Display Types, Positioning, & Floats

Position Types:

- **Static Positioning:**
    - **The default position of every HTML element.**
    - **Positioned using the Box Model (margin, padding, border).**
    - **Unaffected by top, left, right, and bottom CSS properties.**
    - **Static elements are commonly floated left or right to create web layouts.**
- **Absolute Positioning:**
    - **When an element is absolutely positioned, it is taken out of the flow of the page. It is unaffected by the position of other elements and other elements ignore it.**
    - **Can be positioned using the Box Model, but can also be positioned using top, left, bottom, and right properties.**
    - **Absolutely positioned elements are, by default, positioned relative to the viewport (window) dimensions. E.g. `bottom: 0; left: 0` would position an absolute element at the bottom of the height of the browser window.**
    - **When an absolute positioned element is inside of another positioned (absolute, relative, fixed) element, the absolute element is positioned relative to it’s positioned parent instead of the viewport.**
- **Relative Positioning:**
    - **Similar to both Static and Absolute Positioning:**
        - **Similar to Absolute Positioning in that it can be moved with top, left, bottom, and right CSS properties. The difference is that rather than being moved relative to the viewport or an positioned container, it is moved relative to its original static position.**

        - **Similar to Static Positioning in that it takes up space on the page, elements are positioned around it, and it is affected by the position of other elements. The caveat is that when top, left, bottom, and right CSS properties are applied, the relatively positioned element then moves out of the flow of the page and behaves like absolutely positioned elements. Its original static position is still respected by other elements.**

- **Fixed Positioning:**
    - **Similar to Absolute Positioning, except that the position of the element is locked to the viewport. I.e. as the page scrolls, the element doesn't move, but stays in the same position on the screen as the page scrolls.**

Display Types:

- **Block Elements:**
    - **Examples include:**
        - **paragraphs, divs, nav, footer, header, and heading tags.**
    - **Most elements are block elements by default.**
    - **Block elements take up the entire line they sit on. This is why each new paragraph tag produces a line break in-between.**
    - **Even if a block element has a fixed width, it will by default, still want to take up an entire line. Some exceptions are:**
        - **If a block element is positioned fixed, relative, or absolute. Since these elements can be moved in any way described and aren't affected by and don't affect other elements on the page.**
        - **If a block element is floated, since this takes them out of the flow of the rest of the page.**
- **Inline Elements:**
    - **Examples include:**
        - **spans, anchors, strong, and italic tags.**

  - **Inline elements only take up as much space as they need to fit their content.**

  - **Inline elements can live on the same line as other inline elements or inline-block elements.**

  - **Inline elements are positioned using the Box Model, but margin and padding top and bottom are ignored.**

- **Inline-Block Elements:**

  - **Similar to inline elements. Same set of rules apply, except inline-blocks can be positioned using top and bottom margin and padding.**

  - **Only example of an inline-block I can think of/find now is an image.**

Floats:

- **Floats aren't a display or position type. They're something else entirely. Floats are taken out of the normal flow of the page and "floated" either left or right. Elements which are floated behave similarly to inline-block elements in that that can live on the same line as other elements, but have the advantage of being able to be floated left or right.**

- **Floats can be used to wrap text around other elements such as images.**

- **Floated elements can interact with each other, but since they are taken out of the flow of the page, other elements aren't affected by them. This introduces a common problem where the parent container of floated elements doesn't stretch to fit the floated children. To resolve this, floated elements should be cleared. There are a few ways of doing this, but clearing floats effectively brings the flow of the page to where it would have been had the elements been in the flow of the page.**

Question:

- **I still don't fully understand this. What can I do?**

Answer:

- **If you aren't experimenting with this, you won't understand it well. It's important to practice, and refer back to documentation or ask questions when things don't make sense.**

Question:

- **Can you combine different types of positioning and display types?**

Answer:

- **Definitely. Which is why it's important to practice to see what happens in what situations. Generally position types have more control over how an element behave when used together.**

Question:

- **Are these all of the display and position types?**

Answer:

- **No, but these are the most commonly used today and most important to have some understanding of.**

Question:

- **Why do I need to know this? Can't I just use Bootstrap?**

Answer:

- **Sure, to an extent. Often times in a front-end or full stack role, you'll be given a PSD or Sketch Mockup that you're instructed to recreate with HTML and CSS. Usually it'll need to be pixel perfect. So it's important to be able to position elements more subtly than the Bootstrap Grid allows.**

Question:

- **I don't want to be a front end developer, I like JavaScript/Node better. I can just skip this stuff right?**

Answer:

- **You're much more marketable to employers and much more valuable at a company if you're strong on the backend, as well as the front end. Given a little time and practice, this is one of the easiest thing to become better than ~80% of other developers at. So there's a huge ROI on learning this well. Plus, when looking for a developer job, the recruiters checking out your projects probably won't know anything about coding. They won't be able to tell how sophisticated your backend is. They just see the front end. At some point you'll probably make it**

to an interview where you'll talk to a senior developer who will understand your code. But you'll often need to get past that initial recruiter who just knows a good design when they see one.

Question:

- **This is a little discouraging. I thought I sort of knew this. This is week one stuff.**

Answer:

- **Most developers don't understand this well. That has to do with not caring, misinformation on the internet, a reliance on professional designers to fix their code, not wanting to put in some time to practice this stuff etc. So it's okay to feel as though you don't have a good understanding of this now. Just don't get discouraged, practice, practice, practice. You can become better than 80% of developers at this particular skill in under a week of dedicated practice. The same probably can't be said about JavaScript or MySQL or most other technologies.**