World! My name is Devel and I've got a k  
    " to pick with HTML's whitespace behavior. It's'  
    " super confusing and no one understands it!"  
</div>
```

I see your disgusted reaction and honestly, I kinda get it. I don't think this looks very good either and I wouldn't exactly want to write it. It's not particularly elegant and I hate the leading spaces necessary on all but the first line - so inconsistent.

In this proposed syntax, we at least have implicit concatenation like Python so you don't need + signs everywhere. The idea is that all spaces inside the quotes are significant and retained for the user while all spaces outside the quotes are insignificant and used solely for the developer. Whitespace characters like tabs are allowed and preserved when used within quotes. Newlines are banned by default as it would be way too easy to forget a closing quote and introduce unintended significant spacing. Therefore, a quoted newline is a syntax error. An `&lf;` HTML entity would allow newlines to be directly specified or you could use a triple quote syntax to define a multi-line string.

```
<div>
    """
    Here is a newline.
    And another newline.
    One more for fun.
    """
</div>
```

Lots of languages have triple quote syntax like this and my suggestion is to follow [C#'s example](#) by making this "indentation-aware". The opening and closing `"""` tokens need to be the last / first tokens on their respective lines and the text content between them must be indented to match. Therefore the string above is identical to:

Here is a newline.\nAnd another newline.\nOne more f

This formats nicely while still allowing precise whitespace to be added where it is needed.

Aside from adding quotes there's a couple other changes which need to be made:

1. Any text outside of a quoted string is a syntax error.
 - Exactly how it renders is up to the browser, I really don't care what the fallback behavior is given that browsers tend not to throw errors on bad HTML.
 - The main point is that any tool is free to treat unquoted text as a syntax error along the lines of `<div >`.
2. `white-space` needs to be removed/reworked to always use the standard whitespace behavior.
 - It can still control line wrapping and other presentational aspects, it's just the raw text content which should be consistent across all options.
 - This reduces the dependency on CSS to understand how HTML text is parsed.
3. `<pre>` tags should be removed.
 - All text is preformatted (inside quotes), so having a special tag is no longer needed.
 - Formatting arbitrary user input into a quoted string is not meaningfully harder than HTML escaping the string already is today, so there's no value in keeping `<pre>` tags around.

- You could keep this as a backwards compatibility feature which is an exception to the "no unquoted strings" rule, but a purist implementation of this proposal would remove it entirely.

Would this solve the problem? Since whitespace is no longer ambiguous, we don't need whitespace collapsing anymore.

Whitespace outside the quotes is removed altogether, while whitespace inside the quotes is preserved. No collapsing needed! Developers can add multiple spaces without needing ` `:

```
<!-- Just works! Indentation is correctly ignored, but  
spacing between the words is retained. -->  
<span>  
    "Hello,      World!"  
</span>
```

Prettier and other formatters can adjust the developer side of the whitespace as much as they want and even join/split the string literals to move them across lines without changing the rendered output.

```
<div class="wrapper">
  <div class="container">
    <div id="marketing-made-me-add-this">
      <!-- Reformatted text to be shorter. -->
      "Hello, World! My name is Devel and I've
      " got a bone to pick with HTML's whitesp
      " behavior.\nIt's super confusing and no
      " one understands how it works!"
    </div>
  </div>
</div>
```

Content management systems can be greatly simplified as well since there's no whitespace behavior to understand or preserve. Whatever text marketing gives just gets wrapped in quotes and either replaces `\n` with `&lf;` or uses triple quotes. This way it outputs exactly the way marketing intended.

```
<div>
  ""
  These shoes will make you the fastest* runner ar
  * Not legally binding.
  ""
</div>
```

Minifiers also can just drop all the whitespace outside the quotes, then retain and join everything in the quotes together into a single line:

```
<!-- Before -->
<div>
  "Hello"
  " World"
</div>

<!-- After -->
<div>"Hello World"</div>
```

Unfortunately this doesn't quite solve the [Narrator issue discussed earlier](#). Two strings still need to be treated distinctly in a block formatting context, while strings would be implicitly joined in an inline formatting context.

Example 26: Quoted block vs inline.

```
<div>"Re"</div>
<div>"fri"</div>
<div>"ger"</div>
<div>"ator"</div>

<span>"Re"</span>
<span>"fri"</span>
<span>"ger"</span>
<span>"ator"</span>
```

Re
fri
ger
ator

Refrigerator

You need inline elements to join together so it's possible to style and control parts of words. For example, adding emphasis to only part of a word.

Example 27: Emphasized part of word.

```
<em>"in"</em>"conceivable!"
```

*in*conceivable!

I don't see a good solution to that without completely reworking HTML's rendering model such that every tag has a statically-knowable formatting context, which is a much more complicated change to make.

DX Impact

I get that lot of developers would probably push back on adding quotes to everything because of the negative developer experience (DX) impact. To preempt some of those arguments, I'll remind you of a few points I've hopefully made clear by now:

1. Today's confusing whitespace rules is its own DX problem.
 - This proposal eliminates that problem and means you never have to deal with a bad link underlines again.
2. HTML tooling becomes more stable.
 - It is no longer possible for source formatting to affect behavior.
 - Less fighting with your formatter or debugging weird issues, more time solving real problems.
3. HTML already breaks the rules of common text formatting.
 - The idea that you can write HTML today by just typing the text you want [is a lie](#).
4. Every other language already works this way.
 - Adding quotes would not suddenly make HTML unreasonably burdensome to write.
 - Two more characters per element is not going to break your keyboard.

An alternative approach could use whitespace control characters like [Nunjuck's implementation](#) to achieve a different solution to this problem, however I think that actually leads to an [overall worse DX](#).

Ship It?

Since it seems to solve all the problems described, can we ship this? Unfortunately no.

Shipping a breaking change of this magnitude to HTML would be functionally impossible and go against many of the core principles of the open web. The best you could do is introduce a new HTML parsing option which web pages could opt-in to along the lines of `<!DOCTYPE html6>`. Alternatively, you can treat this as an entirely DX problem and invent a new "quoted HTML" file format called `*.qhtml` with tooling to convert it to standard HTML. Even then, the ecosystem effects of this would be incredibly complicated and end up becoming a solution more painful than the problem it's solving.

So what could we ship?

Non-Collapsible Whitespace

The one fix I can think of is to find a drop-in replacement for ` `. If developers want a non-collapsible space which does not come with the baggage of non-breaking and forced-width behavior, then we can come up with a new entity which meets those needs.

My suggestion: Add a new, named HTML entity (`&ncsp;`) as a regular space which does not get collapsed.

This would take on the non-collapsible benefits of ` `, while dropping the non-breaking and forced-width behavior which leads to line wrapping issues. When you "just want to add a space", `&ncsp;` is likely closer to what you actually want and look better when line wrapped.

Given the [layering of the HTML parser and CSS today](#), I suspect this would actually require an entirely new Unicode character representing a "non-collapsible space", distinct from the existing space character. Except Unicode is used in more than just HTML. In theory, other text rendering engines may do some form of whitespace collapsing and a non-collapsible space might be useful in some of those contexts, but I can see some push back to adding a new character specifically to solve an HTML rendering bug.

I [filed an issue](#) with the CSS working group to discuss adding an `&ncsp;` entity. Please share your own thoughts on this particular idea.

Practical Advice

Given that we can't "fix" HTML, what can we do? Understanding the actual whitespace behavior of HTML goes a long way, but as you've probably figured out, it's surprisingly complicated and I don't think it really scales to expect everyone who writes HTML to fully understand this, nor would I expect a typical code reviewer to spot whitespace bugs looking only at HTML code. Fortunately, HTML whitespace does *usually* work and we can often rely on that, it's mainly about minimizing edge case behavior where you need to look up a blog post like this. Rather than *preventing* unexpected collapsing behavior, we can *mitigate* the issue to be less of a problem in practice. To that end, I have a few suggestions, all of which use the term "Avoid" rather than "Never" since they are focused more on mitigation than prevention.

Avoid Leading and Trailing Whitespace in Links

First, avoid leading and trailing whitespace in `<a>` tags or any other underlined text.

```
<div>
  Here is some interesting text with
  <a href="#">a link that exceeds the line length
  but I don't care.
</div>
```

In my experience, links are by far the biggest challenges with whitespace given that they are underlined by default and it's a common style. While spacing may be incorrect in many other situations, the underline is usually where it becomes a noticeable problem that needs to be fixed.

As long as `<a>` tags are written with no leading or trailing whitespace, this isn't an issue and your underlines will always be correct. It does mean that in some situations you might have to exceed the line length limit, and potentially fight with your formatter (don't use `--html-whitespace-sensitivity ignore`), but I think it's worth it given how common of a foot gun this particular use case is.

Avoid Changing Layout Behavior with `display`

Second, avoid changing `display` to a different layout behavior. If you want a `display: block;` element, pick a tag which uses `display: block;` by default. This should reduce the possibility of your whitespace behavior being dependent on CSS invisible to a formatter, however it might be difficult when you need a specific semantic tag. Take for example, if you need an inline `<aside>` tag.

There are two ways to do this:

1. Add a second tag so the one containing the text uses the correct `display` value by default, independent of the semantic `<aside>` tag wrapping it.

```
<aside>
  <span>
    My text content.
  </span>
</aside>
```

2. Pick a tag with the `display` behavior you want by default (`<div>` or ``), and then use `role` to switch it to the semantic element you want. There might be other accessibility implications to this though.

```
<span role="aside">
  My text content.
</span>
```

Both of these approaches work with the `--html-whitespace-sensitivity css` option and avoid a CSS dependency just to understand the text displayed.

Technically the same guidance should apply to [usage of flexbox and grid](#), both of which force their children to be block elements in a way formatters would be unable to detect. Therefore `flex` and `grid` elements should always contain child tags which are block formatted by default (ex. use `<div>` instead of ``), to align with the formatter's expectations.

Avoid Changing Collapsing Behavior with `white-space`

Third, avoid white-space: pre; and prefer a real <pre> tag instead. Again, this reduces dependencies on CSS and might not always be possible, but hopefully is a decent best-practice to follow.

When you need to configure white-space to something other than pre, I recommend only setting it on <pre> tags. The <pre> tag communicates to HTML tooling that its whitespace should be fully retained because its preformatted. Even if you're actually using a different white-space mode in your CSS, the <pre> tag is at least a key signal to HTML tooling that all whitespace should be treated as significant.

Avoid Insignificant Whitespace in <pre> Tags

Fourth, always write <pre> tags with no leading and trailing whitespace, and without indentation. They should generally look like:

```
<div class="outer">
  <div class="inner">
<pre>Some preformatted text
with a new line
and some other content.</pre>
  </div>
</div>
```

Yeah, I don't like the look either. But this approach avoids the confusing leading/trailing newline behavior and prevents indentation accidentally leaking into the content.

None of these suggestions really help HTML tooling maintainers unfortunately since they still need to deal with whitespace regardless. But I warned you at the start, this post is partially for sharing knowledge and partially for venting about how awful whitespace in HTML actually is.

Broken? Yes.

Fixable? Not really.

Manageable?

Shout out to [Nathan Knowler](#) who helped me understand the layering between HTML and CSS, precise behavior of ` `, as well as some of the finer points of the `white-space` property.

Appendix

Whitespace of Flexbox Elements

Did you know that flexbox creates even more unique whitespace behavior? Here's the same code sample, with and without `display: flex;`.

Example 28: Block with `<div>` children.

```
<div>
  <div>First</div>
  <div>Second</div>
</div>
```

First
Second

Example 29: Flexbox with `<div>` children.

```
<div style="display: flex;">
  <div>First</div>
  <div>Second</div>
</div>
```

FirstSecond

`display: flex;` lays out in the inline axis by default, typically horizontally. Even though we have block elements in the `<div>` children, they are placed on the same line. Note the lack of visible whitespace between the elements. Since whitespace between block elements are treated as blocks of whitespace elements, they follow the same rule of "No empty blocks" and are removed, therefore the whitespace before the `<div>` children is considered insignificant.

Let's try the same thing with `` tags!

Example 30: Flexbox with `` children.

```
<div style="display: flex;">
  <span>First</span>
  <span>Second</span>
</div>
```

FirstSecond

Here we see that the flex behavior actually didn't change at all. The `` elements are still side by side *and* there is no whitespace between them. But don't inline elements preserve spaces between them?

This is because flexbox children create new block formatting contexts by default. [Quoting MDN again:](#)

A block formatting context is created by at least one of the following:

- [...]
- Flex items (direct children of the element with `display: flex` or `inline-flex`) if they are neither flex nor grid nor table containers themselves.

This means each child gets its own block rendering context, regardless of the default behavior of its tag. Therefore, whitespace between them is dropped, even though we're using `` elements which are natively `display: inline;`. Grid actually works the same as flexbox in this respect too.

What this tells us is that not just can CSS inform the whitespace processing behavior of an HTML element, but the CSS of a *different* element can have the same influence.

This is especially problematic if you're writing HTML fragments or partials. If you write a file which is not a full HTML page but just a small part of it, and start that file off with a `` tag, it is actually unknowable what the `display` value of that `` will be.

```
<!-- my-component.html -->
```

```
<span>I'm inline right?</span>
```

This is because you can't know what the *parent* of that `` tag will be. If the parent is a `<div>`, then it will use `display: inline;` like `` naturally does. If it is a flexbox or grid, then it will be `display: block;`.

```
<!-- my-other-component.html -->

<div style="display: flex;">
  <!-- Nope, you're a block now. -->
  ${render('./my-component.html')}
</div>
```

The correct answer depends on how that HTML partial is used and can even differ across multiple usages of the same partial!

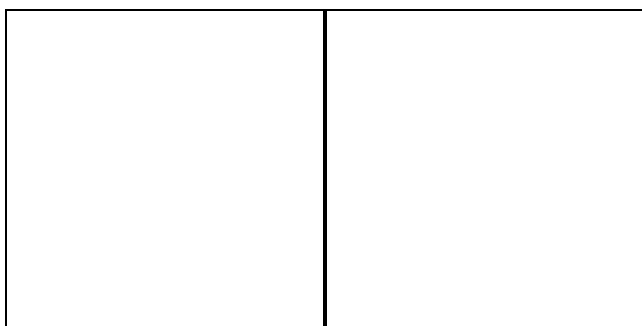
Whitespace of `inline-block` Elements

There's even *more* nuanced whitespace behavior for `inline-block`! `inline-block` elements are treated as inline on the outside and block on the inside. This means whitespace between `inline-block` elements can be significant just like it is for `inline` elements.

Let's look at [this example from MDN](#) with a couple boxes.

Example 31: <inline-block> with whitespace.

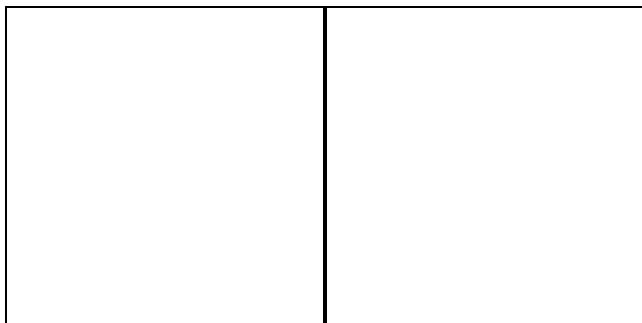
```
<style>
  li {
    display: inline-block;
    width: 2em;
    height: 2em;
    background: red;
    border: 1px solid black;
  }
</style>
<ul>
  <li></li>
  <li></li>
</ul>
```



You should see a small space between the two boxes. I've blown up the font size to make this space more visible. But where does that space come from? There's no margin or padding here. In fact, what if we change the HTML ever so slightly...

Example 32: <inline-block> without whitespace.

```
<ul>  
  <li></li><li></li>  
</ul>
```



Shout out to any developers who had to debug that spacing issue. And also shout out to Firefox which actually displays a "whitespace" element in its DevTools to make this somewhat visible.

** **

If ` ` doesn't work, what can you do instead? Ideally, you could just use a regular space since that would naturally avoid introducing the non-breaking behavior and allow it to be zero-width when line wrapped. However HTML whitespace collapsing prevents you from just inserting arbitrary spaces wherever you want, especially if you want multiple adjacent spaces.

My immediate intuition was to reach for ` `. If you didn't know, ` ` is a *named* HTML entity, however there are unnamed entities as well. Specifically, you can use `&#<num>` to get the character at that Unicode value. 32 happens to be the Unicode value of a standard space, so we can use ` ` just like a space character.

Unfortunately it's so equivalent that it's also subject to the same whitespace collapsing behavior. So you can put as many ` ` as you want, you'll still get at most only one space. My sincerest apologies to the three people reading this who want to put two spaces after the end of a sentence.

Example 33: ` ` spaces.

```
<div>Hello&#32;&#32;&#32;&#32;&#32;&#32;World!</div>
```

Hello World!

This behavior actually feels objectively wrong to me. Whitespace collapsing is a affordance for the developer experience so you don't have to butcher your HTML code to get a reasonable output. But if the developer hand-writes ` `, they clearly care about rendering a space in that slot and are giving up a convenient DX to do it. Whatever developer writes multiple ` ` almost certainly does not expect them to be collapsed together.

I wasn't able to come up with a compelling reason for why a developer would want ` ` to be collapsible, but I think I do understand why it works this way. As [mentioned earlier](#), CSS controls whitespace collapsing, not the HTML parser. This means the HTML parser needs to convert ` ` to literal spaces and retain *all* of them for all elements. It's then up to CSS to decide which spaces are significant. Because of that, CSS can't distinguish between a literal space and a ` ` entity, they're the same thing (Unicode character 32 to be precise).

I can see an argument that this actually is desirable for purposes of consistency. On a certain level, a literal space and ` ` *should* be indistinguishable and lead to the same behavior. However, I think that's a stronger argument for how the spaces are observable at runtime, not how the HTML parser interprets the code. I think it would be reasonable for the HTML parser to distinguish a space character and a ` ` entity to use different collapsing behaviors. For example, `<` and `<` are not the same because the former starts an HTML tag while the latter is a text literal for `<` which explicitly does *not* start an HTML tag. Unfortunately the HTML parser isn't the one doing the collapsing, so we've lost that information by the time CSS handles it. This "bug" with ` ` feels like a side-effect of the decision to allow CSS to influence whitespace behavior.

Isn't HTML Just a Document?

One potential advantage to the way HTML whitespace works is that it makes the format more accessible to non-developers. HTML was intended to represent documents, so just look for the text you want to modify and then update it to whatever new text you want to use. Don't worry about the syntax at all!

That sounds great in theory, but it's also not true. A core aspect of writing plain text is the spacing and formatting of that text. Yet HTML breaks even the most basic rules of how text can be formatted. Take this example of some text a non-developer might try to write without any knowledge of HTML and how it actually renders when thrown into a document unmodified.

34. Plain text.

```
First: Say "hello".
```

```
Second: Say "world".
```

```
Here's my shopping list for today:
```

```
> Apples
```

```
> Oranges
```

```
> Bananas
```

```
First: Say "hello". Second: Say "world". Here's my shopping list for today: > Apples > Oranges > Bananas
```

Everything gets rendered onto one line, `\n` is effectively meaningless and dropped entirely. `First :` had two spaces to align with `Second :`, but they are collapsed together. The list format is completely destroyed. This is all without mentioning that you should escape `>` into `>`, so I would argue it's almost a security issue to think of HTML in this mindset in the first place. This kind of use case requires `<pre>` tags which completely changes the behavior of everything.

Text documents have two intrinsic features to support formatting: spaces and newlines. HTML breaks both of those features with whitespace collapsing. You can't blindly copy-paste from any other text document into an HTML page and expect any kind of sane behavior. Where non-developers do write text content which is rendered in HTML web pages, they typically do it through a content management system (CMS), not by hand-writing HTML files anyways.

You can't think of HTML code like a plain text document.

Whitespace Control Characters

If you really don't like quoting strings in HTML, one potential alternative approach to is to borrow an idea from HTML preprocessors which have whitespace control characters. For example, [Nunjucks](#) uses `{% expr %}` to denote interpolations, but you can use `{%- expr -%}` to [trim whitespace](#) surrounding the content.

```
<div>
  Say hello to
  {%- username -%}
  and welcome them to the team!
</div>
```

Here, the whitespace around `{%- username -%}` is removed because `{%-` and `-%}` are used, meaning this will output:

```
<div>
  Say hello to Devel and welcome them to the team!
</div>
```

You could add a similar syntax to HTML to trim whitespace around particular elements, maybe `<- tag ->` and `<- /tag ->`? I don't know, that looks way worse than quotes to me...

I don't hate this approach, as it does remove the general ambiguity around whether any given space is significant or not and if that's more palatable to the community I could probably be talked into it some form of this being applied to HTML.

However it doesn't address the issue of interior whitespace such as multiple adjacent spaces or spaces from a CMS tool. We'd also still need to make the suggested changes to decouple CSS from whitespace-processing behavior.

I also don't care for this approach because I believe it shifts the burden of understanding whitespace to the developer. They need to recognize "Oh, in this situation I need to use a `<- tag>` because the whitespace after it is important, but the whitespace before isn't". I think just requiring quotes would make this much more obvious to most developers who would get their spacing right the first time, rather than opting into special syntax to fix the problem afterwards.

I do see the impact of quotes to the general DX of HTML and I understand the knee-jerk emotional reaction of a lot of developers. That said, I honestly believe quotes would be a net DX *improvement*, even if it does result in more typing.

Interesting post? Share it!



About the author

Doug is an overly opinionated software engineer on the Angular team. He is passionate about the web and developer tooling, hoping to simplify software development for everyone.

Reach out about anything: [Social links](#)

Fix a typo or help me write more gooder: [dgp1130/blog](https://dgp1130.com/blog)

© Copyright 2024, Douglas Parker. All rights reserved. Powered by Eleventy and Netlify.
Copyright notices and attributions. [Privacy policy](#).