IN4MATX 133: User Interface Software

Lecture 3:
CSS

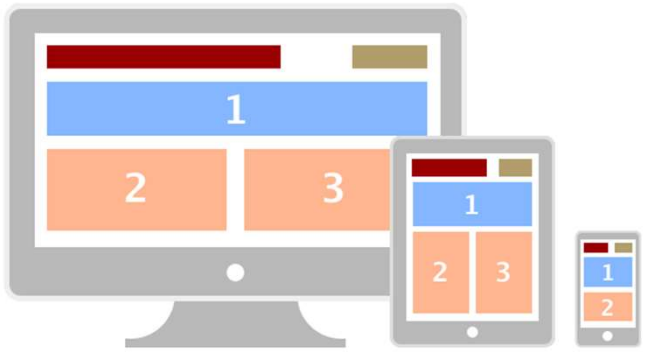
Socrative Quiz!

Enter your UCI Email when prompted name!!!
e.g.,

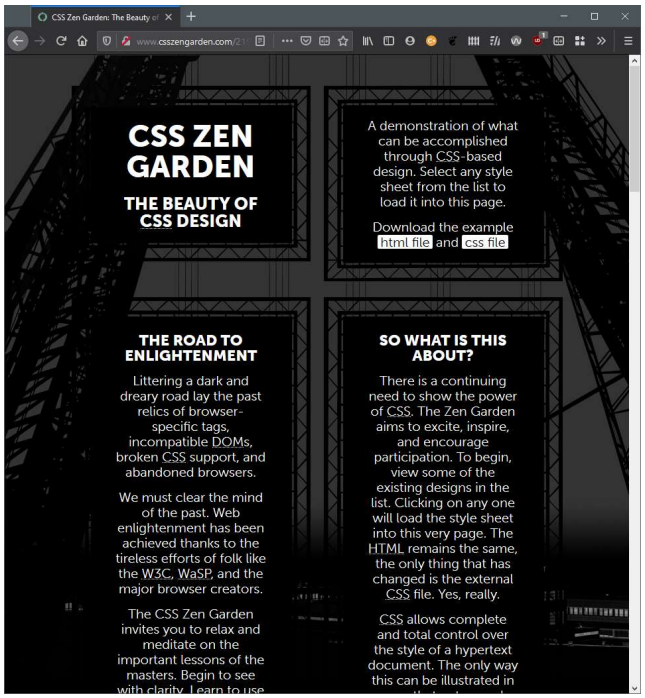
xxxxx@uci.edu

https://api.socrative.com/rc/CvereT

Today: CSS and responsive design



Same page, different stylesheets



http://www.csszengarden.com/

Today's goals

By the end of today, you should be able to…

- Explain the goals of CSS and why it exists as separate from HTML
- Describe the CSS hierarchy and fallback structure
- Utilize the box model and positioning options to arrange content
- Style nested tags with child, adjacent sibling, and general sibling selectors

CSS

Cascading Style Sheets

- Defines rules for styling
- Differs from HTML, which provides structure for the document

CSS: but why?

- Reusability
  - Apply the same style to multiple web pages
- Modularity
  - Include multiple stylesheets that apply to a single page
- Sane management
  - Files can be version controlled, separate from HTML structural content
- Maintainability
  - Styles can be contained in a single type of location (style sheets)

Ok, so how do I write CSS?

CSS syntax

- **Selectors** specify which elements a **rule** applies to
- **Rules** specify what *values* to assign to different formatting **properties**

```
/* CSS Pseudocode */
selector {
  property: value;
  property: value;
  ...
}
```

One rule, many properties

CSS syntax

Apply to all h1 tags

```
h1 {
  font-family: 'Arial';
  color: blue;
  background-color: #ff0000; /*red*/
}
```

"font"

- Link to stylesheets in HTML's **<head>**

```
<head>
  <link rel="stylesheet" href="my-style.css">
</head>
```

relation between
this page and reference

no content,
so no closing tag

Element, ID, and Class selectors

- element: what tag is being styled

```
p {
  font-family: 'Arial';
  color: red;
}
```

- class: a type of element

```
.emphasize {
  font-family: 'Arial';
  color: red;
}
```

- id: one specific element

```
#redtext {
  font-family: 'Arial';
  color: red;
}
```

HTML Class and ID attributes

```
<div class="widget foo" id="baz"></div>
```

- Variable-value just like any other attribute (`href`, `src`)
- An element can have many classes, only one ID
- Each page can have only one element with a given ID
  - Required to pass validation
- Can use the same class on multiple elements
  - And should; it's useful to apply the same style to many elements

https://css-tricks.com/the-difference-between-id-and-class/

HTML Class and ID attributes

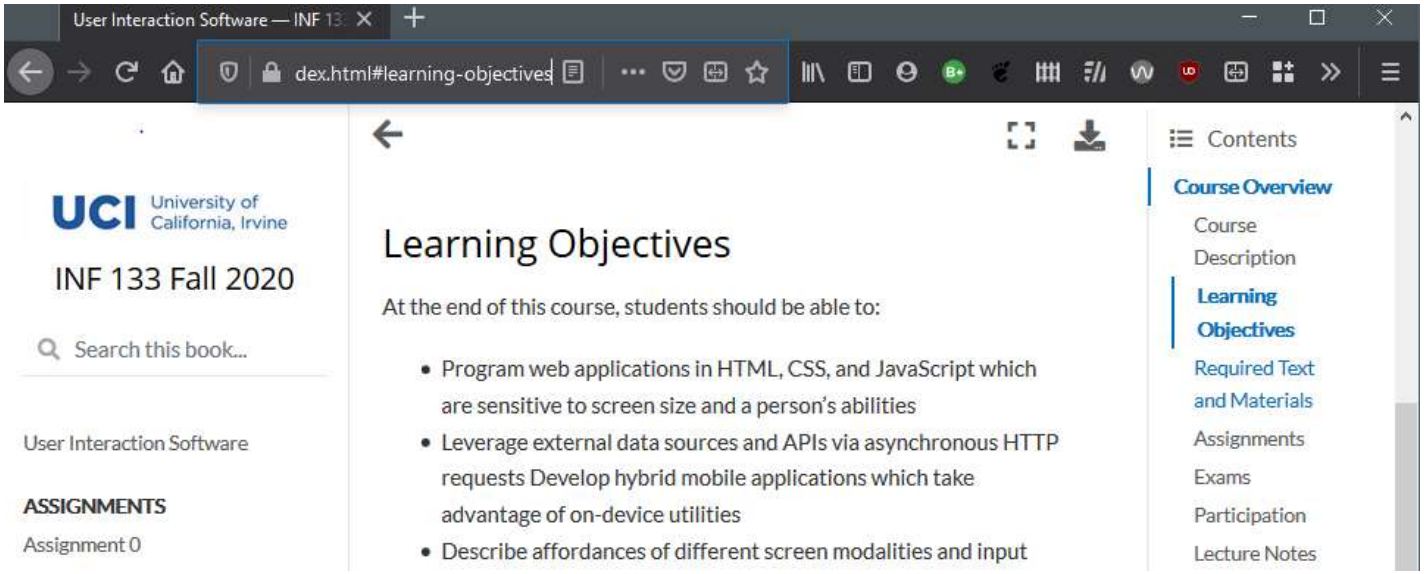
```
<div class="widget foo" id="baz"></div>
```

```
div.widget.foo#baz {
  /*can chain selectors together!*/
}
```

https://css-tricks.com/the-difference-between-id-and-class/

HTML Class and ID attributes

- Fun trick: IDs can be used for navigation
- `http://example.com/#id`



https://css-tricks.com/the-difference-between-id-and-class/

CSS properties

- `font-family`: the “font” (fallback alternatives separated by commas)
- `font-size`: the size of the text
- `font-weight`: boldness
- `color`: text color
- `background-color` (element’s background)
- `opacity` (transparency)
- And much, much more!

http://www.w3schools.com/cssref/default.asp

DEMO

https://repl.it/@m5b/inf133-css-demo

HTML vs. CSS

- HTML specifies the *semantics*
- CSS specifies the *appearance*

HTML vs. CSS

```
<!--HTML-->
<p> This text is <em>emphasized!</em></p>
```

```
<!--HTML-->
<p> This text is also
<i>emphasized!</i></p>
```

Says nothing about appearance

Conflates appearance and semantics

This text is *emphasized*

This text is also *emphasized*

Cascading Style Sheets

- Multiple rules can apply to the same element (in a “cascade”)

```
<!--HTML-->
<p class="big blue">
  This tag has two classes: "big" and "blue"
  (classes are separated by spaces)
</p>

/* CSS */
p { font-family: 'Verdana'; }
.big { font-size: larger; }
.blue { color: blue; }
```


Apply to all <p> tags


Apply to all with class=“big”


Apply to all with class=“blue”

Cascading Style Sheets

- CSS rules are also inherited from parent tags

```
<div class="content"> <!-- has own styling -->
  <div class="sub-div"> <!-- has own styling + .content styling -->
    <ol class="my-list"> <!-- own styling + .sub-div + .content -->
                         <!-- own style is ol AND .my-list rules-->

      <!-- li styling + .my-list + .sub-div + .content -->
      <li>Item 1</li>
      <li>Item 2</li>
      <li>Item 3</li>
    </ol>
  </div>
</div>
```

Cascading Style Sheets

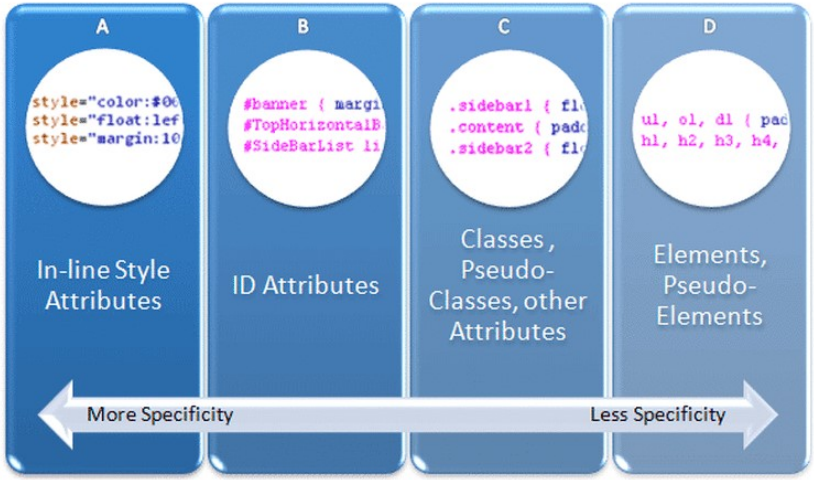
- Rules are applied **in order** (last rule always wins among peer selectors)

```
<!--HTML-->
<p class="red green">
  <em class="blue">Text is blue!</em>
</p>
```

```
/* CSS */
p { font-family: 'Verdana'; }
.red { color: red; }
.green { color: green; }
.blue { color: blue; }
```

Specifying styles

- CSS specificity is *calculated* based on which selector designates it
- General rule: rule that's "closer to the HTML element" applies
- This is difficult stuff, usually trial-and-error resolves most things



https://www.techrepublic.com/blog/web-designer/css-specificity-hierarchy-what-you-need-to-know/

Positioning

- HTML tags are either*:
  - **Block** elements (line break after them)
  - **Inline** elements (no line break)

```
<p>
  This is on a line.
  <em>This is on the same line.</em>
</p>
<p>This will be on a new line.</p>
```

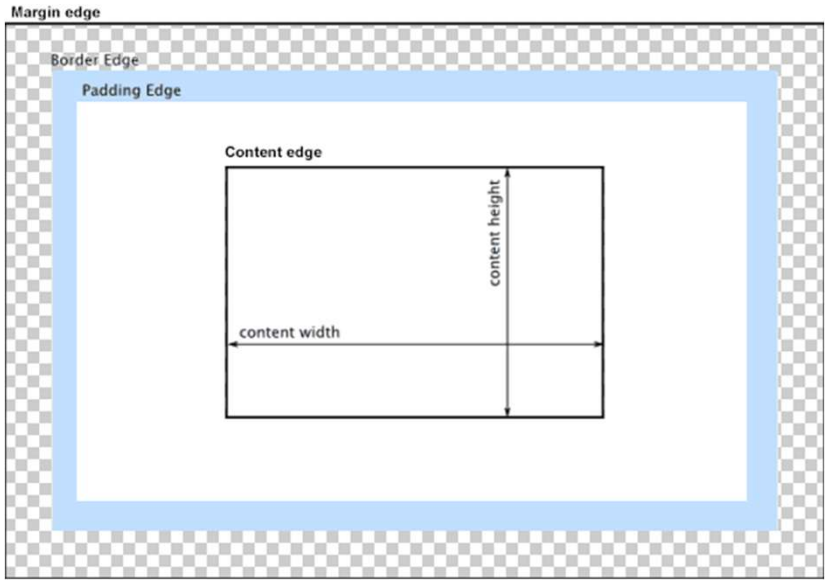
Block

Inline

- Don’t put block elements inside inline elements!

Positioning: box model

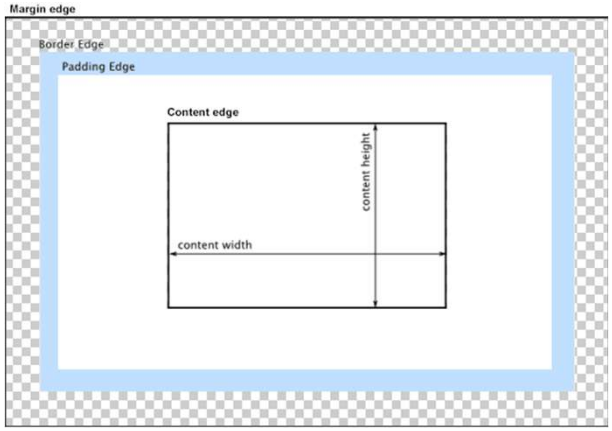
- **Content** contains “real” content
- **Padding** extends content area
- **Border** is similar
- **Margin** is intended to separate elements from neighbors



https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Box_Model/Introduction_to_the_CSS_box_model

Positioning: box model

- Content dimensions are specified with `width` and `height`
- `padding`, `border`, and `margin` have direction properties (e.g., `padding-top`, `margin-right`, `border-left`)
- `border` can have `border-color`, `border-width`, and `border-style`
- Content color (e.g., `background-color`) extends into padding

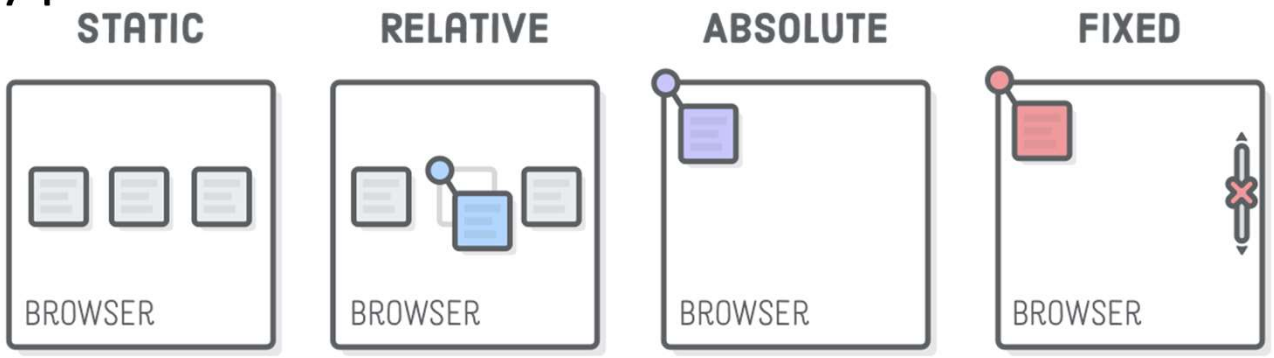


https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Box_Model/Introduction_to_the_CSS_box_model

Positioning

- All positioning is relative to the parent
  - If you nest tags, the child's margins, etc. are all dependent on parent's

Positioning: types



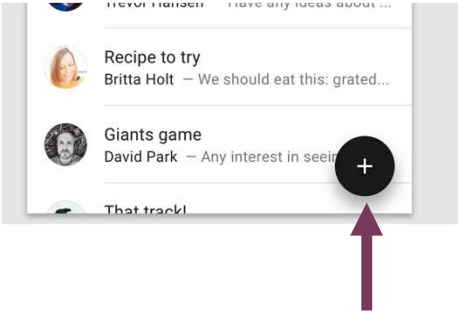
- `static` (default)
- `relative` (offset from default)
- `absolute` (from top-left)
- `fixed` (absolute + floating, fixed to the viewport)

https://internetingishard.com/html-and-css/advanced-positioning/

Positioning: types

- `static` and `relative` follow the overall flow of a page
  - `relative` helps make adjustments to the flow
- `absolute` and `fixed` ignore it entirely
  - But they're helpful in some cases, like floating action buttons (FABs)

Relative position

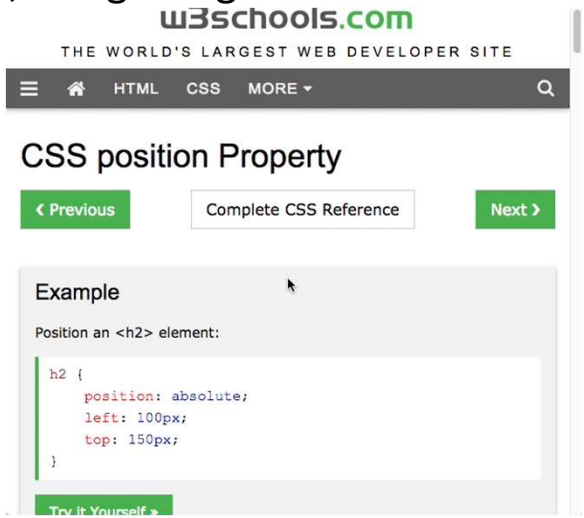


Absolute position

https://css-tricks.com/examples/AbsoluteInsideRelative/

Positioning: types

- `sticky` will stop when a user scrolls past it
  - Useful for menus
  - Not all browsers support it, but getting there



https://css-tricks.com/examples/AbsoluteInsideRelative/

DEMO

https://repl.it/@m5b/inf133-css-demo

Units

- Pixels (`px`), element units (`em`), percentages (`%`), real-world units (`in`, `cm`)
- Use relative units (`em`, `%`) whenever possible
- Helps accessibility, people with low vision change default size (usually 16px)
  - Em fonts scale from the default, a 30px heading stays 30px
- Also useful to vary based on screen size
  - More on how to do that next lecture

	Recommended	Occasional use	Not recommended
Screen	em, px, %	ex	pt, cm, mm, in, pc
Print	em, cm, mm, in, pt, pc, %	px, ex	

https://engageinteractive.co.uk/blog/em-vs-rem-vs-px

Advanced selectors

- Extremely useful for making clean stylesheets
- Add a top margin for all `h2`s that follow a `p`aragraph

```
p + h2 {
  margin-top: 10px;
}
```

- Or only in a particular `div`

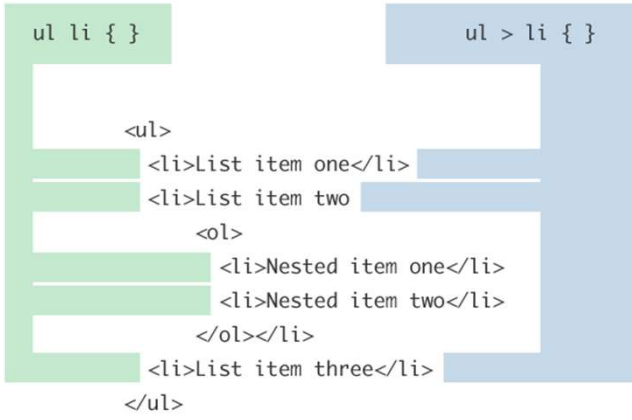
```
div.post p + h2 {
  margin-top: 10px;
}
```

https://www.smashingmagazine.com/2009/08/taming-advanced-css-selectors/

Advanced selectors

Child and Descendant Selectors

- `ui li`
  - Select *all* children and grandchildren
- `ui > li`
  - Select *direct* children (not grandchildren)



https://css-tricks.com/child-and-sibling-selectors/

Fonts & fallbacks

- Browsers will try fonts in order

```
p {
    font-family: "Times New Roman", Times, serif;
}
```

- Google Fonts is a great resource

```
<!--HTML-->
<link
href="https://fonts.googleapis.com/css?family=Roboto"
rel="stylesheet">
/* CSS */
font-family: 'Roboto', sans-serif;
```

https://fonts.google.com/

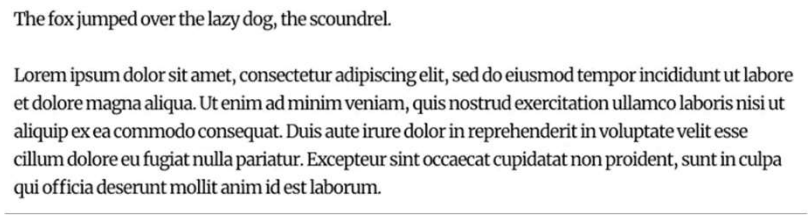
Fallbacks in HTML

- Work similarly to CSS

```
<video autoplay>
  <!--webm not supported in IE or Safari-->
  <source src="lecture3.webm" type="video/webm" />
     <!--mp4 supported in modern browsers, but lower
quality-->
     <source src="lecture3.mp4" type="video/mp4" />
     <!--backup important for some old browsers-->
     <img src="lecture3.jpg" title="Your browser does not
support the <video> tag">
</video>
```

Fallbacks: why?

- Format not supported (webm, ogg, flac)
- Font might not support certain characters
- Might take time to load (“flash of unstyled text”)
    - Pick a similar default font

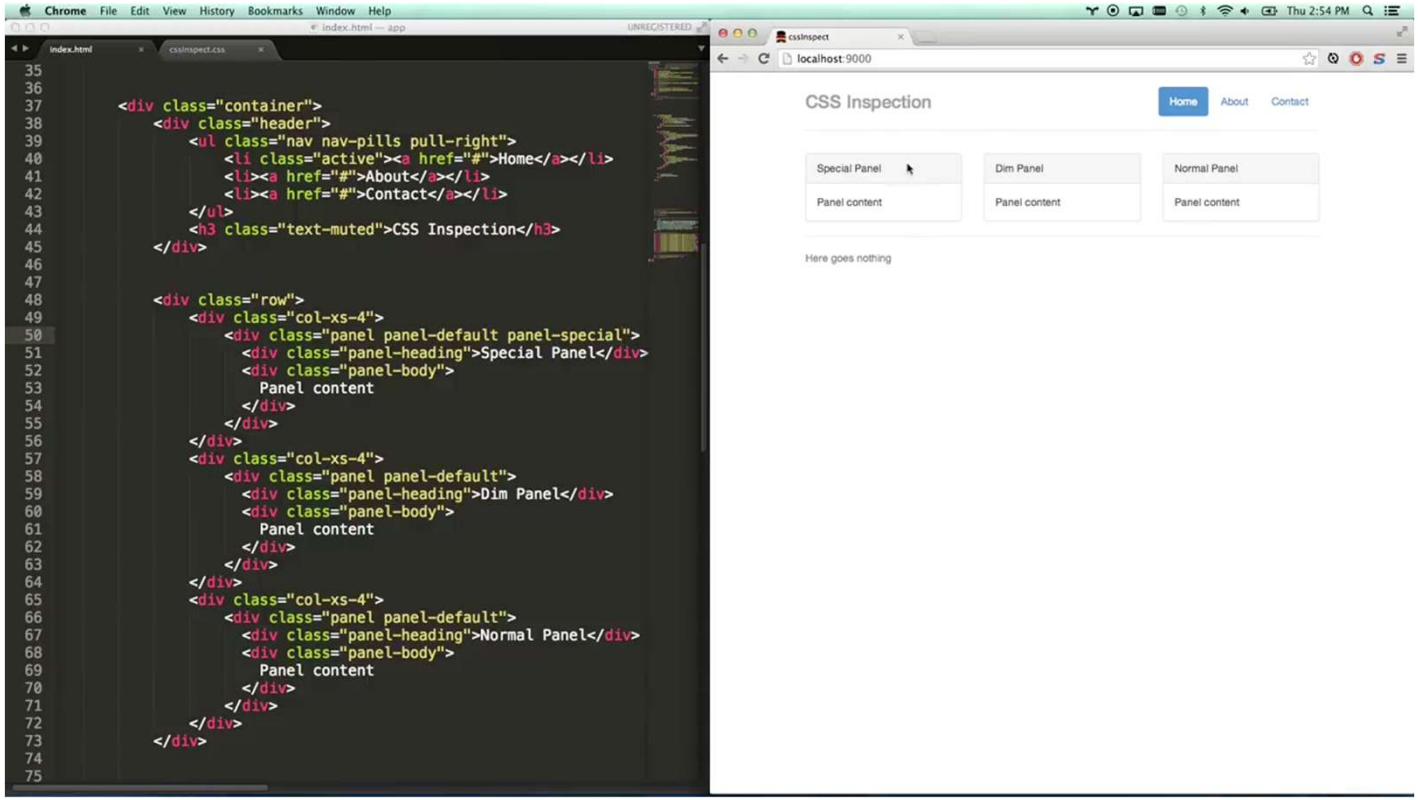


https://css-tricks.com/css-basics-fallback-font-stacks-robust-web-typography/

There’s a lot to CSS.
I can’t create much
from memory alone.

References

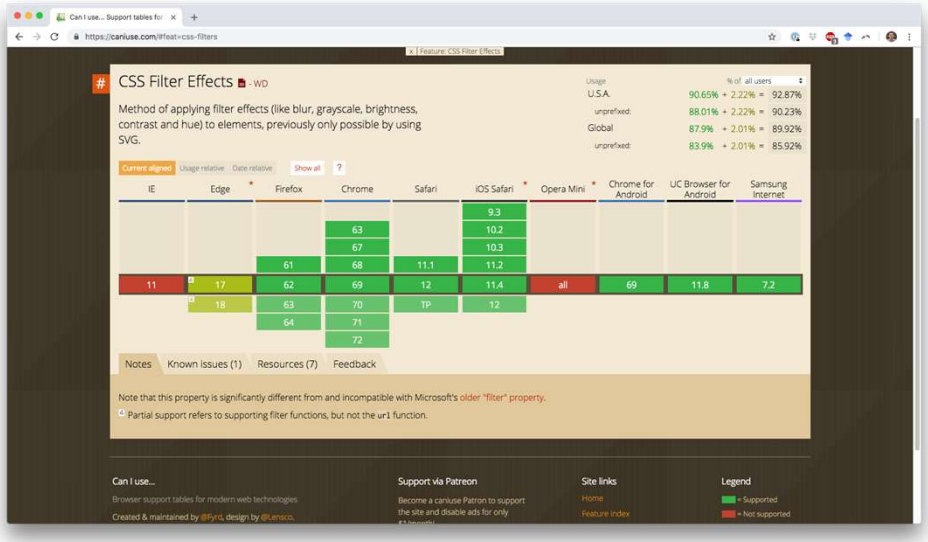
- https://www.w3schools.com/cssref/
- https://cssreference.io/
- https://developer.mozilla.org/en-US/docs/Web/CSS/Reference
- https://www.codecademy.com/learn/learn-css



https://www.youtube.com/watch?v=Z3HGJsNLQ1E

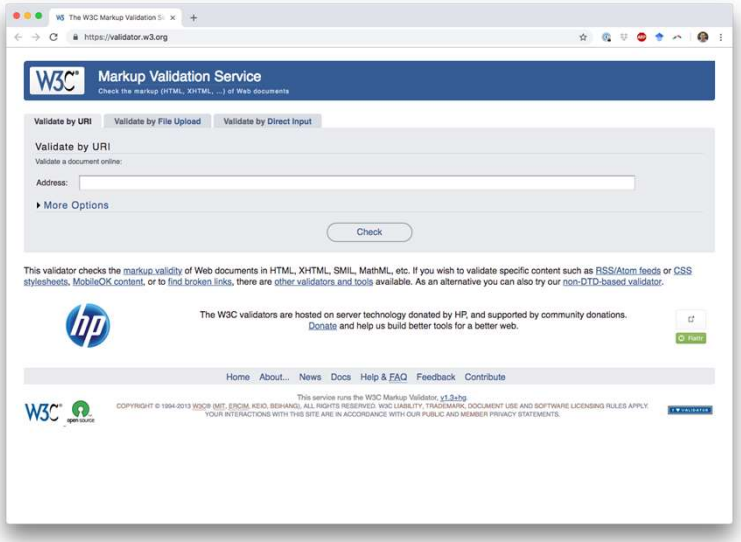
Browser compatibility

Used to be a much bigger issue, but still worth checking



https://caniuse.com/

Validation



https://validator.w3.org

Today's goals

By the end of today, you should be able to…

- Explain the goals of CSS and why it exists as separate from HTML
- Describe the CSS hierarchy and fallback structure
- Utilize the box model and positioning options to arrange content
- Style nested tags with child, adjacent sibling, and general sibling selectors

Additional slides

Specifying styles

Inline Styling

```
<p style="font-family='Arial'; color=red;">
  Red text
</p>
<p style="font-family='Arial'; color=red;">
  More red text
</p>
```

- Supported, but usually bad practice
  - Goes against DRY principles of programming (Don’t Repeat Yourself)

Specifying styles

Internal Styling

```
<head>
  <style type="text/css">
    p {font-family:'Arial'; color:red;}
  </style>
</head>
<body>
  ...
</body>
```

- Just putting CSS into the `<head>` of your HTML

Specifying styles

External Styling

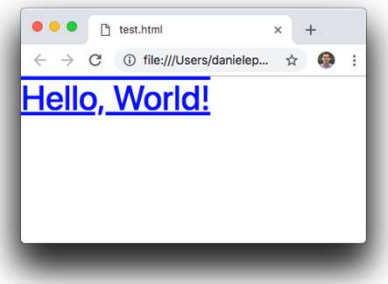
```
<head>
  <link rel="stylesheet" href="./css/style.css">
</head>
<body>
  ...
</body>
```

- Generally a best practice
  - Aligns with the idea of separating structure from style

Specifying styles

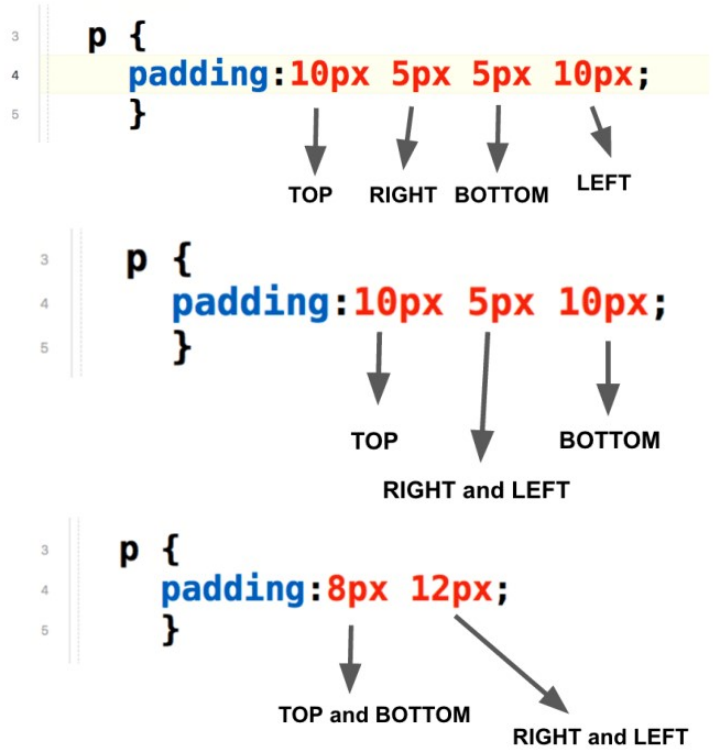
- External styles apply in order, too!

```
<head>
    <link rel="stylesheet"
    href="./css/bootstrap.css">
    <link rel="stylesheet"
    href="./css/style.css">
</head>
<body>
    <h1>Hello, World!</h1>
</body>
```



Positioning: shorthand

- Multiple values can be specified in one line
- Difficult to remember
- Being explicit improves readability at the expense of brevity



https://css-tricks.com/remember-the-order-of-marginpadding-shorthand-with-trouble/

Borders: shorthand

- Multiple values can be specified in one line
- Slightly easier to remember
- Maybe even more readable

```
div {
    border-bottom-width: 3px;
    border-bottom-style: solid;
    border-bottom-color: red;
}

div.equivalent {
    border-bottom: 3px solid red;
}
```

https://www.w3schools.com/css/css_border.asp

Pseudo-classes

- Define a special state of an element

```
/* CSS Pseudocode */
Selector:pseudo-class {
  property: value;
  property: value;
  ...
}
```

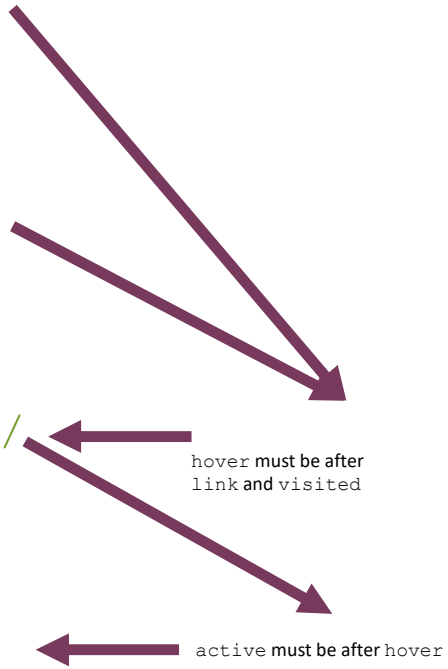
Pseudo-classes

```
a:link { /* unvisited link */
    color: #FF0000;
}

a:visited { /* visited link */
    color: #00FF00;
}

a:hover { /* mouse over link */
    color: #FF00FF;
}

a:active { /* selected link */
    color: #0000FF;
}
```



https://www.w3schools.com/Css/css_pseudo_classes.asp