Floating and Clearing

Another powerful technique you can use to organize your page's layout involves floating and clearing, using the `float` and `clear` properties. Floating an element is a way of moving it out of the normal flow of the document, and is a technique that you use not only for its intended purpose of flowing text around images, but also for creating columns and making block-level elements sit side-by-side. Elements that follow a floated element will move up next to the floated element if there is room for them to do so.

The `clear` property enables you to stop such elements from moving up next to a floated element. If, for example, you have two paragraphs and only want the first to sit next to the floated element, even though both would fit, you can "clear" the second one so it's positioned under the floated element.

Before I cover both these properties in detail, I'll mention that while floating elements is an essential and very useful CSS technique, floating can cause all kinds of headaches for CSS newbies. This is because floated content is not directly in the document flow, and so affects the layout of elements that enclose or follow it in the markup. I have carefully designed the following floating and clearing examples to give you the knowledge that you need to be successful when working with floats.

The Float Property

The `float` property is primarily used to flow text around images, but it also provides the easiest way to create multi-column layouts.

Let's start with an example of how to flow text around an image.

The CSS3 Columns Module specifies how columns can be defined using CSS, but at the time of writing, only Opera and IE10 support them, so for the foreseeable future, floats are the best way to create columns.

Flowing Text Around an Image

For the floating effect to work, the markup should state the floated image first, followed by the text that will wrap around it.

```
<img ..... />

<p>…the paragraph text…</p>
```

Here's the CSS.

```
font specs omitted for brevity ⟶ p {margin:0; border:1px solid red;}

the margin prevents the text from touching the image ⟶ img {float:left; margin:0 4px 4px 0;}
```

This CSS floats the image to the left, so that the text wraps around it to its right (**Figure 3.16**).

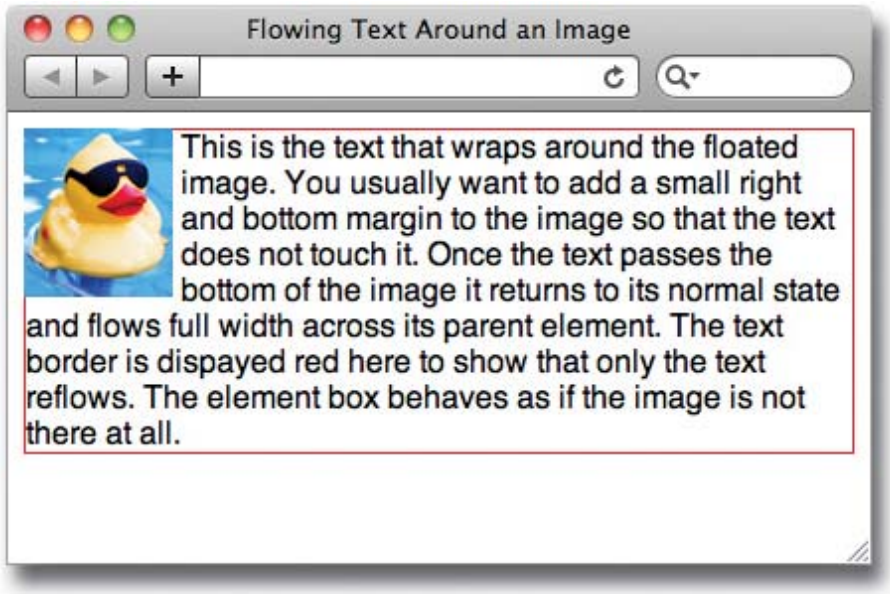


Figure 3.16. A floated image is removed from the document flow. If a text element follows it in the markup, that element's text will wrap around it.

In short, when you float an image, or any element, you are asking it to be pushed, as far as is possible, up and to one side of its parent element; in this

example, that parent element is `body`. In the example here, the paragraph (with a red border) no longer sees the floated element as preceding it in the document flow, so it also takes the top left corner position of the parent; however, its content, the text, wraps itself around the floated image.

When you float an element, you must also set its width, or some unpredictable results can occur. However, images implicitly have width and so don't need to have a width assigned to them when floated.

From here, it's a simple step to use `float` to form columns (**Figure 3.17**). I simply set a width on the paragraph and float it, too.

```
p {float:left; margin:0; width:200px; border:1px solid red;}

img {float:left; margin:0 4px 4px 0;}
```

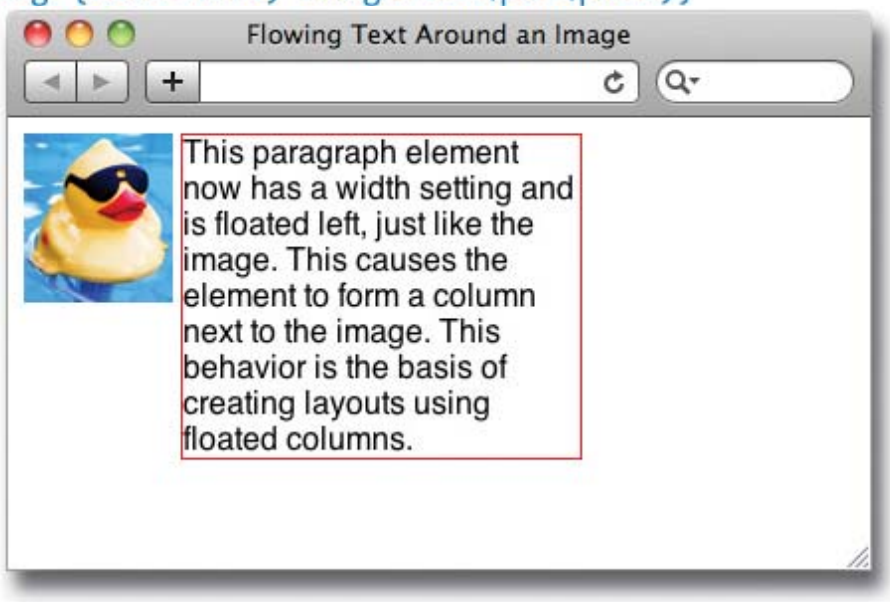


Figure 3.17. When the fixed width paragraph is floated next to the floated image, it forms a column and no longer wraps the image.

When you float both the image and the "widthed" paragraph like this, the text-wrapping effect stops, and the paragraph also tries to move up and as far to the left as possible also, and in this way forms a column next to the image. This is the principle of creating multi-column layouts using `float`. If a set of sibling elements have their widths set, are floated, and there is room for them to do so, they line up next to one another.

If you create three floated, fixed-width elements, they will sit next to each other in this way, giving you a layout of three columns that act as containers into which you can put other elements. I will demonstrate floated layouts in depth in Chapter 5.

Let's look at another gotta-understand aspect of floats—floated elements are "out-of-the-flow" and therefore not enclosed by their parent element: This can have a disruptive effect on your layout.

Three Ways to Enclose Floated Elements

Because a floated element is not directly in the document flow, its parent element doesn't see it and so doesn't enclose it. Because this behavior is not always desirable, I teach you three ways to force elements to enclose their floated children. You need to know all three, so you can use the one that is best for a given situation.

To illustrate this float behavior, what it can do to your layout, and the three ways you can fix it, let's start with an image and its text caption enclosed inside a `section` tag. `section` is followed by `footer` to represent, in this example, the full width footer that is commonly found across the bottom of Web pages.

```
<section>
  <img src="images/rubber_duck2.jpg">
  <p>It's fun to float.</p>
</section>
<footer> Here is the footer element that runs across the
bottom of the page.</footer>
```

So that you can clearly see what is happening, I'll display the element boxes of `section` and `footer`, as shown in **Figure 3.18**.

```
section {border:1px solid blue; margin:0 0 1
p {margin 0;}
footer {border:1px solid red;}
```

removes large top and bottom margins

font specs omitted for brevity

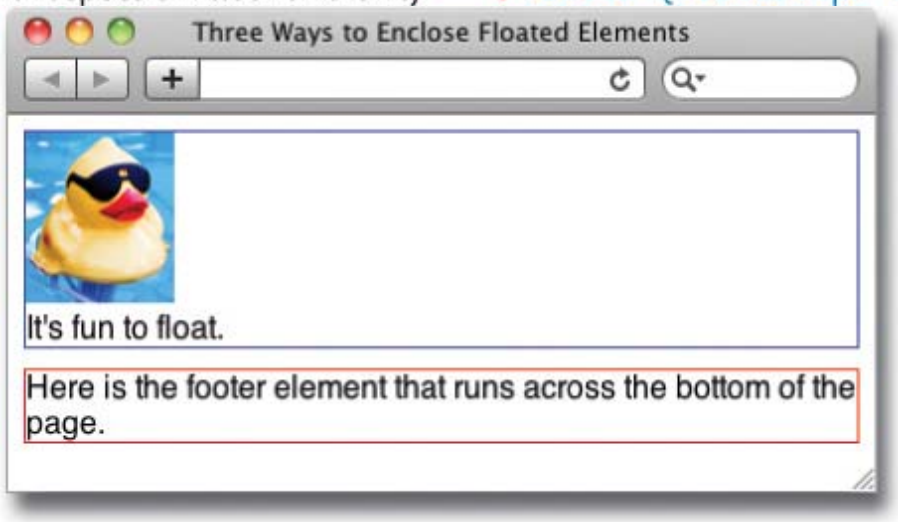


Figure 3.18. Two block-level elements: `section`, containing an image and a caption, and `footer`, stack one below the next in the normal document flow.

What you are seeing here is normal document flow; block-level elements enclose any child elements and stack one below the next down the page. Let's say you want the caption to sit to the right of the image, not below it. As you saw in the previous exercise, the easiest way to do this is to float the image. Let's try that.

```
section {border:1px solid blue; margin:0 0 10px 0;}
img {float:left;}
footer {border:1px solid red;}
```

Figure 3.19 shows the outcome.

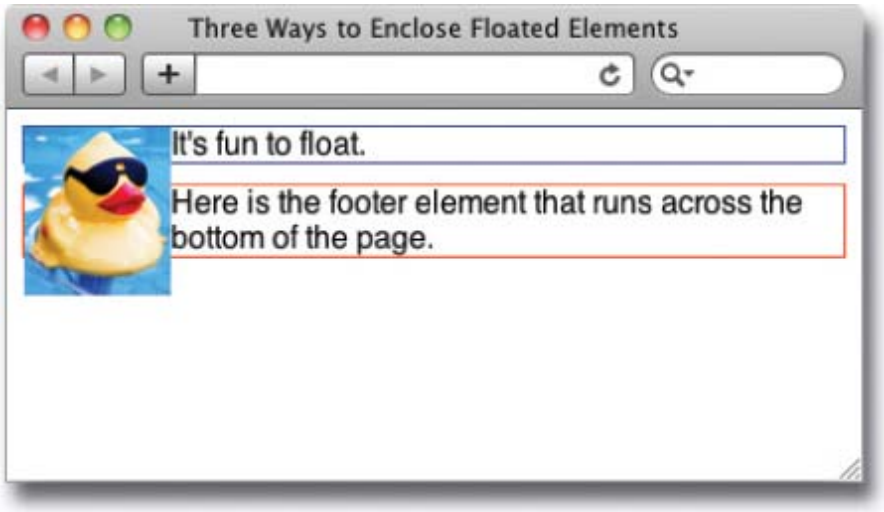


Figure 3.19. When the image is floated to move the caption next to the image, the parent `section` collapses to the height of the non-floated text element.

Oops! The caption is now next to the image as I intended, but `section` no longer encloses the floated element, but the non-floated text element. `footer` then moves up, right under the preceding block-level element, `section`, just like it's supposed to. The result, however, is not what I wanted.

Method 1—Add Overflow:Hidden to the Parent Element

There is a simple, if unintuitive, solution to make `section` enclose the floated element: apply `overflow:hidden` to the enclosing element to force it to enclose the floated content.

```
section {border:1px solid blue; margin:0 0 10px 0;
overflow:hidden;}
img {float:left;}
p {border:1px solid red;}
```

Once the `overflow:hidden` declaration is applied to the container, the footer returns to the desired position (**Figure 3.20**).

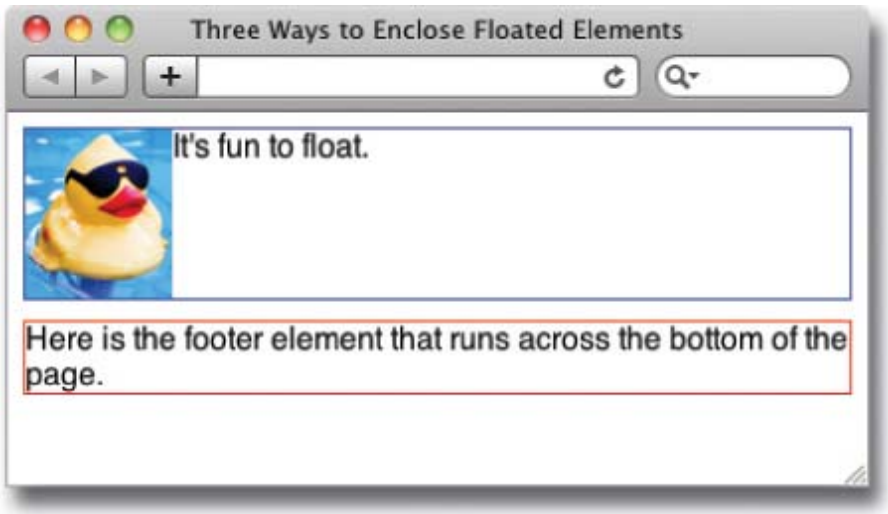


Figure 3.20. With the `overflow:hidden` declaration applied to the container, it now encloses its floated content.

Method 2—Float the Parent, Too!

The second way that you can force a parent element to enclose its floated children is by floating the parent element, too.

```
section {border:1px solid blue; float:left; width:100%;}
img {float:left;}
footer {border:1px solid red; clear:left;}
```

The true purpose of `overflow:hidden` *is to prevent oversize content, such as a large image, from forcing the containing element to expand to show the oversized content. Instead, with* `overflow:hidden` *applied, the container remains at its defined size and the oversize child content is simply cropped by the container. However,* `overflow:hidden` *also reliably serves this useful second role of forcing elements to enclose their floated content.*

When floated, `section` tightly encloses (aka shrink-wraps) its child elements, whether they are floated or not. So, we now need to add `width:100%` to make `section` full width again. Also, because `section` is now floated, `footer` will try to move up next to it, so we need to force `footer` to sit under section by adding `clear:left` to it: A cleared element cannot move up next to a floated element. This code has the same outcome as shown in **Figure 3.20**.

Method 3—Add a Non-Floated Clearing Element

The third way that you can force a parent element to enclose its floated children is by adding a non-floated element as the last child element and clearing it. Because a containing element always encloses non-floated elements, the containing element encloses this element, and the preceding floated elements. There are two ways to add a clearing element as the last child element.

The first, though not ideal, way to do this is simply to add an HTML element directly into the markup as the last child element and apply the CSS property `clear` to it; a `div` is best for this purpose as it has no default styling, and so does not introduce extra space into the layout.

```
<section>
  <img src="images/rubber_duck.jpg">
  <p>It's fun to float.</p>
  <div class="clear_me"></div>
</section>
<footer> Here is the footer element…</footer>
```

Here I add a class to the `div` so I can clear it in the CSS

```
section {border:1px solid blue;}
img {float:left;}
.clear_me {clear:left;}
footer {border:1px solid red;}
```

The floated elements are now enclosed, as shown in **Figure 3.20**. If you'd rather avoid adding purely presentational elements like this, here's how to add a clearing element using only CSS. I first add a class to `section`

```
<section class="clearfix">
  <img src="images/rubber_duck.jpg">
  <p>It's fun to float.</p>
</section>
<footer> Here is the footer element…</footer>
```

This `clearfix` *code, devised by programmer Tony Aslett, adds a cleared, non-floated element that contains just a period (you have to have some content, and a period is the smallest content available). Some additional declarations ensure that this pseudo-element takes up no height and is not visible on the page.*

---

and use the magical `clearfix` CSS code!

```
.clearfix:after {
  content:".";
  display:block;
  height:0;
visibility:hidden;
clear:both;
}
```

The value of `both` *on the* `clear` *property means that* `section` *clears (sits below) elements floated both left and right. I could have used the value* `left` *in this case, but by using* `both` *, if I switch the float on the images to* `right` *later,* `clear` *still works.*

Again, the floated elements are enclosed as shown in **Figure 3.20**, but this time without an extra element being hard-coded into the markup. You can temporarily remove the height and visibility declarations in the `clearfix` code above to see the period that it adds into the markup.

I use the `clearfix` CSS to solve float issues like this in just about every site I create, because floating is the only reliable way (until more browsers support the CSS3 Columns Module recommendations) to create columns. I'll explain exactly why in Chapter 5.

To conclude this section of the chapter, there are three ways to force parent elements to enclose their floated children.

• Apply `overflow:hidden` to the parent.

• Float the parent element.

• Add a non-floated element as the last item within the parent, either by coding that element into the markup, or inserting it adding the `clearfix` class to the parent. (Of course, you need the related `clearfix` CSS in your style sheet.)

Which one of these three you use depends on the circumstance: For example, you can't use the `overflow:hidden` technique on the top-level of a drop-down menu, or the child drop-downs won't display. This is because the drop-downs display outside the area of their parent element, which is exactly what `overflow:hidden` is intended to prevent. As another example, you can't use the "float-the-parent" technique on an element that has been centered with `auto` margins, as it then moves to the left or right, depending on the float value applied. So you need all three of these techniques in your bag of tricks to cover all the different situations in which floated elements need to be enclosed.

Using Clear Without a Containing Element

Sometimes, you will need to clear floated elements when there is no convenient parent wrapper to force around them. The most simple way is to apply the CSS

`clear:both` to the element that is floating up to force it to sit under the floated element. However, when there is room for more than one element to float up, this simple approach may not work and you need to be more creative.

To demonstrate this point, **Figure 3.21** shows a layout with six elements—three images, each with associated descriptive text next to it. This layout is achieved by floating the images, so the text that follows each image in the markup text moves up next to the floated image.

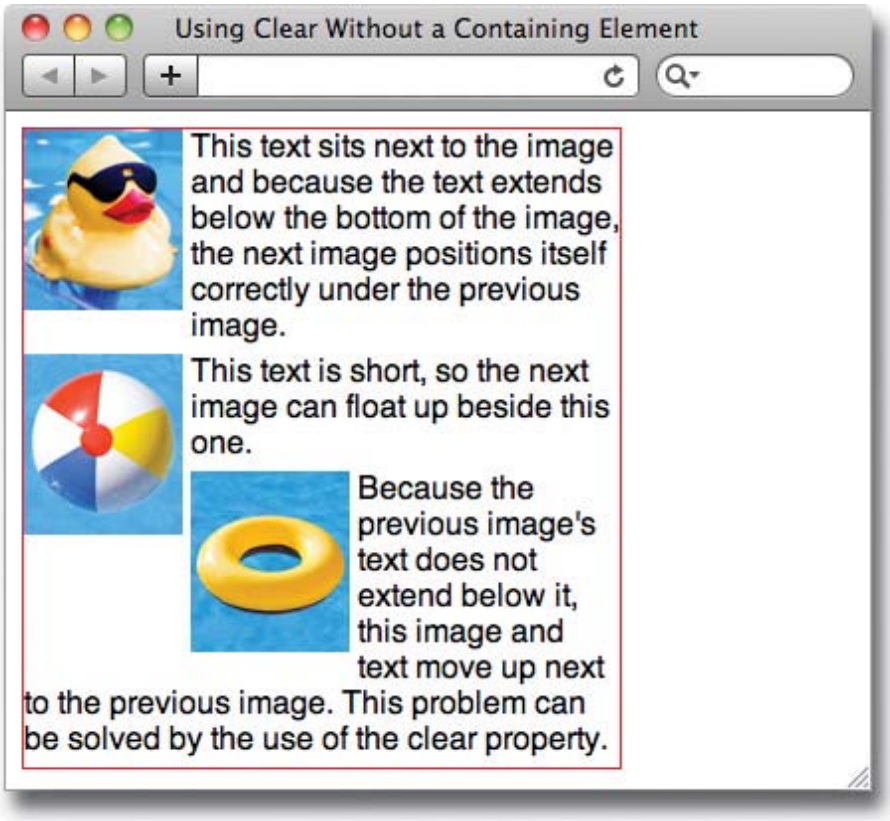


Figure 3.21. Because there is room, the third image and its text can float up next to the second image—not the desired effect.

Here's the HTML for the preceding figure (with the text content edited to save space)

```
<section>
  <img src="images/rubber_duck3.jpg">
  <p>This text sits next to the image and because the…</p>
  <img src="images/beach_ball.jpg">
  <p>This text is short, so the next image can float up…</p>
  <img src="images/yellow_float.jpg">
  <p>Because the previous image's text does not…</p>
</section>
```

to which I apply this CSS

```
section {width:300px; border:1px solid red;}
img {float:left; margin:0 4px 4px 0;}
p {margin:0 0 5px 0;}
```

font specs omitted for brevity

The objective was to have each text block sit next to its image. However, because the second paragraph's text is not long enough to extend below the bottom of the second floated image, the resulting space allows the next image/paragraph pair to float up.

In the preceding example, the layout is technically correct: the third image/paragraph pair has room to sit next to the previous floated element, so they do, because the purpose of floating is to move an element up as high and far to the left or right (depending on the `float` value) as possible. This result isn't what I want visually.

Because there is no containing element around each image/paragraph pair here, I can't use the "force-the-parent-to-enclose" techniques from the previous example. However, I can still use the `clearfix` CSS

```
.clearfix:after {
  content:".";
  display:block;
  height:0;
  visibility:hidden;
  clear:both;
  }
```

like this

```
<section>
  <img src="images/rubber_duck3.jpg">
  <p class="clearfix">This text sits next to the image and
  because the…</p>
  <img src="images/beach_ball.jpg">
  <p class="clearfix">This text is short, so the next image
  can float up…</p>
  <img src="images/yellow_float.jpg">
  <p class="clearfix">Because the previous image's text does
  not…</p>.
</section>
```

As shown in **Figure 3.22**, "cleared" elements are added into the markup after each of the paragraphs. Because the third image and paragraph follow one of these cleared elements in the markup, they can no longer float up, and the desired layout is achieved. I added `clearfix` to *all* of the paragraphs, not just the second one that needs it in this example. This is to illustrate what I would do if this were a real site: in the future, if the text of any of the three paragraphs was made shorter than its image, the layout would not break.

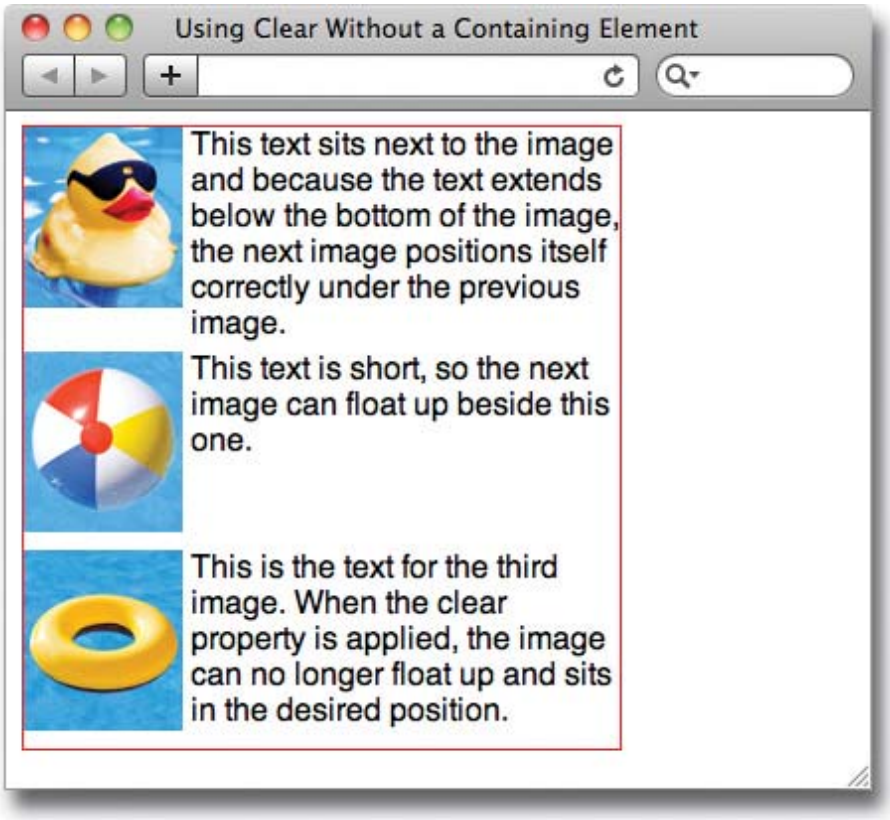


Figure 3.22. Now that the `clearfix` class adds a clearing element, the layout displays correctly.

Now that you have an understanding of the `float` and `clear` properties, let's end this chapter by looking at two other concepts that are key to creating CSS layouts: the `position` and `display` properties.

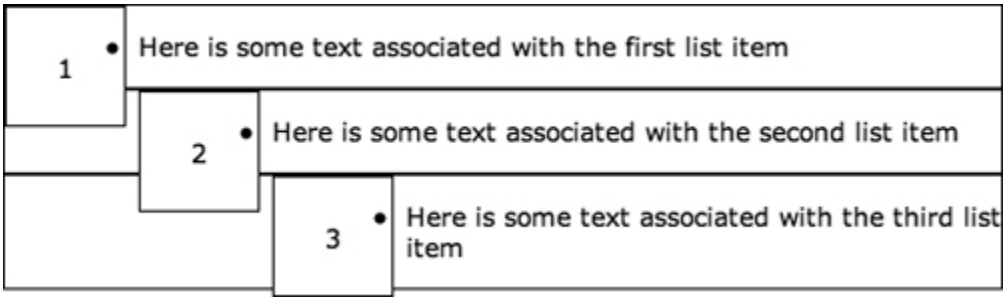
clear

Floating is often used instead of positioning or large margins because the content-wrapping effects are desired, but the wrapping of a tall floated element can often extend into later content in an undesirable way. The `clear` property is used to insert a break when encountering a new section of the layout or another unique record in a list or some other reason for wanting to stop the wrapping effect.

- `none`: The element does not clear any floats.
- `left`: The element will be shifted down to sufficiently clear the bottom edge of any previous left-floated elements.
- `right`: The element will be shifted down to sufficiently clear the bottom edge of any previous right-floated elements.
- `both`: The element will be shifted down to clear and begin after all floated elements.

In **Figure 6.10** you can see what happens when the floated element from one list item bleeds into the following one; things really start to go haywire.

Figure 6.10. The effect of floated elements extending past its parent element into the following content.

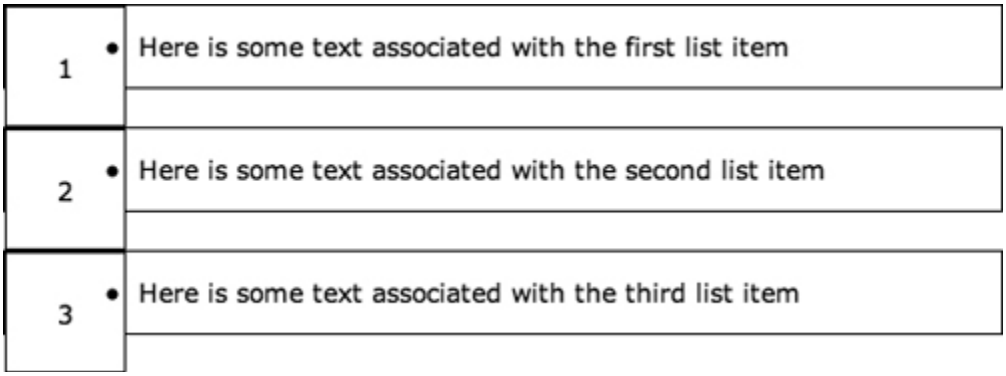



```
li {
   border: 1px solid #000;
}
img {
   float: left;
   margin-right: 0.5em;
}
[...]
<ul>
   <li>
      <img src="images/1.png" width="67" height="67" alt="1" />
      <p>Here is some text associated with the first list item</p>
   </li>
   <li>
      <img src="images/2.png" width="67" height="67" alt="2" />
      <p>Here is some text associated with the second list
   item</p>
```

```
   </li>
   <li>
      <img src="images/3.png" width="67" height="67" alt="3" />
      <p>Here is some text associated with the third list item</p>
   </li>
</ul>
```

You can set the `clear` property to `left` or `both` on the `<li>` element, as shown in **Figure 6.11**, to make sure that for each new item nothing from the previous item interferes.

Figure 6.11. `clear:left` used to stop the previous floats before each new item.




```
li {
   border: 1px solid #000;
   clear: left;
}
```