IT11

INTRODUCING CSS

OVERVIEW

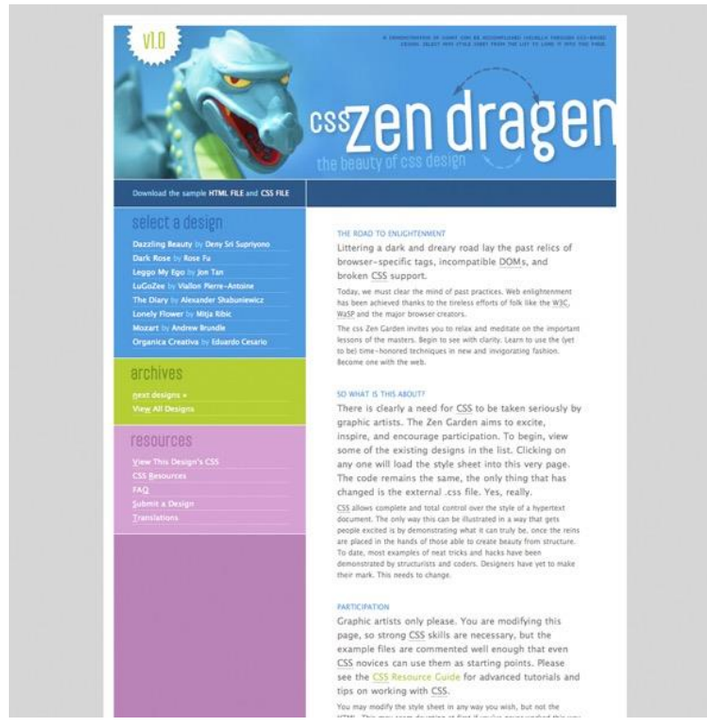
- The benefits of CSS
- Understanding document structure
- Writing style rules
- Attaching styles to the HTML document
- Inheritance
- The cascade
- The box model
- CSS units of measurement

The Benefits of CSS

- Precise type and **layout control**
- **Less work:** Change look of the whole site with one edit
- **Accessibility:** Markup stays semantic
- **Flexibility:** The same HTML markup can be made to appear in dramatically different ways

Style Separate from Structure

These pages have the exact same HTML source but different style sheets:



(csszengarden.com)

How Style Sheets Work

1. Start with a marked up document (like HTML, but could be another XML markup language).

2. Write styles for how you want elements to look using CSS syntax.

3. Attach the styles to the document (there are a number of ways).

4. The browser uses your instructions when rendering the elements.

Style Rules

Each rule *selects* an element and *declares* how it should display.

```
h1 { color: green; }
```

This rule selects all `h1` elements and declares that they should be green.

```
strong { color: red; font-style: italic; }
```

This rule selects all `strong` inline elements and declares that they should be red and in an italic font.

Style Rule Structure

- A style rule is made up of a **selector** a **declaration**.
- The declaration is one or more **property / value** pairs.

declaration

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

declaration block

```
selector {
  property1: value1;
  property2: value2;
  property3: value3;
  }
```

Selectors

There are many types of selectors. Here are just two examples:

```
p {property: value;}
```

Element type selector: Selects all elements of this type (**p**) in the document.

```
#intro {property: value}
```

ID selector (indicated by the # symbol) selects by ID value. In the example, an element with an id of “intro” would be selected.

Declarations

The **declaration** is made up of a **property/value pair** contained in curly brackets `{ }`:

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

Example

```
h2 { color: red;
     font-size: 2em;
     margin-left: 30px;
     opacity: .5;
   }
```

Declarations (cont'd)

- End each declaration with a semicolon to keep it separate from the next declaration.
- White space is ignored, so you can stack declarations to make them easier to read.
- **Properties** are defined in the CSS specifications.
- **Values** are dependent on the type of property:
    - Measurements
    - Keywords
    - Color values
    - More

CSS Comments

```
/* comment goes here */
```

- Content between `/*` and `*/` will be ignored by the browser.
- Useful for leaving notes or section labels in the style sheet.
- Can be used within rules to temporarily hide style declarations in the design process.

Adding Styles to the Document

There are three ways to attach a style sheet to a document:

External style sheets

A separate, text-only *.css* file associated with the document with the **`link`** element or **`@import`** rule

Embedded style sheets

Styles are listed in the **`head`** of the HTML document in the **`style`** element.

Inline styles

Properties and values are added to an individual element with the **`style`** attribute.

External Style Sheets

The style rules are saved in a separate text-only *.css* file and attached via **`link`** or **`@import`**.

Via **`link`** element in HTML:

```
<head>
  <title>Titles are require</title>
  <link rel="stylesheet" href="/path/example.css">
</head>
```

Via **`@import`** rule in a style sheet:

```
<head>
  <title>Titles are required</title>
  <style>
    @import url("/path/example.css");
    p {font-face: Verdana;}
  </style>
</head>
```

Embedded Style Sheets

Embedded style sheets are placed in the `head` of the document via the `style` element:

```
<head>
  <title>Titles are required</title>
  <style>
    /* style rules go here */
  </style>
</head>
```

Inline Styles

Apply a style declaration to a single element with the **`style`** *attribute*:

```
<p style="font-size: large;">Paragraph text...</p>
```

To add multiple properties, separate them with semicolons:

```
<h3 style="color: red; margin-top: 30px;">Intro</h3>
```

Document Structure

Documents have an implicit structure.

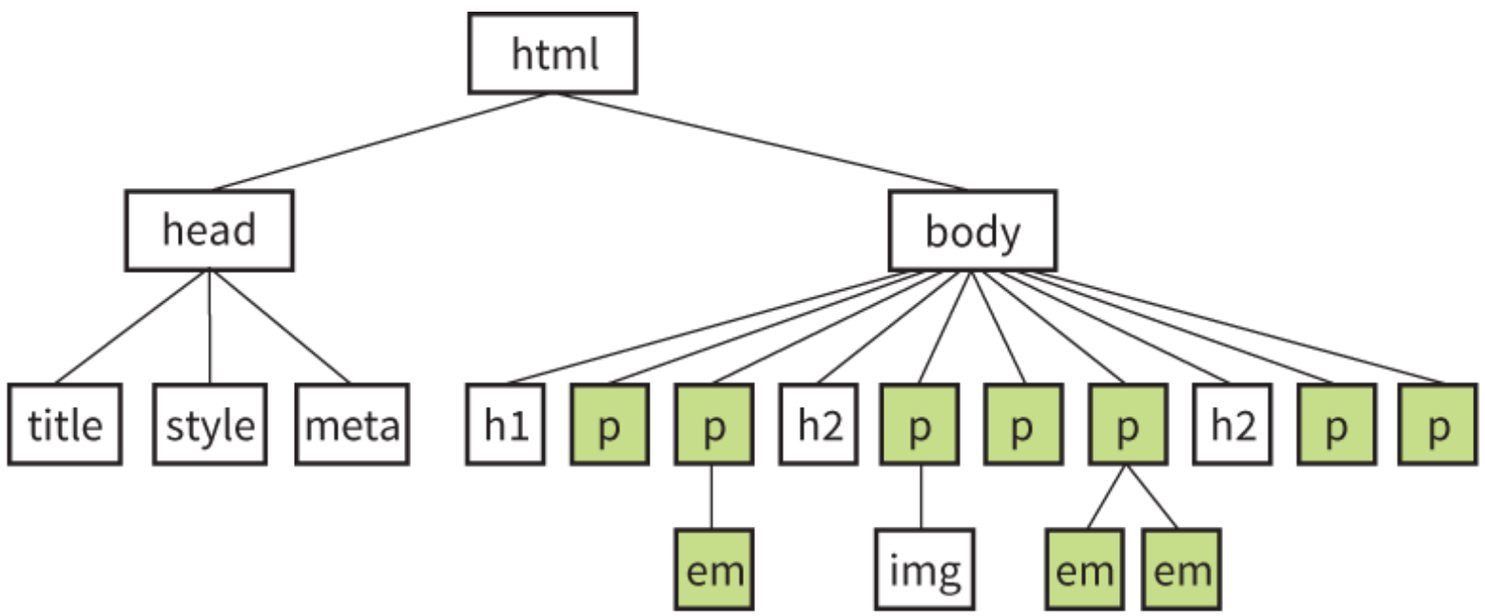
We give certain relationships names, as if they're a family:

- All the elements contained in a given element are its **descendents**.
- An element that is directly contained within another element is the **child** of that element.
- The containing element is the **parent** of the contained element.
- Two elements with the same parent are **siblings**.

Inheritance

- Many properties applied to elements are passed down to the elements they contain. This is called **inheritance**.

- For example, applying a sans-serif font to a `p` element causes the `em` element it contains to be sans-serif as well:




```
p {font-size: large; font-family: sans-serif;}
```

Inheritance (cont'd)

- **Some properties inherit; others do not.**
  Properties related to text usually inherit; properties related to layout generally don't.

- Styles explicitly applied to specific elements override inherited styles.

- You'll learn to use inheritance strategically to keep your style rules simple.

The Cascade

- The **cascade** refers to the system for resolving conflicts when several styles apply to the same element.
- Style information is passed down (it “cascades” down) until overwritten by a style rule with more **weight**.
- Weight is considered based on:
    - **Priority** of style rule source
    - **Specificity** of the selector
    - **Rule order**

The Cascade: Priority

Style rules from sources higher in this list override rules from sources listed below them.

- Any style marked as **`!important`** by the user (to accommodate potential accessibility settings)
- Any style marked **`!important`** by the author (of the web page)
- **Author styles** (style sheets created in web site production)
- **User styles** (added by the reader)
- **User agent styles** (browser defaults)

The Cascade: Specificity

- When two rules in a single style sheet conflict, the **type of selector is used to determine which rule has more weight**.

- For example, ID selectors are more specific than general element selectors.

NOTE: Specificity will be discussed once we have covered more selector types.

The Cascade: Rule Order

- When two rules have equal weight, rule order is used. **Whichever rule appears last "wins."**

```
<style>
  p {color: red;}
  p {color: blue;}
  p {color: green;}
</style>
```

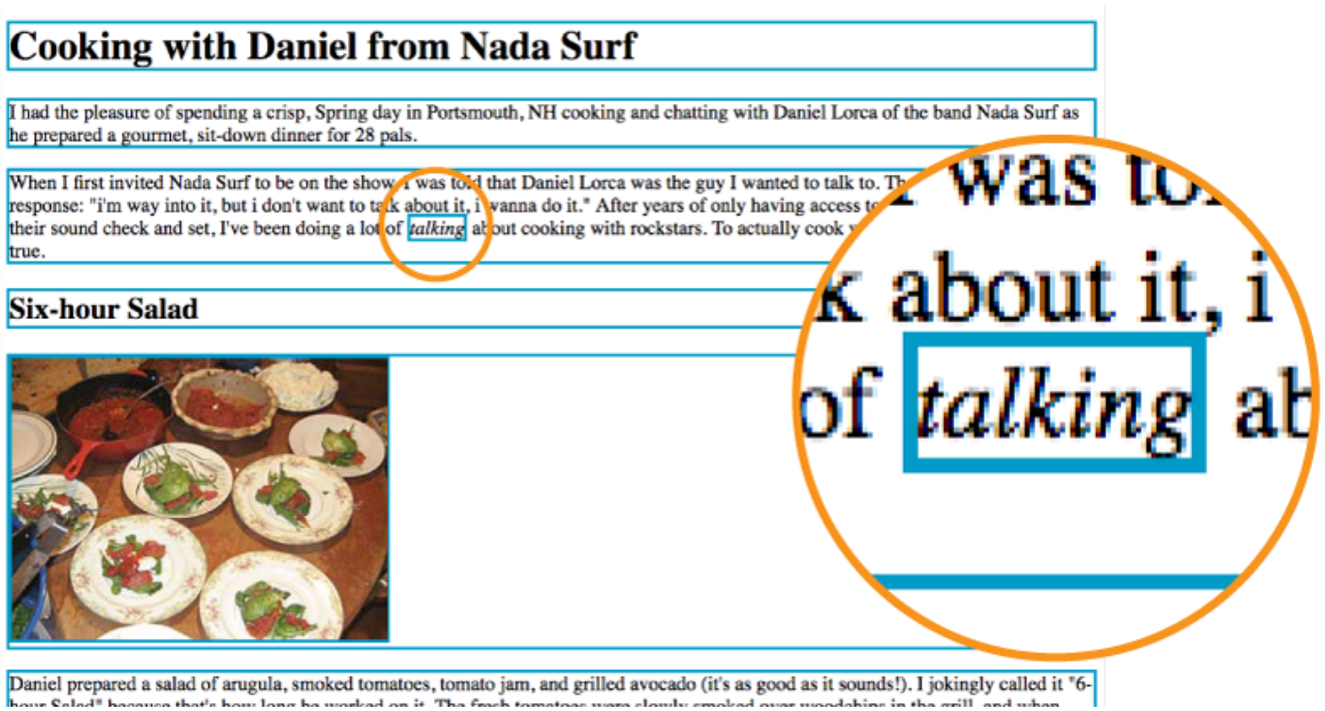
  In this example, paragraphs would be green.

- Styles may come in from external style sheets, embedded style rules, and inline styles. The style rule that gets parsed last (the one closest to the content) will apply.

The Box Model

Browsers see every element on the page as being contained in a little rectangular box. **Block elements and inline elements participate in the box model.**

In this example, a blue border is added to all elements.



The Box Model (cont'd)

- The **box model** is the foundation of CSS page layout.
- Apply properties such as **borders**, **margins**, **padding**, and **backgrounds** to element boxes.
- Position, move, grow, and shrink boxes to create fixed or flexible page layouts.

CSS Units of Measurement

CSS provides a variety ways to specify measurements:

Absolute units
Have predefined meanings or real-world equivalents

Relative units
Based on the size of something else, such as the default text size or the size of the parent element

Percentages
Calculated relative to another value, such as the size of the parent element

Absolute Units

With the exception of pixels, absolute units are not appropriate for web design:

`px`	pixel
`in`	inches
`mm`	millimeters
`cm`	centimeters
`q`	1/4 millimeter
`pt`	points (1/72 inch)
`pc`	pica (1 pica = 12 points = 1/6 inch)

Relative Units

Relative units are based on the size of something else:

`em`	a unit equal to the current font size
`ex`	x-height, equal to the height of a lowercase *x*
`rem`	root em, equal to the font size of the `html` element
`ch`	zero width, equal to the width of a zero (0)
`vw`	viewport width unit (equal to 1/100 of viewport width)
`vh`	viewport height unit (1/100 of viewport height)
`vmin`	viewport minimum unit (value of `vh` or `vw`, whichever is smaller)
`vmax`	viewport maximum unit (value of `vh` or `vw`, whichever is larger)

RELATIVE UNITS

The rem Unit

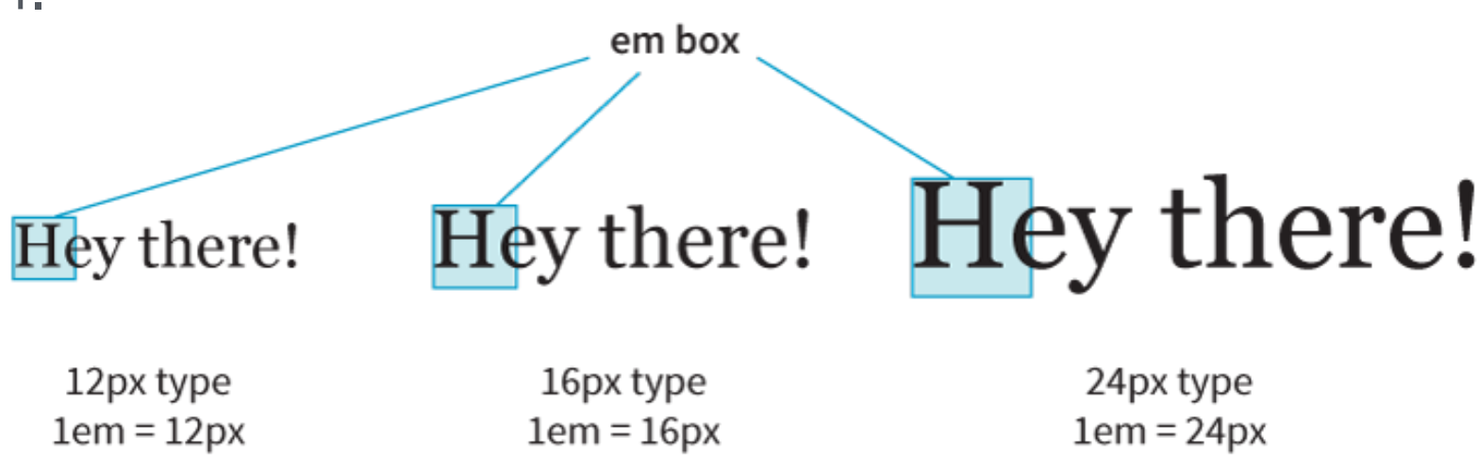
- The **rem** (**root em**) unit is based on the font size of the **`html`** element, whatever that happens to be.
- **Default** in modern browsers: Root font size is 16 pixels, so a **rem = a 16-pixel unit**.
- If the root font size of the document changes, so does the size of a rem (and that's good for keeping elements proportional).

RELATIVE UNITS

The em Unit

- The **em** unit is traditionally based on the width of a capital letter *M* in the font.

- When the font size is 16 pixels,1em = 16 pixels, 2em = 32 pixels, and so on.



NOTE: Because they're based on the font size of the current element, the size of an em may not be consistent across a page.

RELATIVE UNITS

Viewport Percentage Lengths (vw/vh)

Viewport width (**`vw`**) and viewport height (**`vh`**) units are relative to the size of the viewport (browser window):

`vh` = 1/100th width of viewport

`vh` = 1/100th height of viewport

They're useful for making an element fill the viewport or a specified percentage of it. This image will be 50% the width and height of the viewport:

```
img { width: 50vw; height: 50vh; }
```

Browser Developer Tools

Major browsers have built-in tools that aid development:

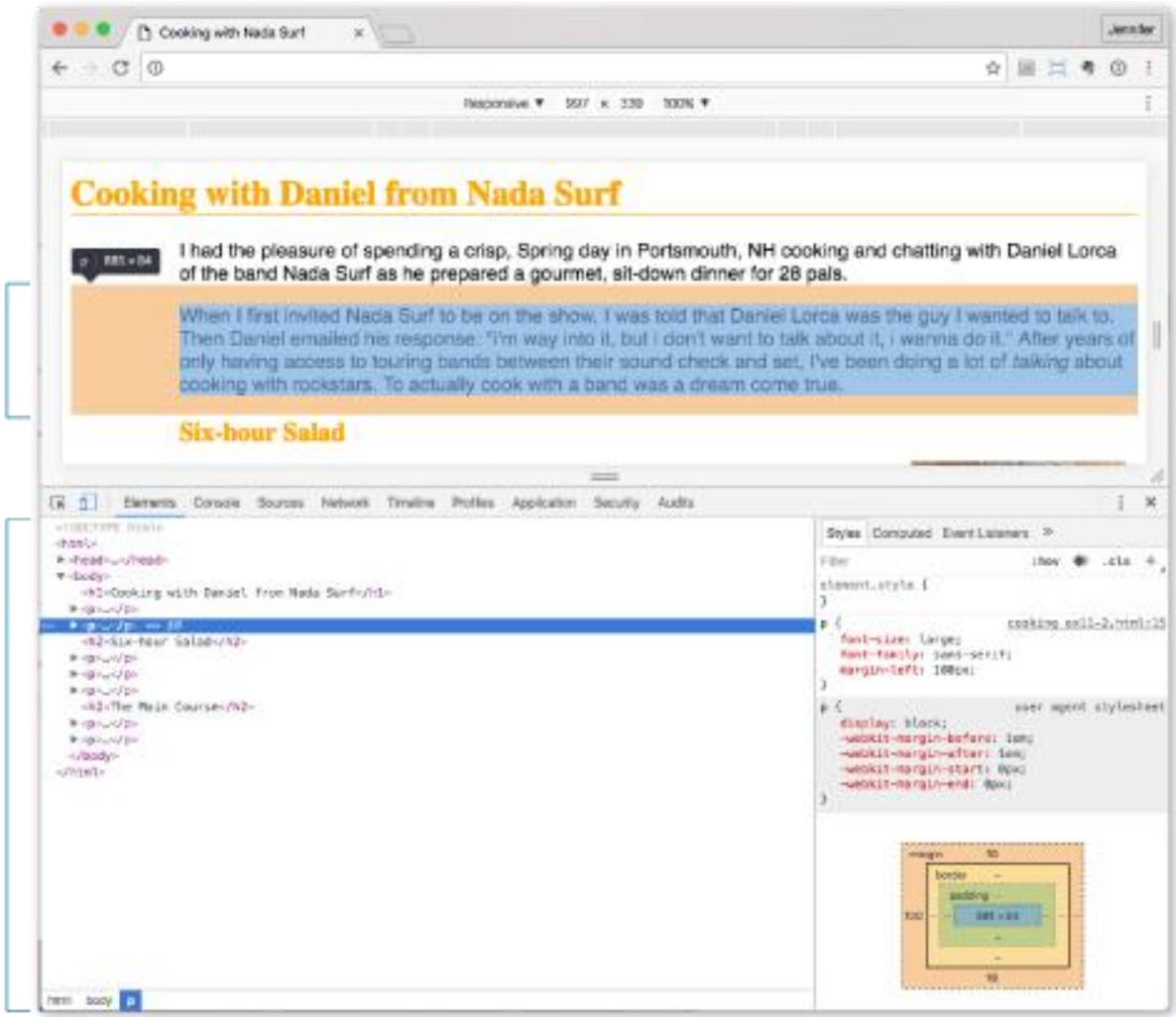
- HTML, CSS, and JavaScript inspectors
- Network speed reports
- Animation tools
- Other helpful features

Browser Developer Tools (cont'd)

Chrome DevTools (View > Developer > Developer Tools)

Elements selected in code are highlighted in the browser view.

HTML source for the page.



All styles that are applied to the selected element.

Margins, borders, and paddings applied to the element.

Firefox, Safari, Opera, and Microsoft Edge also have developer tools.