

HTML & CSS: Week 4

Instructor: Devon Persing
TA: Becky Pezely

Review!

Abbreviations

- **Hex colors** that appear in triplets can be abbreviated
 - #ff0000 can be #f00
 - #666666 can be #666
- Some **CSS properties** that are related to each other can be grouped into a single property

Property abbreviations example

```
p {  
    font-style: italic;  
    font-weight: bold;  
    font-size: 1.4em;  
    line-height: 1.2em;  
    font-family: Georgia, Times, "Times New  
Roman", serif;  
    /* can be */  
    font: italic bold 1.4em/1.2em Georgia,  
Times, "Times New Roman", serif;  
}
```

class and id attributes

- **class** attributes

- can be applied to multiple HTML elements of any kind to assign them the same styles
- HTML elements can have more than one class
- are denoted with a period in CSS (**.example**)

- **id** attributes

- can be applied to a single element on a page to assign it styles
- are denoted with a hash in CSS (**#example**)

- elements can have an **id** and **class(es)**

class example

```
/* a class in CSS */
```

```
.highlight { background: #ff0; }
```

```
<!-- a class in HTML -->
```

```
<h1 class="highlight">My page title</h1>
```

```
<p class="highlight">A paragraph of text.  
</p>
```

id and class example

```
/* an id in CSS */
#title { font-size: 4em; }
/* a class in CSS */
.highlight { background: #ff0; }

<!-- an id and a class in HTML -->
<h1 id="title" class="highlight">My page title</h1>
<p class="highlight">A paragraph of text.</p>
```

id and multiple class example

```
/* an id and multiple classes in CSS */
```

```
#title { font-size: 4em; }
```

```
.highlight { background: #ff0; }
```

```
.offset { font-style: italic; margin-left: 20px; }
```

```
<!-- an id and multiple classes in HTML -->
```

```
<h1 id="title" class="highlight">My page title</h1>
```

```
<p class="highlight offset">A paragraph of text that has  
two classes.</p>
```


class and id descendent selectors

```
#main h2 { /* some styles */ }  
div .highlight { /* some styles */ }  
#main .highlight { /* some styles */ }
```

/* works just like */

```
li span { /* some styles */ }  
div p a { /* some styles */ }
```

combined class and id selectors

```
.highlight.offset { /* some styles */ }  
#title.highlight { /* some styles */ }  
div.main.featured { /* some styles */ }
```

- string **class/id** selectors together *without spaces* to style elements that have:
 - two or more of ***the same* class** attributes
 - a specific **class** ***and*** a specific **id**
 - an **id** ***and*** two or more of ***the same* class** attributes

combined class and id selectors

```
/* CSS ids and classes */  
.highlight.offset { /* some styles */ }  
#title.highlight { /* some styles */ }  
div.main.featured { /* some styles */ }
```

<!-- HTML ids and classes →

```
<div>  
  <h1 id="title" class="highlight"></h1>  
  <p class="highlight offset"></p>  
</div>
```

CSS backgrounds

```
background-color: #eee;
```

```
background-image: url("../img/kitten.jpg");
```

```
/* or */
```

```
background: #eee url("../img/kitten.jpg");
```

```
/* can be transparent! */
```

```
background: rgba(200, 54, 54, 0.5);
```



"a" for alpha
transparency



color opacity
from 0 (invisible)
to 1 (opaque)

background options

```
/* repeating */
```

```
background-repeat: repeat-x; /* horizontal */
```

```
background-repeat: repeat-y; /* vertical */
```

```
background-repeat: no-repeat; /* none! */
```

```
/* position along y and x axes */
```

```
background-position: top right;
```

```
background-position: center center;
```

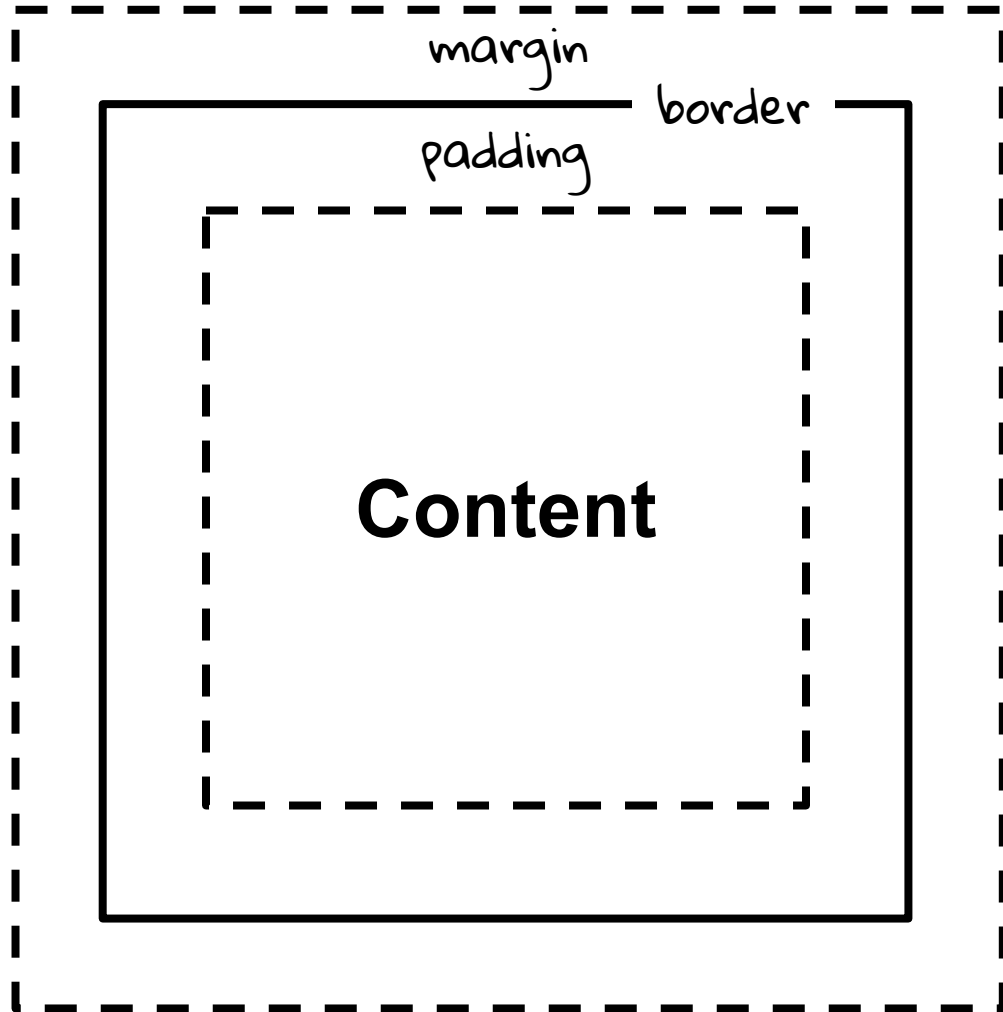
```
background-position: 20px 10px;
```

```
/* combined */
```

```
background: #eee url("../img/kitten.jpg")
```

```
no-repeat bottom right;
```

CSS box model



CSS box model components

- **Content** that defines the default `width` and `height` properties of an element's "box"
- **Border** that goes around the edge of the box
- **Padding** that adds space between the content and border
- **Margin** that adds space outside the border
- Edges are referenced clockwise (top, right, bottom, left)

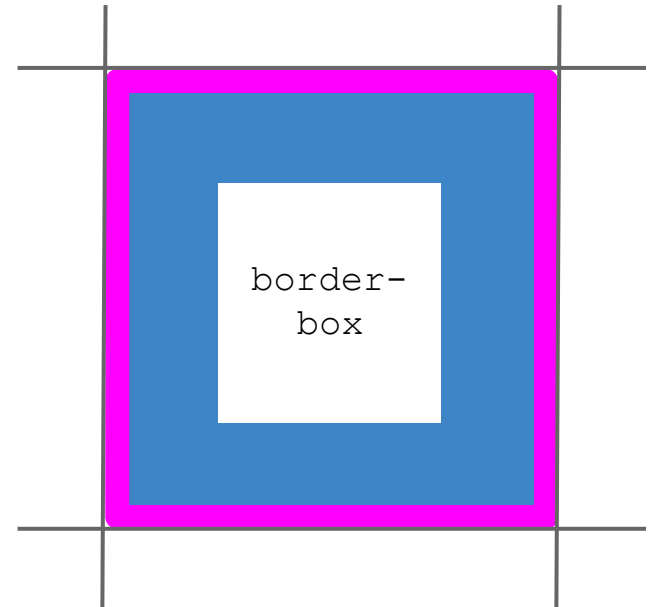
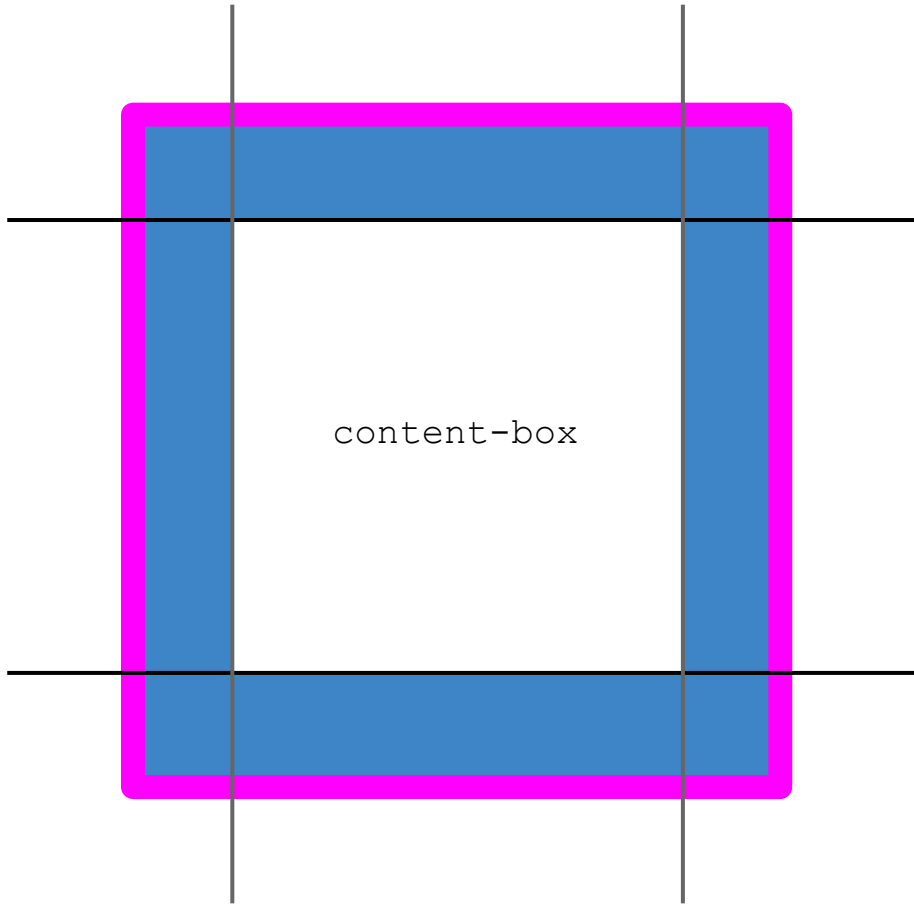
Example border box styles

```
width: 400px; /* reset content width */  
height: 200px; /* reset content height */  
  
/* abbr. border styles: width, style, color */  
border: 2px solid #ff0;  
  
/* margin style written out */  
margin-left: 20px;  
margin-bottom: 10px;  
  
/* abbr. padding styles: 10px top/bottom, 20px right/left  
*/  
padding: 10px 20px;
```


box-sizing to the rescue

- The **box-sizing** property affects how containers are rendered on the page, with `content-box` as the default value
- The `border-box` value renders border and padding **inside** the container instead of **outside**

content-box to border-box



Using box-sizing

```
div {  
    /* Firefox vendor prefix */  
    -moz-box-sizing: border-box;  
    box-sizing: border-box;  
}
```

Questions?

This week

- HTML data tables
- New HTML5 container elements
- Multi-column layouts
- CSS pseudo-classes

HTML data tables

What's a table for?

- Presenting data in a tabular format
- That's it
- That's all
- For example:
 - Listings of people, addresses, etc.
 - Financial data
 - Product feature comparisons

Planning a table

- What **content** needs to be displayed?
- What's the most logical way to **organize and display** that content?
- How should the content be **labeled**?
- How should different parts of the table be **styled**?

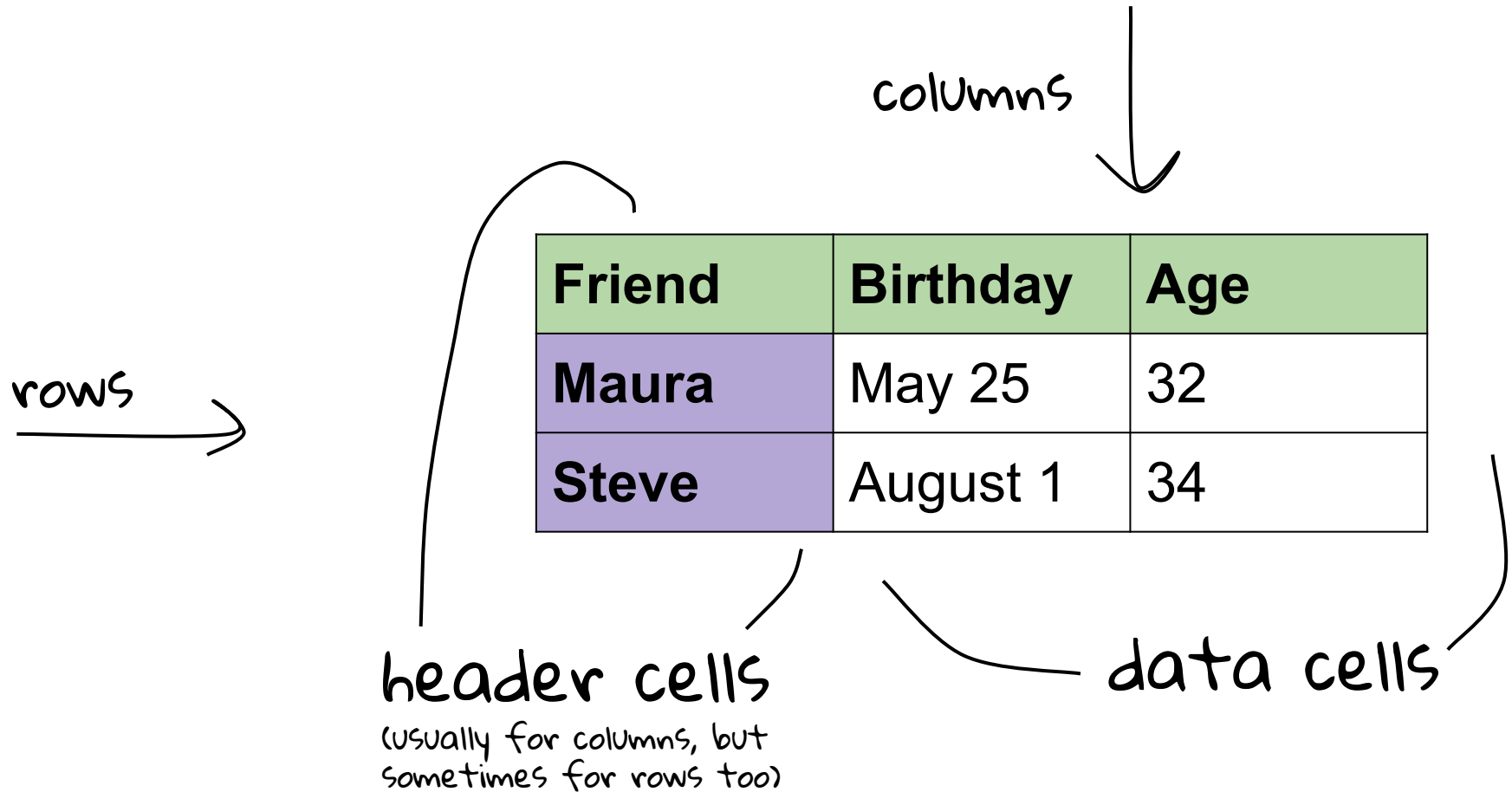
Basic table elements

- `<table>` wraps all table elements
- `<tr>` creates a row of table cells
- `<th>` creates a table header cell within a row
- `<td>` creates a regular table data cell within a row

Code for a basic table

```
<table>
  <tr>
    <th>Column 1 Header</th>
    <th>Column 2 Header</th>
    <th>Column 3 Header</th>
  </tr>
  <tr>
    <td>Column 1 Data Cell</td>
    <td>Column 2 Data Cell</td>
    <td>Column 3 Data Cell</td>
  </tr>
</table>
```

Anatomy of a table



<th> attributes

- For accessibility, it's good practice to use a scope attribute for table header cells:
 - **scope="col"** for table headers that describe a column
 - **scope="row"** for table headers that describe a row
- Creates an explicit connection for data cells that have multiple headers

A basic table with header rows

```
<table>
  <tr>
    <th scope="col">Column 1 Header</th>
    <th scope="col">Column 2 Header</th>
    <th scope="col">Column 3 Header</th>
  </tr>
  <tr>
    <th scope="row">Row 2 Header</th>
    <td>Row 2, Column 2 Data Cell</td>
    <td>Row 2, Column 3 Data Cell</td>
  </tr>
</table>
```

Styling table elements

- **background**, **border**, **margin**, and **padding** styles can all be applied
- **border-spacing** controls space between adjacent cells
- **border-collapse** removes `border-spacing`

```
border-spacing: 10px 5px;
```

```
border-collapse: collapse;
```

Spanning multiple rows/columns

- The **colspan** attribute allows a cell to span multiple columns (**colspan="2"** would span 2 columns)
- The **rowspan** attribute allows a cell to span multiple rows (**rowspan="2"** would span 2 rows)
- In general, cells that span only one row or column are recommended

Let's style a table.

HTML5 containers

Pre-HTML5 structure

- Previously, HTML had semantic values for content (like `<h1>` and `<p>`) but not for groups of content
- We faked it by using `<div>` elements with semantic-sounding `id` attributes like:

```
<div id="header"></div>
```

```
<div id="main"></div>
```

```
<div id="sidebar"></div>
```

```
<div id="footer"></div>
```

HTML5 containers elements

- `<header>`: header of a container
- `<nav>`: navigation links
- `<main>`: primary content
- `<section>`: a group of related content
- `<article>`: what is says on the tin
- `<aside>`: supportive, non-primary content
- `<footer>`: footer of a container

Support for older browsers

- <IE9 and older mobile browsers have no native support for shiny new HTML5 tags
- These browsers can be ~~tricked~~ *gently coaxed* into displaying and styling new HTML5 elements via Javascript
- The most popular method is the **HTML5 shim**: <https://code.google.com/p/html5shim/>

Installing the HTML5 shim

1. Visit <https://code.google.com/p/html5shim/> and download the zip file
2. Unzip it, and add it to your site files (hint: maybe in a `js` folder?)
3. Paste/type this into the `<head>` element of all your pages:

```
<!--[if lt IE 9]>  
<script src="your-file-location"></script>  
<![end if]-->
```

While we're in our files

Let's add a new selector to our reset styles...

```
html, body, div, span, applet, object, iframe, h1, h2, h3, h4, h5, h6, p,
blockquote, pre, a, abbr, acronym, address, big, cite, code, del, dfn, em,
img, ins, kbd, q, s, samp, small, strike, strong, sub, sup, tt, var, b, u,
i, center, dl, dt, dd, ol, ul, li, fieldset, form, label, legend, table,
caption, tbody, tfoot, thead, tr, th, td, article, aside, canvas, details,
embed, figure, figcaption, footer, header, hgroup, menu, nav, output,
ruby, section, summary, time, mark, audio, video, main {
    margin: 0; padding: 0; border: 0; font-size: 100%; font: inherit;
vertical-align: baseline;
}
```

```
/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure, footer, header, hgroup, menu,
nav, section, main {
    display: block;
}
```

<header>

- Wraps introductory content and/or navigation elements
- Can appear in multiple places on the page
- Use it for things like:
 - The global header of a site
 - The header of an article
 - The header of a long section within an article

<nav>

- Wraps major navigation elements
- Can appear in multiple places on the page
- Use it for things like:
 - Global navigation links for a site
 - Pagination links
 - Anything used to get around within a site

<main>

- Wraps the main content of a page
- Is used **only once per page**, unlike other elements
- Use it for things like:
 - A group of blog posts
 - An article with its own header
 - Whatever makes up the main focus of the page
- Came out after our reset styles were made!

<section>

- Wraps thematically related content, often with its own heading
- Can appear in multiple places on the page
- Use it for things like:
 - A group of related blog posts
 - A section within an article
 - A sidebar widget with its own header

`<article>`

- Wraps standalone texts
- A page might have one, several, or none
- Use it for things like:
 - Individual blog posts
 - Individual news articles
 - Individual comments on other articles

<aside>

- Wraps non-primary or tangential content
- A page might have one, several, or none
- Use it for things like:
 - A sidebar of related links
 - A pullquote from an article
 - A supporting fact for an article ("Did You Know?")
 - Things that can stand alone but aren't the main content of a page

<footer>

- Wraps any closing information in a container
- A page might have one, several, or none
- Use it for things like:
 - The global footer of a site
 - The footer of an article

Notes on using HTML5 containers

- HTML5 is an experiment in process and documentation!
- Elements will come and go
- When in doubt, use an online resource like <http://html5doctor.com/> (these will be more up to date than books)
- If you're not sure, use a `<div>`!

**Let's swap out some `<div>`
elements for HTML5 containers.**

Creating CSS layouts

A brief history of web layouts

- Before CSS, we used `<table>` elements to make layouts :(
- With CSS we can use a variety of declarations to arrange elements on the screen by adjusting the flow of the page
 - **Pros:** Any content can be displayed anywhere!
 - **Cons:** Any content can be displayed anywhere!

Centering our page in the browser

```
.wrapper { width: 960px; margin: 0px auto; }
```

- Most websites sit in the middle!
- To do this, give your `<body>` or another container that wraps the whole page:
 - a `width`
 - a left and right `margin` value of `auto`

Resizing images to fit in containers

```
img { width: 100%; height: auto; }
```

- Often images may come in different sizes, but need to be standardized to fit into a layout
- Use the **height** and **width** properties to mix and match % and auto values to scale images based on their default height or width

3 main ways to adjust the page flow

Historically, we've used three main ways to create web layouts. I like to call them:

- 1. Floating Down the River**
- 2. Behavior Modification**
- 3. Dictator for Life**

We'll try them out in our **Layout Laboratory**.

Method 1: Floating Down the River

- The `float` property lets us arrange elements like islands in the flow of the page:
 - To the left of the flow (`left`)
 - To the right of the flow (`right`)
- The `clear` property resets content that comes after floated content, like a bridge across the river (or a dam?)
 - Use `clear: both;` to clear `left` and `right` floats

Let's make a floated layout.

A few more ways to use float

- Position illustrations or photos (``) in the flow of a large block of text
- Position pullquotes or fun facts (`<aside>`) in the sides/margins of an article
- Position any non-primary content that lives inside primary content (social sharing buttons, bylines, dates, etc.)

Method 2: Behavior Modification

- **Inline** elements like `` and `<a>` line up with their neighbors
- **Block** elements like `<div>`, `<p>`, ``, etc. create line breaks
- **Inline-block** elements like `` line up but also maintain the box like block elements
- We can use the CSS `display` property to make elements behave like one or the other

Modify `display` behavior with CSS

- Make any element **behave like a block element**
- Make any element **behave like an inline element**
- Make any element **behave like an inline-block element**

```
a { display: block; }
```

```
div { display: inline; }
```

```
article { display: inline-block; }
```

Let's make a display-inline layout.

The `inline-block` layout trick

- `inline-block` was designed to display text elements like links, so it includes a tiny bit of space at the right for readability
- To fix this, give your inline-block elements a **negative right margin**:

```
div {  
    display: inline-block;  
    margin-right: -4px;  
}
```

A few more ways to use `display`

- Make a normally-inline element **stand out**
- Style the same HTML differently in **different contexts**
- Manipulate how/when content is displayed using **Javascript**

Method 3: Dictator for Life

- The **position** property lets us arrange elements:
 - In relation to the flow (**relative**)
 - In a very specific place outside of the flow or within another **relative** element (**absolute**)
 - In relation to the browser window (**fixed**)
- How **position** is applied depends on to where the element is in the flow by default

Tweaking the position

- We can further dictate where elements go down to the pixel with a few additional elements
- **left**, **right**, **top** and **bottom** add or subtract pixels between positioned elements and their containers

```
div { position: absolute; right: -10px; top: 30px; }
```

Let's make a positioned layout.

Some issues with this method

- Layouts that rely on absolute and fixed positioning require micromanagement:
 - The size of elements must be very precise and may need to be adjusted over time
 - It's difficult to mix other layout methods with this method
- Absolute and fixed positioning scales poorly on smaller devices like tablets and phones

What's the verdict?

CSS pseudo-classes

Pseudo-classes are conditional

- **Pseudo-classes** are added to a selector to add conditional styles to an element
- Most often used to style **states** of **<a>** elements and form elements

```
a:link { /* the default state of a link */ }  
a:visited { /* a link that's been clicked */ }  
a:hover { /* a link that has a mouse hover */ }  
a:focus { /* a link that has keyboard focus */ }  
a:active { /* a link that is currently being clicked */ }
```

:hover versus :focus

- **:hover** is for a link or other interactive element that has a **mouse hover**
- **:focus** is for a link or other interactive element that has **keyboard focus**
- Browsers have their own default **:focus** styles for **accessibility**

```
a:hover, a:focus {
```

```
    /* it's good practice to style :hover and :focus  
    together so they're both accounted for */
```

```
}
```

:hover for non-interactive elements

- **:hover** can be used to style hover states for some non-interactive elements to create a more dynamic experience

```
tr { /* a table row with one background... */  
    background: #99ff66;  
}
```

```
tr:hover { /* ...could have another on hover */  
    background: #ff6600;  
}
```

Some other nifty pseudo-elements

- **:first-letter** styles the first letter of a block of text
- **:first-child** and **:last-child** style the first and last children of a parent
- **:nth-child()** can be used to style even or odd children, or do some math to style every 5th, etc.
- **:before** and **:after** can be used to add style-only content to elements

CSS selectors are evolving

- **Pseudo-classes, pseudo-elements, combinators, and attribute selectors** create extremely targeted ways to style content
- To learn more of these techniques, and see which ones work in which browsers: <http://www.quirksmode.org/css/selectors/>

For next week

- Style your links with **pseudo-classes**
- Implement **HTML5 containers**
- Make and style a **data table**, if appropriate
- Style your **layouts!**
- Let me know if you'd like to **demo**
- ***HTML5 for Web Designers***: ch. 5
- ***HTML and CSS***: ch. 6, 14-15, 17

Next week

- Lingering and new questions
- Project demos?
- Web fonts and other 3rd party services
- Embedding iFrames and media
- A bit about more CSS3 (and CSS4)
- A bit about Javascript and JQuery
- A bit about the present and future of web dev

Questions?

- Visit <http://dpersing.github.io/svc>
 - Class slides
 - Code examples from class
 - Additional general and class-specific resources
- Email me at dep@dpersing.com