TECH

# 10 CSS things I wish I knew when I was a beginner

Himayan Debnath
Aug 31, 2020 · 8 min read

During the initial days of front-end development, in between all the complex problems and solutions that JavaScript (and its frameworks) offers, many programmers tend to under-appreciate the role of CSS in a website.

The reason is quite simple — basic CSS is easy to grasp, and most of the hurdles you come across — you can find the solution in stack overflow. However, CSS problems can get tedious if the initial approach is flawed, and can also result in under-performing websites ruining the user experience.

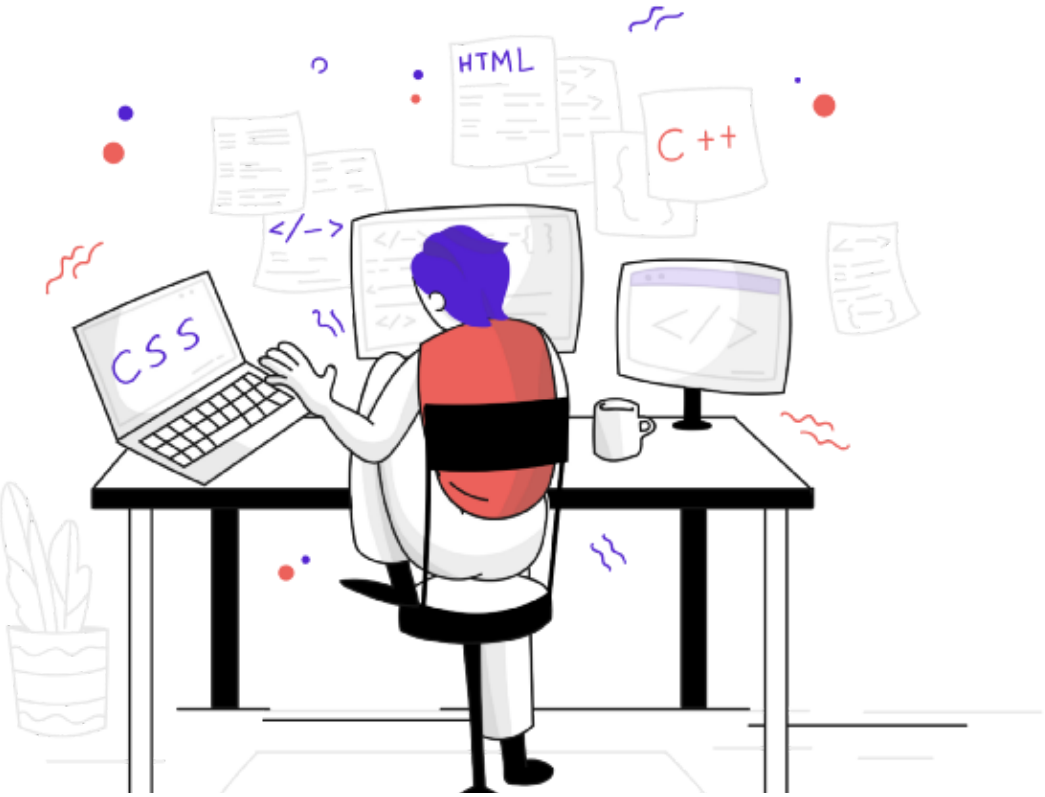


Illustration from pixeltrue

Based on what I have learnt from my mistakes, here are a few things I wish I knew when I began developing websites. The list is sorted considering learning curve and adaptability in mind — the easier stuff will come first.

## 1. Browser Developer Tools

This one might sound simple, but we tend to under-use Dev Tools when it comes to styling.

### Debug better

CSS styles get over-ridden multiple times in a complex website, specially if class names are re-used at different places. Before searching through the code trying to find out why the color of your button is not red, inspect it. It will save a lot of your time while debugging.


Inspect over-ridden styles

### Innovate on the fly

Dev Tools also lets you update styles on the fly. Wondering how it might look like in blue? Well, no need to wonder anymore.

Over-ride styles from Dev Tools

**NOTE** — To open up Dev Tools, just right-click anywhere on the page, and select **Inspect Element** and explore its illustrious list of utilities.

## 2. Wrappers and re-usable classes

This practice, if followed, will help you reap its benefits for years to come. Application development takes time to go from start to finish, and no design stays the way it was at the beginning.

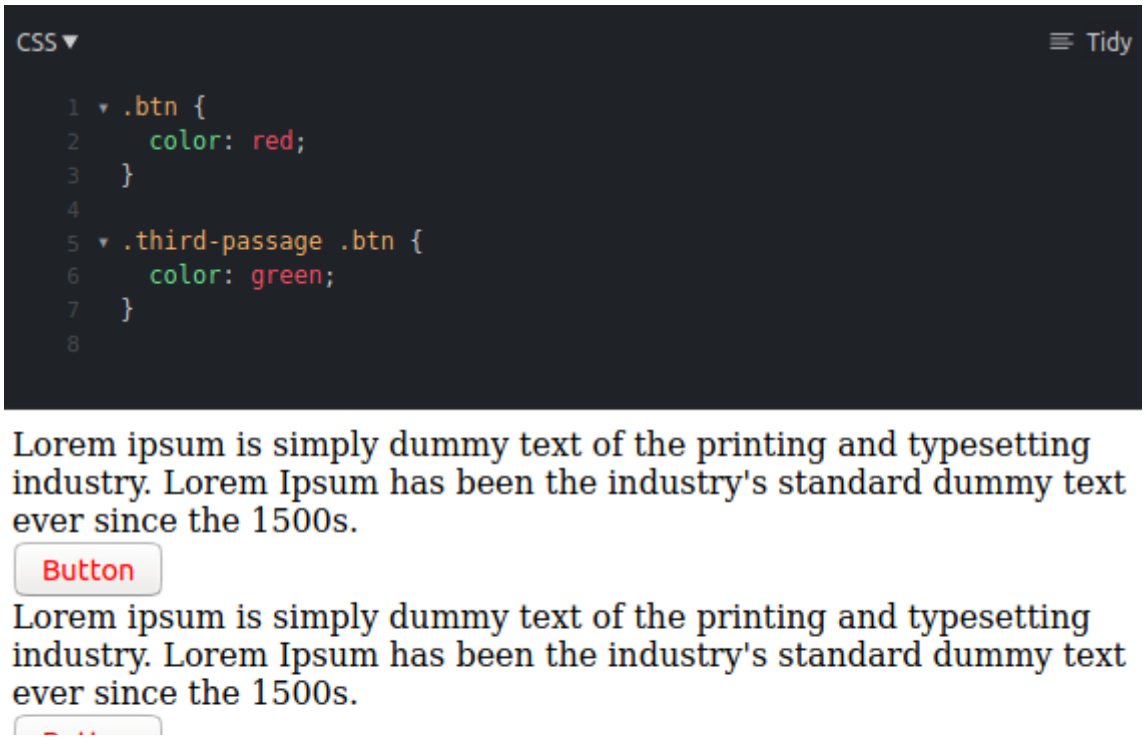
However, the core components would look the same in different pages for consistency, no matter what.

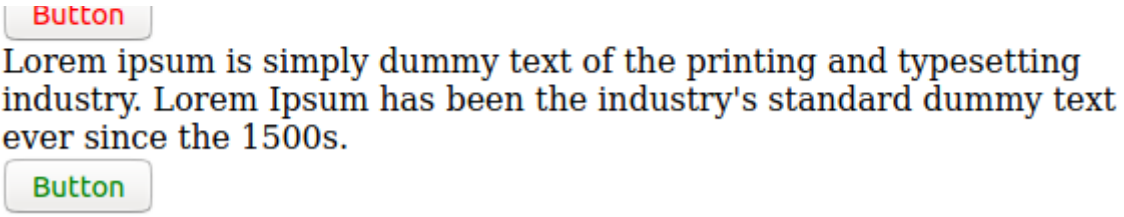
### Re-use classes

Identify the components which are re-used throughout the application (as for example — buttons, modals, headers, footers etc.) and write styles for them first. Re-use these classes wherever you need, so that even if the design changes, you would need to tweak minimum number of lines in your code. This way, you ensure that your code stands the test of time.

### Wrap to over-ride

There might be situations where you would need to change a few things in one of your core-components, just for a specific page. In such cases, make use of the wrapper class.





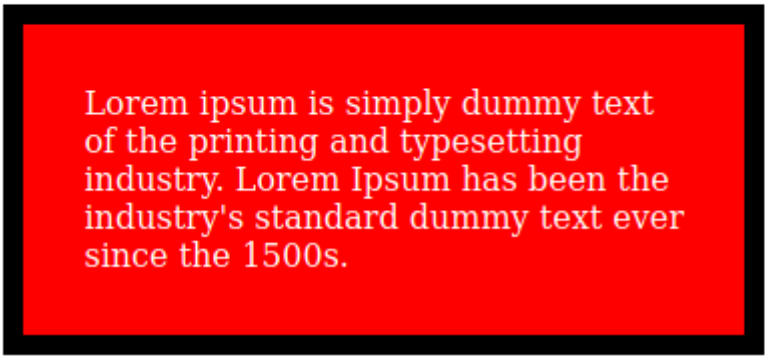

Wrappers and re-usable classes

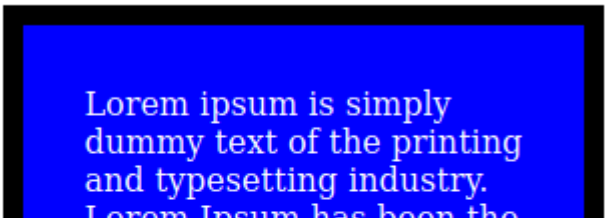
## 3. Property — box-sizing

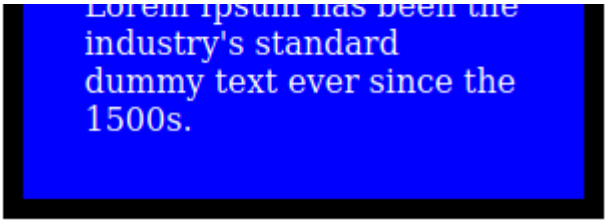
`box-sizing` determines how `height` and `width` of your element gets calculated. One of the more useful ones is `border-box`, which calculates the `width` of an element considering `content`, `padding` and `border-width`.

An example might help clear out the confusion.

```css
.box {
  width: 300px;
  padding: 30px;
  margin: 30px;
  border: 10px solid black;
  color: white;
}
.first-box {
  background-color: red;
}
.second-box {
  box-sizing: border-box;
  background-color: blue;
}
```






An example for box-sizing:border-box

By default, `box-sizing` value is set to `content-box` — which in case of `first-box` above, makes sure that the context width is set to `300px`. However, for the `second-box`, `border-box` ensures that `content-width + padding + border = 300px`. It is quite helpful when we don't want `padding` to interfere with element's defined `width`.

## 4. Mobile-first approach

This might seem a bit trivial at first, but turns out useful in the grand scheme of things, and is quite easy to adapt, if turned into a practice. Mobile compatibility is coming up higher and higher in the checklist of every web designer right now, and developers have two choices — **desktop first** and **mobile first**.

Let me explain the difference first —

```
// desktop first
.btn {
   padding: 10px;
}
@media only screen and (max-width: 768px) {
   padding: 5px;
}

// mobile first
.btn {
   padding: 5px;
}
@media only screen and (min-width: 768px) {
   padding: 10px;
}
```

The output of both these approaches would be exactly the same.

However, in the *desktop-first* approach, for a **mobile** device, it would load the common CSS first and then override those styles after looking into the media query. For a **desktop** though, it wouldn't bother to look at the media query at all, saving some

performance time.
But, this time would be helpful to a mobile more than a desktop, since desktops are generally better equipped.
That is why, it is recommended to follow a *mobile-first* approach from the start, even though it is understandably tempting to go desktop-first while coding.

## 5. !important is NOT important

The `!important` tag might seem very useful at start. As a new developer, I used to use it whenever I faced any challenge, and everything magically used to work. The `!important` tag essentially tells the browser to never let this line of code get over-ridden.

However, in most cases, you would want it to be over-ridden in the near future, and the only way to over-ride one `!important` is by using another one. In no time, your style sheet will be filled with `!important`, and that will stop you from re-using your styles. Instead of using the wildcard, try to find out why the style is not getting applied.

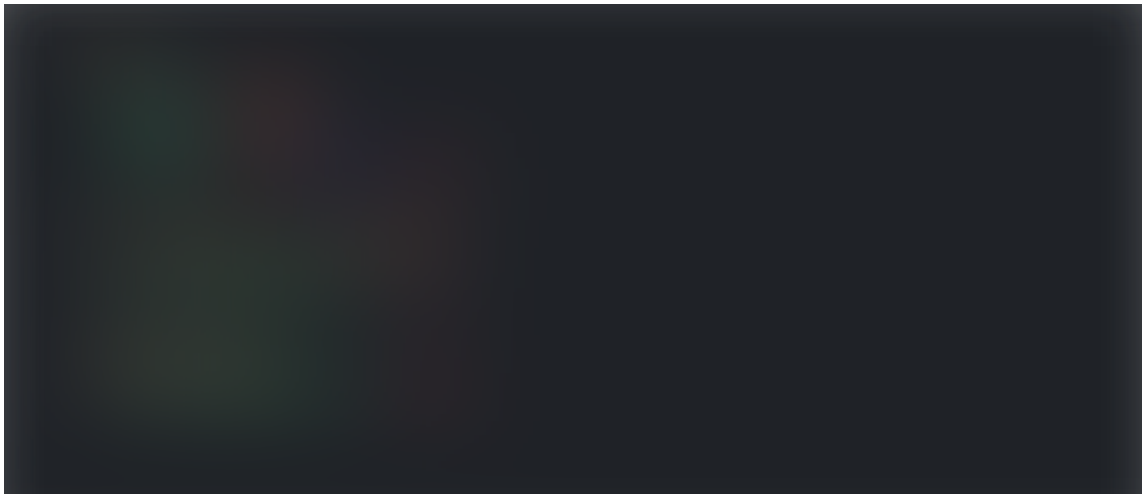
Use the `!important` tag when it is **really important**.

## 6. Positioning — absolute and relative

The concept of `absolute` and `relative` positioning is blurry, even for experienced developers. However, it is a very useful tool, if you know how to use it.
`absolute` positions an element relative to it's parent's position.
`relative` positions an element relative to it's own current position.

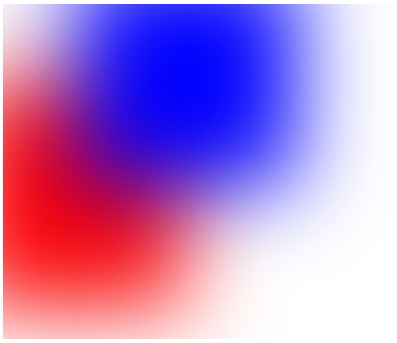
An example might help clear things up.

By default, position is set to `static` —



position: static

When we set it to `relative`, it does not affect the overall layout. A white gap is left at the space it used to occupy, while it moves `50px` from `top` and `left` —



position: relative

When we set it to `absolute`, it affects the overall layout. The space it used to occupy is now gone. I find this really helpful while creating pop-ups and tool-tips —

position: absolute

## 7. Transition and Transform

Transitions make user experience better. They are beautiful and elegant but only when done correctly.

Let's take an example of a progress bar here. As data is fed, it is supposed to go from 0 to 100. It can be done by manipulating the property `width`, but that would result in layout shift and can result in bad user experience. However, using `transform` and `scaleX`, something similar can be achieved which would minimize performance cost.

For more details — you can check out this course which explains the do's and don'ts when it comes to browser rendering optimizations.

## 8. Flex-box and CSS Grid

These are two very useful tools, which are widely popular in displaying an array of children wrapped under a common parent.

Since both of them cover very similar use-cases (such as centering elements, navigation bar, photo gallery), it depends on the developer which one he/she is more comfortable with. Just to provide more clarity, I will try to demonstrate how easy it is to build a very basic navigation bar using `flex-box` (which I prefer) —

### Navigation Bar

Top navigation bars generally have a brand icon on the left, and a range of links on the right.

Navigation bar with flex

There are multiple courses out there, which would help you understand **flex-box** and **CSS grid** . However, I found the following courses from **Wes Bos** really helpful, and hopefully, you would too —

**Flex-box** course — https://flexbox.io
**CSS Grid** course — https://cssgrid.io

## 9. Phase out Bootstrap from your pet projects

### This is going to be controversial

CSS libraries like Bootstrap are super helpful when you are a beginner. Bootstrap is really easy to use, and outputs (more often than not) meet our expectations. Add one class to a button, and it becomes a gorgeous blue one with color transitions on `hover` , `active` etc. It's great.

But, the deal breakers for me were —

1. It is really hard to customize. Over-riding default styles was a pain.
2. It comes with a lot of things, and in a typical pet project, I didn't even use half of it.

I would rather suggest you to write your own styles for your projects, instead of being dependent on a framework. That would help you learn and explore more.

## 10. SASS

CSS is tedious and it gets very boring very fast. Once you know the basics of CSS, don't wait to jump into SASS.

### SASS makes CSS fun

You can do a lot of things with SASS, which will make your life easy — like variables, nested syntax, mixins and a lot more. Start using SASS in your projects and it will help you style faster.

Check out this cheat sheet, to learn about all the features SASS provides.

So, that's it. These are the things I can think of, which I wish I knew when I was a newbie.
Thanks for bearing with me so far. I hope you find this article useful.

Find me on LinkedIn, Github or Twitter.

Cheers!


## References

https://www.mightyminnow.com/2013/11/what-is-mobile-first-css-and-why-does-it-rock
https://uxengineer.com/css-specificity-avoid-important-css
https://medium.com/@leannezhang/difference-between-css-position-absolute-versus-relative-35f064384c6

## Level Up Coding

Thanks for being a part of our community! Subscribe to our YouTube channel or join the **Skilled.dev coding interview course**.