HTML & CSS 2

Intro

Overview

- Semantic HTML
- CSS Selectors
- Creating Layouts

Objectives

- Student understands the benefits of and can utilize semantic HTML.
- Student can add styling to an HTML document using CSS selectors for class and id.
- Student understands and can implement the position properties.
- Student understands and can implement the display properties.
- Students can use flexbox properties to create page layouts.

Semantic HTML

What is semantic HTML?

- HTML that introduces context or meaning to the structure of a webpage
- Certain HTML tags are semantic because the tagname describes the content
- A `div` could be anything, but a `footer` should only be used as a footer
- Examples: `<footer>`, `<header>`, `<nav>`, `<form>`, `<table>`, `<article>`, `<section>`

Why use semantic HTML?

- Appropriate use of semantic HTML helps your site's accessibility
- It can boost SEO when used correctly
- Helpful for you and other developers to figure out a page's structure

Accessibility Tips

When it comes to making your sites more accessible, here are three easy tips to follow:

1. Always use the semantic tag alternative if it's an option
2. Attach labels to each of your `<input />` elements
3. Use a `<button>` tag whenever you use an ***onclick*** event handler

CSS Selectors

Review

- Selectors are how we specify the HTML elements that we want to style
- We can select elements by their tagnames
- We can select multiple elements using a comma
- CSS combinators, such as a space or an angle bracket, help us get more specific
- In this lecture, we'll cover classes and IDs

Class

- If we need to create a group of elements that aren't of the same type, we can give them all the same `class` attribute
- You get to decide the names you give your classes, try to keep them descriptive
- In HTML, elements can have multiple classes, just separate the names with spaces
- In CSS, class names need to be preceded by a `.`

```html
<h1 class="heading">Heading One</h1>
<h2 class="heading content">Heading Two</h2>
<p class="content">Paragraph paragraph paragraph.</p>
<h2 class="heading content">Heading Two Again</h2>
<p class="content">More paragraph paragraph paragraph.</p>
```

```css
.heading {
    font-weight: 800;
}

.content {
    color: darkgray;
}
```

ID

- ID attributes allow us to target one element specifically
- Make sure these names are descriptive as well
- In HTML, an ID should only be given to **one** element on the page
- In CSS, ID names need to be preceded by a `#`

```html
<button>back</button>
<button id="cancel">cancel</button>
<button id="submit">submit</button>
```

```css
button {
    padding: 15px;
    border: none;
}

#cancel {
    background-color: red;
}

#submit {
    background-color: green;
}
```

Cascading and Specificity

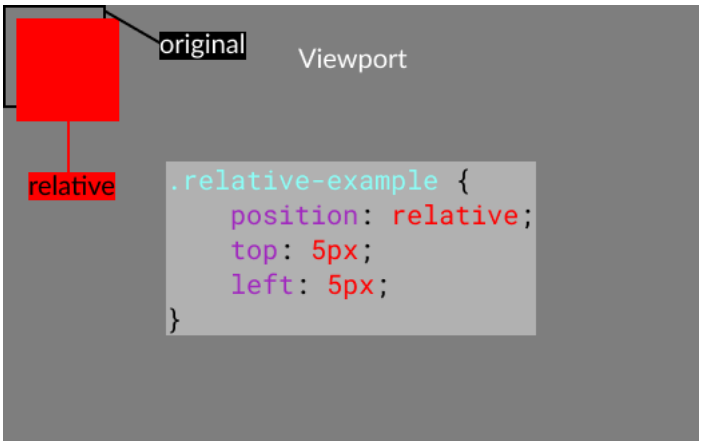
- CSS is read from top to bottom
- Declarations that are lower in will override previous declarations about the same property
- But some declarations are more "specific" than others, so if there is a conflict, the declaration with higher specifcity wins, even if it's declared earlier in the file
- The order from most specific to least is: inline styling, id selector, class selector, element/tagname selector

Creating Layouts

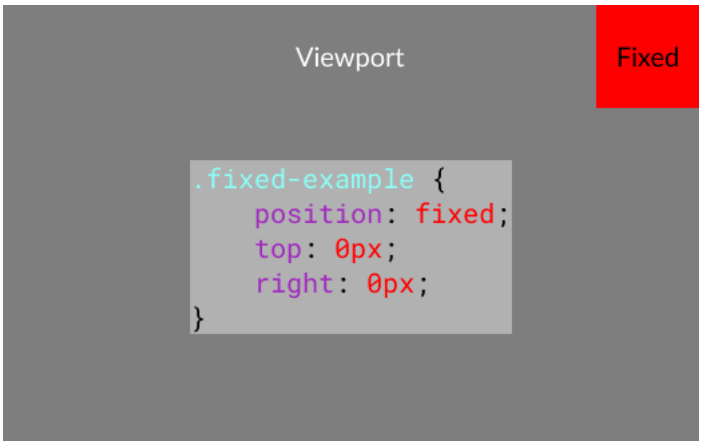
In this section, we'll talk about some more CSS properties that help us create more modern and interesting layouts.

Position

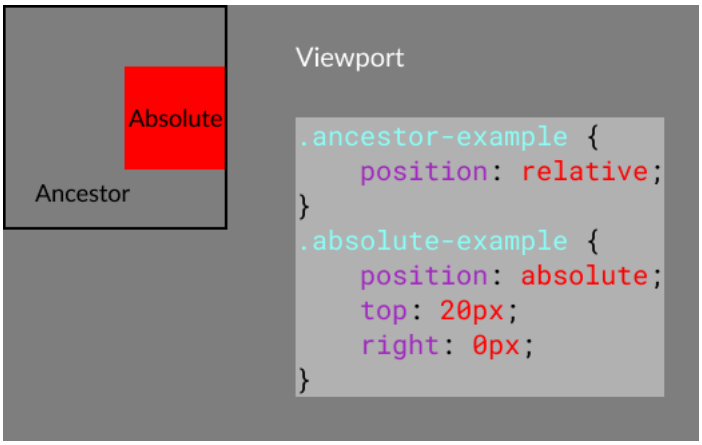
- One way to change where an element appears on the page
- There are 5 possible values: `static`, `relative`, `fixed`, `absolute`, and `sticky`
- `static` is the default value if no other value is set
- The `top`, `right`, `bottom`, and `left` properties are used in conjunction with the 4 other options to move elements
- `relative` position will move an element relative to its original position in the document



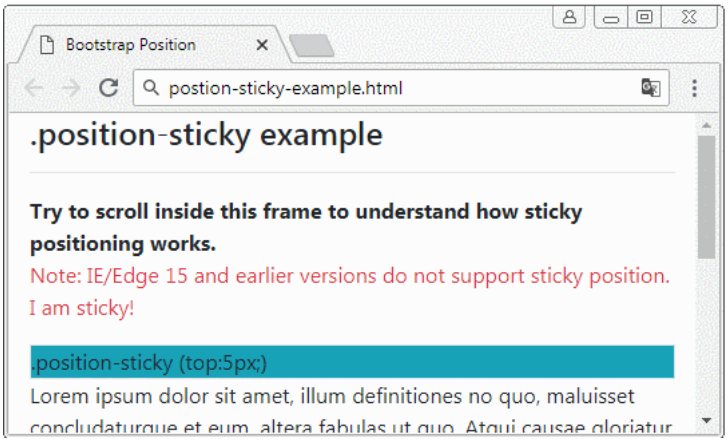
- `fixed` position will move an element relative to the viewport, regardless of scroll position



- `absolute` position will move an element relative to its nearest positioned ancestor
- If there are no positioned ancestors, it will use the viewport

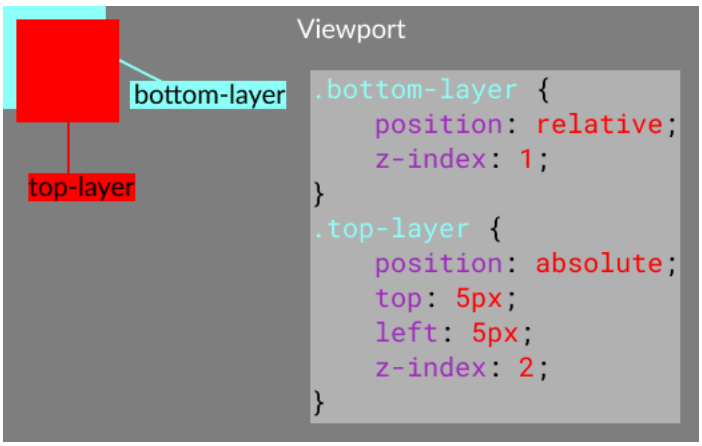


- `sticky` postition will toggle between fixed and relative, based on scroll position
- The element remains relatively positioned until the scroll hits a specified position, then toggles to fixed



Z-index

- Used to determine layers of positioned elements
- An element **must** have a position style in order to use the z-index
- The higher the value, the closer the element will be to the front



Display

- Block and inline are both default display properties, just for different elements
- There are many possible values for display, we'll cover: `block`, `inline`, `inline-block`, and `flex`
- `block` display elements stack vertically and take the whole width available
- You can set the height and width of block elements
- `inline` display elements stack horizontally
- The size of the content determines the height and width
- `inline-block` display elements stack horizontally as well
- But they also allow you to change height and width
- All 3 of these display properties should be applied directly to the element, not a parent

Flexbox

This display property is used on parent elements to arrange their contents.

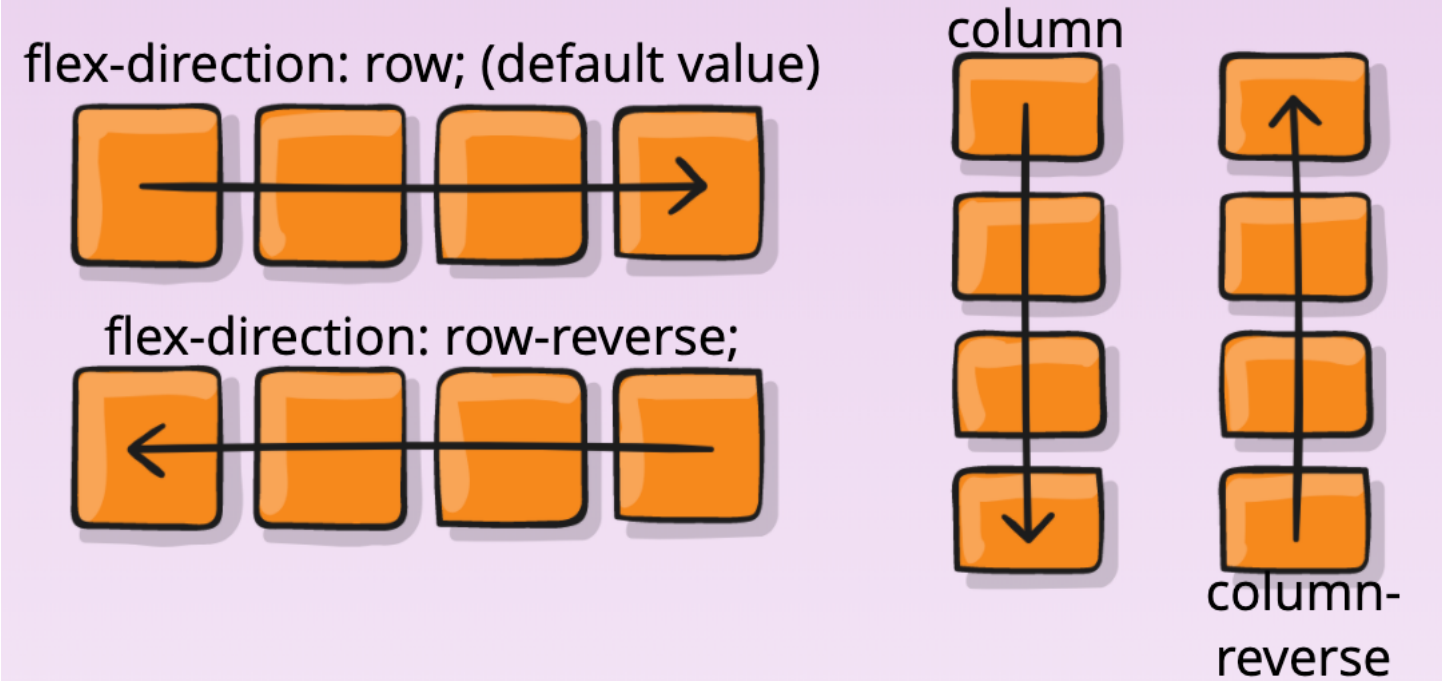
After the `display` property is set to `flex`, we have a lot of other options to customize the layout.

```
.flex-example {
    /*initialize flexbox*/
    display: flex;

    /*available properties*/
    flex-direction: row;
    flex-wrap: wrap;
    justify-content: center;
    align-items: center;
}
```

flex-direction

- The default value is `row`, which will arrange the contents horizontally, even if they're block elements!

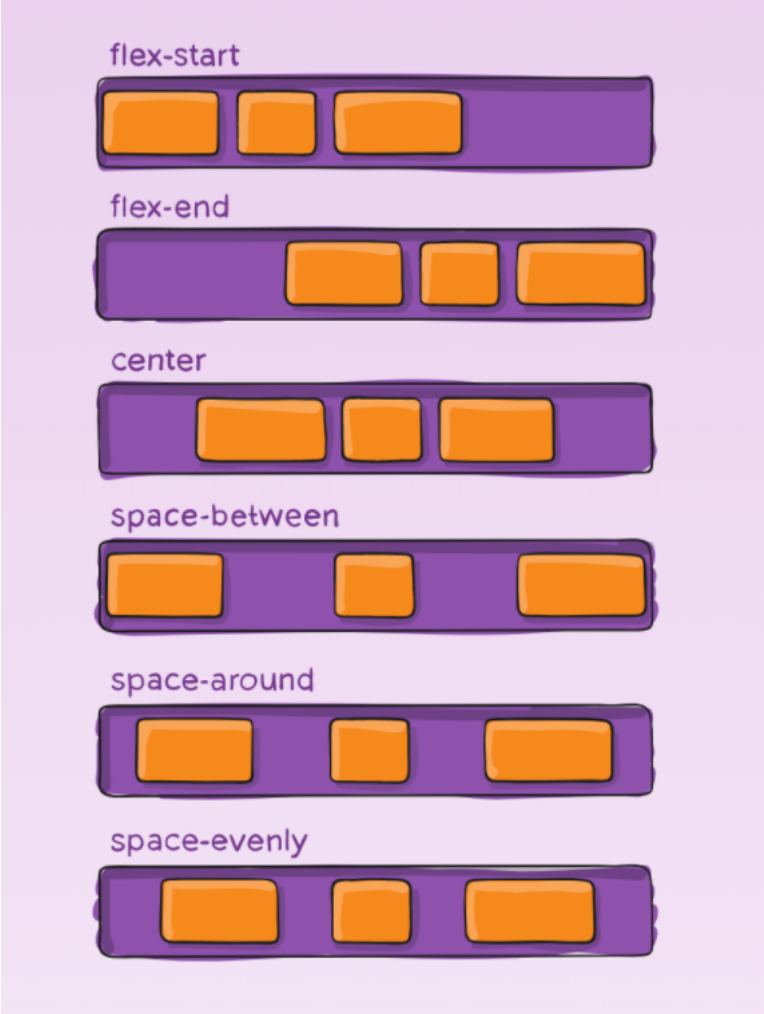


flex-wrap

- This property determines whether elements should wrap in the parent or not
- The default value is `no-wrap`
- To have them wrap, set `flex-wrap: wrap;`

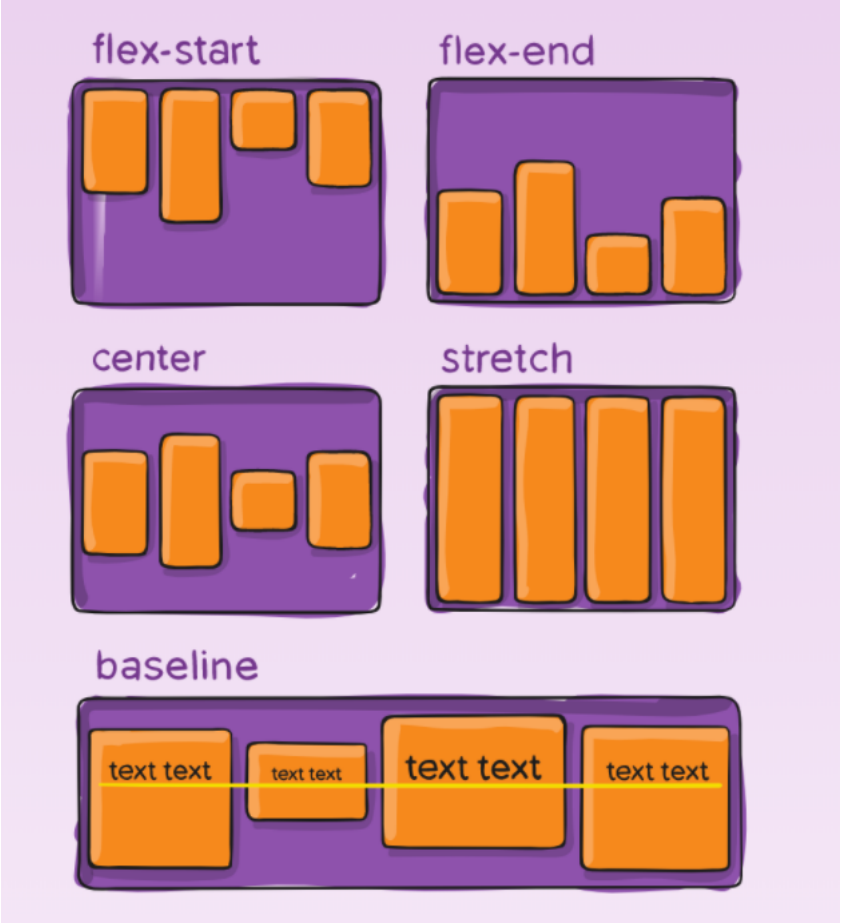
justify-content

- Determines horizontal position and spacing when the direction is `row`
- Determines vertical position and spacing when the direction is `column`
- Default value is `flex-start`



align-items

- Determines vertical position and spacing when the direction is `row`
- Determines horizontal position and spacing when the direction is `column`
- Default value is `flex-start`



Box-sizing

- CSS property that determines how the total width and height of an element is calculated
- This property accepts two values, `content-box` or `border-box`.

`box-sizing: content-box`
This is the default value, the width and height properties include the content; padding, border, and margin are added on

`box-sizing: border-box`
The width and height properties include the content, padding, and border; margin is added on, meaning that padding and border will be inside of the box with the content

```
/*total width and height will be 130 each*/
#content-box {
    box-sizing: content-box;
    width: 100px;
    height: 100px;
    border: solid blue 10px;
    padding: 5px;
    background-color: yellow;
}
```

```
/*total width and height remain 100*/
#border-box {
    box-sizing: border-box;
    width: 100px;
    height: 100px;
    border: solid blue 10px;
    padding: 5px;
    background-color: yellow;
}
```

Reset CSS

- Different browsers come with different default stylings
- This can be difficult to account for
- Reset CSS files remove default styling so that your styles are consistent across browsers
- You must link them before any other stylesheets so that they don't cancel out your styles

Summary

- Semantic HTML is used to give our HTML more meaning
- We can use classes to group together elements for styling
- We can use an id to single out one element
- Position and display properties help us with page layout
- Flexbox is a flexible and convenient way of handling layouts
- Reset CSS files help us make sure styles are consistent across browsers

The End