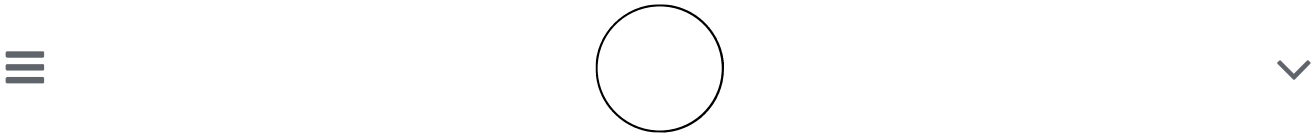

Back

CSS interview questions and answers

29 Apr 2014
Thoughts

A personal take at a fictitious interview.

While browsing through my starred GitHub repositories, I noticed an interesting one I had forgotten about: Front-end Job Interview Questions.

Only once have I had to provide a CSS layout for a job application. But I also happened to rate candidates based on a CSS exercise I set up when Steaw was looking for additional Front-end Developers.

I thought it'd be both entertaining and stimulating to try myself at a fictitious interview based upon the CSS-related questions of that repo. I'll play by the implied rules: answer directly, without putting too much thought into it, write as I'd speak, and leave the potential mix-ups or innacuracies I'd drop in such a situation.

TL;DR: I agree, it's long. But it reads quickly considering I wrote it quickly.

Describe what a "reset" CSS file does and how it's useful.

It's meant for cross-browser consistency. Basically, each browser provides its own user-agent stylesheet that acts as the default one if no stylesheet is present. So you have default font-sizes, margins, colors, paddings… They're all slightly different for each browser. So instead of fighting with these differences when writing your own CSS, you just "reset" them as much as you can. It's like a solid foundation on which you can define your own styles.

I've usually used either Eric Meyer's CSS reset, or the Yahoo one. I just copy paste it on 1 line and forget about it. Recently, I use an HTML5 one that I

found somewhere, and that works quite well. It's a usual "set it and forget it".

Describe Floats and how they work.

Ah, floats… Probably the least understood CSS property. It's hard to handle it because it not only affects the element you're targetting, but also the surroundings: the parent, the siblings, the following elements…

What it does is "push" an element to the side you've chosen, either left or right. That's why you don't have a float: center. It's simple to understand and see how it affects the targeted element. But the issues come from the surroundings: a parent with only floated childs will have a height of zero, as if the children weren't there. The text that follows a floated element will be wrapped around it. If you have float: left, the text will use the remaining space on the right, and then the one remaining below. Visually, it looks like stairs. But if you inspect it, you can see that the element containing this text is actually *behind* the floated element. So, it feels weird.

The thing is, a float will take an element *out* of the flow, a bit like position absolute or position fixed but not exactly. I say not exactly because the surrounding elements are mostly aware that the floated element is still *there* (they will wrap around it), but the parent won't.

The float is directly related to the clear property, which reintroduces the floated element in the flow, by placing a cleared element *after* the floated one. There are other clear techniques, like putting a float or an overflow visible on the parent, but it only delay the issue. So I usually use a cleared element.

Describe z-index and how stacking context is formed.

z-index is for visual depth. Because some elements overlap, because some

elements are nested within each other, there's a natural visual depth that builds, where you can see some elements on top or behind other ones.

The natural depth depends upon a couple of things:

- if an element is nested *within* another one, then it shows up on top of its parent
- if a sibling comes *after* another sibling, it's on top as well

Position absolute or position fixed will mess with that: they'll show up on top *regardless* of their tree position. Well, sometimes… I'm not sure. But because it's sometimes a burden to move elements in the HTML tree, or because the depth behavior can be unreliable, you can use the z-index property to make sure how elements will "stack" upon each other (when they overlap obviously). By the way, you need an element to be positioned (relative, absolute, fixed) for the z-index to work.

The z-index value is relative. So you can have one element with z-index: 1 and another with 2, and the latter will show up on top. Same thing would happen with values of 1000 and 4875.

What I usually do is regroup all the z-index values at the end of my CSS, in order to adjust the values according to the other elements' values. I start with the highest value so that my selectors order reflects the visual depth order. Also, I put multiples of 100 as values, so that if I have to insert up to 99 elements between 2 others, without having to readjust all the other values. In practice, I usually insert 2 or 3.

What are the various clearing techniques and which is appropriate for what context?

The obvious one is by using the clear property, which can have values of none, left, right, or both (left and right). Another one is to float the parent element. Another I've heard of is put overflow: visible on the parent, or

something like that.

In the end, I just rely upon the clear property, because that's what it's meant for. Sometimes I put it directly on an element, sometimes I use a dedicated class for that (the famous "clearfix"), and sometimes I use the :after pseudo-selector.

The thing is: the clear property does *only* that. You rarely apply it along with other properties. Sometimes, it can actually clash>

For example, you have a ul with lots of li's that are floated left, have some height, some background, some padding… You target them by using a ul li selector (or .whateverclasstheulhas li), because you want to make use of the hierarchy selector and not but a class on each li.

Because the li's are floated, the ul will have a height of zero. What do you do?

- float the ul and put a cleared element after that?
- use a clear on ul:after, and cross your fingers for browser compatibility?
- put a clear on the last li?

In every case, the clear property will be used. But for the second option, it will clash with the styling of the li. So you'd need to reset that *only* for that cleared li. It's usually a pain. So you either don't use the hierarchy selector, or don't use li's, or just float the ul and add a "clearfix" div after that.

Because I often work with developers who just want some practical solutions, I have a .clearfix class in my CSS that does just that. It might be semantically irrelevant, but it get things done.

Explain CSS sprites, and how you would implement them on a page or site.

A CSS sprite is a big image containing several background images at once. It's a collection of images combined in one big image. You choose the background image you want to show by playing with the dimensions of the element (which acts like a viewport) and the background position.

Advantages are:

- bandwidth: you just have 1 HTTP request for one image, and the combined image is often lighter than separate images combined
- rendering: as soon as the sprite is called and displayed once (and even cached), you can re-use it with other elements. Works well with the :hover pseudo-state. There's no blinking.
- maintenance: it's easier in Photoshop or Sketch to export a single asset with everything

It's not a new technique. I remember old games having sprites. Even Mario Kart 64 has sprites for the cars!

Because a sprite is an image that's meant to be used on a lot of elements, I consider it a shared property. So I regroup all selectors that use the sprite, and but my CSS code on 1 line for that (with some text-replacement code just in case).

Sprites are often used with icons of the same dimensions. Although you can actually build a sprite for *all* the background images of your site, of various dimensions, it's a pain to maintain. Plus: sprites are mainly used for non-repeated background images. I've done some with horizontally *or* vertically repeated (but not both) images, but it has its limits.

So I usually have only sprites for icons now. There's a tool to generate the sprite: you upload a zip of icons with the same dimensions, set the offset, set a CSS class, and it generates both the sprite *and* the CSS background positions for each selector. It's amazing.

What are your favourite image replacement techniques and which do you use when?

I stick to the one I first learned about: overflow hidden and a huge negative text-indent. I actually use -290486px because that's my date of birth, so it acts like a CSS signature. It's silly but considering you can put any value, why not make it a bit meaningful? Sometimes I Google it and find people who copy-pasted my code. I don't mind: that's what it's meant for.

There are probably other image replacement techniques, but I never bothered to look for an alternative. I mean: this one has worked for years, and I know how to handle it.

How do you serve your pages for feature-constrained browsers?

I guess you mean all that vendor-prefixed properties? There's this term, "progressive enhancement", and its counterpart, that I don't remember actually. It's that permanent question: should I code for the older but still used browsers, and add some magic for the more modern ones, *or* make it look great for the modern ones, and fix it with hacks for the older ones?

I prefer the second approach. It's more future-proof: eventually, the browser market will catch up and you won't have to update anything because the hacks you implemented won't be needed, and won't even be actually called. Plus: it's less frustrating. You can actually design stuff the way you want, make it both look better and with cleaner code. Then I test it on IE to see what's going wrong, call an ie7.css file thanks to the MS conditional tags, and add some hacks there.

That's for really basic CSS properties, that should be available in all browsers *today*. For more advanced or funny styling, I sometimes check the Webkit or Mozilla documentation to see what my browser is capable of.

I love caniuse.com by the way. Stupid designers will say it's ugly. It's actually very well designed: it's quick, well documented, easy to read and scan, and it gets straight to the point. At a glance, I know if the CSS property I intend to use is supported or not.

In the end, if it's a personal project, I won't bother testing with IE. If it's for a client, I'll ask him to test. IE has come a long way, and considering I've had years of IE6 hacking, I'm glad IE 10 is getting there.

What are the different ways to visually hide content (and make it available only for screen readers)?

Well it's the same image replacement technique I mentioned earlier: overflow hidden and text-indent -290486px. The text is there in the code, but doesn't appear on screen. That's another reason why I still use this technique: I'm pretty sure it doesn't affect the screen readers or the SEO.

Have you ever used a grid system, and if so, what do you prefer?

Only with Bootstrap actually. In general, I don't like grid systems because they transfer the styling *from* the CSS *to* the HTML, and my Golden rule is to respect a division between the content (HTML) and the layout (CSS). For example, I'd rather have .button-signup and a .button-upload, and combine them in a selector, rather than having a .button-primary used in both cases because I'd have to dig into the HTML to change the styling, and that's not the point of CSS.

So for a grid system, you'd have things like .col-3 and .col-9 for a sidebar and a "main", in a 12 columns layout. First, the styling is in the HTML. Second, what if you want to put that sidebar on the right and not the left? Well you'd have to modify your HTML structure, instead of changing 2 words in your CSS.

Also, when coding a responsive website, you'd be stuck with 2 columns in this case. You can't go from 4 to 3 to 2 to 1 column, depending upon the space available. You'd have either 2 or 1 column.

In the end, I use a grid when I use Bootstrap, but only because Bootstrap comes with tons of other stuff.

Have you used or implemented media queries or mobile specific layouts/CSS?

Yes, a lot. I love it. It's like a challenge actually. At first, I was just playing with it and thought it was magic. I'd design my desktop website, finish coding it, and then add some rules to make it decent on mobile.

Nowadays, I try to go the "mobile-first" way. It's quite easy: you just follow the HTML flow. But then the desktop layout can end up feeling like a watered down version of what you could have achieved *if* you'd started with that version first.

In any case, I really enjoy media queries. I tend to stick to min-width and max-width though.

Also, I've had issues with synchronizing these media queries values with the ones I use in jQuery. For some reason, some browsers include the scrollbar in the calculated width, some others don't. So it can be tricky to trigger the responsiveness in both you CSS and your jQuery.

Any familiarity with styling SVG?

SVG directly in the HTML code? Not so much. I've seen some people playing with the path and fill attributes, and animating it. It's quite fun. What I like is that you can write the colors in CSS, and avoid color profiles issues! That's really interesting.

I've had SVGs in image tags as well. Had color profile issues so I try to avoid that now.

I haven't tried SVGs much in CSS background-images.

The best use of SVG I've found is icon fonts. There's Icomoon, which is an app where you can upload your SVGs or choose existing one, and generate an icon font. And of course there's Font Awesome, which is *really* convenient. In both cases, what I like is the fact that I can use text CSS properties to style the icons: font-size, color, text-shadow, line-height, vertical-align… And because it's vectorial, it looks great at any size. It's perfect for Retina screens for example.

How do you optimize your webpages for print?

I don't, really. I think I only wrote once a CSS for print specifically. I don't even remember how to do it. Maybe a media query? Or an attribute in the link tag? Something like that.

What are some of the "gotchas" for writing efficient CSS?

I have a few that I've developed over the years. They're quite personal, but in any case I don't think there's a real standard for writing CSS.

Here are a few rules I follow:

- I write my CSS in line. I must be part of a very small minority. The thing is: I absolute love information density. I want to see the CSS structure in front of me, so I make use of the all the horizontal space. When writing 1 property per line (as everyone), it's easy to read the properties. But I don't care about the properties: I know them. I want to read the **selectors**. So I have one set of properties per line, with 1 or multiple selectors. Sometimes I put selectors on several lines, sometimes not. Depends.

- to be able to scan my properties quickly despite the fact that they're on single line, I put them in alphabetical order. It works pretty well with time. I know roughly where to look to find a margin or a width or a text-transform.
- group shared properties with selectors. If several elements share a same set of properties, you could have dedicated class, or repeat yourself, or use a mixin. What I do is just list all the selectors that share this set. Then, I can have specific styles for each of these selectors. *Way* better than having a dedicated class.
- I only use classes. I keep ids only for jQuery stuff. CSS priority is hard to graps, and ids are powerful selectors. They can take over the styling quite easily. To avoid headaches, I just use classes. Ids are meant to be used for unique elements. But nothing prevents from using classes like that. There's no semantic in CSS. So use .header instead of #header, and you'll avoid priority issues. Especially true when having #nav li, because if a li has #nav-home, you can't use *just* #nav-home and it's stupid because it's an id selector, so the uniqueness should work. But it doesn't because #nav li is stronger. So you have to do #nav #nav-home, which looks really stupid.
- Here's how I order my styles: reset, fonts, generic tags, shared properties, common elements (that appear on every page), elements, elements *in* context, colors (background, borders included), z-index, animations, CSS3 vendor-prefix stuff.
- I use a tab to show a variation of the styling of an element.

What are the advantages/disadvantages of using CSS preprocessors? (SASS, Compass, Stylus, LESS)

I tried both SASS and LESS, but never managed to integrate them in my workflow. The things I wanted to use them for is variables mainly. But I found that regrouping selectors works way better for me. One thing is that I can change the **single** value in the Chrome Inspector and see it changing

live *everywhere*.

Also, mixins look nice. But in the end, I feel it's an overkill to have a separate tool for something as simple (but not easy) as CSS. I've written my CSS and HTML in Notepad++ for years, even without auto-completion. This has helped me find ways to write my CSS efficiently, by regrouping stuff, by keeping the HTML structure clean and light, by having a clear flow in my CSS, by keeping a tidy namespace as well.

So preprocessors might just be for developers looking for writing efficient CSS *for them*. But that's not for me. I like the static feel of a text editor and a browser.

How would you implement a web design comp that uses non-standard fonts?

If it's a Google Font, just use a link tag, with the minimum amount of fonts necessary. I never used Typekit. I've often used a font-face generator online. I then put it in a seperate CSS file because the code can be quite long. Also: I try to keep the same font-family name for the various styles and weights. It's easier for example to change the font-weight rather than the font-family.

Explain how a browser determines what elements match a CSS selector?

Well I guess it'll start by looking for the generic tag selector, then the class, then the id. (I won't talk about inline styles or CSS in the head). Also, some properties are inherited. But some elements don't inherit stuff (like links who don't inherit the color). Then you also have to take into account a multiple part CSS selector, like with 2 classes. Which one takes priority? But that's beyond the scope of the question I think.

Explain your understanding of the box model and how you would tell the browser in CSS to render your layout in different box models.

The box model basically determines how much space an element will take on the screen. It includes the height/width (applied or generated), the padding, and the borders. The paddings and borders are usually set. The height/width is often flexible, and is determined by the inner text or child elements.

If you want a different box model, just use IE6 and delete the doctype. You'll trigger Quirks mode. It's like the new box-sizing: border-box. I'm not used to that. I'm used to the content-box, where the width doesn't include the paddings nor borders.

What does * { box-sizing: border-box; } do? What are its advantages?

It's the IE6 Quirks mode: if you set a width, and add paddings and borders, the *total* width won't change. It's the innerwidth that'll adapt.

Advantage? You can play with the paddings and border values without worrying about expanding your box. Very convenient for column layouts. And you can mix percentage and pixel values, so you don't have to rely on a child element for the padding.

Disadvtange? Well, I've spent 7 years designing with the content-box model in mind, and now it seems he border-box model is catching up. I feel like dealing with IE6 again. But apparently, I've heard that the W3C always wanting to make the border-box model the standard. So I guess it's a good idea to get acquainted with it right away. Bootstrap uses it, and my HTML5 reset uses it. And the browser compatibility seems ok.

List as many values for the display property that you can

remember.

Ah well, block and inline. The 2 big ones! list-item, which is quite useless. none, table, table-cell, table-row.

I've been using table-cell a lot lately. Two things it does well is vertical-align, and taking up all the horizontal space (although you need table and table-row for that). It's like designing with tables, but with only the good parts (and the correct semantics).

What's the difference between inline and inline-block?

Ah, forgot about inline-block in the previous question. Well, inline-block is like the best of both worlds (or the worst). Main difference with inline is that you can actually set a height and width on the inline-block. And you can still use text-align center on the parent to center these!

So inline-block is like a mini block that you can insert in text. It's like an image basically, which is the most hybrid HTML element.

What's the difference between a relative, fixed, absolute and statically positioned element?

Static is the default value. The element stays in the flow.
Relative is almost like static, but with some tweaks. It also stays in the flow (which is the magic part), but you can use left/right/top/bottom to actually *offset* the element, **without** the other elements being aware of it! It's like movement in disguise. Also: you can use the z-index. And also (very important): it acts like a point of reference for absolute-positioned child elements.
Absolute takes the element out of the flow. So the other elements act like it's not there. That's why absolute elements appear on top. Then the first positioned parent acts like a reference point: you can use left/right and top/bottom to position the element, like on a grid axis. One trick is to

provide values for the 4 properties, keep height and width to auto, and you have a perfect flexible overlay!
Fixed positioning is like absolute, but the reference is the body, or the viewport actually. Usually used for headers, but is also perfect for overlays, because fixed elements don't scroll.

What existing CSS frameworks have you used locally, or in production? (Bootstrap, PureCSS, Foundation etc.)

Bootstrap only. Used it for a few projects only recently. I think it's great for web apps, because you don't care about semantics, you just need a workable interface that works well on most browsers. And in this case, I'm in a "Just make it work" state of mind, so I don't care about writing/rewriting my HTML for building my layout.

What I do is have an override.css to alter the Bootstrap elements, and have a specific.css for additional classes.

What I mostly like about Bootstrap is the grid system, the form elements (*really* great implementation), the basic responsiveness, the JS plugins, the buttons that work everywhere.

For a quick dashboard, or a simple WordPress theme, I'd use Bootstrap because I want to gain time. But for bigger projects, I have my very own way of writing CSS and imagining its structure, that I'd write everything from scratch. It's more flexible for maintaining, and it also allows me to design more complex layouts.

Have you played around with the new CSS Flexbox or Grid specs?

No.

Next →

Order and hierarchy in menus

Back to top ↑