The goal of this recitation is to learn how to format and beautify your web pages using CSS (cascading style sheets) and to use CSS in a robust manner.

**Stylesheets**

There are three mechanisms to add styling to your page:

1. **Inline styles**
   ```html
   <h1 style="color:blue;margin-left:30px;">This is a heading.</h1>
   ```

2. **Style tags**
   ```html
   <head>
   <style>
   body {
     background-color: linen;
   }
   h1 {
     color: maroon;
     margin-left: 40px;
   }
   </style>
   </head>
   ```

3. **Importing an external style sheet**
   ```html
   <head>
   <link rel="stylesheet" type="text/css" href="mystyle.css">
   </head>
   ```

Separation of concerns dictates that we should prefer external stylesheets. Using inline styles prevents you from easily keeping uniform styles over similar elements. Style tags similarly prevent you from having a uniform style over all of your web pages.

All browsers also have a default stylesheet. If no custom stylesheet is specified, the browser loads this default stylesheet. This provides stylings such as making the `<h1>` bigger than `<h2>` when no custom styles are set.

**Selectors**

There are three types of CSS selectors that you can use to select DOM elements to apply styles to:

1. **Tag Name**
   ```html
   p {
     text-align: center;
     color: red;
   }
   ```
2. Class Name

```css
.center {
    text-align: center;
    color: red;
}
```

3. ID Name

```css
#paral {
    text-align: center;
    color: red;
}
```

There are three ways to combine selectors:

1. Comma separated: This allows you to give multiple elements the same style, in this case, the elements h1, h2, and p.

```css
p, .center, #paral {
    text-align: center;
    color: red;
}
```

2. Space separated: This allows you to specify a style for elements that inherit from a particular parent. In this example, any h2 elements with a parent h1 elem will be colored red.

```css
h1 h2 {
    text-align: center;
    color: red;
}
```

3. Stacked: This allows you to pin-point elements that also have certain class names and ID selectors. The example below will affect only p elements with the class “center”.

```css
p.center {
    text-align: center;
    color: red;
}
```

Pseudo elements/classes add styles to special cases including: hover, link, first-child, first-line, etc. Pseudo-elements have a double colon. Pseudo-classes use single colon.

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

```css
p::first-line {
    color: #ff0000;
    font-variant: small-caps;
}
```
Specificity
Each CSS selector is given a separate point value. Point values are used to determine which style rules take precedence. Order of precedence is inline styles > IDs > classes > tags. Each subsequent selector is used as a tiebreaker. Since IDs are very powerful, they should be used sparingly.

It's much better (and more modular) to layer styles as opposed to creating very specific rules.

```
<a class="btn btn-danger">... </a>
<a class="btn btn-success">... </a>
```

Bad:
```
.btn.btn-danger {
  font-size: 16px;
  background: red;
}
.btn.btn-success {
  font-size: 16px;
  background: green;
}
```
Finally, the !important tag can be appended to any CSS rule to give it the highest priority. This is a bad practice for production code. It should only be used for testing.

**Cascading**

CSS stands for Cascading Style Sheets. Cascading indicates that the rules defined last -- at the bottom of the stylesheet -- take precedence over the rules defined at the top. Cascading precedence is used as the final tie break if conflicting rules have the exact same level of specificity.

It's not considered wise to depend on cascading styles to properly format your web page. The reason is that cascading leads to brittle CSS. Static assets like CSS and JavaScript often undergo compilation and minification. This could potentially reorganize the order of CSS causing elements to display in weird manners.

**The Box Model**

First, the *display* property describes the type of element. The most important are:

1. **block** - starts on a new line. Stretches out as far left and right as possible.
2. **inline** - wraps around text inline. Doesn't disturb the flow of the layout.
3. **inline-block** - The element itself is inline-level. The inside is block-level.

The box-model is represented below. Height and width define the inner level box. Then follows padding, the border, and the margin.
The frustrations of the box model are well-outlined at: http://learnlayout.com/box-model.html. Many designers want to be able to set height and width, and be confident that the width of a box is not modified by padding, border, and margin. You can use the “box-sizing: border-box;” rule to make your CSS more intuitive: http://learnlayout.com/box-sizing.html

**Responsive Design**

Different users will often have different window sizes, especially depending on what kind of device they are using to access your site. Responsive design is the practice of making your webpage adjust to different sized windows, to make it easy for users to view your website.

We won’t go into too much detail here, but you can check out the links below to learn more about different properties that will help with responsive design.

- **width vs. max-width**
  - http://learnlayout.com/max-width.html
- **position: static, absolute, relative, fixed**
  - http://learnlayout.com/position.html
  - http://learnlayout.com/position-example.html
- **float / clear**
  - http://learnlayout.com/clear.html
- **media queries**
  - http://learnlayout.com/media-queries.html
- **flexbox**

**Less CSS Introduction**
Less extends the CSS language to make it more powerful. It adds variables, so you don’t have to remember specific hex values when specifying colors or specific pixel values for margins and padding. This makes it much easier to create themes and change themes, as well. It also allows you to nest selectors, allow mixins, and functions.

```
@base: #f938ab;

.box {
  color: saturate(@base, 5%);
  border-color: lighten(@base, 30%);
  div { .box-shadow(0 0 5px, 30%)
}
}
```

compiles to

```
.box {
  color: #fe33ac;
  border-color: #fdceea;
}
.box div {
  -webkit-box-shadow: 0 0 5px rgba(0, 0, 0, 0.3);
  box-shadow: 0 0 5px rgba(0, 0, 0, 0.3);
}
```

**Frameworks**

Others have written comprehensive CSS frameworks to make your lives easier. Some of the most common frameworks are [bootstrap](http://getbootstrap.com) and [blueprint](http://learn.shayhowe.com/html-css). We aren’t going to cover these in details, but we recommend taking a look at these frameworks. You may be interested in using these for your later projects.

**More References:**

- [http://learnlayout.com/](http://learnlayout.com/)
- [http://getbootstrap.com](http://getbootstrap.com)