HTML & CSS Level 1: Week 2

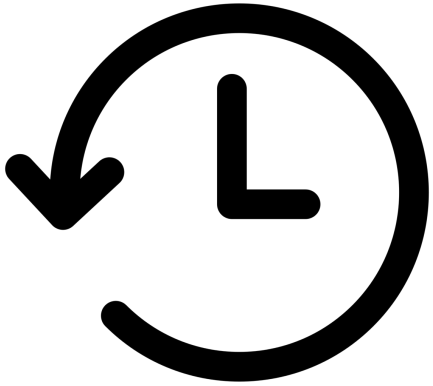
June 18 - July 23, 2014

Instructor: Devon Persing

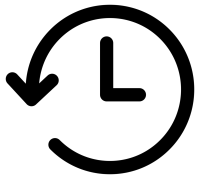
Tonight

- Week 1 review and questions
- Prepping images for the web
- Some more HTML
- Introduction to CSS
- Validating your code
- Making Buster a resume

Review!

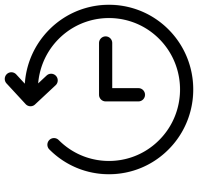


History designed by Ema Dimitrova from the Noun Project



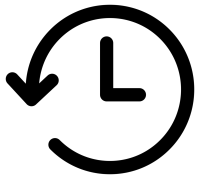
Review: Delivering content

- **HTML** structures content
- **CSS** creates style and layout
- **JavaScript** adds interactivity



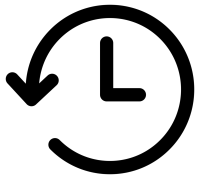
Review: Tidy file structures

- **HTML** files go in the main directory
- **CSS** and media go in subdirectories
- Your homepage is **index.html**
- Paths can be **absolute** or **relative**
- Use relative paths when possible (`img/kittens.png`, **not** `http://example.com/img/kittens/png`)



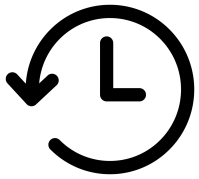
Review: Tidy file names

- **No spaces** in filenames
- **Capitalization** matters
- Use **letters**, **numbers**, **hyphens** and **underscores** only
- Filenames **start with a letter**



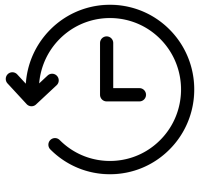
Review: Good coding practices

- **Standardize** your file structures and filenames
- Use `<!-- HTML comments -->` to leave yourself and others notes
- **Indent your code** so its readable



Review: HTML documents

- **`<!DOCTYPE html>`** tells the browser it's serving an HTML file
- **`<html>`** tags wrap the whole document
- **`<head>`** tags wrap all of the metadata
- **`<body>`** tags wrap all of the content
- Most HTML elements have **opening and closing tags**, and some have **attributes**



Review: HTML for content

- **Headings** create an outline: `<h1>...<h6>`
- **Paragraphs** and **lists** structure text: `<p>...<ul>...<ol>`
- **Images** embed jpegs, gifs, etc.:

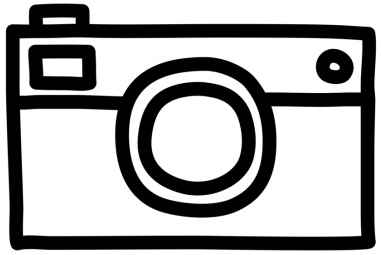
  `<img src="" alt="">`
- **Links** connect pages: `<a href="">`

Questions?

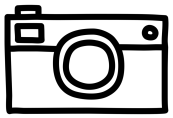
This week's code samples

Let's check'em out

Images for the web

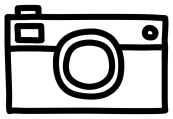


Camera designed by Maria Maldonado from the Noun Project



Why "web-ready" images?

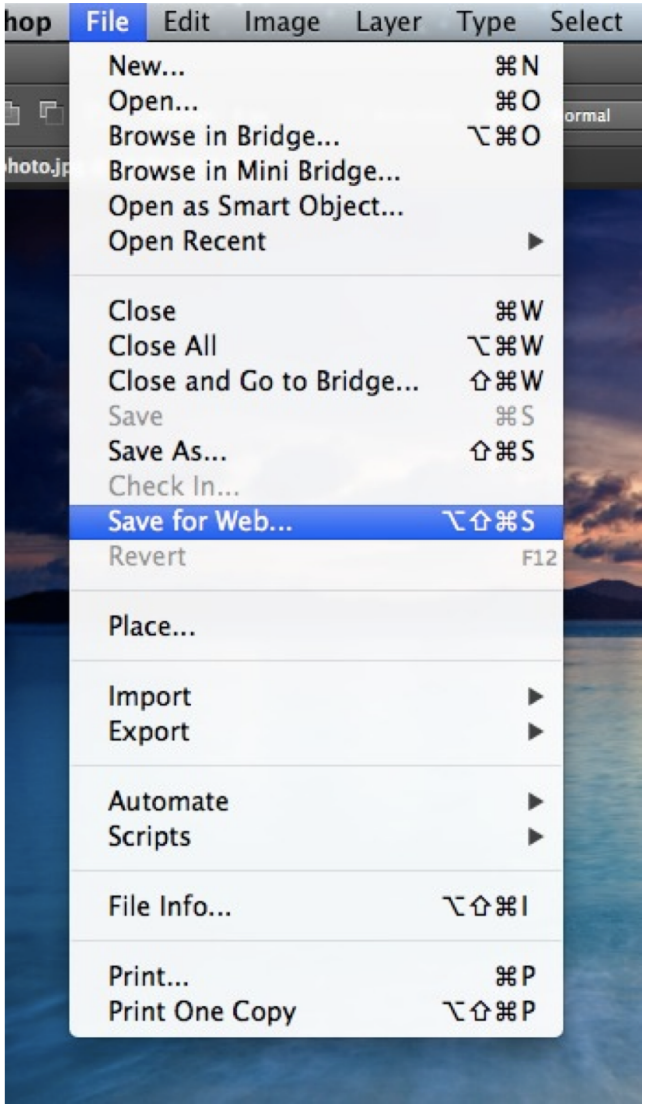
- Strips out **layers** and other **metadata** from your image
- Minimizes file sizes to **help load time** in the browser
- Optimized images for **RGB display** at the correct **resolution** for browsers



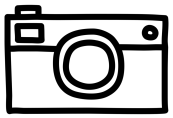
Web image types

- **JPG** or **JPEG** is traditional for photos
- **GIF** is traditional for animation, illustrations, and transparency
- **PNG** was designed specifically for the web for any image type (including transparent images and animations)

"Save for Web" in Adobe CS

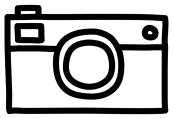


- Adobe products have a **"Save for Web…"** or **"Save for Web and Devices…"** option
- Similar products will have a similar option



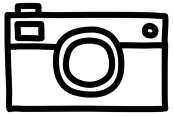
Saving for web

1. Pick "Save for Web..." from the File menu
2. Pick a suitable file format (JPEG, GIF, etc.)
3. Adjust the image dimensions to be the max size you want to display
4. Save!
5. Add your image to your page with an **`<img src="" alt="">`** element



Exceptions!

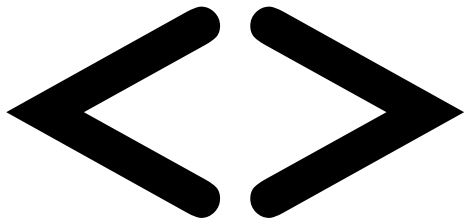
- If the **photo is the content**:
  - Usually we'll use a smaller image as a link, and load a larger resolution image on click
- Newer devices with **retina screens** may support larger resolution images
  - We can use Javascript or server-side code to display different images on different devices



YouTube demos

- Saving for web from Photoshop
- Saving for web from Illustrator
- Look for these on the class site!

Block and inline HTML



<> But first: `<br>` elements

- You found them, didn't you?
- That's okay.
- Pre-CSS, **`<br>`** tags were how we made space between elements
- *Week 3*: using CSS to manage **space around elements**
- *Week 4*: using CSS to **manage layouts**

<> Block and inline elements

Block elements we know:

- Headings (`h1`, `h2`, etc.)
- Paragraphs (`p`)
- Lists (`ul`, `ol`)
- List items (`li`)

Inline elements we know:

- Links (`a`)

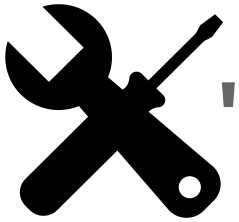
<> Block and inline elements con't.

Block elements:

- Create **linebreaks**
- Take up the **full width** of their container

Inline elements:

- **Don't** create linebreaks
- Take up **only as much space as they need**



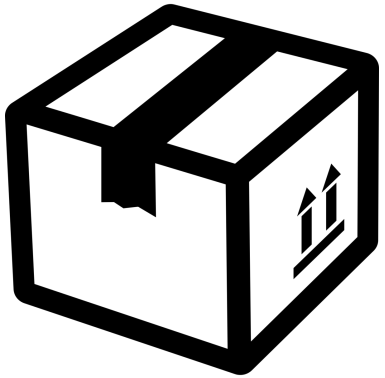
"Seeing" code

- Modern browsers come with a Web Inspector tool that let's you:
  - See the code output and resources that the browser is processing
  - See how much "space" things take up in the browser window
- Right click in the browser and choose "Inspect Element"

Toolset designed by Calvin Goodman from the Noun Project

<> `<div>` elements

- **`div`** elements are **generic block** elements
- Used to **create sections or groupings** in HTML pages for layout and style
- Function **like a box** to put content (or other divs) inside
- Have heights and widths



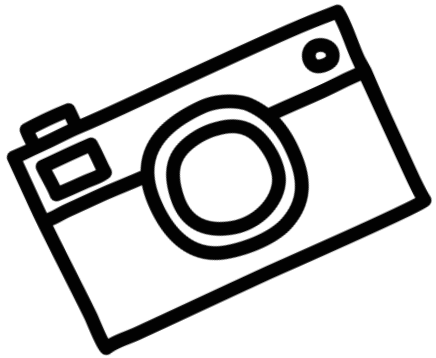
Box designed by Mourad Mokrane from the Noun Project

<> `<span>` elements

- **`span`** elements are **generic inline** elements
- Can **nest inside** other block or inline elements
- Used to **style other inline content** or **content inside block elements**
- **Flow** with the content around them

<> The rare inline-block element

- **Inline-block** elements behave a bit like both block and inline elements:
  - Take up height and width like block elements
  - Flow with content around them
- So far we know `img` elements



<> More inline elements

- **`em`** elements are used to show the equivalent of *spoken emphasis*
- **`strong`** elements are used to show **importance in context**

```
<p>"Oh, great. Someone ate <em>my only clean socks</em>."
</p>

<p>"Was it <strong>the cat</strong>?"</p>

<p>"No, it was <strong>the dog</strong>.</p>
```

<> What about `<i>` and `<b>`?

- Older versions of HTML used `i` *(italic)* and `b` **(bold)** tags
- They're still legit but have sort of squishy explanations
- Theories about why they might have been replaced by `em` and `strong`?

Intro to CSS

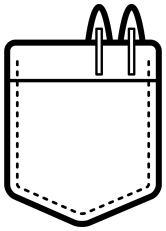
{ }

{ } Getting ready for our CSS

1. Open **`index.html`** in your text editor.
2. After the **`<title>...</title>`** element, make a new line.
3. Type: **`<style></style>`**
4. Save your **`index.html`** file.
5. We'll put all of our style rules inside **`<style></style>`.**

{ } Cascading Stylesheets

- **CSS** brings style, formatting, and layout to HTML
- Provides consistent and scalable ways to style **single elements**, **single pages**, or **entire sites**
- **Separates look and feel from content** so that sites can be restyled more easily



CSS overload alert

- There are **a lot** of CSS properties
- We will **not get anywhere close** to covering all of them
- **Practice the basics** before getting fancy
- We'll cover **common CSS** for text styles, colors, positioning, layout, and some fun extras

{ } Anatomy of a CSS rule

```
selector { property: value; }
```

- **Selector** is the thing you want to style
- **Property** is what aspect you want to style
- **Value** is how you want to style it
- **Property** + **value** = **declaration**
- **Declarations** end in semicolons (`;`)

{ } Example CSS rule

```
h1 { font-size: 2em; }
```

- **Selector** is `h1` (yes, an `<h1>`!)
- **Property** is `font-size` **and value** is `2em`
- **Note:** Property and value options come from a list of specific options that browser support

{ } CSS comments

- Just like HTML, CSS can have **comments**

```
<style>
   /* I am a CSS comment! */

   h1 { /* I am also a CSS comment */
      color: #ff0000;
   }
</style>
```

{ } Common font properties

- **`font-size`**: a number followed by a measurement of how tall the element's text is, usually in ems (**`em`**) or pixels (**`px`**)
- **`font-family`**: the name of a typeface, or typefaces
- **`font-style`**: **`italic`**
- **`font-weight`**: **`bold`** | values of bold!
- **`line-height`**: a number followed by a measurement of how tall the element's line of is, usually in ems (**`em`**) or pixels (**`px`**)

{ } Common text properties

- **`text-align: left`** | **`right`** | **`center`** | **`justify`**
- **`text-transform: capitalize`** | **`uppercase`** | **`lowercase`** | some others
- **`text-decoration: underline`** | **`overline`** | **`line-through`** | some others
- **Note:** A lot of properties will take a value of **`none`**

{ } Colors

- To set **text colors**, the property is **`color`**
- To set **background colors**, the property is **`background-color`**
- The value can be done a few ways:
  - Hex: **`#ff0000`**
  - RGB: **`rgb(255,0,0)`**
  - Also possible but not preferred: **`red`**
- Need a random hex number?
- Want to convert hex to RGB?

{ } More about rules

- **Type selectors** point to a specific *type* of HTML element (a paragraph: `p`)
- **Descendant selectors** point to an element that is the *child* of another element (a link inside a paragraph: `p a`)
- Descendent selectors are **more specific**, since they target more particular stuff

{ } Multiple selector/property ex.

```
ul, ol {
  font-size: 1.2em;
  font-weight: bold;
  color: #39887c;
}
```

{ } Using styles in multiple places

- **Inline styles** are applied to only a single element (and are uncommon)
- **Internal styles** are added in the head of a page and style only that page (what we've done so far)
- **External styles** are called into multiple pages, to style a whole site

{ } Making an external stylesheet

1. Create a **new file** in your text editor.
2. **Copy and paste** our styles from inside the **`<style>...</style>`** element into our new file.
3. Save our new file as **`styles.css`**, and put it in your **`css`** folder.
4. Save your **`index.html`** file.
5. Then...

{ } Linking to external CSS

```
<link href="css/styles.css"
rel="stylesheet">
```

- Tells an HTML page to go find and load the CSS file
- Goes inside the **<head>** element
- Needs to go in every page that should load the stylesheet

{ } The "cascading" part

Q: If properties can live in multiple places, and some can be more specific than others, how does the browser know what order they should go in and which ones should be ignored?

A: Inheritance, rule order, specificity, location

{ } Children are specific

- Children elements usually **inherit** styles from their parents but can **override** parents with their own styles

```
body { color: #60dd47; } /* parent */

p { color: #2f4251; } /* child */
```

{ } Selectors can be more specific

- If one style is **more specific** than another, it takes precedence

```
p { color: #2f4251; } /* paragraphs in general */
a { color: #e7c0c8; } /* links in general */
p a { color: #c4fe46; } /* a in p */
div p a { color: #a5dd5e; } /* a in p in div */
```

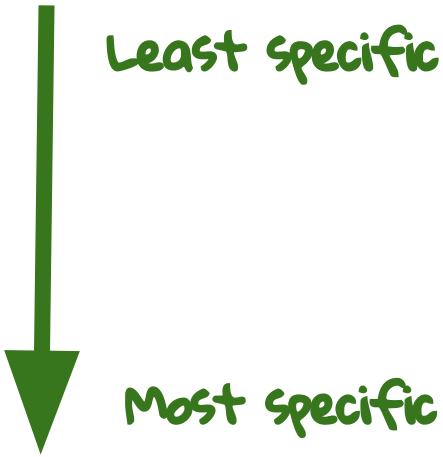
{ } Top to bottom

- If the same property is styled multiple times for the same selector, **the last one sticks**

```
p { color: #2f4251; }
ul { color: #5b416d; } /* some other stuff
*/
p { color #daa645; } /* overrides first! */
```

{ } Stylesheet "location"

- Styles that are "closer" to the elements they style take precedence
  - Browser defaults
  - External styles (in a `.css` file)
  - Internal styles (in the `<head>`)
  - Inline styles (in an element)



{ } "How will I remember all this?"

- **You won't.** I don't!
- Use online references like these:
  - Mozilla's Getting Started with CSS guide
  - Mozilla's CSS Reference index
- Use your Web Inspector to see load order for rules, and what's ignored

Validating your code

Award designed by Luc Poupard from the Noun Project

Validating HTML and CSS

- **Validation** is an easy way to make sure your code is properly formatted
- HTML: html5.validator.nu
- CSS: jigsaw.w3.org/css-validator

"Homework" for next week

1. Start grouping content in your pages with **`<div>`** elements
2. Style your pages with an **external stylesheet**
3. **Validate** your HTML and CSS
4. *Optional*: Create a header image/logo for your site
5. *Optional reading: HTML and CSS* ch. 10-12

Next time

- Questions and review from Week 2
- New HTML5 elements!
- More CSS and ways to write it
- The CSS box model (aka intro to layouts)
- Overriding browser defaults for styles

NO CLASS NEXT WEEK!

Fireworks by Michael B. on Flickr

Questions? Comments?

- Visit dpersing.github.io/svc for:
  - Class slides
  - Code samples
  - Resources
- Email me: dep@dpersing.com
- Tweet at me: @devonpersing