CSS Layouts

Chapter 4

- Debugging CSS or HTML can be difficult.
- There are no print statements to help us debug our code.
- Use Firebug for Firefox or Chrome.
- Internet Explorer has a similar built-in feature.
- Right-click on the content you wish to troubleshoot and click on "Inspect Element."

- HTML has a rather boring default layout.
- When modifying the layout, it's useful to draw out a rough sketch first.
- **div** - a block element that represents a major division on a page.
- **span** - inline element that represents a meaningful portion of text.
- A div or span has no inherent appearance. However, when we apply id's or classes to an element, we can use CSS to give the text a new appearance.
- HTML5 introduces new tags to describe a div better: article, aside, figure, footer, header, nav, section.

- **Context Selector** - targets an element if it is inside some specified element.
- **Descendant Combinator** - matches any descendant of the outer selector. For example, *.outerList li* below.
- * can be used as a wildcard to specify any element.

Example

```
.outerList li {
    background-color: blue;
    font-weight: bold;
}
In the HTML file:
<ul class="outerList">
    <li>content</li>
    <li>content</li>
    <li>content</li>
</ul>
```

- **Direct Context Selector** - only match the inner elements directly inside the outer element. Also called a **child combinator**.

Example

```
.outerList > li {
    background-color: blue;
    font-weight: bold;
}
```

In the HTML file:

```
<ul class="outerList">
    <li>content</li>
        <ul><li>not formatted</li></ul>
    <li>content</li>
    <li>content</li>
</ul>
```

- What color is the text in the following example?

Example

In HTML file:

```
<div id="begin">
   <p class="myClass">Color?</p>
</div>
```

In CSS file:

```
div p { color: red; }
#begin > p { color: blue; }
p { color: yellow; }
.myClass { color: green; }
```

- Similar to before, the most specific rule wins.
- CSS applies rules of *specificity* to determine which rule wins.
- Rules are roughly scored based on the following:
    - +1 for each HTML element mentioned in the rule.
    - +10 for each class mentioned in the rule.
    - +100 for each ID mentioned in the rule.
- Rules are found on page 97 or at http://www.w3.org/TR/selectors/#specificity.

- Each block element is laid out from top to bottom and separated by a line break.
- **Box Model** - describes the rectangular regions occupied by HTML elements. Every element's layout is composed of:
  - Content Area
  - Border around the content
  - Padding between the content and the border (inside the border)
  - Margin between the border and other content (outside the border)
- Complete specification at http://www.w3.org/TR/css3-box/

- Borders have four sides: top, bottom, left and right.
- A border can have:
  - thickness
  - style
  - color
- The border property is a shortcut for the following properties in this order: border-width **border-style** border-color.

Example

```
p {
   border: 10px solid red;
   border-left: thick dotted yellow;
   border-bottom-color: green;
   border-bottom-style: double;
}
```

- There are many border properties. A complete listing can be found at http://www.w3schools.com/css/css_border.asp
- CSS3 allows a rounded border, shadows and images: http://www.w3schools.com/css/css3_borders.asp
- Useful for styling borders around tables. Can use the *border-collapse* element. This avoids double borders.

- **Padding** - is the spacing between the content and the border.
- **Margin** - separation between neighboring elements.
- **Margin collapse** - when a block element is on top of another block element, the margins are combined.
- Similar to borders, padding and margins can be applied to: top, bottom, left, right.

Example

```
p {
    border: 10px solid red;
    padding: 1px;
    padding-left: 5em;
    margin: 2px;
    margin-bottom: 3pt;
}
```

- The default layout may not be good enough.
- An important property for floating elements is *width*. The width property applies only to block elements and img tags.
- The default width is the entire page width.

Example

```
#someID {
    border: 2px solid black;
    width: 15em;
}
p {
    width: 20em;
    border: 2px solid black;
}
```

- **float** - the float property lifts an element up from it's normal content flow and shifts it to the left or right side of the page.
- Nearby elements text wraps around the floating element.
- Align content if you want it on the left or right side of the page. Float content if you want it to hover on some side of the page with other content wrapping around it.
- You use the *clear* property to stop the float.

Example

In the CSS file:

```
#someID {
   float: right;
}
```

In the HTML file:

```
<p id="someID">This is my paragraph that is on the right side of the
page.</p>
```

- Floating content may ruin how you want the content to appear. Consider the following.

Example

In the CSS file:

```
img.sidebar{
    float: right;
}
p {
    border: 2px solid black;
}
```

In the HTML file:

```
<p><img class="sidebar" src="telecope.png" />A lot of content to wrap
around image but not enough.</p>
```

- A solution is to use the *overflow* property. Values include visible, hidden, scroll and auto.

Example

In the CSS file:

```
img.sidebar{
    float: right;
}
p {
    border: 2px solid black;
    overflow: hidden;
}
```

In the HTML file:

```
<p><img class="sidebar" src="telecope.png" />A lot of content to wrap
around image but not enough.</p>
```

- If multiple items are floated right, the first one will be the rightmost and they stack horizontally.
- This allows for multiple columns.
- CSS3 add a set of properties for columns that you can read about at http://www.w3schools.com/css/css3_multiple_columns.asp

Example

In the CSS file:

```
.rightStuff {
    float: right;
    width: 20%;
    border: 2px solid black;
}
```

In the HTML file:

```
<div class="rightStuff">Rightmost <br />column.</div>
<div class="rightStuff">Second rightmost <br />column.</div><div
class="rightStuff">Leftmost <br />column.</div>
```

- **width** - changes the width of an element.
- **height** - changes the height of an element.
- Can use margins to center a block (example below). Only works if width is set, why?

Example

```
.positioning {
    height: 10em;
    width: 50pt;
    margin-left: auto;
    margin-right: auto;
    border: 2px solid black;
    overflow: scroll;
}
```

Inline elements are a bit different.

- size properties are ignored.
- margin-top and margin-bottom are ignored.
- padding-top and padding-bottom work but neighboring lines are not spaced further apart.
- vertical-align aligns it vertically within its block box.

Example

```
.positioning {
    height: 10em;
    vertical-align: middle; /* top, middle, bottom, sub, super */
}
```

- The **position** property allows us to break the standard flow of content. Values include:
  - **static** - default property
  - **relative** - relative to normal static position
  - **absolute** - absolute position on page, removed from normal flow. positioned relative to the entire page*.
  - **fixed** - removed from normal flow and is relative to the entire page, won't move even if the user scrolls.

Example

```
.positioning {
   position: relative;
   left: 1em;
   top: 2em;
   border: 2px solid black;
}
```

- **z-index** - a property that determines which element appears on top of another element.
- **display** - property that lets you set an element to block, inline or none.
- **visibility** - property that lets you hide elements. Values include visible and hidden.
- **opacity** - sets degree of visibility. 0.0 is transparent, 1.0 is opaque.
- display: none ≠ visibility: hidden
  visibility: hidden = opacity: 0.0

Example

```
.change {
    opacity: 0.5;
}
```