Placement Prep Resource- CSS Concepts Assignment

1. What is CSS and how does it work with HTML?

Ans: CSS stands for Cascading Style Sheets with an emphasis placed on “Style.” While HTML is used to structure a web document (defining things like headlines and paragraphs, and allowing you to embed images, video, and other media), CSS comes through and specifies your document’s style—page layouts, colours, and fonts are all determined with CSS. Think of HTML as the foundation (every house has one), and CSS as the aesthetic choices (there’s a big difference between a Victorian mansion and a mid-century modern home).

2. What are selectors and what are their different types?

Ans:

CSS selectors are used to "find" (or select) the HTML elements you want to style.

We can divide CSS selectors into five categories:

- Simple selectors (select elements based on name, id, class)
- Combinator selectors (select elements based on a specific relationship between them)
- Pseudo-class selectors (select elements based on a certain state)
- Pseudo-elements selectors (select and style a part of an element)
- Attribute selectors (select elements based on an attribute or attribute value)

3. What is CSS selector specificity and how does it work?

Ans: MDN put it nicely (as always):

“Specificity is the means by which browsers decide which CSS property values are the most relevant to an element and, therefore, will be applied.”

That means that CSS specificity is a set of rules used by browsers in determining which of the developer-defined styles will be applied to a specific element. For a style to be applied to a particular element, the developer has to abide by the rules, so that the browser knows how to apply the style.

When two or more styles target a particular element, the style with the highest specificity is the one that gets applied.

4. Describe z-index and how stacking context is formed.

Ans: The **stacking context** is a three-dimensional conceptualization of HTML elements along an imaginary z-axis relative to the user, who is assumed to be facing the viewport or the webpage. HTML elements occupy this space in priority order based on element attributes.

Using z-index, the rendering order of certain elements is influenced by their z-index value. This occurs because these elements have special properties which cause them to form a *stacking context*.

A stacking context is formed, anywhere in the document, by any element in the following scenarios:

- Root element of the document (<html>).
- Element with a position value absolute or relative and z-index value other than auto.
- Element with a position value fixed or sticky (sticky for all mobile browsers, but not older desktop).
- Element that is a child of a flex container, with z-index value other than auto.
- Element that is a child of a grid container, with z-index value other than auto.
- Element with a opacity value less than 1 (See the specification for opacity).
- Element with a mix-blend-mode value other than normal.
- Element with any of the following properties with value other than none:
  - transform
  - filter
  - perspective
  - clip-path
  - mask / mask-image / mask-border
- Element with a isolation value isolate.
- Element with a -webkit-overflow-scrolling value touch.
- Element with a will-change value specifying any property that would create a stacking context on non-initial value (see this post).
- Element with a contain value of layout, or paint, or a composite value that includes either of them (i.e. contain: strict, contain: content).

Within a stacking context, child elements are stacked according to the same rules previously explained. Importantly, the z-index values of its child stacking contexts only have meaning in this parent. Stacking contexts are treated atomically as a single unit in the parent stacking context.

5. Describe BFC (Block Formatting Context) and how it works.

Ans: A **block formatting context** is a part of a visual CSS rendering of a web page. It's the region in which the layout of **block** boxes occurs and in which floats interact with other elements.

Some examples I have found online on how to use block formatting context:

- The root element of the document (<html>).
- Floats (elements where float isn't none).
- Absolutely positioned elements (elements where position is absolute or fixed).
- Inline-blocks (elements with display: inline-block).
- Table cells (elements with display: table-cell, which is the default for HTML table cells).

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

Ans: The difference between grid and flex is grid is 2D meaning it has row and column but flex box has only row. To make flex system 2d you have to put flex inside flex .

For me i don't use flex better i go with bootstrap because its made of flex and easy to use and you don't have to write your code from stretch.

For conclusion i think it depend upon the layout design. But if you compare with two then i think grid is best than flex.

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

Ans: yes, the Media Queries in CSS3 take this idea and extend it. Rather than looking for a *type* of device they look at the *capability* of the device, and you can use them to check for all kinds of things.

8. Explain how a browser determines what elements match a CSS selector.

Ans: This part is related to the above about writing efficient CSS. Browsers match selectors from rightmost (key selector) to left. Browsers filter out elements in the DOM according to the key selector and traverse up its parent elements to determine matches. The shorter the length of the selector chain, the faster the browser can determine if that element matches the selector.

For example with this selector p span, browsers firstly find all the <span> elements and traverse up its parent all the way up to the root to find the <p> element. For a particular <span>, as soon as it finds a <p>, it knows that the <span> matches and can stop its matching.

9. Describe pseudo-elements and discuss what they are used for.

Ans: A CSS pseudo-element is used to style specified parts of an element.

For example, it can be used to:

- Style the first letter, or line, of an element
- Insert content before, or after, the content of an element

- The syntax of pseudo-elements:

-
  ```
  selector::pseudo-element {
    property: value;
  }
  ```

10. Explain your understanding of the box mode.

Ans: In CSS, the term "box model" is used when talking about design and layout.

The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content. The image below illustrates the box model:

Explanation of the different parts:

- **Content** - The content of the box, where text and images appear
- **Padding** - Clears an area around the content. The padding is transparent
- **Border** - A border that goes around the padding and content

- **Margin** - Clears an area outside the border. The margin is transparent

The box model allows us to add a border around elements, and to define space between elements.

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

Ans: Since the result of using the box-sizing: border-box; is so much better, many developers want all elements on their pages to work this way.

The code below ensures that all elements are sized in this more intuitive way. Many browsers already use box-sizing: border-box; for many form elements (but not all - which is why inputs and text areas look different at width: 100%;).

Applying this to all elements is safe and wise

12. What is the CSS display property, and can you give a few examples of its use?

Ans

The Display property in CSS defines how the components (div, hyperlink, heading, etc) are going to be placed on the web page. As the name suggests, this property is used to define the display of the different parts of a web page.

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

Ans: **display: inline-block** brought a new way to create side by side boxes that collapse and wrap properly depending on the available space in the containing element. It makes layouts that were previously accomplished with floats easier to create. No need to clear floats anymore.

Compared to display: inline, the major difference is that **inline-block** allows to set a width and height on the element. Also, with display: inline, top and bottom margins & paddings are not respected, and with display: inline-block they are.
Now, the difference between display: inline-block and display: block is that, with **display: block**, a line break happens after the element, so a block element doesn't sit next to other elements.

14.What's the difference between the "nth-of-type()" and "nth-child()" selectors?

Ans: The **nth-child()** and **nth-of-type()** selectors are “structural” pseudo-classes, which are classes that allow us to select elements based on information within the document tree that cannot typically be represented by other simple selectors.

In the case of **nth-child()** and **nth-of-type()**, the extra information is the element’s position in the document tree in relation to its parent and siblings. Although these two pseudo-classes are very similar, they work in fundamentally different ways.

The **nth-child()** pseudo-class is used to match an element based on a number, which represents the element’s position amongst it’s siblings. More specifically, the number represents the number of siblings that exist before the element in the document tree

*The **nth-of-type()** pseudo-class, like **nth-child()**, is used to match an element based on a number. This number, however, represents the element's position within* only those of its siblings that are of the same element type.

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

Ans: **relative**

This type of positioning is probably the most confusing and misused. What it really means is “relative to itself”. If you set position: relative; on an element but no other positioning attributes (top, left, bottom or right), it will have no effect on it’s positioning at all, it will be exactly as it would be if you left it as position: static; But if you *do* give it some other positioning attribute, say, top: 10px;, it will shift its position 10 pixels *down* from where it would *normally* be. I’m sure you can imagine, the ability to shift an element around based on its regular position is pretty useful. I find myself using this to line up form elements many times that have a tendency to not want to line up how I want them to.

There are two other things that happen when you set position: relative; on an element that you should be aware of. One is that it introduces the ability to use z-index on that element, which doesn’t work with statically positioned elements. Even if you don’t set a z-index value, this element will now appear **on top** of any other statically positioned element. You can’t fight it by setting a higher z-index value on a statically positioned element.

The other thing that happens is it **limits the scope of absolutely positioned child elements**. Any element that is a child of the relatively positioned element can be absolutely positioned within that block. This brings up some powerful opportunities which I talk about here.

absolute

This is a very powerful type of positioning that allows you to literally place any page element exactly where you want it. You use the positioning attributes top, left, bottom, and right to set the location. Remember that these values will be relative to the next parent element with relative (or absolute) positioning. If there is no such parent, it will default all the way back up to the <html> element itself meaning it will be placed relative to the page itself.

The trade-off (and most important thing to remember) about absolute positioning is that these elements are **removed from the flow** of elements on the page. An element with this type of positioning is not affected by other elements and it doesn't affect other elements. This is a serious thing to consider every time you use absolute positioning. Its overuse or improper use can limit the flexibility of your site.

fixed

A fixed position element is positioned relative to the *viewport*, or the browser window itself. The viewport doesn't change when the window is scrolled, so a fixed positioned element will stay right where it is when the page is scrolled.

This might be used for something like a navigation bar that you want to remain visible at all times regardless of the pages scroll position. The concern with fixed positioning is that it can cause situations where the fixed element overlaps content such that is is inaccessible. The trick is having enough space to avoid that, and tricks like this.

static

This is the **default** for every single page element. Different elements don't have different default values for positioning, they all start out as static. Static doesn't mean much; it just means that the element will flow into the page as it normally would. The only reason you would ever set an element to position: static; is to forcefully remove some positioning that got applied to an element outside of your control. This is fairly rare, as positioning doesn't cascade.

16. What existing CSS frameworks have you used locally, or in production? How would you change/improve them?

Ans: I don't use frameworks in my work, rather prefer my own, lighter-weight CSS to dragging around lots of un-used code. I've been at it a long time, understand how semantic markup, viewports, media queries, and wildcard classes work together to make mobile-first and nicely responsive websites because I've got to teach this stuff.

I do encourage students to learn a framework or two, usually starting with BootStrap because it's so well-kinown and has a good intro at W3Schools and the Bootstrap site is very usable. Some students choose Materialize or Skeleton as light-weight frameworks and get god results with minimum effort.

Using frameworks is certainly a very professional approach. They've been written and maintained by smart people who have 'normalized' them as HTML5 and CSS3 have been rolled out and more standardized so there are fewer 'vendor-specific' rules that need to be applied. If you use a framework properly, the result is a mobile-friendly, nicely responsive site without hours and hours of testing and debugging only to find some cross-browser issue pops up and there's more time debugging and trying stuff.

Some students who love complexity are drawn into the heavier-weight frameworks like Foundation or Angular. They'll work 10X harder learning them and setting up the environment to run them, seem satisfied with the results, but I seldom see any added value for all the complexity they involve.

If I'd change anything, I'd like to see Bootstrap and other frameworks' CSS run through W3C CSS Validator without errors. I preach the value of valid code and have to make exceptions for Bootstrappers' projects where others are required to validate.

And, I'd like to see all the examples they post using semantic page layout elements where they exist, for example using <header, <nav, <main, and <aside tags rather than naming classes as header, nav, main, or aside. It's hard to teach proper semantic mark-up with so many demo sites that abuse it.

17.Have you used CSS Grid?

Ans: CSS Grid Layout enables two-dimensional layout without requiring the additional mark-up for row wrappers. As a two-dimensional layout method you can cause elements in your design to span rows, in a predictable and robust way.

You can achieve some nice design effects. For example, having items in your design that are *at least* a particular height but will still expand to contain content that is taller.

You can easily mix fixed width elements with flexible units, with the fr unit in grid. This makes it easier to deal with items in your layout that you want to maintain at a fixed size.

You can redefine the layout from the grid container, making responsive design more straightforward, giving you the ability to fine tune the design at different breakpoints.

You can layer items on the grid, items respect z-index so you can cause different items to occupy the same grid cells with plenty of scope for creativity.