Edit Manage Stats

Ingo Steinke
Posted on Mar 28

Understanding the Benefits of "Quirky" Web Languages

#webdev #javascript #programming #beginners

This is a post to help beginners learn to understand the peculiarity of web technologies. I will share some basic concepts from my own perspective as an early adopter of HTML, JavaScript, and CSS. Tutorial, talks, and even songs and poems that might help you get a more natural feeling about those quirky old web languages.

Quirks, Inconsistencies, and deliberate Fault-Tolerance

Apart from the unintended quirks and inconsistencies betraying a chaotic and controversial history before the current standards and recommendations, HTML and CSS are special due to their deliberate fault-tolerance, following the principle of robustness.

More than One Way to to it

Quoting an old Perl motto: in many programming languages, there's more than one way to do something. Code style, paradigms, patterns, naming things, and the choice of

tools and frameworks.

Code style may have become the slightest problem, a matter of taste and aesthetics, also of readability, but mostly solved by conventions and recommendations and automatable using linters and IDEs. Paradigms and patterns can also depend on taste, experience, and requirements, but there some rather universal "design patterns" useful to know for any kind of software development.

Strictness and Fault-Tolerance

But there is a more fundamental dissent than the preference of tabs, spaces, or how to align braces and brackets. Many people love programming for its clarity and abstractness, while others, especially web developers, have learned to code using imperfect tools, hacks, and shortcuts that look like erroneous code to those with an academic background or preference.

The product logos in this article's cover image include different languages and technologies some of which are still relevant for web development today: HTML, CSS, JavaScript / ES / TypeScript (and the DOM), SVG, PDF, PHP, SQL / mySQL / MariaDB, mongoDB, and Node.js, the most successful server-side implementation of JavaScript so far.

For the sake of simplicity, I won't divert into the topic of syntax variations and extensions in this post. TypeScript, CoffeeScript, Sass, and less will be covered in another article soon.

But does anyone still remember Perl, Flash, and ActionScript? And do you recognize the similarity of React and ColdFusion, or at least the similarity of their logos?

The Secret to Success

Some technologies come and go, while others are there to stay. But why? HTML, JavaScript, and CSS have come a long way, and apart from built-in browser support, there might be another secret to their success.

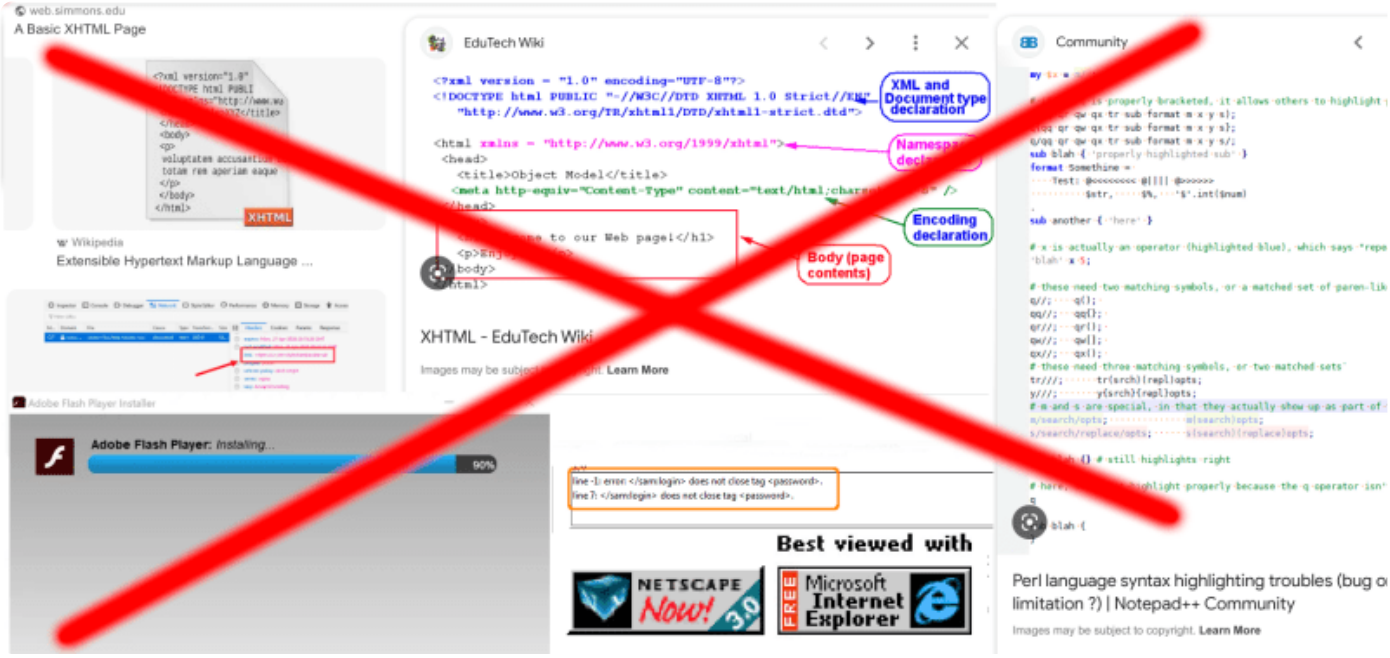
Web technology, at least in the front-end, has mostly strived to follow the robustness principle, also known as Postel's Law. There have been a several, more or less valid, reasons to do so, and some kind of dead-end after abandoning robustness with XHTML, based on XML. The need for a well-formed document structure means that we might have to discard a useful and usable piece of information if it contains a single invalid character.

UTF-8 to facilitate Global Collaboration

When the "western" character set, standardized as ANSI, got superseded by UTF-8, an unofficial ANSI extension introduced by Microsoft was defined as invalid. A handful of characters did not conflict with anything else and there were similar characters in the new encoding, but according to official standards, there was no valid way to convert those characters to their corresponding UTF-8 codepoints.

Nerdy, fragile, unreadable: some technology has vanished for good.

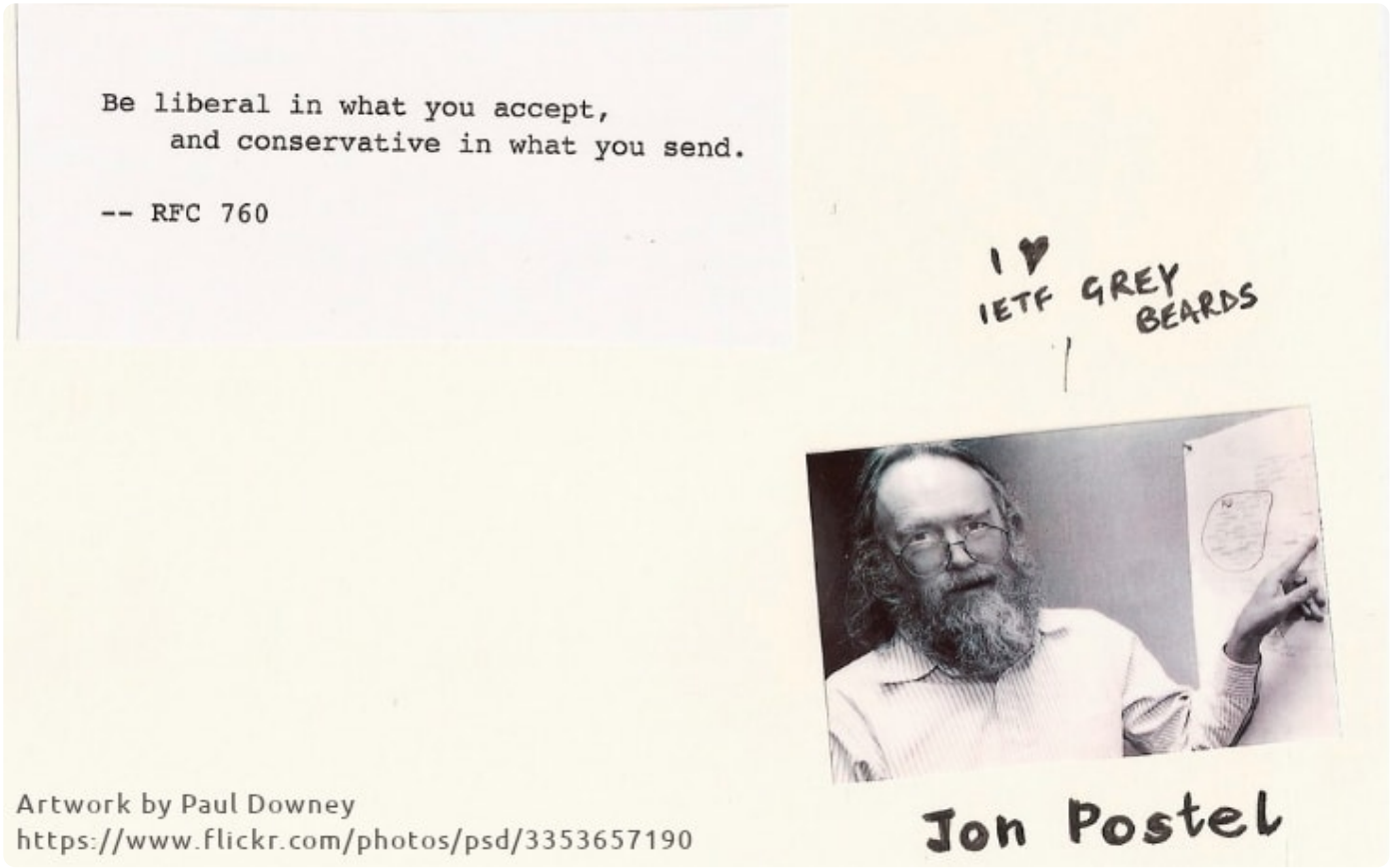
I have been working with SOAP interfaces which did break due to formally invalid characters several times, causing overtime work and hotfixes of a running system. I also remember when storage was scarce, and I needed a batch of floppy disks to store large files. The disk spanning process was prone to break at a single error as well, which - according to Murphy's law - seemed most likely to happen when inserting the last disk to complete the restoring process. I dislike indention syntax (like in Python and YAML) and JSON files for the same reason.



Fault-tolerant processing can help to preserve valuable data despite errors, and it can save storage or bandwidth by omitting redundant data.

Some practices have become obsolete for a good reason, as clarity and maintainability is obviously more important than optimizing some milliseconds of loading time.

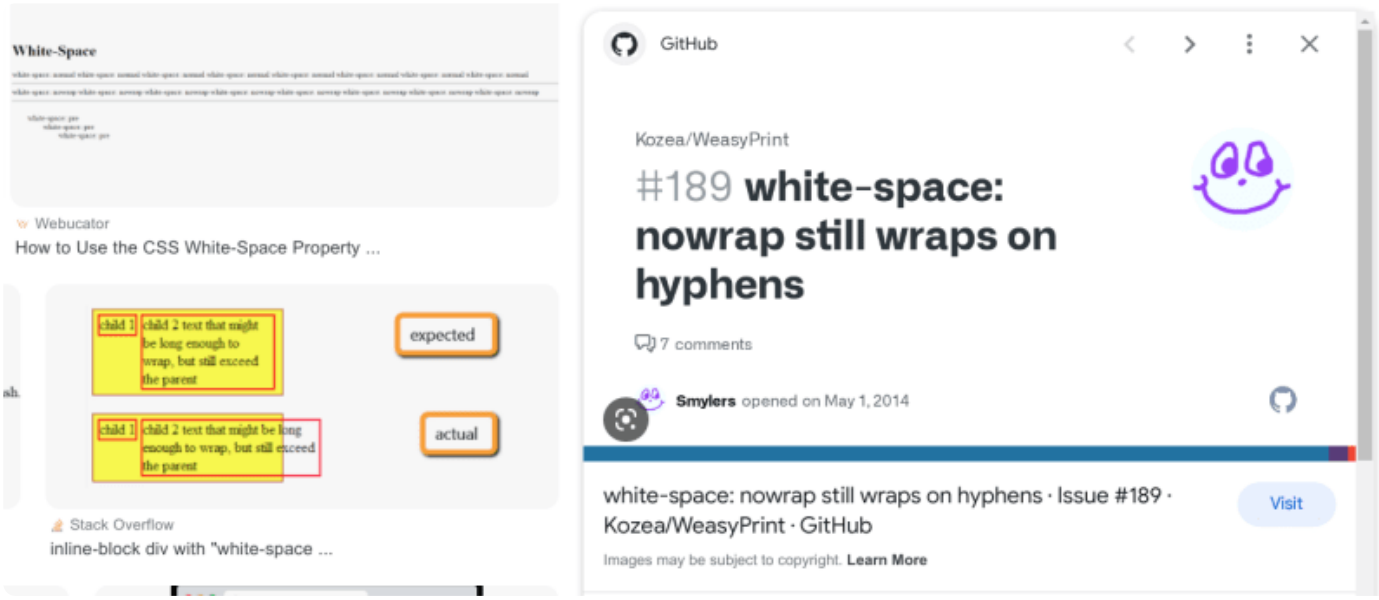
As we can see, there are different types of non-strict behavior. I am not advocating for weak or untyped variables, although that's another aspect of web technology in its original form and the reason why we needed TypeScript and PHP Standards Recommendations (PSR). Fault tolerance might invite people to overoptimize, making code harder to understand and introduce new potential sources of error. But that can be fixed by advocating strict and orderly source code, which might or might not be transpiled to a more compact output. Actually this is the main point of robustness: "**be liberal in what you accept, and conservative in what you send**".



Strict + Tolerant = Principle of Robustness

Robustness should not be an excuse or invitation for writing sloppy code, but instead make clients resilient against possible mistakes, but also against syntax variations and upgrades. Unlike JavaScript, which, despite its sloppiness, is still the strictest and fault-intolerant one of the classic front-end web technologies, HTML and CSS allow to be extended by future syntax, so clients must ignore elements that are formally correct but unknown to the current implementation. This syntactical principle is a foundation to add custom tags and define web components today.

Where to put the Hyphen in white-space: nowrap



Unfortunately, unlike optional features of HTML, CSS, and JavaScript, history has left its syntactical mark in ways that leave us no choice but to learn or rely on documentation and code completion. Like in natural languages, inconsistencies that seem illogical and inconsistent to adult learners, feel right and natural and go unnoticed by early adopters. While I still struggle to memory some CSS flex and grid properties, I know where to put the hyphens in "white-space: nowrap" and I can even tell you that #a31e04" is a reddish color just by looking at the hexadecimal notation. While CSS in particular might be a quite peculiar language, I love it for being mostly declarative, and of course for its robustness features as well.

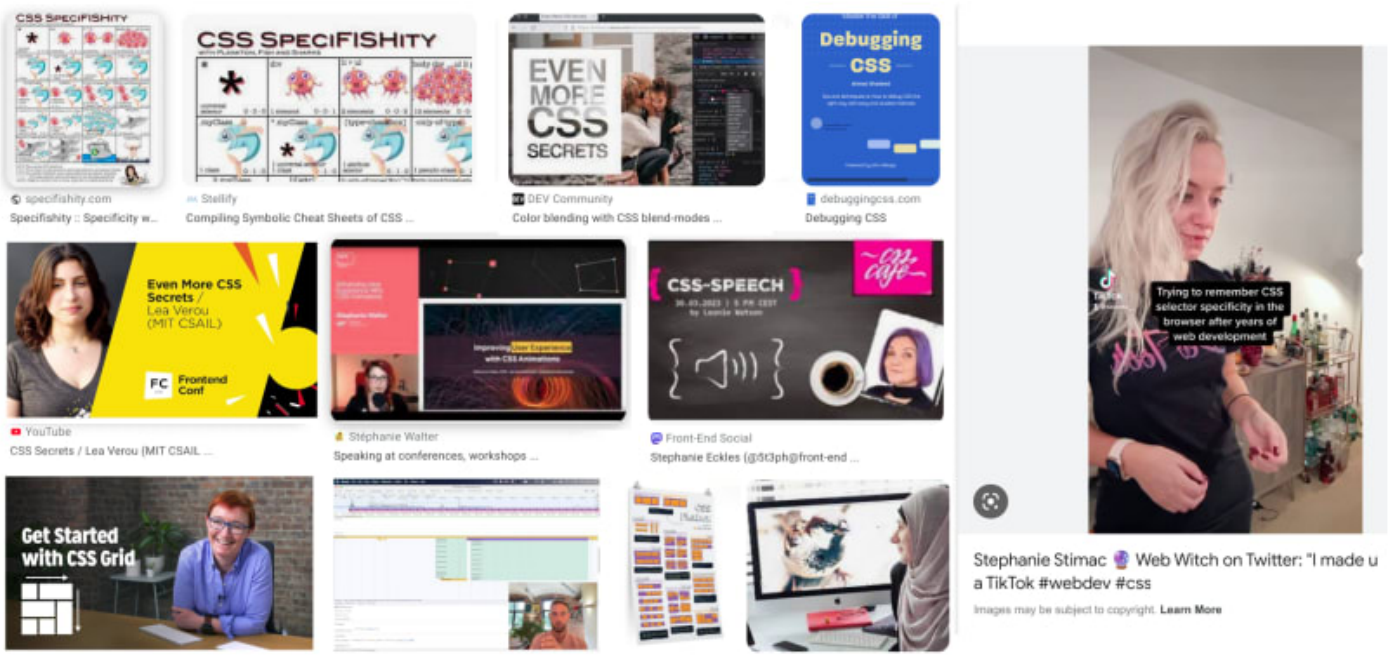
Learn to Love those Quirky Old Languages

How can we learn to love those quirky old languages? Well, we can start by regarding them as such. Quirky old languages, like ancient Greek, Latin, or the fictional ones invented by authors like J.R.R. Tolkien. But maybe I should rather compare CSS and HTML to a spoken language, which is still quite ancient and full of inconsistencies: English. Non-native speakers will know, Britons and Americans might guess when changing perspective: have you ever wondered how to pronounce boom, zoom, choose, or footer? Why do we even have to look that up or listen carefully in the first place?

Talks, Tutorials, Songs, and Poetry

In natural languages we have songs and poetry, or a proverb that our grandparents or teachers told us. In web development, we have great talks and tutorials, and we even have songs and poetry as well. If you are a student or a freelancer alone at your desk, put on your headphones and watch all those beautiful videos, and if you can, go to talks and conferences as well. There are great lectures at FrontendMasters, Udemy,

and YouTube, and there are DevToks (not only on TikTok) where you will find the songs and poems about web development mentioned before.



Some names and links: Lea Verou, Sara Soueidan, Sara Drasner, Heydon Pickering, Michelle Barker, Stephanie Stimac, Stéphanie Walter, Ahmad Shadeed, Rachel Andrew, and so many more. There are also excellent demos to be discovered on CodePen and elsewhere, and some graphical tutorials like the legendary SpeciFISHity tutorial.

Coding with all of our Senses like a Craftsperson

So we can see, read, and hear our code, and there have even been reports of "code smell". But to get in touch with coding, we have to start typing and enter letters, numbers and symbols using our own hands and review and refine the results of our work and discover the artisan or even artistic aspect of web development.

Conclusion

It doesn't matter if you see web technology as "programming", but it might make them easier to learn if you see them as natural languages that you have to discover with all of your senses to get a "natural" feeling how to use them correctly.

---

fruntend • Apr 3

Congratulations 🥳! Your article hit the top posts for the week - dev.to/fruntend/top-10-posts-for-f...
Keep it up 👍