week::seven

Introduction to CSS Layout

Getting Started by Writing Good HTML

Create a Clear Content Structure

- Make sure HTML is structured correctly and organized in the general order they should be presented visually.
- Use indents to give visual hierarchy to HTML (helps you navigate the code).

Organize Code Using HTML5 Semantic Tags

- Wrap logical areas of content in HTML5.
- If you use an HTML5 tag more than once, give them names using a class attribute.
  - `<nav class="primary"> … </nav>`
- Use `<div>` or `<section>` elements as needed to created “boxes” to contain content, or even boxes to contain other boxes.

Modify HTML As Needed

- Never feel locked-in to your initial HTML structure — if you need to re-write parts of your page or add/remove HTML, don’t be afraid to do so.

Utilize an Existing Site Structure

Web developers rarely start from scratch

- They almost always use previous work or existing frameworks available online.

Download the class framework from the course website:

- https://hoelter.chemeketa.vc/vc237/links.html

Use a *Reset* or *Normalize* Stylesheet

- Use the **Reset Style Sheet** provided by the instructor (`reset.css`), or…
- To completely reset the browser defaults, use Eric Meyer’s reset style sheet:
  - `http://meyerweb.com/eric/tools/css/reset/`
- Or use **Normalize.css** to increase cross-browser consistency but maintain helpful browser defaults like font sizes for header elements.

A Reminder About Document Flow

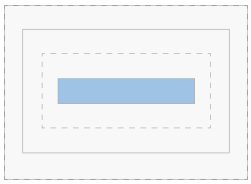
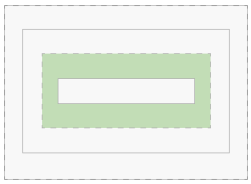
- By default, text flows as far right as it can before returning and beginning a new line. Without any constraints, this is always the right-hand side of the browser window.
- As the browser window is re-sized, the text on a page re-flows to fit the new window size.
- Generally speaking, page elements are not positioned outside of the document flow—all elements affect the position of each other.
  - However, it is possible to position elements outside the document flow—think objects on Web site that do not scroll and stay in a fixed place.
- Document flow is becoming more important as the industry begins to adapt their designs to mobile devices.

The CSS Box Model

In Web layout, everything is a box

Components

- All **block-level** elements include these four controllable components:

			
Content Area The area in which HTML content may be displayed.	**Padding** Space between the content area and the border. Can be styled with background images and background colors; No HTML content.	**Border** A border surrounding all sides of a box.	**Margin** Empty white space between the border and other elements. Cannot be styled other than the amount of space. No HTML content

Key CSS Layout Properties

`width`

- Best set in percentages; can also be ems and pixels.
- `width: 65%;`

`height`

- *Avoid* setting height unless necessary
- Set in px, em or vh
- `height: 20em;`

`max-width` *and* `min-width`

- Set in ems, pixels, or percentages
- `max-width: 61.5em;`

`max-height` *and* `min-height`

- *Avoid* setting height unless necessary
- Set in px, em or vh
- `min-height: 10em;`

`display`

- Controls the display of and role that an element plays in a page.
- Values include *block*, *inline*, and *none*
- ```
  display: block;
  display: none;
  ```

`border`

- Set width in px; Use solid as style
- ```
  border: 1px solid red;
  border-bottom: 2px solid #ccc;
  border-width: 2px 0;
  ```

`margin`

- Shorthand options:
  - `top right bottom left`
  - `top/bottom left/right`
  - `all_sides`
- ```
  margin: 0 0 1em .5em;
  margin-right: 8%;
  ```

`padding`

- Shorthand options:
  - `top right bottom left`
  - `top/bottom left/right`
  - `all_sides`
- ```
  padding: 1em 0;
  padding-top: 2.25em;
  ```

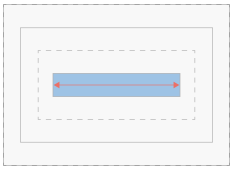
`float`

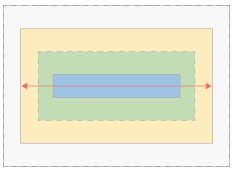
- left, right, or none
- Use as few of these as possible
- `float: right;`

`clear`

- Clears floats
- both, left, right
- `clear: both;`

Understanding Element Width

Browser Default `box-sizing: content-box;` Note that, by default, the `width` property controls only the ***content*** area of an element. Padding, Border, and Margin are *added* to the width.	`content_width = Element_Width`	

Class Framework Default
`box-sizing: border-box;`
The `width` property factors in the ***content, padding, and border widths*** to set the element width.
Margin is *not factored* into the width.
	`content_width +`
	`padding_width +`
	`border_width = Element_Width`
		

Using Outer & Inner Wrapper Classes

Advantages

- Allows for some elements to span the full width of a browser window, while others are constrained to a common width (responsively, of course).
- Easier to create visual vertical stratification of content, a common current design trend.
- Paves the way for reusable chunks of layout code.

Disadvantages

- Works for individual sections of a page, but sometimes cumbersome for the primary article or main page content (can break things up in awkward ways).

Example Code:

CSS:

```
.outerwrap {
    width: 100%;
    clear: both;
    display: block;
}
.innerwrap {
    width: 94%;
    max-width: 68.75em;
    margin: 0 auto;
    display: block;
}
```

HTML:

```
<div class="outerwrap">
    <section class="innerwrap">
        <h1>Hello World</h1>
    </section>
</div>
```

Flexible Images

The Magic CSS Rule

- Use this rule to have all images scale to match their context:
- This rule is included as part of our class template.

```
img, embed, object, video { max-width: 100%; }
```

CSS Units of Measurement

Overview

- When entering numbers in CSS, always specify the units of measurement.
- The only time you can skip the units is with zero.

Percentages (`%`)

- Used mainly for width.
- Calculates a value based on the available space using the available space within the current element.
- Modern layout techniques (responsive web design) are moving towards the use of percentages for defining widths and heights, using pixels less and less.

Ems (`em`)

- Most common unit of measurement for type, including size and leading (`line-height`).
- Allow for flexible typography and layouts whose size is based on the current default font size of the device being used to view the page.
- Based on the relative size of type in the containing element.
- Also used for width and height in some circumstances.

Root ems (`rem`)

- Same as ems, but calculated from the *root* element of the page.
- Eliminates variations of size based on their parent element's type size.
- Can be a more predicable method of setting type sizes, but comes with its own set of issues.

Viewport Width & Viewport Height (`vw & vh`)

- Similar to percentage, but are always sized relative to the root element.
- 1vw equals 1% of the root element; 100vw equals 100%
- Very helpful for elements that need to take up a specific part or all of a device's screen.
  - Viewport width and height are better units than percentages for this because they are based solely on the Viewport, not the parent element's size.

Pixels (`px`)

- Common for widths, heights, margin, padding, and borders.
- Is technically a relative unit of measurement, though most desktop browsers treat it as absolute (monitor pixels).
- On HiDPI screens, the pixel is abstracted based on the PPI of the device's screen.

Recommendations:

- Use **ems** for all typography and margin/padding rules where the exact spacing is not critical, and where it's best that sizes scale with the page typography.
- Use **percentages** (%) as much as possible for widths, as they are inherently flexible, adapting to different screen sizes as needed. Percentages help create flexible (and ultimately, responsive) layouts—ones that are *not* exactly the same across every browser and device.
- Avoid percentages for heights as they are unpredictable.
- Use **pixels** (px) or **percentages** (%) when exact spacing and layout is required and setting widths of some container elements.
  - For percentages, divide the pixel values by the pixel width of the container elements to get exact percentage values.

Additional Reading

- `https://css-tricks.com/the-lengths-of-css/`

Common CSS Layout Properties

This page presents just an overview of a few common CSS properties. We will explore all of these much more in-depth in future classes.

Name	Property	Description	Example CSS Code
Background Color	`background-color:`	Sets the background color of an element	`background-color: #dfdfdf;`
Element Border	`border:`	Creates a border around an element	`border: 1px solid #ccc;`
Rounded Corners	`border-radius:`	Rounds the corners of elements	`border-radius: 20px;`
Padding	`padding:`	Space between content and the edge of the box	`padding-left: 20px;` `padding-top: 13px;`
Margin	`margin:`	Transparent space around a content box	`margin-right: 45px;` `margin-bottom: 100px;`
Element Display Type	`display:`	Controls the display type of an element. In this example, the CSS code hides the element.	`display: none; /*Hides element*/`
Element Width	`width:`	Width of a block element	`width: 300px;`
Element Height	`height:`	Height of a block element	`height: 400px;`
Maximum Width	`max-width`	Sets the maximum width of an element	`max-width: 68.75em;`
Minimum Width	`min-width`	Sets the minimum width of an element	`min-width: 50%;`
Maximum Height	`max-height`	Sets the maximum height of an element	`max-height: 10em;`
Minimum-Height	`min-height`	Sets the minimum height of an element	`min-height: 2em;`
Bulleted List Style	`list-style-type:`	Controls the appearance of bullets in lists.	`list-style-type: circle;`
Element Float	`float:`	Removes an element from the document flow.	`float: right;`