The CSS Mindset

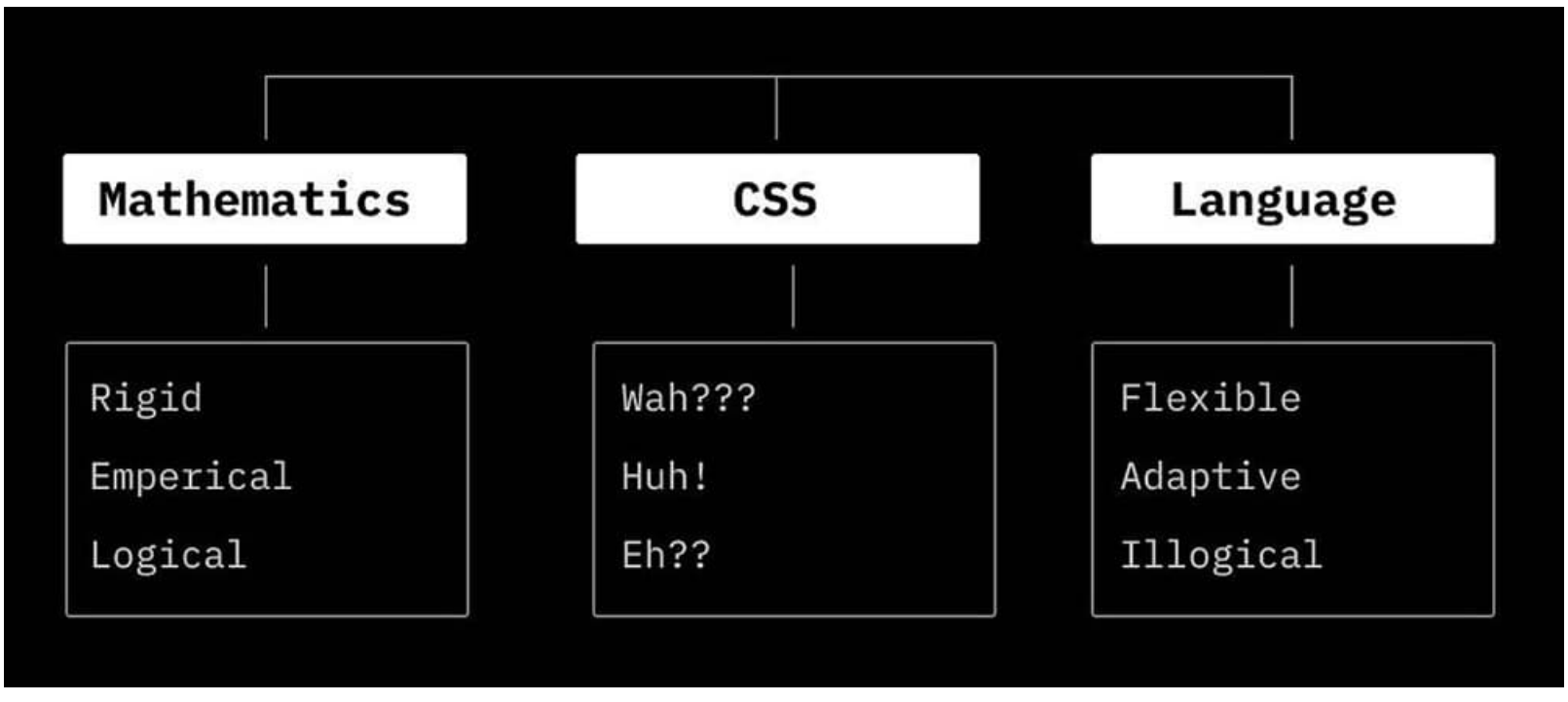
Max Böck 05 Jun 2019

Ah yes, CSS. Hardly a week passes without it being the topic of a heated online discussion. It's too hard. It's too simple. It's unpredictable. It's not a real programming language. Peter Griffin struggles with blinds dot gif.

I don’t know why CSS sparks so many different emotions in developers, but I have a hunch as to why it can sometimes seem illogical or frustrating: You need a certain **mindset** to write good CSS.

Now, you probably need a mindset for coding in general, but the declarative nature of CSS makes it particularly difficult to grasp, especially if you think about it in terms of a “traditional” programming language.

Robin Rendle makes a very good point in this CSS-Tricks Newsletter where he finds that CSS lives somewhere between rigid, logical systems like Math and flexible, adaptive systems like natural languages:



Comparison of Math, CSS and Language by Robin Rendle, 2019

Other programming languages often work in controlled environments, like servers. They expect certain conditions to be true at all times, and can therefore be understood as concrete instructions as to how a program should execute.

CSS on the other hand works in a place that can never be fully controlled, so it has to be flexible by default. It’s less about “programming the appearance” and more about translating a design into a set of rules that communicate the intent behind it. Leave enough room, and the browser will do the heavy lifting for you.

For most people who write CSS professionally, the mindset just comes naturally after a while. Many developers have that “aha!” moment when things finally start to click. It’s not just about knowing all the technical details, it’s more about a general sense of the ideas behind the language.

This is true whether you write CSS-in-JS, Sass or plain vanilla stylesheets. The output will always be CSS - so knowing these concepts will be helpful regardless of your tooling choice.

I tried to list some of them here.

Everything is a Rectangle

This seems obvious, given that the box model is probably one of the first things people learn about CSS. But picturing each DOM element as a box is crucial to understanding why things layout the way they do. Is it inline or block level? Is it a flex item? How will it grow/shrink/wrap in different contexts?

Open your devtools and hover over elements to see the boxes they're drawing, or use a utility style like `outline: 2px dotted hotpink` to visualize its hidden boundaries.

The Cascade is your Friend

The Cascade - a scary concept, I know. Say “Cascade” three times in front of a mirror and somewhere, some unrelated styling will break.

While there are legitimate reasons to avoid the cascade, it doesn’t mean that it’s all bad. In fact, when used correctly, it can make your life a lot easier.

The important part is to know which styles belong on the global scope and which are better restricted to a component. It also helps to know the defaults that are passed down, to avoid declaring unnecessary rules.

As much as necessary, as little as possible

Aim to write the minimal amount of rules necessary to achieve a design. Fewer properties mean less inheritance, less restriction and less trouble with overrides down the line. Think about what your selector should essentially do, then try to express just that. There's no point in declaring `width: 100%` on an element that's already block-level. There's no need to set `position: relative` if you don't need a new stacking context.

Avoid unnecessary styles, and you avoid unintended consequences.

Shorthands have long effects

Some CSS features can be written in "shorthand" notation. This makes it possible to declare a bunch of related properties together. While this is handy, be aware that using the shorthand will also declare the default value for each property you don't explicitly set. Writing `background: white;` will effectively result in all these properties being set:

```
background-color: white;
background-image: none;
background-position: 0% 0%;
background-size: auto auto;
background-repeat: repeat;
background-origin: padding-box;
background-clip: border-box;
background-attachment: scroll;
```

It's better to be explicit. If you want to change the background color, use `background-color`.

Always Be Flexible

CSS deals with a large amount of unknown variables: screen size, dynamic content, device capabilities - the list goes on. If your styles are too narrow or restrictive, chances are one of these variables will trip you up. That's why a key aspect in writing good CSS is to embrace its flexibility.

Your goal is to write a set of instructions that is comprehensive enough to describe what you want to achieve, yet flexible enough to let the browser figure out the **how** by itself. That's why its usually best to avoid *"magic numbers"*.

Magic numbers are random hard values. Something like:

```
.thing {
    width: 218px;
}
```

Whenever you find yourself tapping the arrow key in your devtools, adjusting a pixel value to make something fit - that's probably a magic number. These are rarely the solution to a CSS problem, because they restrict styles to a very specific usecase. If the constraints change, that number will be off.

Instead, think about what you actually want to achieve in that situation. Alignment? An aspect ratio? Distributing equal amounts of space? All of these have flexible solutions.

In most cases, it's better to define a rule for the intent, rather than hard-code the computed solution to it.

Context is Key

For many layout concepts it's imperative to understand the relationship between elements and their container. Most components are sets of parent and child nodes. Styles applied to the parent can affect the descendants, which might make them ignore other rules. Flexbox, Grid and `position:absolute` are common sources of such errors.

When in doubt about a particular element behaving different than you'd want it to, look at the context it's in. Chances are something in its ancestry is affecting it.

Content will change

The number one mistake made by designers and developers alike is assuming that things will always look like they do in the static mockup. I can assure you, they will not.

Strings may be longer than intended or contain special characters, images might be missing or have weird dimensions. Displays may be very narrow or extremely wide. Those are all states you need to anticipate.

Always be aware that what you see is just one UI state in a bigger spectrum. Instead of styling the thing on your screen, try to build a "blueprint" of the component. Then make sure that whatever you throw at it won't break your styling.

Find Patterns and re-use them

When you set out to turn a design mockup into code, it's often helpful to take inventory of the different patterns included first. Analyse each screen and take note of any concept that occurs more than one. It might be something small like a typographic style, or large like a certain layout pattern. What can be abstracted? What's unique?

Thinking of pieces in a design as standalone things makes them easier to reason about, and helps to draw the boundaries between components.

Use consistent Names

A surprisingly large part of programming in general is coming up with good names for stuff. In CSS, it helps to stick to a convention. Naming schemes like BEM or SMACSS can be very helpful; but even if you don't use them, stick to a certain vocabulary. You'll find that certain component patterns come up over and over - but is it called a "hero" or a "stage"? Is it "slider" or "carousel"?

Establish a routine in how you name parts of your UI, then stick to that convention in all your projects. When working in a team, it can be helpful to agree on component names early on and document them somewhere for new team members.

All these things were important for me to understand, but your personal experience as to what matters most might be different. Did you have another "aha" moment that made you understand CSS better? Let me know!

Update: I did a talk about the CSS Mindset at CSS-Minsk-JS in September. There's also a video recording available, if you prefer that.

Further Reading

- How to learn CSS by Rachel Andrews
- The Secret Weapon to learning CSS by Robin Rendle
- CSS doesn't suck by Andy Bell