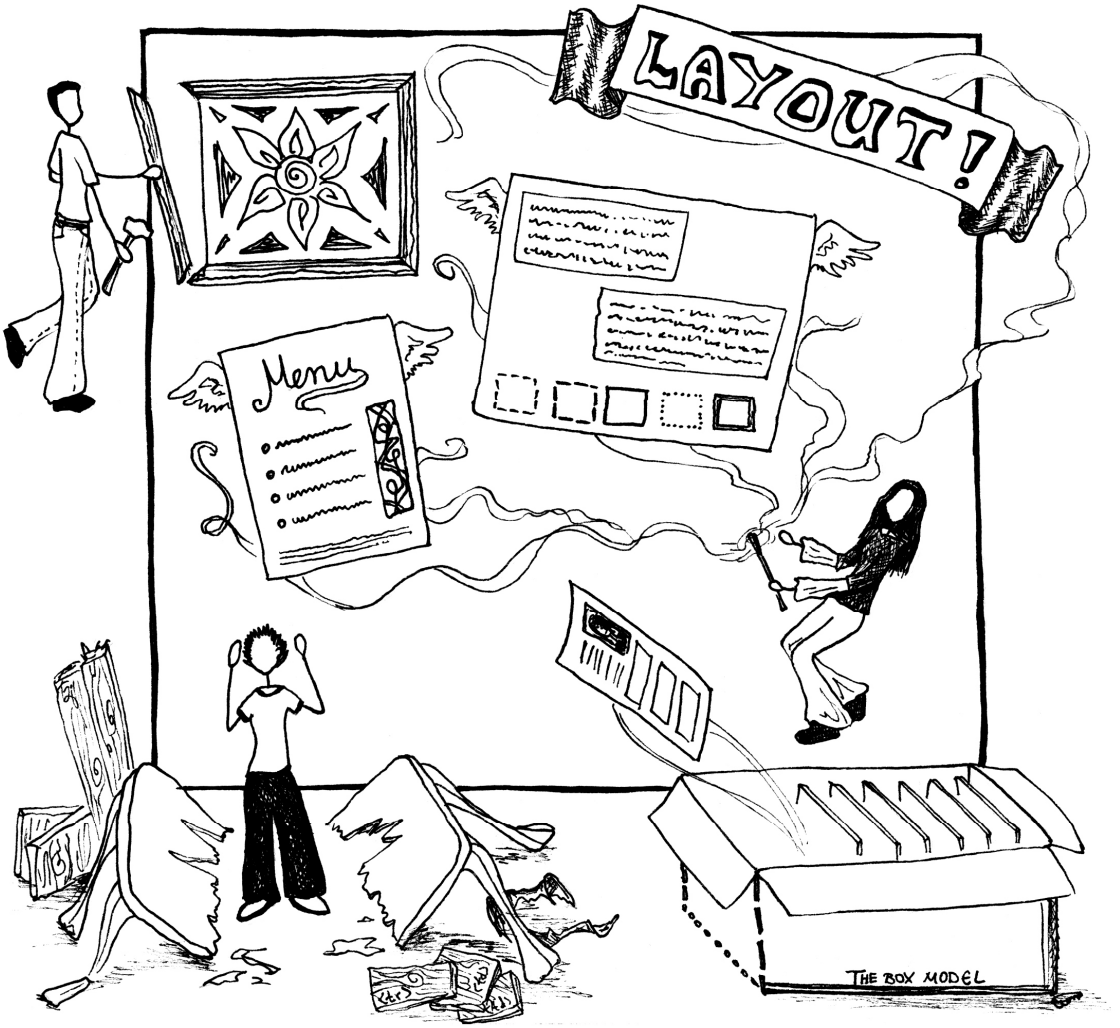
# Chapter 4  Page Layout



## Introduction

In this chapter we'll learn more HTML and CSS with a focus on customizing the layout of web pages. Custom layouts allow us to break the browser's somewhat boring default pattern where each block element appears below the last.

The core of web page layout is a set of CSS rules and properties collectively referred to as the CSS box model. Learning about the box model will enable us to change the size and position of elements and the gaps and margins between them.

We will also learn about "floating" elements which position themselves at the left or right edge of a page. Another useful layout tool is absolute and fixed positioning, which allow you to put elements anywhere on the page that you like.

Before studying layout and the box model, we'll first need to learn some more CSS and HTML to give identification labels to particular elements and to group elements together. This will enable us to select particular elements or groups of elements, so that we can apply a layout style to precisely the desired part of the page.

Unfortunately Internet Explorer does not properly support HTML and CSS layout standards. One section of this chapter is devoted to mentioning several prominent problems in IE and learning about ways to work around them.

This chapter contains a larger "Case Study" example of a site for an ultimate Frisbee club. We'll come up with a design for this site's layout and implement it over the course of this chapter.

### Chapter Outline

## 4.1 Styling Page Sections

Let's consider the task of making a web site for an ultimate Frisbee club we've formed with our friends. The site will contain announcements about upcoming matches, news articles about ultimate, and links to various other useful ultimate sites.

The page has a heading at the top. Announcements will scroll down the center of the page, each with a styled underlined heading. Some announcements contain images, and we'd like them to hover to the right of the text. News stories go in boxes on the left, nested within the main section of the page. A calendar of events appears on the right. Since the calendar and events are important, we'd like that section always to be visible even when the user scrolls down the page. As the user scrolls the page, the events calendar should not move. The page's desired appearance is shown in Figure 4.1.

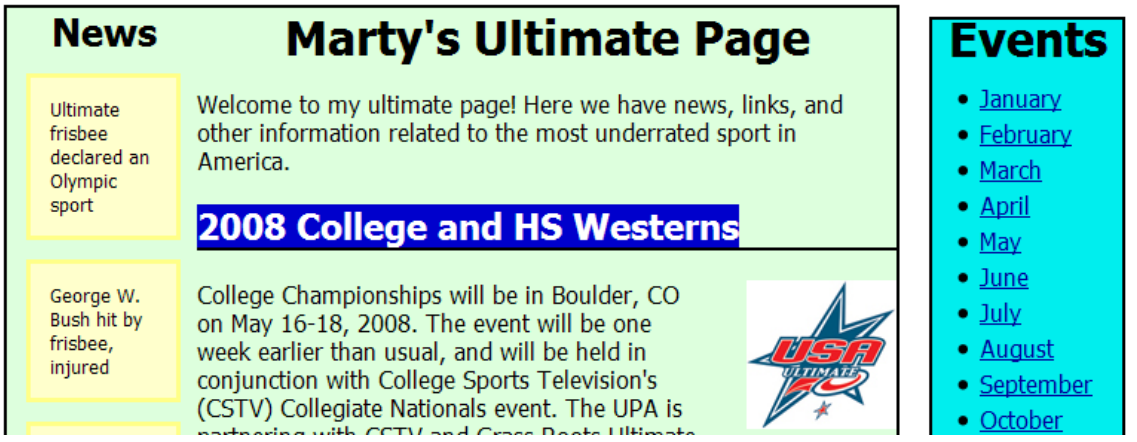



**Figure 4.1 Ultimate Frisbee web page, desired appearance**

To achieve such a layout, we'll need to be able to group contents of the page into major sections and apply styles to each section. In our page, the sections are the main central announcements, the news items on the left, and the calendar on the right. It can be helpful to draw out a rough sketch of the sections of a page. Figure 4.2 shows a rough sketch of those sections. Before we can create the Ultimate Frisbee site, we'll need to learn a bit more HTML and CSS. We'll learn new HTML tags to represent sections of a page and how to write more precise CSS selectors for styling.

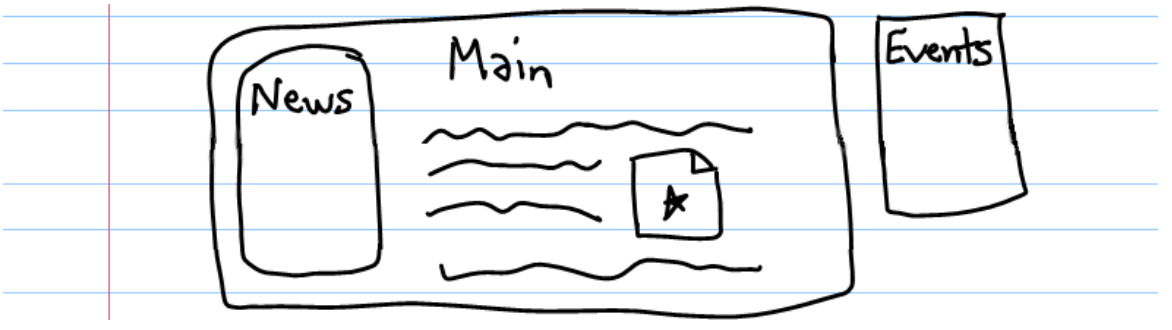



**Figure 4.2 Frisbee page layout sections**

### 4.1.1 Page Sections (div)

| Element | div |
|---|---|
| Description | Section of a page (block) |
| Syntax | <br>  **content**<br> |

HTML has two elements that are very useful for layout. One of these is `div`, a block element that represents a major division or section of a page. The other is `span`, an inline element that represents a meaningful span of text. These elements don't have as much specific meaning as an element like `ul` or `a`; they exist mainly to serve as targets of CSS rules. You can apply a style to a region of your page by wrapping it in a `span` (if it is a short inline part of the page) or a `div` (if it is a large section encompassing one or more block elements), and then applying CSS styles to that `span` or `div`.

The `div` element is a block element that represents a division or section of a page. When we write our ultimate frisbee page, we can wrap each major section's group of paragraphs, headings, and other content in a `div`. By default, a `div` has no onscreen appearance, but if we give it a `class` or `id`, we can apply CSS styles to it and its contents as a group. Even if we don't want to apply any styles to the section of the page, using a `div` is still a good idea because conceptually it adds a bit of semantic information about the organization of the page contents. Example 4.1 demonstrates a `div`.

```
<div id="main">
  <h1>Marty's Ultimate Page</h1>

  <p>blah blah blah... welcome to my ultimate page!</p>

  <h2>Annual Members Meeting</h2>
  <p>
    The Annual Members Meeting of the Ultimate Players Association
    will take place on Sunday, January 20, ...
  </p>
</div>
```

```
#main {
  background-color: #ddffdd;
  font: 12pt "Tahoma", "Arial", sans-serif;
}
```


**Example 4.1 Page sections with `div`**

### 4.1.2 Spans of Text (span)

| Element | span |
|---|---|
| Description | Short section of content (inline) |
| Syntax | content |

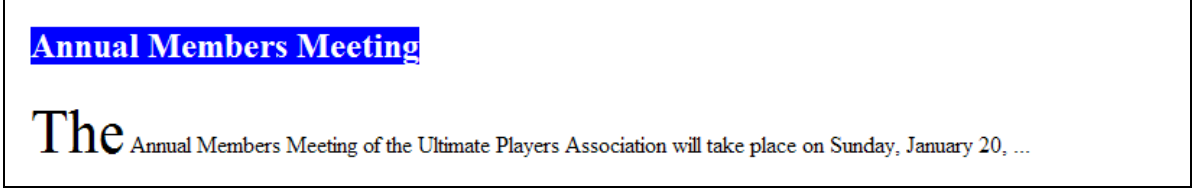
When we have a short inline piece of text that we want to apply a style to, but none of the existing HTML elements make sense semantically, we can wrap the text in a span. Like a `div`, a `span` has no onscreen appearance by default, but we can apply CSS styles to it.

For example, notice that the first word of each announcement appears in a larger font. We can achieve this effect by wrapping those first words in `span`s and then applying a font style to them.

Also, if you look closely at the desired appearance of the headings on our Frisbee page, they use white text on a blue background; you might think we should create an `h2` style that sets these colors. But if we do this, the blue background will extend all the way across the page; the desired appearance is for the blue to appear only behind the text itself. The way to get such an effect is to wrap the headers' text in `span` elements and apply the color styling to the `span`s.

```
<h2>
  <span class="announcement">Annual Members Meeting</span>
</h2>

<p>
  <span class="firstword">The</span> Annual Members Meeting of the
  Ultimate Players Association will take place on Sunday, January 20, ...
</p>
```

```
.announcement {
  background-color: blue;
  color: white;
}

.firstword {
  font-size: 32pt;
}
```




**Example 4.2 Inline text spans**

### 4.1.3 CSS Context Selectors

Suppose you have an unordered list (`ul`) full of ultimate frisbee-related news events (each represented by an `li`). You decide that you'd like to apply a particular style to each event, such as a yellow background color and bold font. You don't want this style on every `li` on the page, just every event in this particular list.

One way to achieve this would be to create a CSS class and apply it to every item in the list, as in the following example. This is not an ideal solution, because there could be a large number of items in the list, and it would be cumbersome and redundant to apply the style to each of them.

```
<ul>
  <li class="newsitem">Ultimate frisbee declared an Olympic sport</li>
  <li class="newsitem">George W. Bush hit by frisbee, hospitalized</li>
  <li class="newsitem">Frisbee catches dog</li>
  <li class="newsitem">Study: Frisbee players more wealthy, virile</li>
</ul>
```

**Example 4.3 Redundant `class` attributes (don't do this!)**

**context selector**
A CSS rule that applies only to elements that reside inside another particular element.

A better way to solve this problem is to place a `class` on the entire `ul` list, then use CSS to target only `li` items inside that list. We can do this by using a *context selector*, which is a selector that only targets an element if it is inside some other element. The syntax for a context selector is shown in Example 4.4.

```
outerSelector innerSelector {
  property: value;
  property: value;
  ...
  property: value;
}
```

**Example 4.4 Syntax template for context selector**

The browser processes such a CSS rule by first looking for elements on the page that match the outer selector, then looking for elements inside them that match the inner selector. An improved version of Example 4.3's code is shown in Example 4.5.

```
<ul class="newslist">
  <li>Ultimate frisbee declared an Olympic sport</li>
  <li>George W. Bush hit by frisbee, hospitalized</li>
  <li>Frisbee catches dog</li>
  <li>Study: Frisbee players more wealthy, virile</li>
</ul>
```

```
.newslist li {
  background-color: yellow;
  font-weight: bold;
}
```

- **Ultimate frisbee declared an Olympic sport**
- **George W. Bush hit by frisbee, hospitalized**
- **Frisbee catches dog**
- **Study: Frisbee players more wealthy, virile**

**Example 4.5 Context selector**

Now imagine that we want our ultimate Frisbee news items to use the previous style, but if any news item has its own sub-list, we don't want the sub-list to have the style. For cases like this, it's pos-

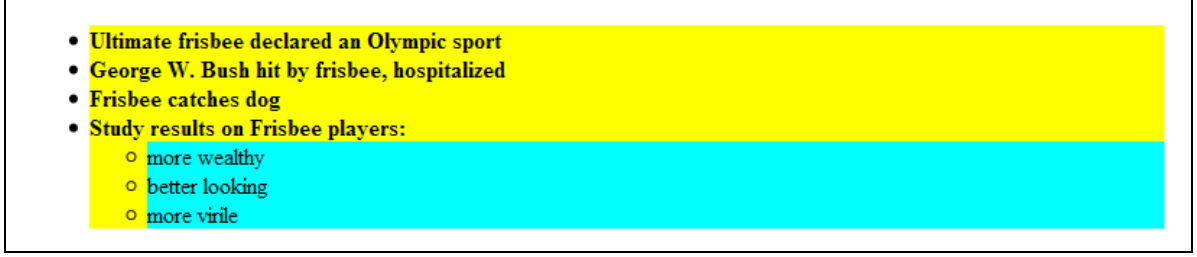
sible to create a *direct context selector* that will match only inner elements that reside directly inside the outer element (as opposed to being nested several elements deep inside it). This is done by placing a > character between the outer selector and inner selector. The syntax is shown in Example 4.6. The HTML and CSS code in Example 4.7 shows a set of nested lists that use a direct context selector.

```
outerSelector > innerSelector {
  property: value;
  ...
  property: value;
}
```

**Example 4.6 Syntax template for direct context selector**

```
<ul class="newslist">
  <li>Ultimate frisbee declared an Olympic sport</li>
  <li>George W. Bush hit by frisbee, hospitalized</li>
  <li>Frisbee catches dog</li>
  <li>Study results on Frisbee players:
    <ul>
      <li>more wealthy</li>
      <li>better looking</li>
      <li>more virile</li>
    </ul>
  </li>
</ul>
```

```
li {
  background-color: cyan;
  font-weight: normal;
}
.newslist > li {
  background-color: yellow;
  font-weight: bold;
}
```




**Example 4.7 Direct context selector**

It's also legal to use * as a wildcard to specify any element. This is most commonly used with context selectors. For example, if you have a `div` with an `id` of `banner`, and you want to give every element inside that `div` a black border (but not give the `div` itself such a border), you could write:

```
div#banner * {
  border: 2px solid black;
}
```

### Specificity and Conflicts

In the previous chapter we mentioned that browsers apply various rules of precedence when style rules conflict. It gets much more complicated when you have class selectors, ID selectors, and context selectors in your CSS file. Consider the HTML and CSS code shown in Example 4.8. All of the rules apply to the paragraph shown, and they all conflict. Which one will be used?

```
<div id="top">
      <p class="new">Where do I go?</p>
</div>
```

```
div p { text-align: left; }
#top > p { text-align: center; }
p { text-align: right; }
.new { text-align: justify; }
```

Where do I go?

**Example 4.8 CSS specificity and conflicts**

**specificity**
The measure of how tightly a rule matches a given element; used to decide which rule to use in case of a conflict.

The answer is that the second rule "wins" in this case because it is considered to be the most specific. CSS applies rules of *specificity* to decide which one should win when two or more rules conflict. Each rule's overall selector is given a score based upon approximately the following rules. The rule with the highest score wins if there is a conflict.

- Any HTML element mentioned in the rule scores 1 point.
- Any class mentioned in the rule scores 10 points.
- Any ID mentioned in the rule scores 100 points.

Based on these rules, we show the specificity scores for several selectors in Table 4.1.

| CSS selector | Specificity |
|---|---|
| `p` | 1 (one HTML element selector) |
| `div > p` | 2 (two element selectors) |
| `.banner` | 10 (one class selector) |
| `p.banner` | 11 (one element and one class selector) |
| `div.box > p` | 12 (two elements and one class selector) |
| `div.box > p.banner` | 22 (two elements and two class selectors) |
| `#logo` | 100 (one ID selector) |
| `body #logo .box p.banner` | 122 (one ID selector, two classes, two elements) |

**Table 4.1 Specificity examples**

Most web programmers don't have all of these rules of specificity memorized. We just vaguely remember that IDs are very specific because they target an individual element, so rules with IDs usually win. Classes are also fairly specific, because paragraphs with class `"foo"` are less common than all paragraphs overall. But classes are clearly not as specific as IDs, since a class can apply to several elements. And plain old elements are the least specific rules of all. So in this way the rules make sense.

The rule shown in the last chapter still applies here: If two rules with the same selector are given, or if two rules with the same specificity apply to the same element, the one declared last wins.

For a humorous take on this subject, check out web designer Andy Clarke's page CSS Specificity Wars, a good explanation of all this using Star Wars characters to represent the different levels of specificity, linked in our references section at the end of this chapter.

## Self-Check

1. What is the difference between a `div` and a `span`? Which is more appropriate for each of the following cases?
    a) A few words at the start of each line represent the title of a movie. We want to color those and make them appear in a different font.
    b) Every three paragraphs of the page constitute a section about a particular author.
    c) Certain words in our news flashes are very important. We want to emphasize them with a bold style and red color.
2. Which CSS rule is more general (matches potentially more elements on the page):
    a) `element1 element2`
    b) `element1 > element2`
3. What color (foreground and background) will be used for each element below?

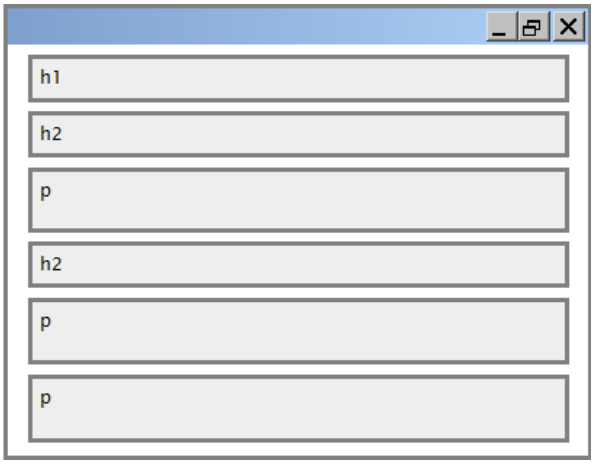
```
<body>
  <p>
    I'm a paragraph; what color am I?
  </p>

  <div class="central">
    <p>
      I'm another paragraph; what color am I?
    </p>
    <ul>
      <li>I'm a list item; what color am I?</li>
    </ul>
  </div>
</body>
```

```
body {
  background-color: yellow;
  color: blue;
}
body > p, ul {
  color: red;
}
.central > li {
  color: green;
}
.central p, .central, ul .central li {
  background-color: cyan;
}
```

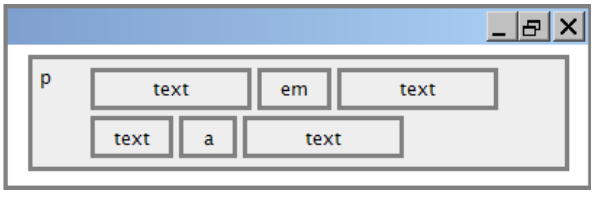
## 4.2 Introduction to Layout

Browsers use a standard layout for pages. Block elements such as `p` and `h1` are laid out in the order they appear in the document, from top to bottom. Each new block element causes a line break. A block element's width is equal to the entire page width in the browser. A block element's height is just enough to fit its text and contents. Within a block element, inline elements and text flow from left to right, top to bottom, wrapping to the next line as needed (except within certain elements such as `pre`). Figure 4.3 illustrates various layouts of block elements, inline elements, and pages.

```
<body>
  <h1>...</h1>
  <h2>...</h2>
  <p>...</p>
  <h2>...</h2>
  <p>...</p>
  <p>...</p>
</body>
```




```
<p>
  Today, <em>24 hrs only</em>,
  blowout sale! See our <a
  href="products.html">
  Products</a> page for info.
</p>
```




an overall page

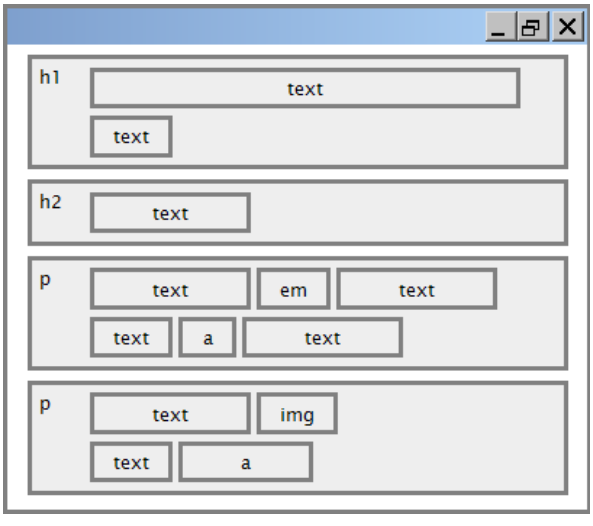



**Figure 4.3 Block element layout**

### 4.2.1 The CSS Box Model

**CSS Box Model**
The set of rules that governs the size, shape, spacing, borders, and margins of page elements.

A set of rules collectively known as the *CSS Box Model* describes the rectangular regions occupied by HTML elements. The W3C CSS specification at http://www.w3.org/TR/REC-CSS2/visuren.html describes in detail the kinds of boxes that exist and how their layout can be manipulated. The W3C's site is a bit of a long read, but it is very complete about the various layout rules. The main idea is that every element's layout is composed of:

- the actual element's *content area*
- a *border* around the element
- a *padding* between the content and the border (inside the border)
- a *margin* between the border and other content (outside the border)

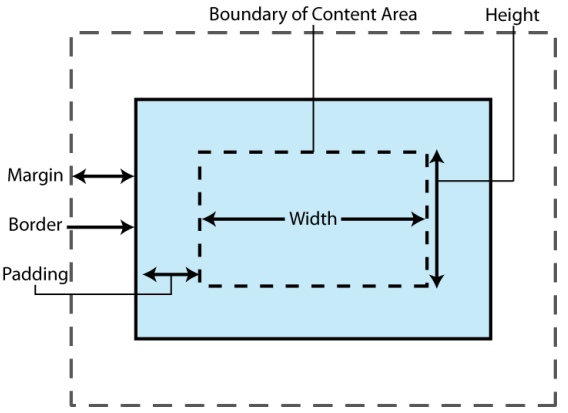



**Figure 4.4 Box model diagram**

You can think of an element's box as being like an HTML table with only one cell in it. Figure 4.4 summarizes the box given to each element on the page by the browser.

The true overall width and height of an element onscreen are the following:

- width = content width + left/right padding + left/right border + left/right margin
- height = content height + top/bottom padding + top/bottom border + top/bottom margin

Many new web developers have trouble remembering the difference between padding and margin. Here's a mnemonic device: The more food you eat, the more padding you get inside your belly. Padding is inside the border, and margin is outside the border.

#### Borders

Each element can have a surrounding line called a *border*. The element's four sides can accept a border: top, bottom, left, and right. A border has:

- a **thickness**, specified in `px`, `pt`, `em`, `%`, or a general width: `thin`, `medium`, or `thick`
- a **style**, which is one of the following: `none`, `hidden`, `dotted`, `dashed`, `double`, `groove`, `inset`, `outset`, `ridge`, or `solid`
- a **color** (specified as seen previously for text and background colors)

Using a variety of border-related CSS properties, you can specify any or all of the three above items for any or all of the four border regions.

| Property | Meaning |
|---|---|
| `border` | all properties of all four borders |
| `border-color`, `border-width`, `border-style` | color/thickness/style of all four borders |
| `border-bottom`, `border-left`, `border-right`, `border-top` | all properties of bottom/left/right/top border |
| `border-bottom-color`, `border-bottom-style`, `border-bottom-width`, `border-left-color`, `border-left-style`, `border-left-width`, `border-right-color`, `border-right-style`, `border-right-width`, `border-top-color`, `border-top-style`, `border-top-width` | specific properties of border on a particular side |
| `border-collapse` | sets whether a table's borders are collapsed into a single border or detached (default) |

**Table 4.2 Border CSS properties**

Each side's border properties can be set individually or as a group. If you omit some properties, they receive default values (e.g. `border-bottom-width` in the following example). Example 4.9 sets several border properties of level 2 headings.

```
<h2>I'm HEADING your way!</h2>
```

```
h2 {
  border: 5px solid red;
  border-left: thick dotted #cc0088;
  border-bottom-color: rgb(0, 128, 128);
  border-bottom-style: double;
}
```


**Example 4.9 Borders**

In the Tables section of the HTML chapter we mentioned that there were better ways of styling table borders other than setting the `table` element's `border` attribute. All the above CSS properties can be used to put a border around a table and table cells. One in particular applies only to tables: `border-collapse`. By default, if a table has borders on both the table and the cells within the table, you will see double borders as in Example 4.10. `border-collapse` merges table borders into one border as shown in Example 4.11.

```
table, td, th {
  border: 2px solid black;
}
```

| Beverage | Caffeine (mg) |
|---|---|
| Brewed Coffee | 80 - 135 |
| Brewed Tea | 60 |

**Example 4.10 Table with borders**

```
table, td, th {
  border: 2px solid black;
  border-collapse: collapse;
}
```

| Beverage | Caffeine (mg) |
|---|---|
| Brewed Coffee | 80 - 135 |
| Brewed Tea | 60 |

**Example 4.11 Table with collapsed borders**

### Padding

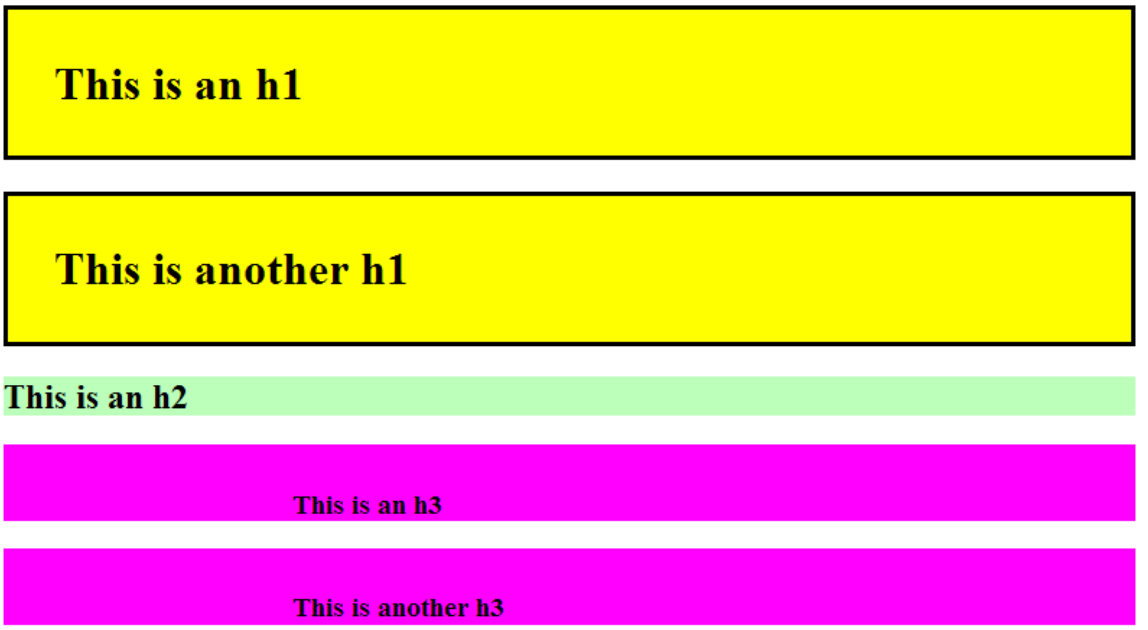
| Property | Meaning |
|---|---|
| `Padding` | padding on all four sides |
| `padding-bottom`, `padding-left`, `padding-right`, `padding-top` | padding on a particular side |

**Table 4.3 Padding CSS properties**

*Padding* gives blank space between the inline contents of an element and its border. As with borders, padding can be applied to any of the following four regions: bottom, left, right, and top. Padding is specified simply as a size, in the standard CSS size units such as `px`, `pt`, `em` or `%`. Example 4.12 sets padding on several elements. Notice that the padding is inside the element's border and shares the background color of the element.

```
<h1>This is an h1</h1>
<h1>This is another h1</h1>
<h2>This is an h2</h2>
<h3>This is an h3</h3>
<h3>This is another h3</h3>
```

```
h1 { padding: 1em; background-color: yellow; border: 3px solid black; }
h2 { padding: 0em; background-color: #BBFFBB; }
h3 { padding-left: 200px; padding-top: 30px; background-color: fuchsia; }
```




**Example 4.12 Padding**

### Margins

| Property | Meaning |
|---|---|
| `margin` | margin on all four sides |
| `margin-bottom`, `margin-left`, `margin-right`, `margin-top` | margin on a particular side |

**Table 4.4 Margin CSS properties**

A *margin* gives a separation between neighboring elements. As with padding, a margin is specified as a size, and can be set on any or all of the four sides of the element. Example 4.13 sets several margins. Notice that the margins are outside the element, and therefore they're always transparent; they don't contain the element's background color.

```
<p>This is the first paragraph</p>
<p>This is the second paragraph</p>
```

```
p {
  margin: 2em;
  background-color: yellow;
}
```





**Example 4.13 Margins**

Many block elements already receive default margins if you don't explicitly set them. For example, many browsers render paragraphs and lists with approximately a 1em top and bottom margin.

**margin collapse**
Vertical margins that are combined.

One subtlety about margins is that the overall web page body usually has a margin of its own. A common request is to create a page whose content begins flush against the top/left corner of the page. To do this, create a style with body as its selector that sets the margin width to 0px.

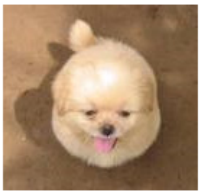
When one block element appears below another, the two elements' top and bottom margins are combined, a phenomenon called *margin collapse*. Their shared margin is the larger of the two individual margins.

**Don't Be a Newb**
Avoid spacer images

A classic web programming hack from the '90s that sadly is still around today is to use a "spacer" image to achieve horizontal or vertical separation between elements. This hack comes from the days when CSS wasn't as mature, so it wasn't easy to space elements with a margin.

The idea was to create a tiny invisible GIF image, and then to place it on the page but with varying `width` and `height` attributes in the HTML. Doing this causes the browser to draw an invisible gap of that size between the surrounding elements. Example 4.14 shows this poor technique.

```
<!-- a spacer image of 200px between two other images -->
<img src="images/smiley.png" alt="smiley face" />
<img src="images/pixel.gif" width="200" height="1" alt="spacer" />
<img src="images/puppy.jpg" alt="cute puppy" />
```



**Example 4.14 Using spacer images (poor style)**

Nowadays we look back in shame at such an egregious hack. The right way to do it is to set a 200px margin on the side of one image to separate it from the other. The code in Example 4.15 looks the same but has much better style.

```
<img src="images/smiley.png" alt="smiley face" />
<img id="puppy" src="images/puppy.jpg" alt="cute puppy" />
```

```
#puppy {
  margin-left: 200px;
}
```

**Example 4.15 Spacing images using CSS margins (better style)**

## 4.2.2 Finding Box Model Problems with Firebug

We recommend Firefox for web development, and if you use Firefox for web programming, you simply must install Firebug. It's an extremely powerful add-on that, among many other things, allows you to "inspect" the code of any page to see a visual layout of its box model, all of its CSS properties, and generally learn anything you want about the content on the screen. It's a phenomenal tool for answering those, "Why does it look like that?" questions.

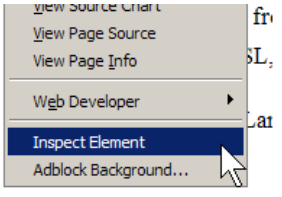



**Figure 4.5 Inspecting an element with Firebug**

The quickest way to find box model problems is to right-click an element in Firefox and choose Inspect Element, as shown in Figure 4.5. You'll see the HTML on the left and the CSS styles that apply to the element on the right. Inspect those styles to make sure the ones you expect are there. If the size or location of the element aren't right, click Firebug's Layout tab on the right to see the element's size and shape, including its padding and margins, colored highlights of each region, and a handy ruler overlay, as shown in Figure 4.6.

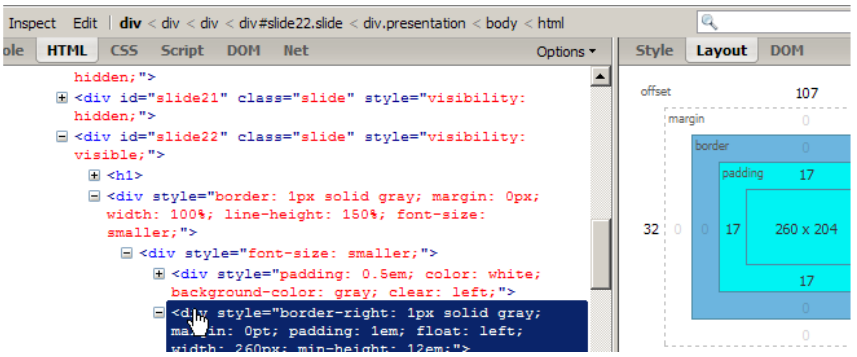



**Figure 4.6 Using Firebug**

Firebug is a wonderful tool for web development. Install it from http://www.getfirebug.com/.

## Self-Check

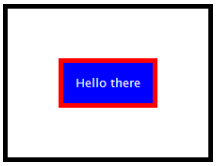
4. If you specify for an element to have a width of 10em, a padding of 2em, a border-width of 1em, and a margin of 1em, how much total horizontal space is occupied by the element?
   a) 10em
   b) 12em
   c) 14em
   d) 16em
   e) 18em
   f) 20em
5. What's the difference between margin and padding?
6. Consider the following HTML/CSS code:

```
<div> <p>Hello there</p> </div>
```

```
div {
  border: 5px solid black;
  padding: 1em;
}
p {
  background-color: blue;
  border: 5px solid red;
  color: white;
  margin: 1em;
  padding: 4em;
}
```

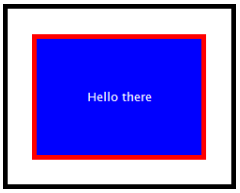
Which of the following best matches the appearance the code would have in the browser?

a)




b)


c)



## 4.3 Floating Elements

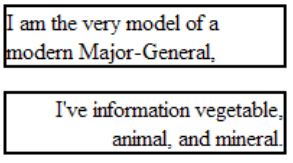
There are times when the browser's normal layout simply won't position an element the way you want. For example, you may want multiple columns of text. You might want an image on the side of your page with text wrapping around it. You might want a sidebar of useful links. None of these layouts can be achieved with the CSS properties we've seen so far. In the following sections we'll learn about floating layouts, which are useful for precisely these kinds of situations.

| Property | Meaning | Value |
|---|---|---|
| `width` | how wide to make the element's content area | a size (`px`, `pt`, `%`, `em`) |

Before talking about floating layouts, we must mention the CSS `width` property. Normally a block element is given a width equal to the entire page width. But if you specify some other width by setting a width property value, you can control how wide that element and its content appear in the browser. The `width` property applies only to block elements and to the `img` element; it is ignored for other inline elements. Example 4.26 demonstrates two elements with `width` settings.

```
<p id="ex1">I am the very model of a modern Major-General,</p>
<p id="ex2">I've information vegetable, animal, and mineral.</p>
```

```
#ex1, #ex2 {
  border: 2px solid black;
  width: 12em;
}
#ex2 {
  text-align: right;
}
```




**Example 4.16 Element width**

Notice that if the `text-align` is set to `right`, it makes the text within the element right-aligned, but the overall element itself is still on the left edge of the page. We could right-align the overall `div` using appropriate left and right margins. We mention the `width` property here because it is very important when creating floating layouts.

### 4.3.1 The `float` Property

| Property | Meaning | Value |
|---|---|---|
| `float` | whether to use "float" positioning to lift this element from the normal content flow | `left`, `right`, `none` (default) |

The CSS `float` property lifts an element up from the normal content flow and shifts it to either the left or right edge of the page, where it hovers or floats above other content. Any nearby elements' text wraps around the floating element as necessary.

A floating element's vertical position is the same as it otherwise would have been on the page. Its horizontal position is flush against the left or right edge of the document. If the floating element is contained within another block element, it floats against the edge of that element instead. If you want it to float a bit of distance away from the edge, you can set a margin on the floating element. There is no way to float content in the center of the page, only to the left or right edge, which can be frustrating for new web developers.

**floating element**

One that is lifted out of the normal page layout and placed against the far left or right edge of the page or its containing element.

Floating elements are great for elements that you want to "hover" on one edge of your page, with multiple lines of text wrapping around them. Example 4.17 demonstrates a floating element, and Figure 4.7 shows a sketch of the layout that would result in the browser.

```
<div id="sidebar">...</div>
<h1>...</h1>
<p>...</p>
<p>...</p>
```

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

**Example 4.17 Floating element**

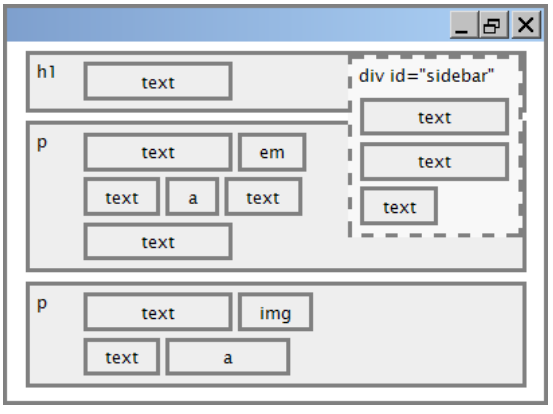


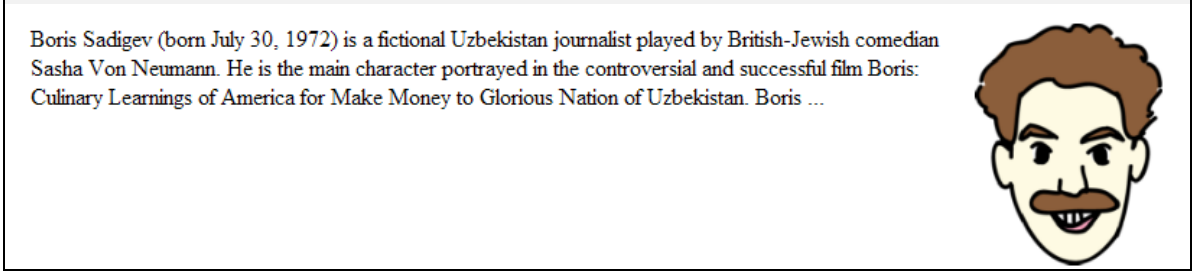

**Figure 4.7 Floating element (output)**

Many new web programmers don't understand the difference between floating and alignment. They decide that if they want something on the right side of the page, they should always float it right. But floating is a more drastic measure than simple alignment. A floating element is removed from the normal block flow of the document; other block elements lay themselves out around and underneath the floating element, ignoring it for layout purposes. But the inline content within those block elements does respect the floating element and wraps neatly around it. The rule of thumb is, if you simply want something on the left or right side of the page, align it. If you want it to hover on that side of the page with other content wrapping around it, float it.

Normally we float block elements such as a `div`, and in fact when any element is floated it is thereafter treated as a block box. This means that it can have a `width` setting and horizontal margins, unlike most inline content. One inline element that is often floated is `img`, producing a hovering image next to a multi-line section of text. This also avoids the nuisance of placing a tall image inline with a large amount of shorter neighboring text. Example 4.18 demonstrates a floating image.

```
<p>
  <img class="hovericon" src="images/boris.png" alt="Boris" />

  Boris Sadigev (born July 30, 1972) is a fictional Uzbekistan
  journalist played by British-Jewish comedian Sasha Von Neumann. He is
  the main character portrayed in the controversial and successful film
  Boris: Culinary Learnings of America for Make Money to Glorious Nation
  of Uzbekistan. Boris ...
</p>
```

```
img.hovericon {
  float: right;
  width: 130px;
}
```




**Example 4.18 Floating image**

If a floating block element has lengthy inline content, it should have a `width` property value to constrain its width. If no `width` is specified, the floating element's content will occupy 100% of the page width, and there will be no room for content to wrap around it. Figure 4.8 shows several floating elements, some of which have `width` settings and some that do not. Notice that the second paragraph, which floats but has no width setting, occupies 100% of the width of the page.

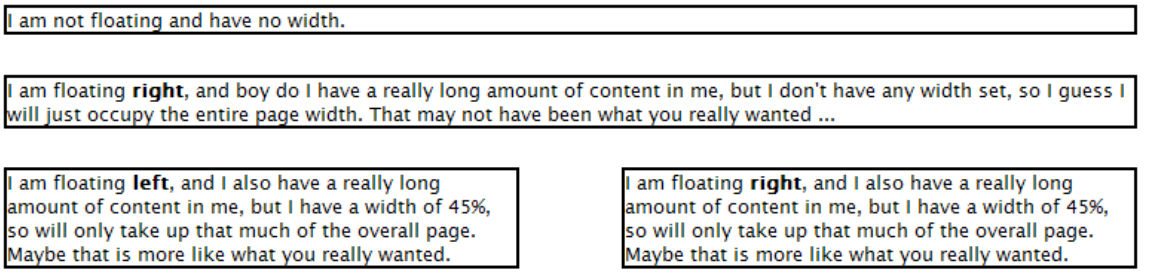



**Figure 4.8 Floating elements with widths**

## 4.3.2 The `clear` Property

| Property | Meaning | Values |
|---|---|---|
| `clear` | whether to move this element below any prior floating elements in the document | `left`, `right`, `both`, `none` (default) |

The `clear` property disallows any floating elements from overlapping some other element. You place this element below any left-floating elements by setting `clear` to `left`, or place it below any

right-floating elements by setting `clear` to `right`. You can place the element below any floating elements on either side by setting `clear` to `both`. Example 4.19 shows a cleared element and Figure 4.9 shows the resulting page layout.

```
<div id="sidebar">...</div>
<h1>...</h1>
<p id="section2">...</p>
<p>...</p>
```

```
#sidebar {
  float: right;
}
#section2 {
  clear: right;
}
```

**Example 4.19 Clear**

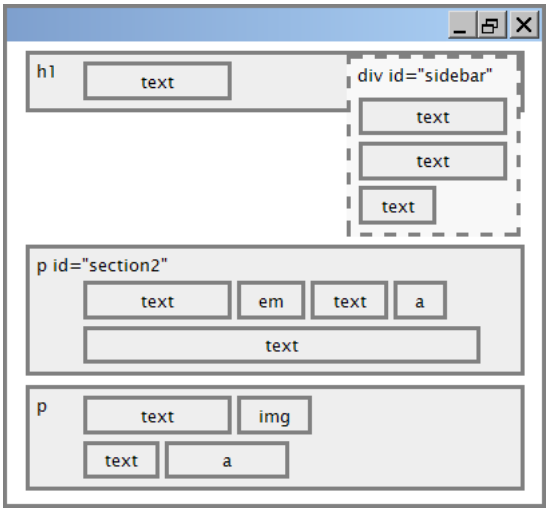



**Figure 4.9 Clear (output)**

The effect of `clear` is that the element will appear below any previous floating content, rather than side-by-side. It's your way of saying, "Stop the float, I want to get off!" Example 4.20 shows a floating image and a heading that clears. Notice that the yellow "My Starhome Sprinter Fan Site" text drops down below the image of the character, which it wouldn't do without a `clear` setting.

```
<p>
  <img class="hoveringicon" src="images/starhome.png" alt="starhome" />
  Starhome Sprinter is a Flash animated Internet cartoon. It mixes
  surreal humour with references to 1980s and 1990s pop culture,
  notably video games, classic television and popular music.
</p>

<h2>My Starhome Sprinter Fan Site</h2>
```

```
img.hoveringicon {
  float: left;
  margin-right: 1em;
}
h2 {
  clear: left;
  background-color: yellow;
}
p {
  background-color: fuchsia;
}
```

Starhome Sprinter is a Flash animated Internet cartoon. It mixes surreal humour with references to 1980s and 1990s pop culture, notably video games, classic television and popular music.

My Starhome Sprinter Fan Site

**Example 4.20 Heading with `clear`**

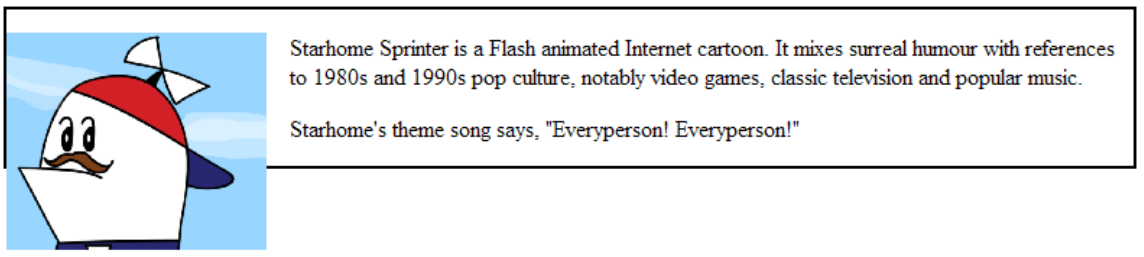
### 4.3.3 Making Floating Elements Fit

One annoyance about floating elements is that because they're lifted up out of the normal document flow, they have no effect on the width or height of the block element containing them. In other words, if you place a tall floating element inside a block element without much other content, the floating element may hang down past the bottom edge of the block element that contains it.

If we get rid of the cleared `h2` from Example 4.20, you can see this problem in action. Example 4.21 makes this modification to the code. Notice that the image hangs down below the black border for the `div` that contains it. This probably isn't the appearance the author intended. We'd rather have the border extend downward to enclose the floating image.

```
<div id="main">
  <p>
    <img class="hoveringicon" src="images/starhome.png" alt="starhome" />
    Starhome Sprinter is a Flash animated Internet cartoon. It mixes
    surreal humour with references to 1980s and 1990s pop culture,
    notably video games, classic television and popular music.
  </p>
  <p>Starhome's theme song says, "Everyperson! Everyperson!"</p>
</div>
```

```
img.hoveringicon {
  float: left;
  margin-right: 1em;
}
#main {
  border: 2px solid black;
}
```




**Example 4.21 Floating image that doesn't fit**

One workaround for this problem is to place a final empty element at the bottom of the `main` section and give it a suitable `clear` value. This will cause the `div` to extend its height downward far enough to accommodate the floating image and the empty element below it. However, this is poor style, since the element with its `clear` property is added to the page entirely for layout purposes and has no semantic value.

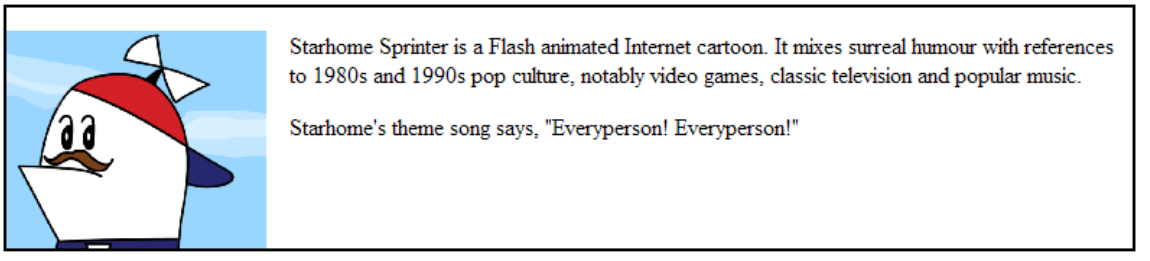
| Property | Meaning | Values |
|---|---|---|
| `overflow` | action to take if element's content is larger than the element itself | `visible` (default), `hidden`, `scroll`, `auto` |

A better workaround is to use the CSS `overflow` property. This property specifies what an element should do if its content is too big for it. Essentially the property is used to enable and disable scrollbars on elements like text boxes and tall `div`s. But in this case it's also useful for telling the browser not to let the floating content hang off the bottom of an element.

By setting the outer `div`'s `overflow` property to either `hidden` or `auto`, the browser will make the element large enough to fit the floating content. Various online tutorials suggest that using `hidden` is the best choice since it is most compatible with various web browsers such as Internet Explorer 7. Example 4.22 demonstrates this idea. Notice how the enclosing `div` is now tall enough to fit the entire image.

```
<div id="main">
  <p>
    <img class="hoveringicon" src="images/starhome.png" alt="starhome" />
    Starhome Sprinter is ...
  </p>
  <p>Starhome's theme song says, "Everyperson! Everyperson!"</p>
</div>
```

```
img.hoveringicon {
  float: left;
  margin-right: 1em;
}
#main {
  border: 2px solid black;
  overflow: hidden;
}
```




**Example 4.22 Floating image that fits**

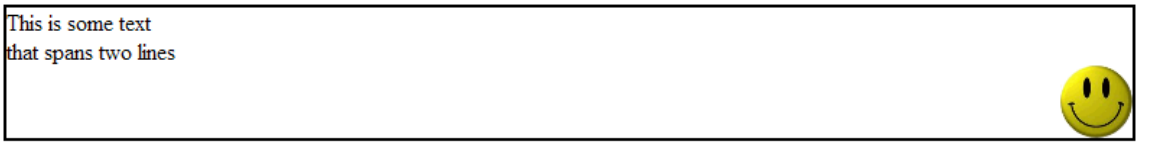
**Common Error**

Floating element too late in the page

If you specify a floating piece of content after another block element, it will appear below that element. Web developers new to using the float property encounter this when trying to make a floating element appear directly to the right of another element just before it on the page. Notice how the smiley face appears below the paragraph in Example 4.23.

```
<div id="main">
  This is some text <br /> that spans two lines
  <img id="logo" src="images/smiley.png" alt="smiley" />
  <div class="spacer"></div>
</div>
```

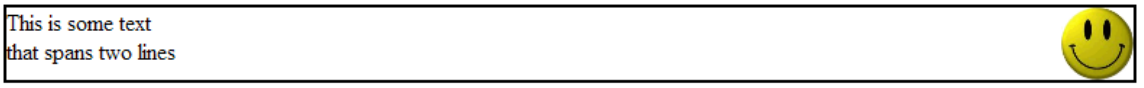
```
#logo {    float: right; }
#main {    border: 2px solid black; }
.spacer {  clear: both; }
```




**Example 4.23 Common error: floating element too late in page**

The problem occurs even if we put the two lines of text into a paragraph and set a small `width` value on the paragraph. The proper fix is to define the logo image before the text in the HTML, so its floating position is established before the text is drawn by the browser, as shown in Example 4.24.

```
<div class="urgent">
  <img id="logo" src="images/smiley.png" alt="smiley" />
  This is some text <br /> that spans two lines
  <div class="spacer"></div>
</div>
```




**Example 4.24 Corrected error with floating image**

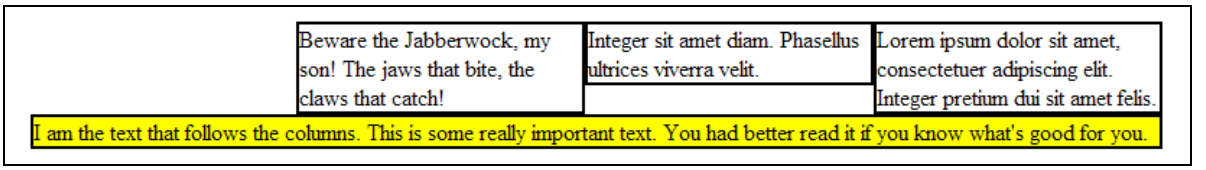
### 4.3.4 Multi-Column Floating Layouts

When more than one element floats in the same direction, they stack horizontally. The first one is the furthest toward the float direction (for example, if all the elements float right, the first one will be the furthest right).

This stacking can be useful for creating pages with multi-column layouts. To do this, create multiple `div`s, each with a `float` and `width` attribute. Example 4.25 demonstrates such a multi-column layout. Since the columns float, the paragraph following the columns is given a `clear` value to make sure it is placed below the columns.

```
<div class="column">
  Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
  Integer pretium dui sit amet felis.
</div>
<div class="column">
  Integer sit amet diam. Phasellus ultrices viverra velit.
</div>
<div class="column">
  Beware the Jabberwock, my son!
  The jaws that bite, the claws that catch!
</div>
<p id="aftercolumns">
  I am the text that follows the columns.
  This is some really important text.
  You had better read it if you know what's good for you.
</p>
```

```
div, p {
  border: 2px solid black;
}
.column {
  float: right;
  width: 25%;
}
#aftercolumns {
  background-color: yellow;
  clear: both;
}
```




**Example 4.25 Multi-column layout**

It's important to note that any content to follow these columns should clear the previously set `float`. If this is not done, the following content tries to flow around the floating columns, which usually produces the wrong appearance. If Example 4.25's code had no `clear` attribute set on the last paragraph, it would have the appearance shown in Figure 4.10. The final paragraph actually appears to the left of the column `div`s, which is likely not the appearance intended. Interestingly the paragraph's yellow background also bleeds through into the `div`s. To avoid this we could explicitly give the column `div`s a `background-color` of `white`.

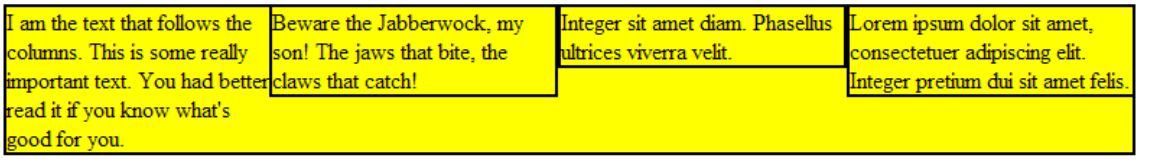


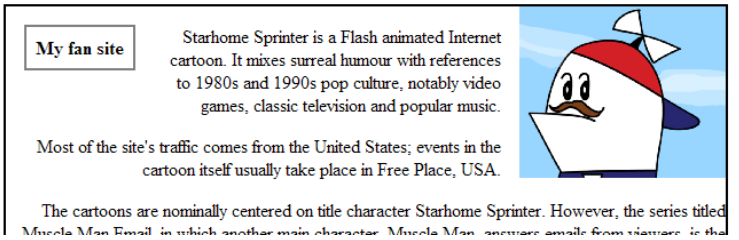

**Figure 4.10 Multi-column layout without `clear`**

## Self-Check

7. What is the difference between setting `float: right;` and `text-align: right;` ?
8. If you float two consecutive elements to the same side of the page, what happens? Where will the two elements appear with respect to each other?
9. Why is it important to set a `width` on a floated element? What can happen if it is not set?
10. What is the meaning of the `overflow` property? How is it related to floating elements?
11. Given the following code (abbreviated):

```
<div id="main">
  <h1>My fan site</h1>

  <img src="images/starhome.png" alt="starhome" />

  <p>Starhome Sprinter is a Flash animated Internet cartoon...</p>
  <p>Most of the site's traffic comes from the United States...</p>
  <p>The cartoons are nominally centered on title character ...</p>
  <p>Starhome's theme song says, "Everyperson! Everyperson!"</p>
</div>
```

What CSS code would be necessary to make the page have the following appearance?



## 4.4 Sizing and Positioning

We've already seen that each element has a default position and size. The default position of block elements is stacked vertically down the page. The default position of inline elements is in a left-to-right flow from top to bottom within their enclosing block elements. A block element's default width is 100% of the page width, and its default height is the height of its inline content. An inline element's default size is just large enough to fit its contents. It's possible to change these defaults using the CSS properties shown in this section.

### 4.4.1 Width and Height

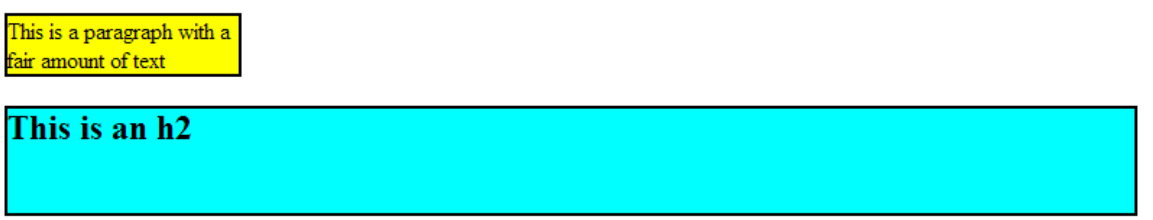
| Property | Meaning |
|---|---|
| `width`, `height` | how wide or tall to make the element's content area |
| `max-width`, `max-height`, `min-width`, `min-height` | the minimum or maximum size of this element |

The size given to an element's content can be changed by setting its CSS `width` and `height` properties. We have already mentioned the `width` property when discussing floating elements. It's also legal to specify minimum and maximum sizes for the element. When a minimum or maximum is set, the browser will try to use the element's default size, but if that would exceed the minimum or maximum, the size is restricted to that bound. All of the properties listed above apply only to block elements and to the `img` element; they are ignored for other inline elements (unless they are floating).

Example 4.26 demonstrates sizing of several elements. Notice that since `h2` is given a height of 3em, its box is now too tall for its contents, so the text inside the heading appears at the top of the element with some empty shaded space underneath. Later in this chapter we'll see how to change that with the `vertical-align` property if we want to do so.

```
<p>This is a paragraph with a fair amount of text</p>
<h2>This is an h2</h2>
```

```
p {
  width: 10em;
  background-color: yellow;
  border: 2px solid black;
}
h2 {
  height: 3em;
  background-color: aqua;
  border: 2px solid black;
}
```



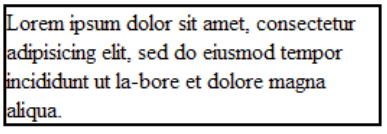

**Example 4.26 Width and height**

### Centering Block Elements

It's also possible to center a block element on the page using margins. To do so, give a `width` to the element and then set its `margin-left` and `margin-right` to the special value `auto`. A value of `auto` means to let the browser decide the margins. In this case, setting them both to `auto` has the effect of making them both the same size, which places the element horizontally in the center of the page. Example 4.27 illustrates this technique.

```
<p>
  Lorem ipsum dolor sit amet, consectetur adipisicing elit,
  sed do eiusmod tempor incididunt ut la-bore et dolore magna aliqua.
</p>
```

```
p {
  border: 2px solid black;
  margin-left: auto;
  margin-right: auto;
  width: 33%;
}
```




**Example 4.27 Centering with auto margins**

Centering with `auto` margins only works if the element has a `width` setting; if not, the element occupies 100% of the page width and is too wide to be centered.

It's important to understand the difference between centering a block element and centering the inline content inside it. Notice that the text in the preceding example isn't center-aligned; each line is left-aligned, just not with respect to the left edge of the page. To center inline text within a block element, give the block element a `text-align: center;` instead.

### Inline Elements

You can also adjust the size and position of inline elements, though there are some distinctions between them and block elements:

- The various size properties (`width`, `height`, `min-width`, `max-height`, etc.) are ignored for inline boxes, except when used with the `img` element.
- The `margin-top` and `margin-bottom` properties are ignored. (`margin-left` and `margin-right` do work.)
- The `padding-top` and `padding-bottom` properties cause a padding to appear between an inline element and its border as expected, but they do not cause neighboring lines to be spaced any further apart. This can cause the inline element's border or padding to run over into the lines above and below. The `img` element is the one exception to this rule; `img` elements with vertical padding do cause spacing between themselves and neighboring lines. (The `padding-left` and `padding-right` properties behave as expected on an inline element; they do cause neighboring text content to be spaced farther to the left and right.)
- The containing block element's `text-align` property controls horizontal position of inline boxes within it.

- Each inline box's `vertical-align` property aligns it vertically within its block box.

The `vertical-align` property is a new one we haven't discussed yet, and it has some subtleties, so it deserves its own section.

### Vertical Alignment

| Property | Meaning | Values |
|---|---|---|
| `vertical-align` | vertical alignment of an inline element | `baseline` (default), `top`, `middle`, `bottom`, `sub`, `super`, `text-top`, `text-bottom`, or a size value or % |

The `vertical-align` property specifies where an inline element should be aligned vertically, with respect to other content on the same line within its block element's box. The default value of `baseline` aligns the content vertically with the bottom or baseline of the non-hanging letters of inline text. (Hanging letters are ones such as g or j that have parts that protrude below the normal bottom line of the text.)

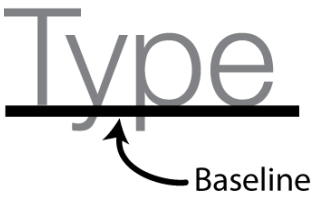
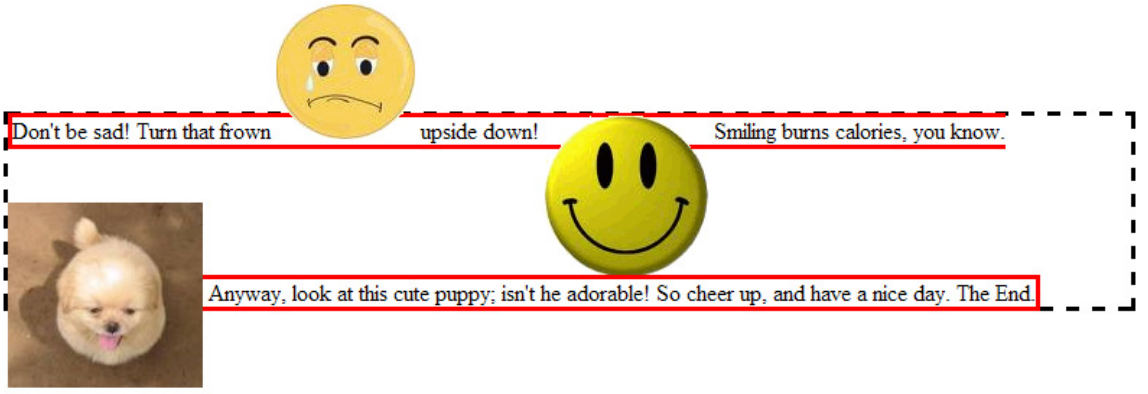



**Figure 4.11 Text baseline**

Example 4.28 shows several images with different vertical alignments and the way they look with respect to the neighboring text. The paragraph's inline content is wrapped into an overall `span` with a `class` of `inlinestyles`, so that we can apply a border to it to help you see the relative vertical alignment of the images.

```
<p>
  <span class="inlinestyles">
    Don't be sad!  Turn that frown
    <img src="images/sad.png" alt="sad" /> upside down!

    <img class="happy" src="images/smiley.png" alt="smile" />
    Smiling burns calories, you know.

    <img class="puppy" src="images/puppy.jpg" alt="puppy" />
    Anyway, look at this cute puppy; isn't he adorable!  So cheer up,
    and have a nice day.  The End.
  </span>
</p>
```

```
p {
  border: 3px solid black;
}
.happy {
  vertical-align: bottom;
}
.inlinestyles {
  border: 3px solid red;
  vertical-align: top;
}
.puppy {
  vertical-align: middle;
}
```




**Example 4.28 Vertical alignment**

**Common Error**
Space under image

There are cases where you'll want an image inside a block element that is exactly as tall as that image. (We see this a lot when we're placing images into tables. We'll discuss tables in a later chapter.) You'd think it would work to make the block element exactly the right size to fit the image, by setting its padding to 0em. But when you do so, there is a bit of space under the image, as shown in Example 4.29.

```
<p class="alert">
  <img src="images/smiley.png" alt="smile" />
</p>

.alert {
  background-color: red;
  margin: 0px;
  padding: 0px;
}
```

**Example 4.29 Common error: space under image**

There is red space under the image, despite padding and margin of 0. This is because the image is vertically aligned to the baseline of the paragraph, which isn't the same as the bottom. Setting `vertical-align` to `bottom` as shown in Example 4.30 fixes the problem because it causes the image to also occupy the previously empty baseline area.

```
<p class="alert">
  <img class="bottom" src="images/smiley.png" alt="smile" />
</p>

.alert {
  background-color: red;
  margin: 0px;
  padding: 0px;
}
.bottom {
  vertical-align: bottom;
}
```

**Example 4.30 Corrected error with space under image**

Notice that now the image is flush against both the top and bottom of the area containing it, with no red space showing underneath. Another fix for this problem is to set the `line-height` property to `0px`, which places the baseline into the same position as the bottom, but this is more hackish.

## 4.4.2 Positioning

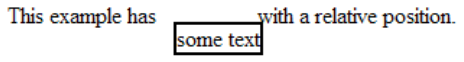
| Property | Meaning | Values |
|---|---|---|
| `position` | location of element on page | `static`: default position<br>`relative`: offset from its normal static position<br>`absolute`: at a particular offset within its containing element<br>`fixed`: at a fixed location within the browser window |
| `top`, `bottom`, `left`, `right` | offsets of element's edges | a size in `px`, `pt`, `em`, or `%` |

There are times when the standard flow of content on the page isn't ideal for presenting your page's information. For example, your page may have a sidebar of links that you want always to be visible even after the user scrolls down the page. Or you may have a pair of block elements that you want to appear next to each other horizontally. To achieve these kinds of effects, we can use the powerful CSS `position` property, which chooses from several models to position an element.

**relative positioning**
Setting the location of an element to an offset from its normal static position.

The default position is `static`, meaning to place the element within the normal document flow. But we can lift the element out of the normal flow in many different ways. Example 4.31 demonstrates *relative positioning*, which lets you shift an element's position slightly relative to its normal static position.

```
<p>
  This example has <span id="lifted">some text</span>
  with a relative position.
</p>

#lifted {
  position: relative;
  left: 0.5em;
  top: 1em;
  border; 2px solid black;
}
```




**Example 4.31 Relative positioning**

After setting the `position` to `relative` on the element, you can set its `top`, `bottom`, `left`, or `right` properties to specify the relative adjustments to those respective edges of the element's box.

### Absolute Positioning

**absolute positioning**
Setting the location of an element to an offset from the block element containing it.

Elements with a `position` of `absolute` are given an *absolute position* on the page and are removed from the normal flow. They are positioned at an offset relative to the block element containing them, assuming that block also uses `absolute` or `relative` positioning. (If it doesn't, the position is relative to the edges of the overall web page.)

Other elements on the page completely ignore the absolutely positioned element when laying themselves out. It's as though the element isn't even there, as far as their layout and positioning is concerned. Example 4.32 demonstrates absolute positioning, and Figure 4.12 shows the appearance.

```
<div id="area1">...</div>
<div id="area2">
  ...
  <div id="menubar">...</div>
</div>
<p>...</p>
```

```
#menubar {
  position: absolute;
  top: 20px;
  right: 40px;
  width: 100px;
}
```

**Example 4.32 Absolute positioning**

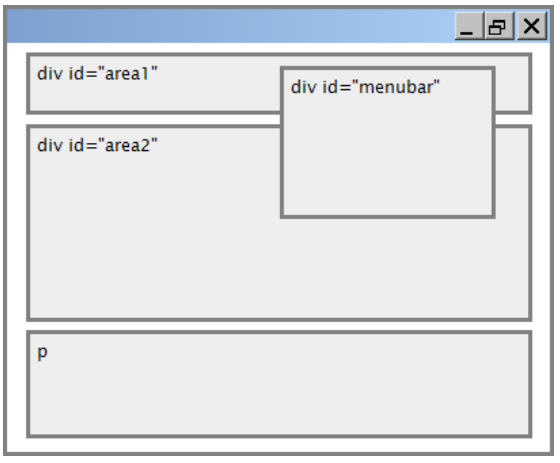



**Figure 4.12 Absolute positioning (output)**

As with relative positioning, the element's placement is determined by the values of its `top`, `bottom`, `left`, and `right` properties. You should generally specify a `width` property as well.

Generally the coordinates of an absolutely positioned element are relative to the entire page. If you want to make them relative to a particular element on the page instead, enclose your absolutely positioned element inside another element that uses absolute or relative positioning. A common idiom is to use an enclosing `div` with relative position, but no `left`, `top`, `right`, or `bottom` values. This will make the outer `div` stay at its normal static position, but also serve as a point of reference for any absolutely positioned elements inside it. Example 4.33 demonstrates this technique.

```
<div id="area1">...</div>
<div id="area2">
  <div id="menubar">...</div>
</div>
<p>...</p>
```

```
#area2 {   /* menubar will be relative to this div's position */
  position: relative;
}
#menubar {
  position: absolute;
  top: 20px;
  right: 40px;
  width: 100px;
}
```

**Example 4.33 Absolute positioning with relative containing element**

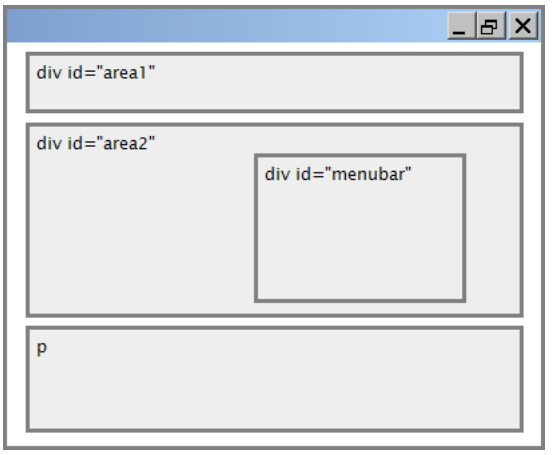



**Figure 4.13 Absolute positioning with relative containing element (output)**

If you use a percentage for your absolute position, you may not see the correct layout in some browsers unless you style the `body` to have a `width`/`height` of `100%`, as shown in Example 4.34.

```
body {
  width: 100%;
  height: 100%;
}
#menubar {
  position: absolute;
  top: 20px;
  right: 40px;
  width: 20%;
  height: 50%;
}
```

**Example 4.34 Body width and height styles**

An absolute position may sound a lot like floating an element. But a crucial difference is that, unlike absolutely positioned elements, floating ones still affect the flow of nearby inline content. A floating element is removed from the normal block flow of the document, and block elements lay themselves out underneath the floating element, ignoring it for layout purposes. But the inline content within those block elements does respect the floating element and wraps neatly around it. Absolutely positioned elements are ignored by other elements on the page during layout.

### Fixed Positioning

**fixed positioning**
Setting the location of an element to an absolute location on the browser screen.

Elements with a `position` value of `fixed` are given a *fixed position* on the screen and are removed from normal flow. This is much like absolute positioning, except that fixed elements are positioned relative to the overall browser window, not the page or their containing element. In other words, a `fixed` element will stay in exactly the same place on the screen, even if the user scrolls up and down the page. A fixed position is useful for content that you always want to be visible on the page, such as a sidebar of links or (shudder) an advertisement.

The CSS code in Example 4.35 is the same as in Example 4.32 seen previously, except with a fixed instead of absolute position. The screen diagram in Figure 4.14 shows the page after the user

has scrolled down. The `menubar` section of the page remains in its original place and does not scroll with the other page contents underneath it.

```
<div id="area1">...</div>
<div id="area2">
  <div id="menubar">...</div>
</div>
<p>...</p>
```

```
#menubar {
  position: fixed;
  top: 20px;
  right: 40px;
  width: 100px;
}
```

**Example 4.35 Fixed positioning**

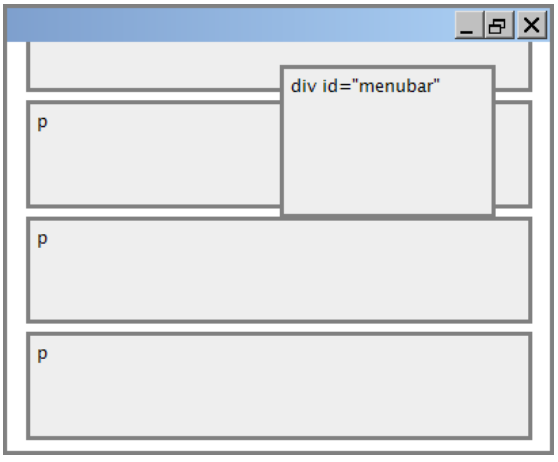



**Figure 4.14 Fixed positioning (output, after scrolling down the page)**

The left, right, top, or bottom value can be negative to cause an element to sit outside the visible browser window. This can create a nice "hanging" effect of an element that's partially off the page. Example 4.36 shows the necessary CSS and Figure 4.15 shows a diagram of the resulting appearance.

```
#menubar {
  position: fixed;
  top: 20px;
  left: -50px;
  width: 100px;
}
```

**Example 4.36 Fixed position with negative offset**


**Figure 4.15 Fixed position with negative offset (output)**

### 4.4.3 Z-indexing

| Property | Meaning | Values |
|---|---|---|
| `z-index` | element's 3-dimensional ordering | `auto` (default), or an integer |

The `z-index` property sets which `absolute` or `fixed` positioned element will appear on top of another that occupies the same space. An element with a greater `z-index` appears on top of any others on the same part of the screen.

### 4.4.4 Element Visibility

CSS has two properties named `display` and `visibility` that let you hide elements on a page. There are a few differences between the two properties, which we'll discuss in the following sections.

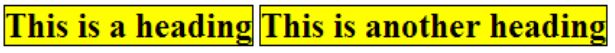
#### The `display` Property

| Property | Meaning | Values |
|---|---|---|
| `display` | whether an element is displayed as a block or inline element, or not at all | `block` (display as a block element),<br>`inline` (display as an inline element),<br>`none` (don't display this element),<br>... |
| `visibility` | whether an element's content can be seen on the page | `visible` (default), or `hidden` |

The `display` property sets the type of CSS box the browser should use to display a particular element. This is a powerful property that lets you "play God" a bit with your HTML. For example, you can set an `li` to display as an inline element even though the browser would normally display it as a block element. Example 4.37 demonstrates headings that display as inline elements.

```
<h2>This is a heading</h2>
<h2>This is another heading</h2>
```

```
h2 {
  background-color: yellow;
  border: 1px solid black;
  display: inline;
}
```




**Example 4.37 Displaying a block element as inline**

Setting the `display` property has no effect on what is/isn't valid XHTML. For example, even if you set a `span`'s `display` to `block`, the W3C XHTML validator will complain if the `span` is not placed inside a block element. Use `display` sparingly, because it can radically alter the page layout.

One common use of `display` is to make the contents of a list display horizontally rather than vertically. Most new web developers think of a `ul` as showing on the page as a bulleted list, with each item on its own line. But by setting the list items to display inline, the list flows horizontally with no bullet next to each item. Example 4.38 illustrates this technique.

```
<ul id="menubar">
  <li>News</li>
  <li>Links</li>
  <li>Members Only</li>
  <li>Join</li>
</ul>
```

```
li {
  display: inline;
  padding: 0.5em;
  border: 2px solid gray;
}
```


**Example 4.38 Displaying a list as inline**

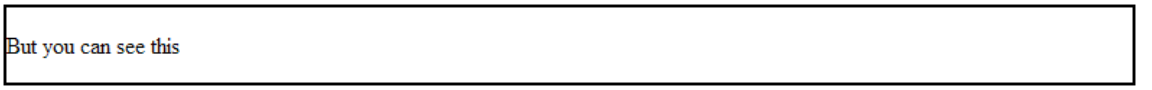
You might ask why we're using an unordered list in Example 4.38, rather than, say, a paragraph with four `span` elements inside it for the list items. But remember that we should choose our HTML tags based on the meaning of the content. The above section headings are a list, not a paragraph. Tagging them as list items is more semantically descriptive than simply labeling them as spans of text.

There are other values `display` can have (such as `compact` and `table-column-group`), but most of them are obscure values dealing with tables, which we'll cover later in this textbook.

An element whose `display` is set to `none` is not shown on the page at all. For example, the code in Example 4.39 produces no visible output in the browser. Right now it might not make sense to set `display` to `none`, rather than just deleting the element from the page's source code altogether or commenting it out. But in later chapters we'll see a common usage of `display` where we'll show and hide an element dynamically in response to user events using JavaScript.

```
<div>
  <p class="secret">No one will be able to see this! :-(</p>
  <p>But you can see this</p>
</div>
```

```
p.secret {
  display: none;
}
```




**Example 4.39 Non-displayed element**

### The `visibility` Property

The `visibility` property sets whether an element should be shown onscreen. The default visibility for an element is `visible`, but when an element's visibility is set to `hidden`, the element will not be shown on the page. The main difference between the `display` and `visibility` properties is the following:

- `display: none;` means the element does not occupy space on the page ("It's not there")
- `visibility: hidden;` means the element still occupies space on the page, but its contents are not drawn and it is invisible to the user ("It's there, but I can't see it")

Example 4.40 shows a hidden element. It's not very exciting, because you can't see the element. But notice that the area containing the hidden paragraph is large enough to fit the paragraph, unlike the previous example with `display` set to `none`, in which the outer area was smaller.

```
<div>
  <p class="secret">No one will be able to see this! :-(</p>
  <p>But you can see this</p>
</div>
```

```
p.secret {
  visibility: hidden;
}
```


**Example 4.40 Hidden element**

Later we'll use this property to show and hide dynamic HTML content on the page in response to user events in JavaScript. Sometimes a `visibility` of `hidden` is better than a `display` of `none` in such cases because the page layout already has allocated space for the hidden element, so the layout won't change when we instruct the element to appear.

## Self-Check

12. What CSS code would center a paragraph horizontally on the page, making it occupy half the page's width, and with the text right-aligned within the paragraph?
13. What CSS code would place a paragraph against the right edge of the page, making it occupy half the page's width, but with the text left-aligned?
14. Which of the following CSS properties have an effect, when set on an inline element? Choose all that apply.
    a) `margin`
    b) `margin-top`
    c) `margin-left`
    d) `width`
    e) `height`
    f) `padding`
    g) `padding-bottom`
    h) `padding-right`
15. What `vertical-align` value does each of the following images have?


I am fine

16. What is the difference between absolute and fixed positioning? Which is more appropriate for each of the following items you might want to put on your page?
    a) a bar of links that should be visible on the page at all times
    b) a left column of the page's layout showing various news and updates; the updates might be longer than the browser window is tall
    c) an image that you want to appear to the right of some text in a paragraph
    d) a pop-up ad that shows on top of the normal page content and can't be dodged by scrolling down the browser window
17. What is the difference between setting `display: none;` and `visibility: hidden;` on an element? How can you tell them apart by looking at the page?

## 4.5 Internet Explorer Layout Quirks (optional)

One of the sad realities of web programming is that Microsoft's Internet Explorer browser simply does not follow the rules. There are a large number of major bugs and incompatibilities in IE that can make us web coders want to tear our hair out. Microsoft has chosen simply not to follow web standards. In particular, IE does not implement the CSS box model the way it is supposed to be implemented. A page that looks one way in IE may look a completely different way in Firefox or Safari. A page that shows up in one browser may be blank or badly misformatted in another. This leaves web programmers with a tough decision: Which users do I want to be able to see my page correctly?

The worst part is, despite the fact that Microsoft's IE team continues to release new versions of their browser (version 8 is being readied as this book goes to press), they choose not to correct these problems. It's not clear whether this is because they want to retain compatibility with existing pages that use the old incorrect behaviors, or as a way of intentionally balkanizing the web and making it harder for the web as a platform to reduce the importance of Microsoft's Windows operating system. Either way, it can be a royal pain to deal with IE, as you'll see if you try to do so as a web dev.

Fortunately, if you are aware of some of IE's most major deficiencies and workarounds for them, you can often create a page that works in both IE and standards-compliant browsers. In this section, we list some of the more common quirks of IE and suggest ways to deal with them. The section is intentionally kept short, because we generally object to Microsoft's decisions here and want to focus on and follow web standards as much as possible.

### The Broken Box Model

Various versions of Internet Explorer misinterpret the `width` and `height` properties. It considers them to include the padding and border around the element, which it shouldn't. This is disastrous for web developers who want a consistent appearance to their pages, because the width you set for a standards-compliant browser simply won't match that seen in IE, unless your elements have no padding. The element will appear too thin in IE, since the same number of pixels now have to accommodate the padding and border. Thanks a lot, IE development team.

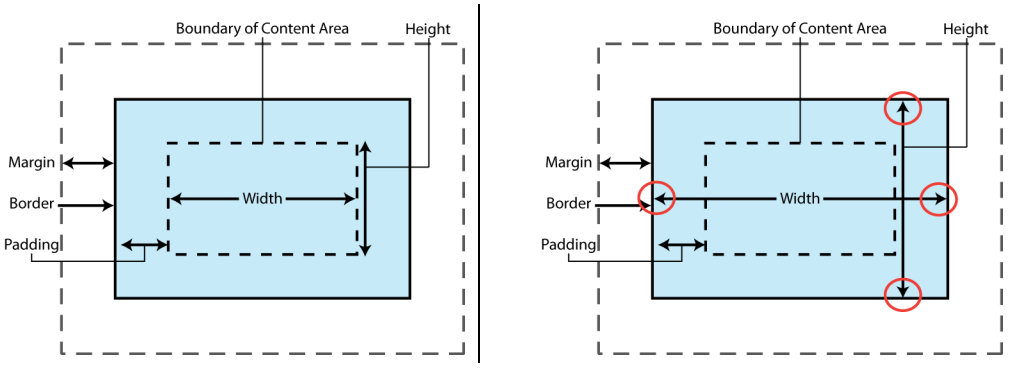



**Figure 4.16 Box model, standard-compliant (left) and IE (right)**

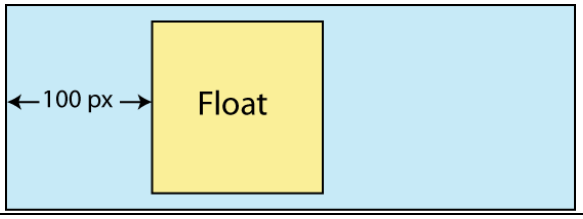
To make matters worse, Internet Explorer 6 also doesn't support the `min-width`, `min-height`, `max-width`, or `max-height` properties. What a pain! Many web developers have come up with hacks to try to emulate the effect of properties like `min-width` in IE, such as the following sites:

- http://www.cssplay.co.uk/boxes/minwidth.html
- http://www.webreference.com/programming/min-width/

### The Broken Float Model

If you float an element inside another block element and want to space it from the edge a bit, you can set a margin on the appropriate side. For example, the code in Example 4.41 is supposed to space a `div` 100px from the page's left edge. It works properly in standards-compliant browsers.

```
div#floating {
  float: left;
  margin-left: 100px;
}
```




**Example 4.41 Float with left margin**

In Internet Explorer 6 when an element is floated left, its left margin is mistakenly doubled by the browser. In this case our floating element actually appears 200px from the left edge.

This is not the only bug related to floating elements; there is a lengthy list of such bugs. For example, when you float elements, often IE6 will mistakenly indent the first line of nearby text with the same amount of indentation as the floating element's `margin-left` setting.

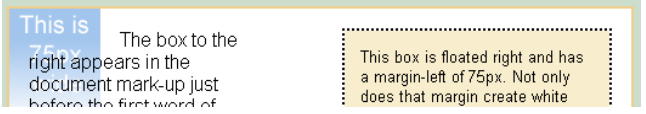



**Figure 4.17 IE6 float text indentation bug**

A suggested workaround for IE is to set the floated element's `display` to `inline`, which does mysteriously fix the bug. But this is simply a hack and shouldn't make such a drastic change to the page's layout.

```
div#floating {
  display: inline;
  float: left;
  margin-left: 100px;
}
```

**Example 4.42 Display inline hack for float bug**

Here's another float bug. If you float an element to the left and then position other block content underneath it, the block element is supposed to lay itself out normally. But in IE, if the underlying block element has a `width` declaration, it will also be mysteriously turned into a floating element and placed to the side of the floating one. This bug still exists in IE7 (it is fixed in IE8), so the majority of users of the web potentially face this issue. IE also often adds 3px or so of margin between floating elements and other elements, even when they are supposed to touch flush against each other. On and on. The following site has a floating elements test that IE7 and below fail miserably:

- http://css-class.com/articles/explorer/floats/floatandcleartest1.htm

As you can see, getting floating layouts to render correctly in IE can be a grueling experience.

## 4.5.1 Workarounds for IE Flaws

A lot of times you can make a page look right in Internet Explorer if you can apply different CSS rules to it in IE than you do to other browsers. The idea is to supply one set of CSS rules to IE that account for its misbehavior, and another set to everybody else who follows the rules.

### The "Underscore Hack"

Over the years a bunch of CSS "hacks" have been found that can be used to target IE exclusively. For example, recall how IE's box model deals with widths and heights improperly. Some crafty CSS developer discovered that IE will pay attention to CSS rules even if they have certain bad characters in them, such as underscores. For example, if you try to set a property named `_width`, standards-compliant browsers will completely ignore it, but IE will happily set the element's width. People use this to set IE to use a larger width to give space for the element's horizontal padding and borders, as in the code shown in Example 4.43.

```
div {
  width: 100px;
  padding: 10px;
  border: 10px solid black;
  _width: 140px;
}
```

**Example 4.43 IE "underscore hack" (not recommended!)**

There are other equally objectionable hacks, such as calling it `w\idth` or using a bogus context selector that only IE will recognize, such as `* html div { ... }`. We strongly discourage against using hacks like these. Many of them will break your CSS file so that it does not pass the W3C validator. IE support isn't worth that.

Fortunately there's a better way to get this same effect of providing a special set of style rules to IE. This can be achieved by a special IE-only proprietary HTML feature called conditional comments. A *conditional comment* is a comment that can contain an `if` statement in it, causing a piece of HTML to be included only if some condition is met. The syntax for conditional comments is:

```
<!--[if condition]>
  HTML code
<![endif]-->
```

The most common thing to write under condition is simply the letters `IE`, meaning, "if this browser is any version of IE." All other browsers will ignore this text and treat it as a large HTML comment. Another variation of a condition checks for a specific IE version, such as `lte IE 6`, since different versions of IE have had different bugs and require different fixes.

Example 4.44 demonstrates Internet Explorer's conditional comments. The code uses them to link to a file ie_hacks.css on all versions of IE, and to an additional file ie6_more_hacks.css if the browser is specifically IE6. The ie_hacks.css file contains the 140px width to correct for IE's broken box model, so that the actual content width of the element will be the desired 100px to match the standard-compliant browsers.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <!--[if IE]>
      <link href="ie_hacks.css" type="text/css" rel="stylesheet" />
    <![endif]-->

    <!--[if lte IE 6]>
      <link href="ie6_more_hacks.css" type="text/css" rel="stylesheet" />
    <![endif]-->
    ...
```

```
/* ie_hacks.css file contents */
div {
  width: 140px;
}
```

**Example 4.44 Internet Explorer conditional comments**

## 4.6 Case Study: Ultimate Frisbee

At the start of this chapter, we posed the problem of laying out an ultimate Frisbee site. Recall the appearance we wanted it to have, as shown in Figure 4.1. Now, with our new knowledge of CSS ids and classes, `div` and `span`, the box model, and layout, we can achieve the desired layout.

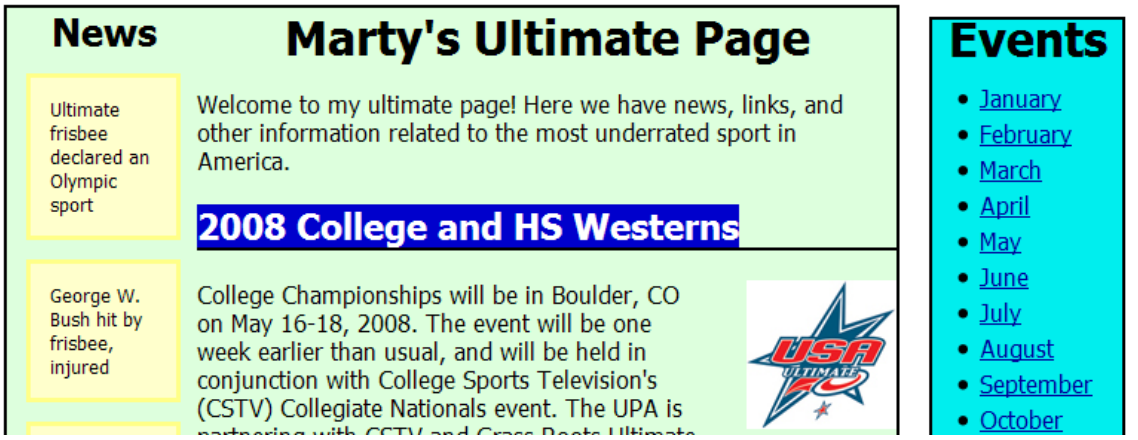



**Figure 4.18 Ultimate frisbee page, desired appearance**

Let's assume that we're starting out with the HTML code shown in Example 4.45 (abbreviated), and that we want to change the HTML code as little as possible in our process of styling the page.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Ultimate Frisbee</title></head>
  <body>
    <h1>News</h1>
    <ul>
      <li>Ultimate frisbee declared an Olympic sport</li>
      ...
    </ul>

    <h1>Marty's Ultimate Page</h1>
    <p>Welcome to my ultimate page! ...</p>

    <h2>2008 College and HS Westerns</h2>
    <p><img src="ultimate.jpg" alt="ultimate" />
      College Championships will be in Boulder, CO ...</p>
    ...
    <h1>Events</h1>
    <ul>
      <li><a href="">January</a></li>
      <li><a href="">February</a></li> ...
    </ul>
  </body>
</html>
```

**Example 4.45 Initial HTML code for Ultimate Frisbee page**

Let's come up with a strategy to approach this problem step-by step. The following order of operations should help us create the desired page appearance:

1. Look at the desired appearance picture and decide what are the major sections of the page. Change the HTML code to semantically represent these sections as `div` elements.
2. Link the page to a CSS file that we'll create incrementally.
3. Add general styles not related to tricky layout, such as fonts, colors, and alignment.
4. Tackle the layout for each section, starting with the Events calendar, then the News section and central announcements.

## 4.6.1 Page Sections and General Styles

First let's denote the major sections of the page and wrap them in `div`s as appropriate. The news at left and the calendar at right seem to be major page sections, so they each deserve a `div`. Let's also place a `div` on the central section containing the announcements. We'll give each one an `id` for styling, as shown in Example 4.46.

```
<div id="news">
  <h1>News</h1>
  <ul><li>Ultimate frisbee declared an Olympic sport</li> ...
  </ul>
</div>

<div id="announcements">
  <h1>Marty's Ultimate Page</h1> ...
</div>

<div id="calendar">
  <h1>Events</h1>
  <ul><li><a href="">January</a></li> ...
  </ul>
</div>
```

**Example 4.46 Ultimate frisbee page sections**

Now let's take a stab at a few initial styles, without worrying about the complex layout of the page just yet. We'll edit the HTML page's header to include a link to our CSS file:

```
<head>
  <link href="ultimate.css" type="text/css" rel="stylesheet" />
```

### General Styles

The overall page body should use a 12pt Tahoma sans-serif font. The central announcements section has a green background and a black border. The level-1 headings on the page should be centered and have reduced margins. Also note how the level-2 headings have a blue background color and white text, and their first word is in a large font. Earlier in this chapter we discussed how to achieve such styles by wrapping text in `span`s. All these initial styles are shown in Example 4.47.

```
body {
  font: 12pt "Tahoma", "Arial", sans-serif;
}
.announcement {  /* text inside h2 headings */
  background-color: #0000cc;
  color: white;
}
#announcements {
  background-color: #ddffdd;
  border: 2px solid black;
}
h1 {
  margin: 0em 0em 0.5em 0em;
  text-align: center;
}
```

**Example 4.47 Initial page styles**

### Image, Heading, and List Styles using Context Selectors

Some of the announcements have images associated with them, which should hover on the right side of the `main` pane. Rather than styling each of those images with a CSS class, we can use a context selector to float all `img` elements within the `announcements` area. (We'll set a `clear` property on the `h2` elements for each announcement, so that one announcement's image never hangs down into the next announcement.) Example 4.48 shows the related CSS styles.

```
#announcements img {
  float: right;
  margin-left: 2em;
}

#announcements h2 {
  border-bottom: 2px solid black;
  clear: right;
}
```

**Example 4.48 Styles for headings and floating images**

The calendar and news sections have several non-layout styles that we can apply now. The news area should have a small font. Also, you'll notice that in our page the list of news items is a bulleted list, while in the expected page appearance there are no bullets and boxes around each news item. This might make you think that `ul` is the wrong tag for this list of items, but that would be faulty logic. We don't choose our HTML tags based on how content looks; we choose them based on the meaning and semantics of the content. This is still a list of items, so `ul` is the right tag. We'll just remove the bullets by setting a `list-style-type` property on the list in the News area. We don't need to give an `id` to the list to achieve this; we can do it with a CSS context selector that matches any `ul` inside the `#news` area. The list items themselves (`li`) inside this list can be given borders and background colors to make them match the desired appearance. Example 4.50 shows these styles.

```
#calendar {
  background-color: #00eeee;
  border: 2px solid black;
  line-height: 1.5em;
}
#news { font-size: smaller; }
#news > ul {
  list-style-type: none;
  padding-left: 0em;
}
#news li {
  background-color: #ffffcc;
  border: 3px solid #ffffc88;
  margin-bottom: 1em;
  padding: 1em;
}
```

**Example 4.49 News and calendar area styles**

After all the preceding styles have been applied, the page has the appearance shown in Figure 4.19 as a split screenshot to show part of the top and bottom of the page.

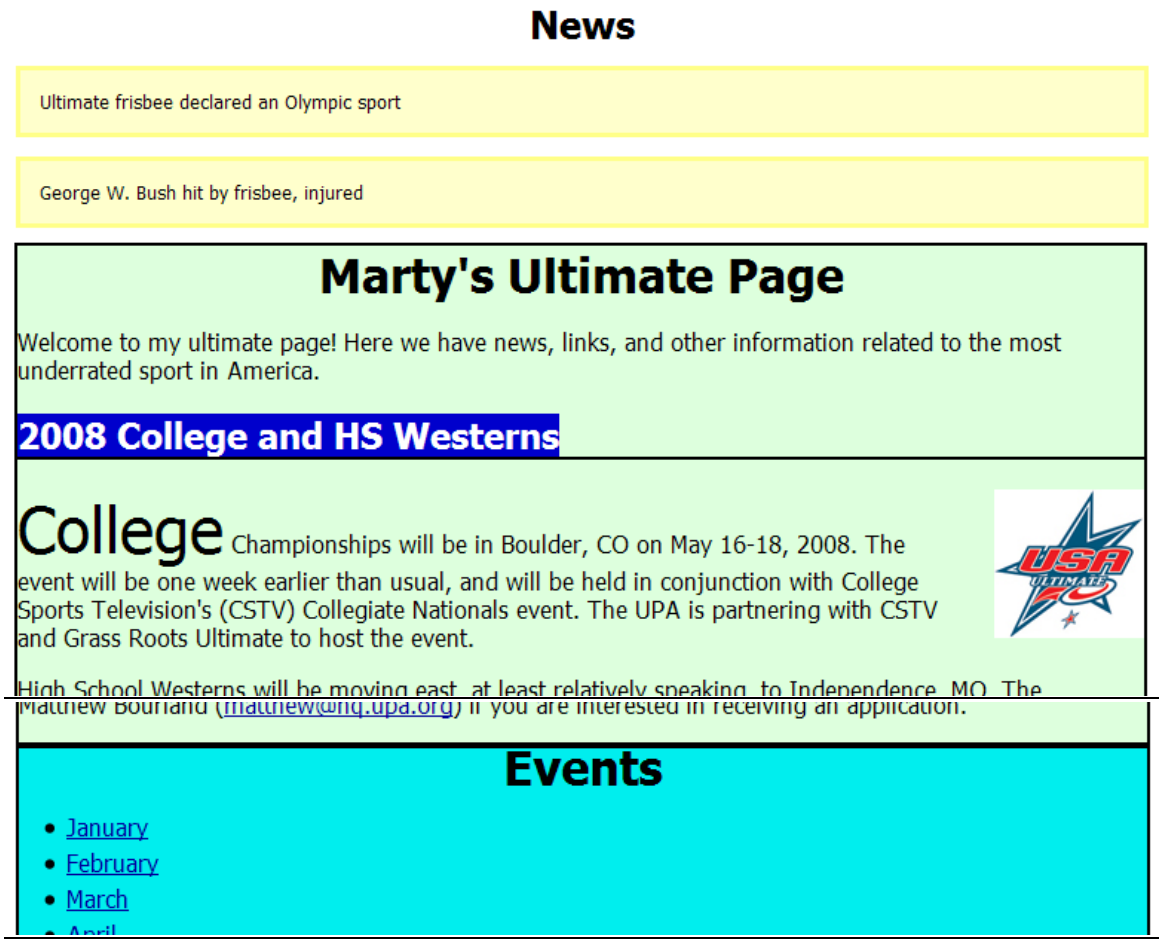



**Figure 4.19 Ultimate Frisbee page appearance, take 1**

### 4.6.2 Page Layout

Now let's do a bit of layout work. The calendar at right should remain in the same position on the page even after the user scrolls down. To do this, we'll give the calendar area a `fixed` position near the top-right edge of the page. We'll set its width to be 8em, just right for the width of its text.

```
#calendar {
  background-color: #00eeee;
  border: 2px solid black;
  line-height: 1.5em;
  position: fixed;
  right: 1em;
  top: 1em;
  width: 8em;
}
```

**Example 4.50 Calendar area styles**

After these new styles have been applied to the page, it has the appearance shown in Figure 4.20. It still has several glaring flaws. For one, the announcements area is too wide and reaches underneath the fixed Events calendar. Second, the News section is still at the top of the page, rather than on the left where it should be. We'll need to create additional styles to fix these problems.


**Figure 4.20 Ultimate Frisbee page appearance, take 2**

First let's fix the announcements area so it doesn't reach underneath the Events calendar. Since the calendar area is 8em wide and 1em from the right edge of the page, we can set a right margin on our announcements area of 10em, which will keep it from colliding.

```
#announcements {
  margin-right: 10em;
}
```

The news bar at left is a little trickier. It should move as the page scrolls, and it should also be contained inside the central announcements area, with the announcement text wrapping around it. This is an ideal case to use a floating layout.

```
#news {
  float: left;
}
```

However, if the `float` style is the only one we set, the appearance still isn't quite right. We get the appearance shown in Figure 4.21, with the News items awkwardly injected into the main announcements area. This is why it's so important to always set a width on any floating content, to keep it from widening and interfering with other contents on the page.


**Figure 4.21 Ultimate Frisbee page appearance, take 3**

### News and Announcement Section Layout

Let's apply some additional styles to the news section to clean it up. It should have a width of 8em, like the event calendar. Let's also place a 1em margin around it on all sides to distance it from neighboring content.

```
#news {
  float: left;
  font-size: smaller;
  margin-left: 1em;
  width: 8em;
}
```

After these changes, we have the following layout, shown in Figure 4.22 in two sections so you can see the top and bottom of the news area. The appearance is pretty close to what we want, but the text after the News area moves over too far to the left. We want it to stay over to the right of the news section all the way down the page. Also there isn't any spacing between the News section and the announcement text, and the black borders behind the level-2 headings extend left under News.


**Figure 4.22 Ultimate Frisbee page appearance, take 4**

Remembering the right margin we used to separate the announcements section from the events section, we could try setting a similar left margin on the announcements section to distance it from the floating news area. This helps, but it causes the green background and border to move away from the News section, making it differ from the desired appearance, as shown in Figure 4.23.

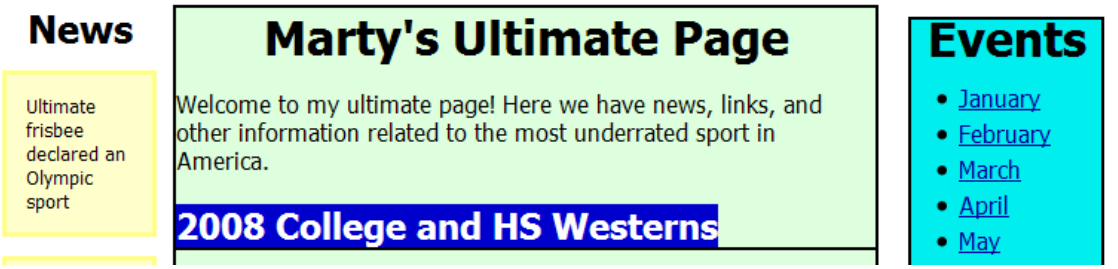



**Figure 4.23 Ultimate Frisbee page appearance, take 5**

We want to put a left margin on the announcements area, but not the overall green box and border. How can we do this? The solution is to add an additional outer `div` layer to represent the green background and black border, and place both the News and Announcements sections inside it. Then we can place a margin on the announcements section without moving the border or background.

```
<body>
  <div id="main">
    <div id="news">
      <h1>News</h1> ...
    </div>

    <div id="announcements">
      <h1>Marty's Ultimate Page</h1> ...
    </div>
  </div>

  <div id="calendar">...</div>
</body>
```

**Example 4.51 Adding `main` section to page for styling**

The styles applied to the main section and announcements section must be chosen carefully. We want the overall background and border on the new `main` section. We want the left margin on the `announcements` section inside `main`, so the margin doesn't move the border. But the right margin, the one to distance the central content from the calendar at right, *must* be on the main section and not the announcements section. If it's placed on the announcements, the green background and border will extend to the right underneath the calendar. Example 4.52 shows the new CSS styles. After adding these final styles, we have the desired appearance from our original screenshot.

```
#main {
  background-color: #ddffdd;
  border: 2px solid black;
  margin-right: 10em;
}
#announcements { margin-left: 8em; }
```

**Example 4.52 New CSS styles for main page section**

### 4.6.3 Final File Contents

After all of the changes and code in the previous sections, we end up with final files ultimate.html and ultimate.css, shown in Example 4.53 and Example 4.54 respectively.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <!-- Chapter 4 case study: XHTML code for Ultimate frisbee page -->
  <head>
    <title>Ultimate Frisbee</title>
    <link href="ultimate.css" type="text/css" rel="stylesheet" />
  </head>

  <body>
    <div id="main">
      <div id="news">
        <h1>News</h1>
        <ul>
          <li>Ultimate frisbee declared an Olympic sport</li> ...
        </ul>
      </div>

      <div id="announcements">
        <h1>Marty's Ultimate Page</h1>
        <p>Welcome to my ultimate page!  Here we have news, ...</p>

        <h2><span class="announcement">2008 College and
            HS Westerns</span></h2>

        <p><img src="ultimate.jpg" alt="ultimate" />
          <span class="firstword">College</span> Championships
          will be in Boulder, CO...</p>

        <h2><span class="announcement">Annual Members Meeting</span></h2>

        <p><span class="firstword">The</span> Annual Members ...</p>
      </div>
    </div>

    <div id="calendar">
      <h1>Events</h1>
      <ul>
        <li><a href="">January</a></li>
        <li><a href="">February</a></li>
        ...
        <li><a href="">December</a></li>
      </ul>
    </div>
  </body>
</html>
```

**Example 4.53 Ultimate Frisbee page code `ultimate.html`**

```
/* Chapter 4 case study: CSS styles for Ultimate frisbee page */
body { font: 12pt "Tahoma", "Arial", sans-serif; }
.announcement {
  background-color: #0000cc;   color: white;
}
#announcements {                    /* text inside h2 headings */
  margin-left: 8em;
  padding: 0em 1em;
}
#announcements h2 {
  border-bottom: 2px solid black;
  clear: right;
}
#announcements img {
  float: right;    margin-left: 2em;
}
#calendar {
  background-color: #00eeee;
  border: 2px solid black;
  line-height: 1.5em;
  position: fixed;
  right: 1em;
  top: 1em;
  width: 8em;
}
#calendar ul { padding-left: 2em; }
.firstword   { font-size: 32pt; }   /* first word of h2 headings */
h1 {
  margin: 0em 0em 0.5em 0em;
  text-align: center;
}
#main {
  background-color: #ddffdd;
  border: 2px solid black;
  margin-right: 10em;
}
#news {
  float: left;
  font-size: smaller;
  margin-left: 1em;
  width: 8em;
}
#news ul {
  list-style-type: none;
  padding-left: 0em;
}
#news li {
  background-color: #ffffcc;
  border: 3px solid #ffff88;
  margin-bottom: 1em;
  padding: 1em;
}
```

**Example 4.54 Ultimate Frisbee page styles `ultimate.css`**

## Chapter Summary

- Block elements in an HTML document are laid out in a standard flow from top to bottom. Inline elements are laid out within block elements from left to right, top to bottom.
- The CSS Box Model describes the regions HTML elements occupy. An element's box has an optional border, with padding inside the border and margins outside it.
- Padding and margins are specified as a size; borders have a thickness, color, and style.
- Elements can be given a width and height. Content's horizontal and vertical position can be affected by setting `auto` margins or changing the `vertical-align` property, respectively.
- Elements can be given custom positions by setting their `position` property. An `absolute` position specifies a particular offset within the surrounding region. A `fixed` position specifies an offset within the browser window.
- The `display` and `visibility` properties set whether an element is displayed as a block element, inline element, or not at all.
- Floating elements are lifted out from the normal document flow and pushed over to the left or right corner of the page.

## References

CSS Specifications and References:

- W3C CSS2 Specification: http://www.w3.org/TR/REC-CSS2/
- W3C Schools CSS2 Reference http://www.w3schools.com/css/css_reference.asp
- W3Schools CSS Tutorial: http://www.w3schools.com/Css/default.asp
- EchoEcho.Com Tutorial: http://www.echoecho.com/css.htm
- CSS by Quirksmode: http://www.quirksmode.org/css/contents.html
- CSS Specificity Wars:
  - http://www.stuffandnonsense.co.uk/archives/css_specificity_wars.html
- CSS Float Theory: Things You Should Know by Smashing Magazine:
  - http://www.smashingmagazine.com/2007/05/01/css-float-theory-things-you-should-know/
- CSS Positioning by Relatively Absolute
  - http://www.autisticcuckoo.net/archive.php?id=2004/12/07/relatively-absolute
- CSS Positioning in 10 Steps
  - http://www.barelyfitz.com/screencast/html-training/css/positioning/
- Web Design from Scratch: Block and Inline elements
  - http://www.webdesignfromscratch.com/css-block-and-inline.cfm
- CSS Zen Garden: http://www.csszengarden.com/
- Most Useful 50 CSS Tips And Tools For Webmasters by Emma Alvarez
  - http://www.emmaalvarez.com/2008/04/most-useful-50-css-tips-and-tools-for.html
- Internet Explorer Layout Hacks and Fixes:
  - http://www.tdrake.net/ie7-hacks/
  - http://www.positioniseverything.net/ie-primer.html
  - http://www.satzansatz.de/cssd/onhavinglayout.html
- Should I Use Tables for Layout?: http://shouldiusetablesforlayout.com/