HTML & CSS

CS142 Section 1

remember to record

Agenda

I. HTML vs. CSS (high level review) ~ *5 min*
II. HTML ~ *10 min*
    A. Tags
    B. Attributes
III. CSS ~ *35 min*
    A. Rules
    B. Must-Knows (Rules, Display Models, Useful Features)

Agenda

I. HTML vs. CSS (high level review) ~ *5 min*
II. HTML ~ *10 min*
    A. Tags
    B. Attributes
III. CSS ~ *35 min*
    A. Rules
    B. Must-Knows (Rules, Display Models, Useful Features)

HTML vs. CSS

HTML = scaffolding / structure

- apply "meaning/structure" to content (rather than style)
- i.e. ⇒ Heading & Paragraph
- **NOT** ⇒ 16pt font; Arial

Example:

- **Bad** ⇒ `<i>` & `<b>`
  - styling tags
- **Preferred** ⇒ `<em>` & `<strong>`
  - better semantics
  - accessibility-friendly

HTML vs. CSS

CSS = painting / styling / (some) interactivity

- styling for the semantic content expressed in HTML
- CSS Rule ⇒ Selector + Declaration(s)

"Don't Repeat Yourself"

- Selector lets you apply the same style to multiple elements
- Modify the declaration to change them all at once

Agenda

I. HTML vs. CSS (high level review) ~ *5 min*
II. HTML ~ *10 min*
    A. Tags
    B. Attributes
III. CSS ~ *35 min*
    A. Rules
    B. Must-Knows (Rules, Display Models, Useful Features)

Agenda

I. HTML vs. CSS (high level review) ~ *5 min*
II. HTML ~ *10 min*
    A. Tags
    B. Attributes
III. CSS ~ *35 min*
    A. Rules
    B. Must-Knows (Rules, Display Models, Useful Features)

HTML

XHTML Structure Review

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <p>Hello world!</p>
  </body>
</html>
```

Use lines verbatim

HTML: Tags (<head>, <body>)

<head> = contains title of page, link to CSS stylesheet, sometimes scripts

<body> = content rendered by the browser (text, tables, images, etc.)

*See Lecture 2, Slides 16-18 for more common tags

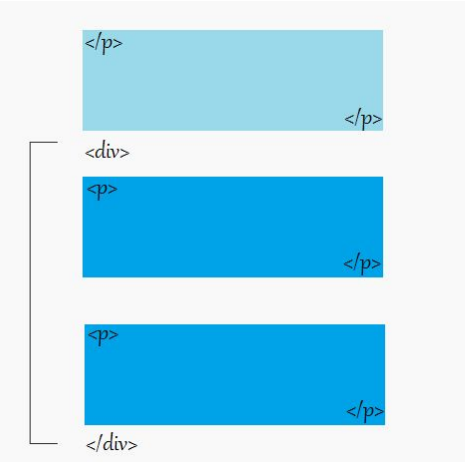
HTML: Tags (,)

** & **

- define presentation of document (no semantic meaning)
- used to "group" sections of the document

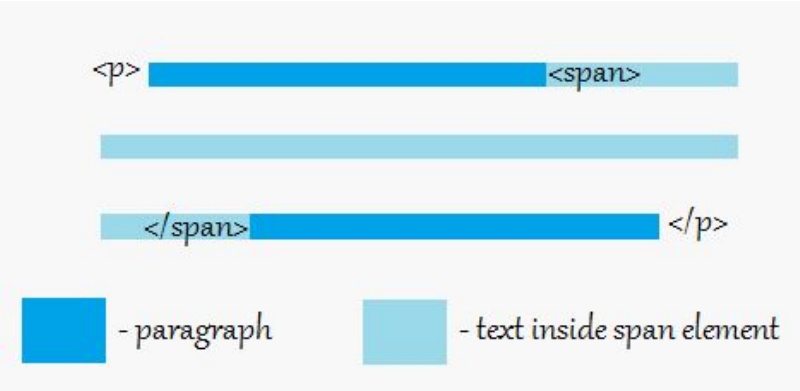
 = **block-level** element

- usually used as container *for elements*



 = **inline** element

- usually used as container *for text* within larger text



HTML: Attributes

Gives element a custom "attribute" so that it can be picked out by a CSS selector

Two methods:

1. **ID:** *uniquely* identifies object within a document *[DO NOT REUSE]*
2. **Class:** a "group" an element belongs to *[CAN (and, when possible, should) REUSE]*

HTML

```
<div id="large">
<div class="large">
<div class="large small">
```

CSS

```
#large { // CSS styling by id }
.large { // CSS styling by class }
.small { // More CSS styling by class}
```

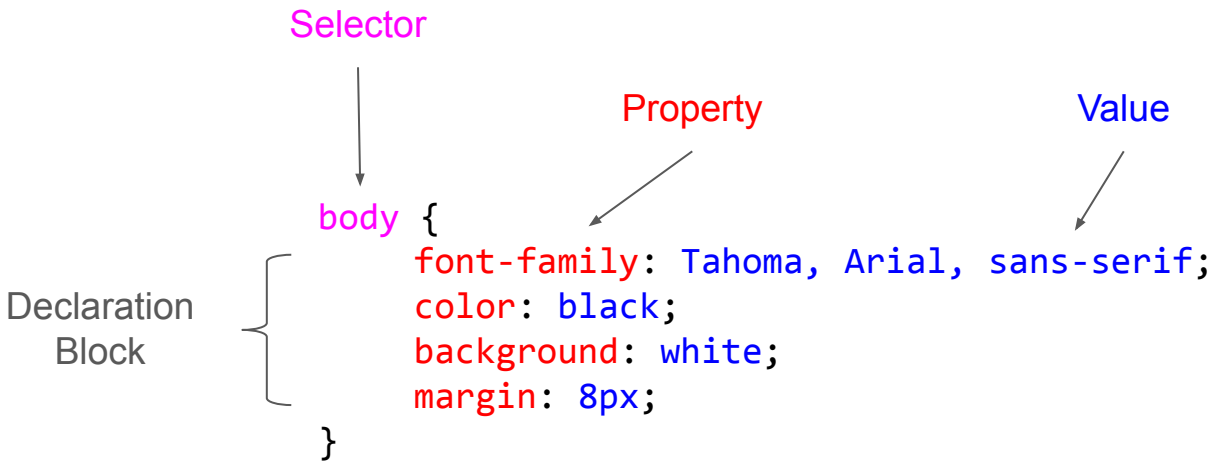
Agenda

I. HTML vs. CSS (high level review) ~ *5 min*
II. HTML ~ *10 min*
    A. Tags
    B. Attributes
III. CSS ~ *35 min*
    A. Rules
    B. Must-Knows (Rules, Display Models, Useful Features)

Agenda

I. HTML vs. CSS (high level review) ~ *5 min*
II. HTML ~ *10 min*
    A. Tags
    B. Attributes
III. CSS ~ *35 min*
    A. Rules
    B. Must-Knows (Override Rules, Display Models, Useful Features)

CSS: **Rules** (defined in your .css file)



CSS: **Rules** (defined in your .css file)

Selector: pick out element(s) by tag, class, or id.

- Ex: `p {}, .large {}, #container {}`
- Pseudo-classes: `:hover {}`
- Combination: `p.large {}, div:hover {}`

Declaration: property-value pair, specifying the property

- Ex: `height: 300px; font-family: Arial`

Declarations live inside Selectors (see previous slide)

CSS: Must-Knows

Override Rules ⇒ Cascade, Inheritance

CSS Display Models ⇒ Box Model, Flexbox

Useful Features ⇒ Positioning, Shorthand, Pseudo-classes

CSS: Must-Knows

Override Rules ⇒ Cascade, Inheritance

CSS Display Models ⇒ Box Model, Flexbox

Useful Features ⇒ Positioning, Shorthand, Pseudo-classes

Override Rules ⇒ Cascade

CSS is "cascading" because of fixed priority of rules if >1 conflicting rule refers to the same object.

Note — if two rules don't conflict, *both* are applied!

1. **Importance**
   Ex: `color: red !important;`
   - `!important` properties override normal ones (**but strongly discouraged**)
   - user stylesheets (e.g. browser default) overridden by author stylesheets
2. **Specificity**
   Ex: `<div style="font-size: 10px;">`
   - Declaration in `style` attribute has highest priority
   - Priority: id > class/pseudo-class > element type (e.g. `#first > .tab > div`)
3. **Source Code Order**
   - Last one wins

https://www.w3.org/wiki/Inheritance_and_cascade

Override Rules ⇒ Inheritance

CSS

```
div {
    font-size: 15px;
}
```

HTML

```
<div> <p> This font is 15px. </p> </div>
```

But not everything is inherited! ⇒ (Recall Lecture 3, Slide 16)

Override Rules ⇒ Inheritance

Some properties inherit automatically (i.e. font size, font color) while others do not (background color)

Force inheritance by using the `inherit` property:

```
div { background-color: inherit; }
```

Percentages are a computed value and imply inheritance, so parent property needs to be defined)

```
div { font-size: 75%; }
```

More on cascade and inheritance:
https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/Cascade_and_inheritance

CSS: Must-Knows

Override Rules ⇒ Cascade, Inheritance

CSS Display Models ⇒ Box Model, Flexbox, Grid

Useful Features ⇒ Positioning, Shorthand, Pseudo-classes

Display Models ⇒ Box Model

`width/height` properties refer to the **content only** (by default)

- `box-sizing: content-box`
- "Actual width" = content width + padding + border

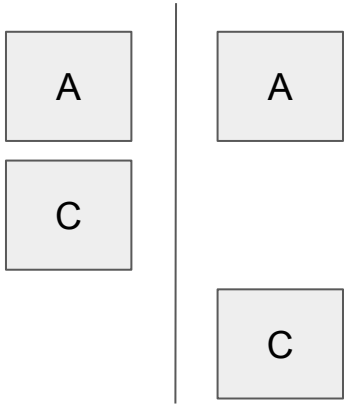
If you want `width/height` to include padding and border, set

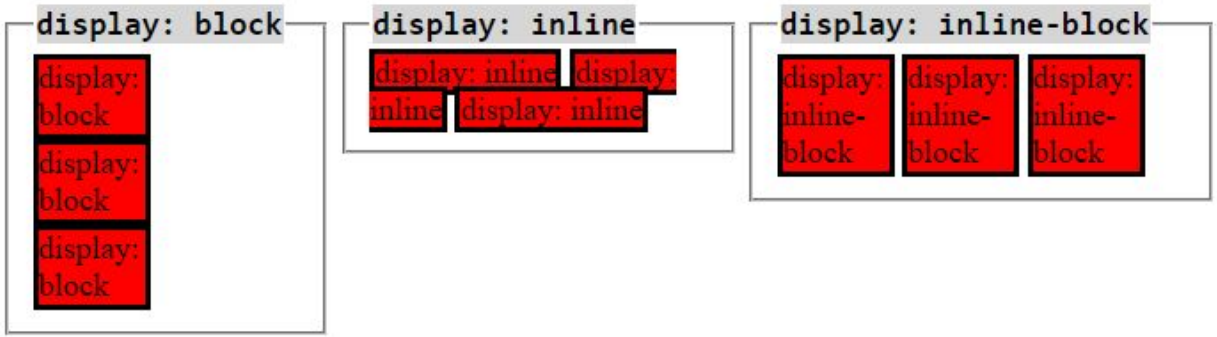
- `box-sizing: border-box`

Display Models ⇒ Display

```
#element {
    display: none;
}
```

- none (as opposed to visibility: hidden)
- block, inline, inline-block, float
- *flex
- *grid



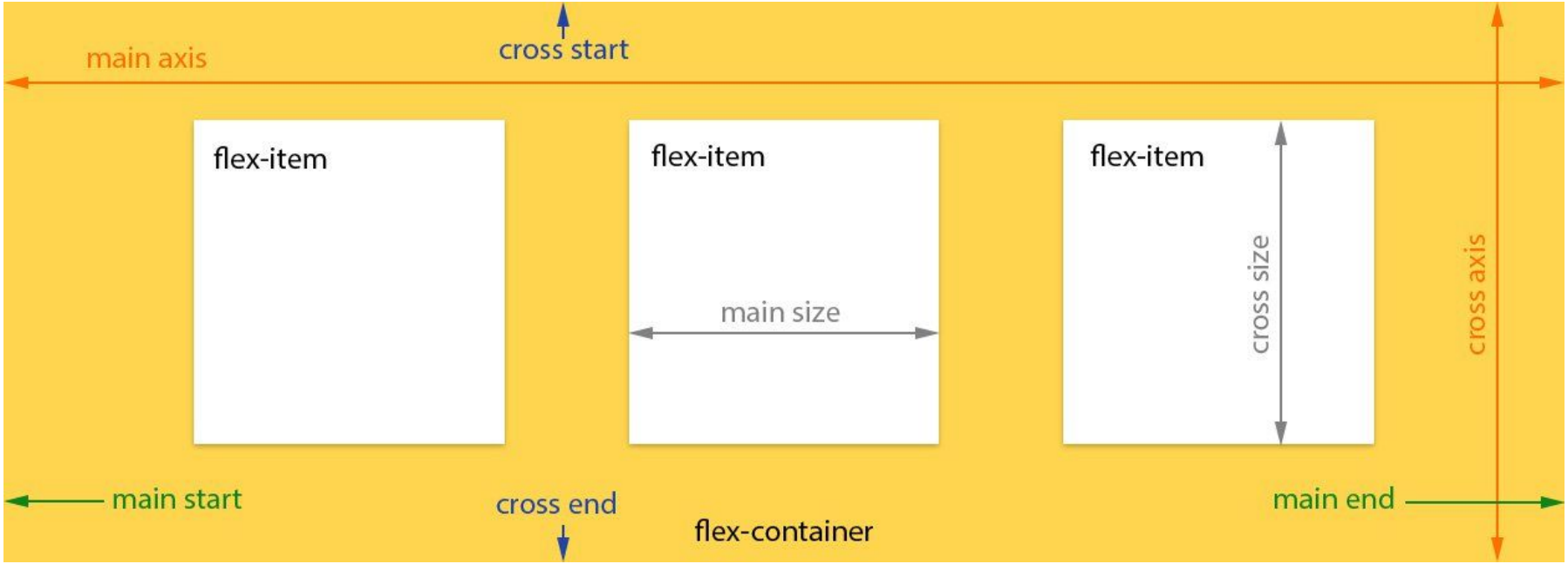


http://dustwell.com/div-span-inline-block.html

Display Models ⇒ display: flex

```
<div style="display: flex">
    <div>1</div>
    <div>2</div>
    ...
</div>
```

`cross` and `main` instead of `width` and `height`

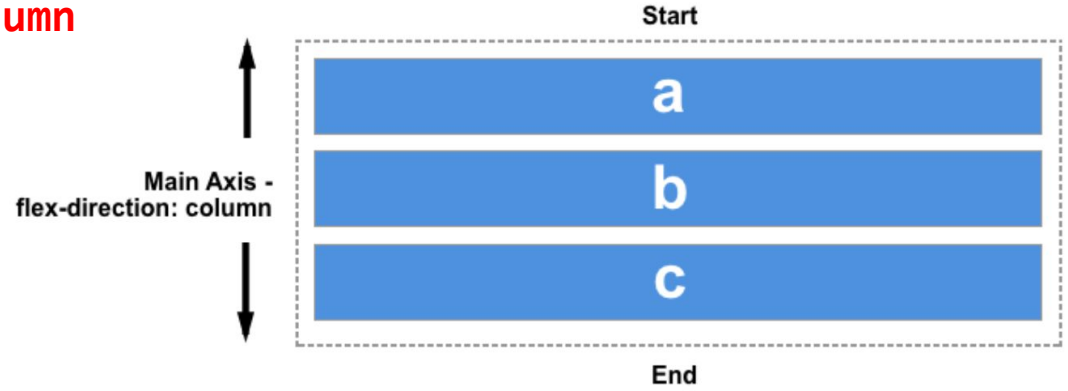


https://scotch.io/tutorials/a-visual-guide-to-css3-flexbox-properties
https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Display Models ⇒ display: flex *(flex container direction)*


```
display: flex
flex-direction: // row or column
```

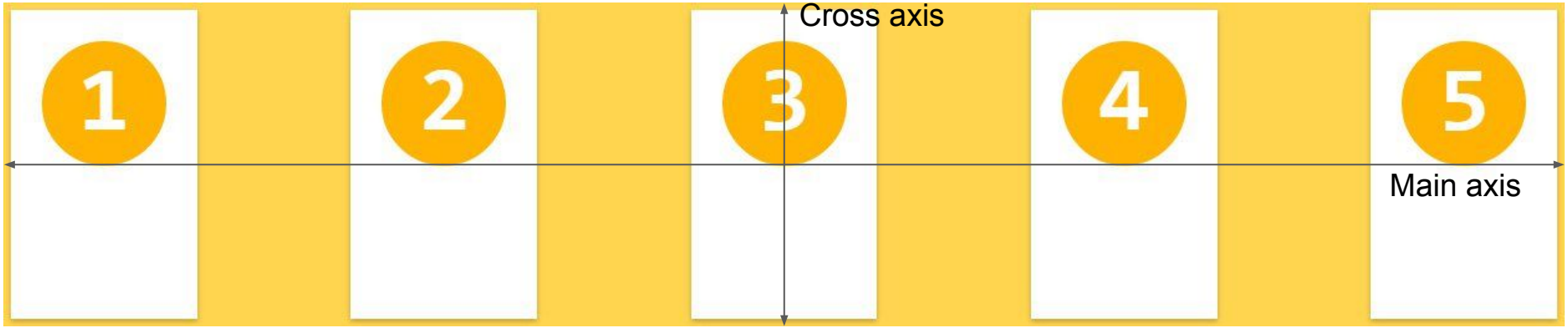
- 1-Dimension Layout



https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Ordering_Flex_Items

Display Models ⇒ display: flex *(alignment)*

- **`justify-content:`** determines how items spaced out along *main* axis
  - `left/right/center, space-between, space-around, etc.`
- **`align-items:`** determines how items aligned on *cross* axis



`flex-direction: row`

Flex documentation: https://scotch.io/tutorials/a-visual-guide-to-css3-flexbox-properties
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbo
Justify Content Demo: https://developer.mozilla.org/en-US/docs/Web/CSS/justify-content

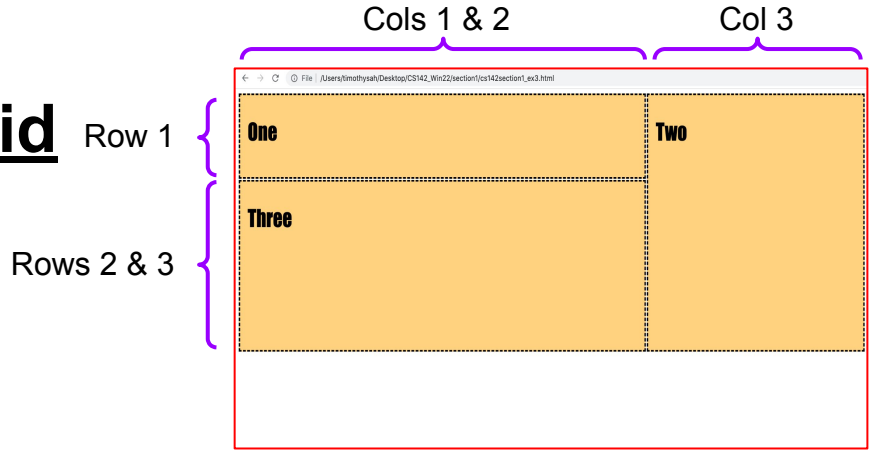
Display Models ⇒ display: flex *(sizing)*

Elements can expand (or shrink) to fill available space

https://scotch.io/tutorials/a-visual-guide-to-css3-flexbox-properties

Display Models ⇒ display: grid

- 2-Dimension layout
- Useful for dividing up available space equally among elements, alignment, and automatically handling different window/display sizes
- *grid-column* and *grid-row* are (inclusive) / (exclusive)



HTML	CSS

```html
<div class="grid-container">
    <div class="one">One</div>
    <div class="two">Two</div>
    <div class="three">Three</div>
</div>
```

```css
.grid-container {
  display: grid;
  grid-template-columns: repeat(3, auto);
  grid-template-rows: repeat(3, 150px);
  grid-gap: 2px;
}

.one {
  grid-column: 1 / 3;
  grid-row: 1;
}

.two {
  grid-column: 3;
  grid-row: 1 / 4;
}

.three {
  grid-column: 1 / 3;
  grid-row: 2 / 4;
}
```


**Review documentation for more — https://www.w3schools.com/css/css_grid.asp

CSS: Must-Knows

Override Rules ⇒ Cascade, Inheritance

CSS Display Models ⇒ Box Model, Flexbox

Useful Features ⇒ Positioning, Shorthand, Pseudo-classes

Useful Features: CSS Positioning

`position: static;`
- Item is positioned as it falls in the flow of the document (default).

`position: relative;`
- Item is *offset* relative to where it would be with `static`, using `top, left, right,` and/or `bottom` properties.

`position: fixed;`
- Item is positioned relative to browser window (viewport) with `top, left, right,` and/or `bottom` properties. This means if you scroll, item stays put.

`position: absolute;`
- Item is positioned relative to closest "positioned" (non-`static`) ancestor, offset by `top, left, right,` and/or `bottom` properties.

https://css-tricks.com/absolute-relative-fixed-positioining-how-do-they-differ/

Useful Features: CSS Shorthand

```
background-color: #000;
background-image: url(images/bg.gif);
background-repeat: no-repeat;
background-position: top right;
```

Can consolidate

```
background: #000 url(images/bg.gif) no-repeat top right;
```

Consolidated

```
margin: 10px 20px 20px 10px;
    // (top, right, bottom, left)
```

*If unsure of shorthand order, check documentation

Useful Features: Pseudo-Class Selectors

```
:nth-child(odd) // even, 8

:first-child :last-child

:hover :focus
```

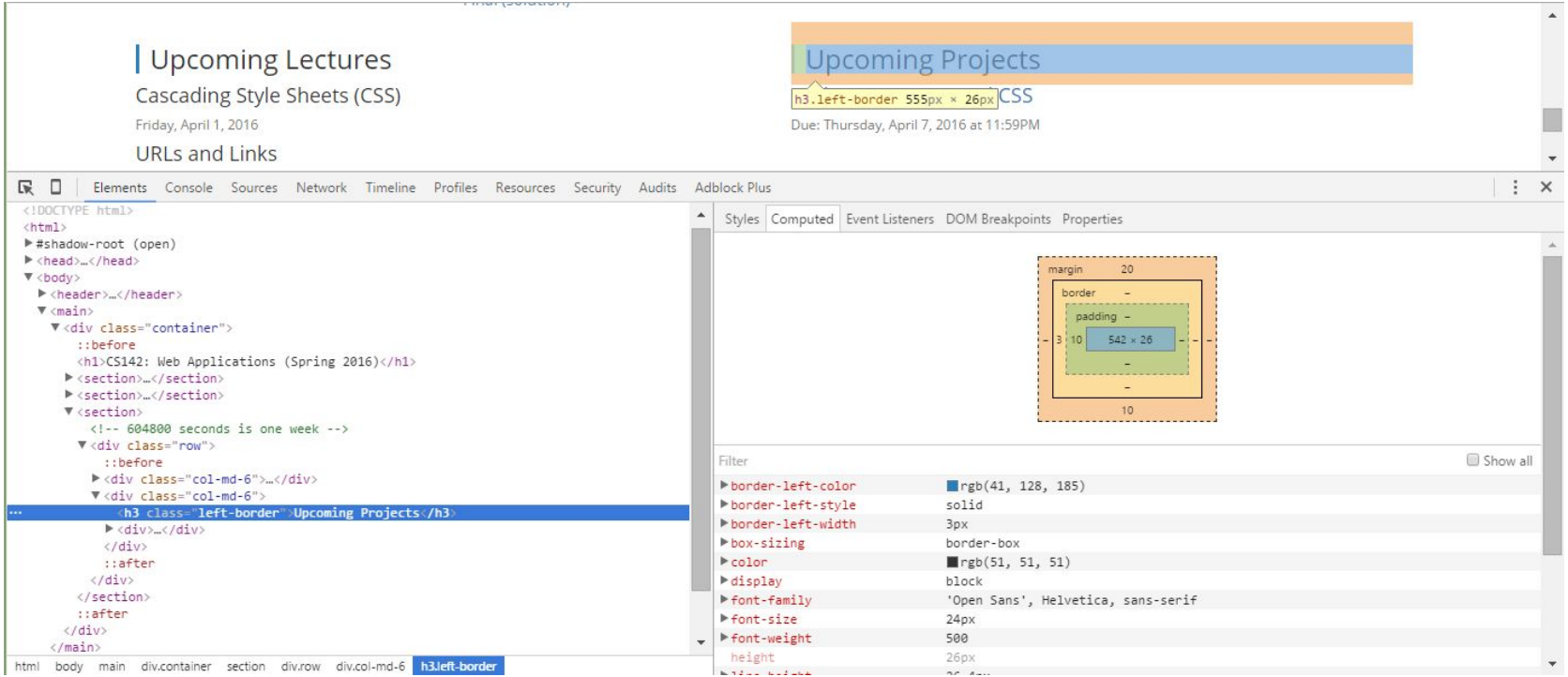
Useful Features: Validation

XHTML 1.0 validated at

https://validator.w3.org/

Useful Features: Debugging CSS (Chrome Inspector)

Chrome Inspector (right-click > Inspect)



Additional References

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

https://www.w3schools.com/cssref/

https://css-tricks.com/snippets/css/a-guide-to-flexbox/