Deep Dive Fullstack Web

Intermediate CSS

CSS Positioning

In order to understand CSS and page layout effectively, one must understand the rules governing how elements are positioned on a page. In practice, explicitly implementing these rules should be done with careful consideration as **we will be using different (and much more modern) ways to lay out our page elements** (grid systems and flexbox, for example), **but these concepts are important for any front-end developer and web designer to understand**.

There are 4 rules we will cover:

- position: static;
- position: relative;
- position: absolute;
- position: fixed;

position: static;

This is the default for all HTML elements. This means the element sits where it normally would in the page, exactly as it was laid out in the HTML document. Normally, you wouldn't have to specify this unless you needed to override another positioning rule that has been previously set.

position: relative;

If you specify `position:relative;` , then you can use top and bottom, and left or right to move the

When you specify `position:absolute;` , an element is "removed" from it's default place in the document, and placed exactly where you tell it to go. The space that is formerly occupied collapses, as if the element was removed from it's original place.

position: fixed;

This will fix an element to a specific location in the browser window, and it will not move regardless of page scroll.

z-index

`z-index: 9999;`

Z-Index property specifies the stack order of elements that are positioned absolute, relative, or fixed. An element with a higher z-index always overlaps an element with a lower z-index. This is sometimes used with navigation dropdowns, so they always sit on top of other elements.

Using relative and absolute positioning together.

If we set relative positioning on a parent element, and absolute positioning on a child element, *we are able to position a child element absolute inside it's parent*. This can be very useful in specific and often tricky use cases.

Examples

Mozilla Developers Network has a great comparison that allows you to see several position attributes in action: https://developer.mozilla.org/en-US/docs/Web/CSS/position

You can see some source code for position examples here.

Flexbox

Flexbox provides a method to align and scale items using a "flexible box" layout. Flexbox is one-dimensional, and let's you organize child elements in a row or column inside a parent. The size and space distribution of the child elements becomes flexible across screensizes. Child elements can also be reordered, realigned, scaled, and stretched within their parent container.

Flexbox is generally best used on specific components or content areas of a page. CSS Grid is

HTML

```html
<div class="flexbox">
  <div class="flex-item">Item 1</div>
  <div class="flex-item">Item 2</div>
  <div class="flex-item">Item 3</div>
  <div class="flex-item">Item 4</div>
  <div class="flex-item">Item 5</div>
</div>
```

CSS

```css
.flexbox {
  display: flex;
  display: -webkit-flex;
  flex-wrap: wrap;
}

.flex-item {
  flex-grow: 1;
}
```

CSS Grid

CSS Grid should not be confused with pre-built "CSS Grid Systems" like Bootstrap. *Native* CSS Grid is a way to use "flat CSS" to create two-dimensional page layouts using rows and columns.

Flexbox and CSS grid can work very well together. CSS Grid is best used to create comprehensive and complex page layouts while Flexbox works best with simpler page elements.

See: CSS Tricks Complete Guide to Grid

HTML

```
  <div class="grid-item">Item 3</div>
  <div class="grid-item">Item 4</div>
  <div class="grid-item">Item 5</div>
  <div class="grid-item">Item 6</div>
  <div class="grid-item">Item 7</div>
  <div class="grid-item">Item 8</div>
  <div class="grid-item">Item 9</div>
</div>
```

CSS

```
.css-grid {
  display: grid;
  grid-template-rows: 1fr 1fr 2fr;
  grid-template-columns: 1fr 1fr 3fr;
}

.grid-item {
  border: 1px solid #777;
  padding: 1rem;
}
```

Adding Custom Fonts Using @font-face

There are several ways we can add custom fonts to a web page. The easiest way is to use a Web Font Embedding Service like Google Fonts or Adobe Edge Web Fonts.

Alternatively, you can include your own custom font files in your project and embed them using the CSS `@font-face` property.

1. First, upload your selected custom font files with your project in a web friendly format such as .woff, .woff2, and .ttf (This will support IE9+. For IE 6-8 support, include an .eot file). `.woff` is considered the established web font file format at this time. See: caniuse.com
2. Second, add the following to your CSS stylesheet at the TOP, before any other rules:@font-face { font-family: 'MyWebFont'; src: url('lib/myfont.woff2') format('woff2'), url('lib/myfont.woff') format('woff'), url('lib/myfont.ttf') format('truetype'); }
3. Third, apply CSS font-family rules as follows: `body { font-family: 'MyWebFont', Helvetica, sans-serif; }`

Retina Support

Tutorial: Line25.com - How to create Retina graphics.

Using the following method, we can swap out images set via CSS (background-image) by using media queries.

1. You'll need to use an image editing program to create a set of images that are double the pixel height and width as the originals, and add these to your images folder. Append the file names of these new, larger images with `@2x` . Example: *my-photo@2x.jpg*
2. In your CSS write the following media query:@media only screen and (-webkit-min-device-pixel-ratio: 2), only screen and (min--moz-device-pixel-ratio: 2), only screen and (-moz-min-device-pixel-ratio: 2), only screen and (-o-min-device-pixel-ratio: 2/1), only screen and (min-device-pixel-ratio:

   2), only screen and (min-resolution: 192dpi), only screen and (min-resolution: 2dppx) { #my-element { background-image: retina-image@2x.jpg; background-size: *original height original width***; } }
3. ** For `background-size` , input the ORIGINAL height, followed by the ORIGINAL width of the image. This will be ½ the height and width of the @2x image. For example, for a @2x image with a dimension of 3000 x 2000 set the following: `background-size: 1500px 1000px;`

The full set of @media expressions listed in the example above will ensure broad-scope compatibility across devices and browsers.

For images added into your HTML using the <img> tag, the easiest way is to use the retina.js plugin which will check your server automatically for images with @2x in the filename, and automatically swap them out on devices with a high pixel density.

Calculating CSS Specificity Values

Let's look at an easy way we can calculate specificity of different CSS selectors. This is a points-based system.

Inline CSS	ID Selectors	Class, pseudo-class, & attribute selectors.	Element Selectors
__________,	__________,	__________,	__________,

For the above diagram:

- For each inline style: Add 1,0,0,0 points.
- For each ID value in the selector: Add 0,1,0,0 points

The commas remind us that this isn't actually a base-10 number (ones don't carry over).

Example:
`ul#nav li.active a` 0,1,1,3
`<a style="color: red;">` 1,0,0,0

Important Notes:

- The universal selector (*) has no specificity value (0,0,0,0).
- Pseudo-elements (e.g. :first-line) get 0,0,0,1 point value, unlike their psuedo-class brethren which get 0,0,1,0.
- The pseudo-class :not() adds no specificity by itself, only what's inside it's parentheses.
- `!important` appended to a CSS property value is an automatic win. It overrides even inline styles from the markup. The only way an !important value can be overridden is with another !important rule written later in the stylesheet with an equal or greater specificity. You could think of it as adding 1,0,0,0,0 to the specificity value.

Example: Calculate the specificity, and determine which CSS selector is the winner.

1. `div#navbar ul.navbar-nav > li > a:hover {...}`
2. `header.header-home .navbar-nav > li > a:hover {...}`

Answer:

1. Rule #1: 0,1,2,4. (Winner)
2. Rule #2: 0,0,3,3.

CSS Organization and Methodologies

There are numerous methods and frameworks that have been created to specifically address the issue of CSS file organization and scalability. One popular methodology is **SMACSS**, and it's a great place to start. Consider familiarizing yourself with SMACSS, along with other CSS methodologies and frameworks such as BEM and OOCSS if you are interested in doing in-depth front-end work, or are working on larger sites.

- **SMACSS**: Scalable and Modular Architecture for CSS. https://smacss.com/
- **OOCSS**: Object Oriented CSS. OOCSS is both a CSS framework, and a methodological

CSS. Strict. Created by Yandex (a large Russian internet company, and the 4th largest search engine worldwide). Designed for use on large projects. https://en.bem.info/method/

Consider integrating some ideas from SMACSS and other CSS methodologies when writing your CSS. Think about writing your styling rules from the "top down": large, global base styles followed by finer details. Create meaningful class and ID names that adhere to a standardized and efficient naming convention (even if it's your own). Consider using classes to style elements, and leaving IDs for behaviors and hooks. Modulate your styling patterns in a way that is simple and re-usable. Complex stylesheets can quickly become a nightmare across larger project with many people working on them.

Refer to our Coding Style Guide for CSS Best Practices requirements. Here are some additional stylistic criteria to keep in mind when writing CSS:

- One selector and one rule per line for readability and neatness.
- Keep your CSS files well-commented and organized using comment blocks.
- Run your CSS through a linter to check for errors.
- Keep things simple, and don't rely on excessive CSS overrides to achieve layout goals.