Day 3: Positioning, Flexbox, and Sizing

Lesson Objectives:

By the end of this lesson, learners should be able to:

- Understand and use different CSS positioning methods.
- Control the stacking of elements using `z-index`.
- Use floating and clearing to manage layout.
- Apply the basics of Flexbox to create responsive layouts.

Lesson Outline:

1. CSS Positioning
2. Z-index and Layering
3. Floating and Clearing Elements
4. Introduction to Flexbox
5. Practice Project

---

1. CSS Positioning

CSS provides several ways to position elements on the page. Each method gives different control over how and where elements are placed.

1.1 Static Positioning

Static is the default position value. Elements with `position: static` are placed according to the normal document flow, meaning they follow the natural order of HTML elements.

1.2 Relative Positioning

Relative positioning moves an element relative to its normal position in the flow of the document.

```
div {
    position: relative;
    top: 20px;  /* moves the element 20px down */
    left: 10px; /* moves the element 10px to the right */
}
```

> **Key Note:** The space that the element originally occupies remains reserved in the layout.

1.3 Absolute Positioning

Absolute positioning places an element exactly where you want it, relative to its nearest positioned ancestor (`relative`, `absolute`, or `fixed`), or relative to the document if no such ancestor exists.

```
div {
    position: absolute;
    top: 50px;
    left: 100px;
}
```

> **Key Note:** Absolute elements are removed from the normal document flow and don't affect the layout of other elements.

1.4 Fixed Positioning

Elements with `position: fixed` are positioned relative to the browser window. They stay fixed in position, even when the page is scrolled.

```
div {
    position: fixed;
    bottom: 0;
    right: 0;
}
```

> **Common Use Case:** Fixed elements are often used for headers or footers that remain visible as you scroll through the page.

> **Activity:** Create a `div` and try applying `static`, `relative`, `absolute`, and `fixed` positioning. Observe how each property affects the layout and placement of the element.

---

2. Z-index and Layering

The `z-index` property controls the stacking order of elements. Elements with a higher `z-index` value appear on top of elements with a lower value.

- **Default Layering:** Elements are stacked in the order they appear in the HTML.
- **Z-index:** Allows you to override this default stacking order.

```
div {
    position: relative;
    z-index: 2;  /* this element will appear above elements with z-index
1 */
}
```

> **Important Note:** `z-index` only works on positioned elements (`relative`, `absolute`, `fixed`).
>
> **Activity:** Create two overlapping `div` elements and assign different `z-index` values to them. See how the layering changes when you modify the `z-index`.

---

3. Floating and Clearing Elements

Floating is an older method of creating layouts, often used before Flexbox and Grid became popular. While not as commonly used for layout today, it's still important to understand.

3.1 Float

The `float` property moves an element to the left or right and allows text and other elements to flow around it.

```
div {
    float: left;
}
```

3.2 Clear

The `clear` property is used to prevent elements from wrapping around floated elements.

```
div {
    clear: both;  /* prevents the div from being affected by floated
elements */
}
```

Common Use Case: Float is often used for creating layouts with sidebars. Clear is used to ensure that footer or next content block doesn't overlap with floated elements.

Activity: Create two `div` elements: one floated left and one floated right. Add some text content around them and experiment with the `clear` property to see how it affects layout.

---

4. Introduction to Flexbox

Flexbox is a modern layout system that makes it easy to design flexible, responsive layouts without using float or positioning tricks. It's ideal for aligning elements in a row or column and distributing space.

4.1 Flex Container and Flex Items

To start using Flexbox, you need a **flex container**. All the direct child elements of the flex container become **flex items**.

```
.container {
    display: flex;
}
```

4.2 flex-direction

The `flex-direction` property controls the main axis of the flex container, determining whether the flex items should be placed in a row (default) or a column.

- `row` (default) aligns items horizontally.
- `column` aligns items vertically.

```
.container {
    display: flex;
    flex-direction: row; /* or column */
}
```

4.3 justify-content

The `justify-content` property aligns the flex items along the main axis (horizontally if `flex-direction` is `row`, vertically if `flex-direction` is `column`).

Options include:

- `flex-start` (default): Items align at the start.
- `center`: Items are centered.
- `space-between`: Items are evenly distributed, with space between them.

```
.container {
    display: flex;
    justify-content: center;
}
```

4.4 align-items

The `align-items` property aligns items along the cross axis (perpendicular to the main axis).

Options include:

- `flex-start`: Aligns items to the top.
- `center`: Centers items vertically.
- `flex-end`: Aligns items to the bottom.

```
.container {
    display: flex;
    align-items: center;
}
```

> **Activity:** Create a `div` with three child elements (e.g., three boxes). Use Flexbox to align them in a row, center them using `justify-content`, and experiment with `flex-direction` to switch between row and column layouts.

---

5. Practice Project

Let's put everything together with a small project that combines positioning, layering, and Flexbox.

Goal:

Create a simple navigation bar using Flexbox with some positioned elements.

Steps:

HTML structure:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <link rel="stylesheet" href="styles.css">
    <title>Flexbox Navigation</title>
</head>
<body>
    <header>
        <div class="logo">MySite</div>
        <nav class="navbar">
            <a href="#">Home</a>
            <a href="#">About</a>
            <a href="#">Services</a>
            <a href="#">Contact</a>
        </nav>
        <div class="user-profile">Profile</div>
    </header>
</body>
</html>
```

1.

CSS (styles.css):

```
/* Header Styling */
header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 10px;
    background-color: #333;
    color: white;
}

/* Logo and Profile */
.logo, .user-profile {
    position: relative;
}

/* Navbar Styling */
.navbar {
    display: flex;
```

```
    gap: 15px;
}

.navbar a {
    color: white;
    text-decoration: none;
}

.navbar a:hover {
    text-decoration: underline;
}
```