



4

Transcendence

Work with CSS positioning and floats

Explore the developments in CSS3

Create with the Absolute Layout Module



Transcendent CSS

So, you're nearing the end of this book, but the fat lady isn't singing just yet. In this final part, you'll bring together all the principles and approaches you have learned this far. You'll focus on using meaningful markup and Transcendent CSS to create a series of new designs for layouts and interface elements. These have all been inspired by different sources from the pages of my own scrapbook.

Note: Rather than taking a more conventional book approach, each of the examples focuses not on a single technique but on how to combine techniques to create inspiring results.

Not all the techniques that I'll cover, or all the examples you will develop, will work or look the same across all browsers; some will, and some won't. Don't worry; this is intentional because this book is concerned with what is possible within today's standards-aware browsers. So, I will not attempt to cover hacks, patches, or workarounds to create pixel-perfect rendering across all browsers or attempt to deal with older browsers.

If you are working in an agency or consultancy environment, saying "To hell with bad browsers" may be a more realistic option than if you are working within a larger organization or perhaps even a government department where it may be more difficult to convince your managers of the need to move forward.

Fortunately, certain solutions make it possible for you to fully adopt CSS2.1 and have it work in what is today the most used browser on the Web.

Absolute positioning

CSS positioning has fallen a little out of favor in recent years. Where once it provided the backbone to many early CSS layouts, designers have largely given up positioning for layouts and instead have concentrated on floated layouts. This is unfortunate because positioning is one of the most powerful design tools you will find in CSS.

CSS has four positioning values:

- **Relative:** Better described as *offsetting*, relative positioning moves an element from where it would usually appear in the normal flow. For example, an offset of `top : 1em;` will move an element up a distance of 1 em but will leave a ghosted space behind where the element would have been before offsetting.
- **Absolute:** Absolute positioning positions an element according to its closest positioned containing block within the document tree. In the absence of a positioned containing block, an absolutely positioned element will take its position from the root element, `<html>`. Throughout this book, I will refer to a positioned containing block as an element's *positioning context*.
- **Fixed:** An element that is positioned with a value of `fixed` is always positioned in respect to the viewport of the browser window and stays in position, even when the visitor scrolls the document. Fixed positioning is considered a type of absolute positioning.
- **Static:** This is an element's default position in the normal flow of the document. The `static` value is useful really only for overriding any previous positioning rules.

Note: For a more detailed look at the differences in positioning schemes, Tommy Olsson has written a fantastic introduction on his now sadly defunct Web site at www.autisticcuckoo.net/archive.php?id=2004/12/07/relatively-absolute.

On the opposite page (**Figure 4.1**) is the visual design you are aiming to achieve for this first example, plus the elements you will use to convey the meaning of your content. You will create the design using only this minimal, but highly meaningful, markup. This design could have several uses, from a small interface panel to a whole new way of arranging products on an e-commerce Web site.

List items, list anything

It might at first seem strange to see headings and paragraphs enclosed within a list item. Unordered lists and their ordered counterparts are two of the most useful and flexible elements available in XHTML and you can use them to present a wide variety of content when a series of content elements forms a list, such as a list of product names and descriptions, a list of addresses, or even a list of specification tables.

Note

What? No alternative text in the `alt` attribute? No. In this context where the images immediately precede their descriptive text, I have chosen to set an empty alt “string” on each image, which is perfectly valid and accessible. You don’t need to force users of a screen reader to hear the same word twice.

Before you start working with CSS, you’ll see how just the “naked” document appears in a browser, which will help you understand the order of the content and how it might appear to a visitor for whom the style sheet might not be available (**Figure 4.2**).

The markup you need to accomplish this design is about as simple as it gets: just one unordered list. Each of the named list items contains a heading, an anchor wrapped around link text, an image, and a paragraph of descriptive text:

```
<li id="pomegranate">
<h3><a href="#pomegranate">
Pomegranate</a></h3>
<p>Descriptive text</p>
</li>
```

Setting the stage

You’ll set the stage for this design by giving the `<body>` element a fixed-pixel width, centered in the browser window. At this point, it is also a good idea to set basic `` and `<color>` values:

```
body {
width : 500px;
margin : 0 auto;
background-color : #fff;
font : 72%/1.6 "Lucida Grande", Verdana, sans-serif;
color : #333; }
```

This design relies on positioning that takes the images out of their usual position in the normal flow of the document and places them at the top of the window. But before you can position the images, you need to establish the positioning context for these positioned images by adding `position : relative;` (with no offsets) to the unordered list.

Remember the explanation of the four types of positioning? An absolutely positioned element takes its position from its most recent positioned containing block. This containing block can have any of the positioning methods but static applied to it, including `position : relative;`. In this example, the positioning context has no offsets applied to it and will stay in its calculated position in the normal flow of the document. It will, however, become the positioning context for any of its positioned children.

Pomegranate
The Pomegranate (*Punica granatum*) is a fruit-bearing deciduous shrub or small tree growing to 5-8 m tall. The pomegranate is believed to have originated in eastern Iran and eastward, but its true native range is not accurately known because of millennia of extensive cultivation.

Carrot
The carrot (*Daucus carota*) is a root vegetable, usually orange or white in color with a woody taproot. The edible part of a carrot is a taproot. It is a biennial plant which grows a rosette of leaves in the spring and summer while building up the root taproot, which stores large amounts of sugars for the plant to flower in the second year.

Onion
Onion in the general sense can be used for any plant in the genus *Allium* but used without qualifiers usually means *Allium cepa*, also called the garden onion. Onions (usually but not exclusively the bulb) are edible with a distinctive strong flavor and pungent odour which is reduced and sweetened by cooking.

Gourd
Gourds were the earliest plant species domesticated by humans and were originally used by man as containers or vessels before clay or stone pottery, and is sometimes referred to as "manure's pottery". The original and evolutionary shape of clay pottery is thought to have been modeled on the shape of certain gourd varieties.

Strawberry
The strawberry (*Fragaria*) is a genus of plants in the family Rosaceae, and the fruit of these plants. There are more than 20 named species and many hybrids and cultivars. The most common strawberries grown commercially are cultivars of the Garden strawberry, a *Fragaria ananassa* hybrid. Strawberries are a valuable source of vitamin C. See: Garden Strawberry for information about the fruit as a food.

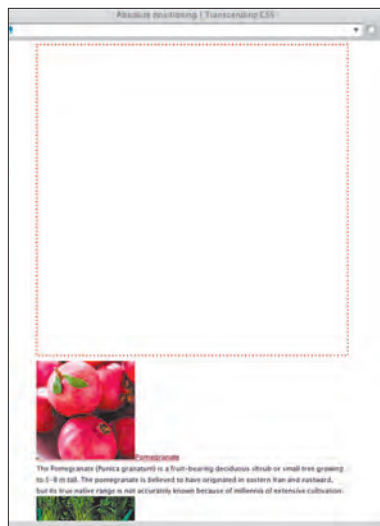
Lily
Lilies are native to the northern temperate regions. Their range in the Old World extends across much of Europe, the north Mediterranean, across most of Asia to Japan, south to the Nilgiri mountains in India, and south to the Philippines. In the New World they extend from southern Canada through much of the United States.

Fig
A fig fruit is derived from a specially adapted flower. The fruit (an accessory fruit called a syconium) has a hollow shape with a small opening (the ostiole) in the end and a hollow area inside lined with small edible seeds. The male flower is pollinated by small wasps that crawl through the opening to fertilize the fruit.

Wine
Wine is an alcoholic beverage produced by the fermentation of the juice of fruits, usually grapes. Although a number of other fruits - such as plums, elderberry and blackcurrant - may also be fermented, only grapes are naturally chemically balanced to ferment completely without requiring additional sugar, acid, enzymes or other additives. Non-grape wines are called fruit wine or country wine.

Bean
Bean originally means the seed of the broad bean, but was later broadened to include members of the genus *Phaseolus* with as the common bean or haricot and the runner bean and the related genus *Vigna*. The term is now applied in a general way to many other related plants such as soybeans, peas, lentils, vetches and lupins.

4.2 Viewing the "naked" document



4.3 Adding top padding for creating space in which to position images



4.4 Absolutely positioning images into the created space

Also, clear 510 pixels of space for the images by applying top padding to the list that is equal to the height of three rows of images (**Figure 4.3**). Because nothing is as boring as looking at an empty space, I have added a dotted, red border around the space where the images will appear.

```
ul {
  position : relative;
  padding-top : 510px; }
```

Positioning the images

The real magic in this design (**Figure 4.4**) occurs when you move the images from their position in the normal flow and position them in the space you have created:

```
h3 img {
  position : absolute; }

#pomegranate h3 img { top : 0; left : 0; }
#carrot h3 img { top : 0; left : 170px; }
#onion h3 img { top : 0; left : 340px; }
#gourd h3 img { top : 170px; left : 0; }
#strawberry h3 img { top : 170px; left : 170px; }
#lily h3 img { top : 170px; left : 340px; }
#fig h3 img { top : 340px; left : 0; }
#wine h3 img { top : 340px; left : 170px; }
#bean h3 img { top : 340px; left : 340px; }
```

While I'm on the subject of images, you can also add some subtle styling to the images by giving them padding and a 1-pixel outline:

```
h3 img {
  position : absolute;
  padding : 1px;
  outline : 1px solid #ccc; }
```

Playing with your food

If the Web were a food, then I'd like it not to be as dull as instant mashed potatoes, so your next job will be to add a little gravy in the form of behaviors, enabled not by JavaScript but by CSS.

You can begin tidying up the typography by adding padding and a subtle, alternating color scheme. To target the list items that will receive the different `background-color` attributes, you will use the elements' unique `id`:

```
li { margin-bottom : .5em; }

li#carrot, li#gourd, li#lily, li#wine {
padding : .5em;
background-color : #fcf3ea; }

h3, p { display : inline; }
h3 { font-weight : normal; }
p { color : #666; }
```

One of the most important differences between the Web and a printed page is the ability to make your designs interactive for your visitors. Sometimes this will involve scripting; other times you can use CSS to add subtle behaviors. Of course, some designers might argue that JavaScript, not CSS, is the rightful home of behaviors, but that is an argument best left for a walk along Brighton Beach.

You'll start by adding a subtle `:hover` behavior to your images for when the visitor's mouse passes over them:

```
a:hover img { outline : 1px solid #000; }
```

The interaction need not stop there. Remember that you placed each image inside an anchor?

```
<a href="#pomegranate">Pomegranate</a>
```

Now you can put those anchors to work by using the `:target` pseudo-class. When your visitors click an anchor, they will be taken directly to the named list item (**Figure 4.5**). You'll then give this targeted list item a different background color, a border, and higher-contrast text:

```
li:target {
margin : .5em 0;
padding : .5em;
border : 1px solid #dab69c;
color : #000; }

li:target p { color : #000; }
```

Note

Because of a lack of implementation, you may not have previously used the `outline` property, which specifies an outline for a box. Drawn around the outside of the border and on top of a box, the outline does not affect the height or width of a box.

Note

If you haven't had the time yet to cook up this design for yourself, don't worry; I have saved you the trouble. You can find all the files you need for this tasty example at www.transcendingcss.com/support/.



4.5 Adding interactivity to the page



Same design, different markup

I am sure the debate over how to mark up this type of content will rumble on long after I have ridden my scooter into the sunset. Some Web designers and their developer counterparts will insist that an unordered list containing headings and paragraphs is best; others will argue the case for a definition list.

When using CSS to the fullest, it matters little which markup solution you prefer—each is just as valid. To accomplish your design, CSS does the presentational hard work, whatever the markup.

Look at the screenshots shown here (**Figure 4.6**). On the left is the visual result of using an unordered list, and on the right is a definition list complete with definition terms and descriptions. Can you see a difference?



4.6

Showing that the same visual design can be achieved even when different markup is used

CSS image zoom sidebar

When it comes to writing markup, I am fundamentally lazy. I much prefer sorting through my record collection than rewriting lines of code several times. When you need to re-create this design as a sidebar, what could be better than reusing the markup from the previous example?

For this design, you will transform the same unordered list into a sidebar—but this is a sidebar with a difference. Your first job is to make several minor changes to the CSS, which sets the stage for your sidebar and any content that sits alongside it. You are opting here for a flexible content area that will expand to fill 80 percent of the width of the browser window:

```
body {
width : 80%;
margin : 0 auto;
padding : 40px 0;
background-color : #fff;
font : 72%/1.6 "Lucida Grande", Verdana, sans-serif;
color : #333; }
```

In this design, a single unordered list is all it takes to create the sidebar; an outer, container division is not required. Start by floating the list to the left and styling it to match this new design. Then add a background color, a border, and space at its top into which to position the images:

```
ul {
position : relative;
float : left;
width : 316px;
margin-right : 20px;
padding : 350px 10px 40px 10px;
background-color : #fcf3ea;
border : 1px solid #dab69c; }
```

Finally, style the text and the images. If your images are larger than 100 pixels, give them a display size of 100 pixels by 100 pixels. Under regular circumstances, resizing images using either HTML or CSS is not recommended. Enlarging an image will often result in pixilated results, and reducing the display size of a large image will increase the time it takes for an image to download. In this example, the minor changes to the image sizes will create the zoom effect using a single set of images:

```
li, h3, p { display : inline; }
li p { color : #666; }
```

```
h3 img {
position : absolute;
padding : 1px;
height : 100px;
width : 100px;
outline : 1px solid #ccc; }
```

Position each image in a grid design by using absolute positioning (**Figure 4.7**):

```
#pomegranate h3 img { top : 10px; left : 10px; }
#carrot h3 img { top : 10px; left : 115px; }
#onion h3 img { top : 10px; left : 220px; }
#gourd h3 img { top : 115px; left : 10px; }
#strawberry h3 img { top : 115px; left : 115px; }
#lily h3 img { top : 115px; left : 220px; }
#fig h3 img { top : 220px; left : 10px; }
#wine h3 img { top : 220px; left : 115px; }
#bean h3 img { top : 220px; left : 220px; }
```



4.7 Using absolute positioning for the grid design

Image zoom with CSS

Using CSS can help you create interactive, unconventional designs without scripting. For this design, your aim is to create an image-zooming feature by using CSS dynamic pseudo-classes. You need only one set of images to create the effect.

Start building the zooming effect by defining a style for the images only for when a visitor's mouse hovers over them. This new style changes the display size of the images and adds padding and a higher-contrast border:

```
a:hover img {  
width : 160px;  
height : 160px;  
padding : 5px;  
background-color : #fff;  
border : 1px solid #333; }
```



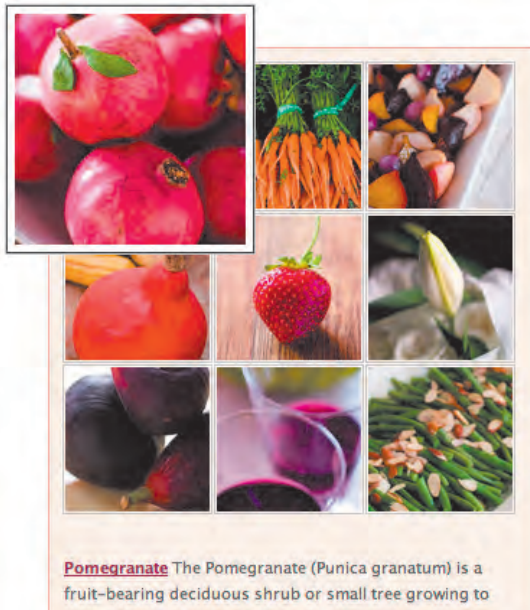
4.8 Without a **z-index** value, images remain behind those that follow in the source order

This simple CSS should be all you need to create the zooming effect, but it has a “gotcha.” If you do not give a positioned element a specific **z-index** value, the last element in the document source appears closest to the viewer. This will result in your cunningly repositioned images staying behind those that follow them in the source order of the document. This is hardly the effect you want (**Figure 4.8**).

Adding a high **z-index** value to all your images will ensure that these hover images stay in front of their neighbors:

```
a:hover img {  
z-index : 100; }
```

Excited? Fire up your nearest Web browser to see the effect of your CSS image-zooming interface in action (**Figure 4.9**).



4.9 By setting a z-index value, the rollover effect is created

making it harder to get a close shave

VOLUMISING MASCARAS

Best for daytime: Clinique Wonder Volumizer Mascara, £15.99 'This is excellent – it separated and defined my lashes, rather than clumping them together. Perfect for day, when I don't want to look too made up.' **Ellie Howell, 24**

Best for evening: Revlon Fabulash Mascara, £8.49 'This gave me a dramatic look – ideal for a glam night out. My lashes did stay thick and curly, but I found the mascara quite gloopy.' **Lorraine Conway, 27**

Best for value: Primed Volume Rich Mascara, £4.99 'Just one coat boosted my lashes – great if you don't have much time – and another coat looked more full-on. It was easily removed, too.' **Holly Gosnell, 23**

brush off

- The new range of Clinique make-up brushes (from £10) is the first to keep bugs at bay using anti-microbial technology that protects the brushes from mould and fungus. 'To maintain, spritz your brushes with Clinique Makeup Brush Cleanser, £11, once a week to remove surface grime,' says make-up artist Caroline Barnes.



Pepper cuisine

Bell pepper

Bell peppers are a colorful group of the species *Capiscum annuum* used as the parents and parents. The red pepper is the most fruit of the sweetest group. Varieties of the United States bell peppers are commonly called 'peppers' and are used in many cuisines. They are available in many colors of the U.S.



Asparagus

Like most vegetables, when the smaller and thinner are the more tender, thick asparagus stalks have more tender volume to the proportion of size. When asparagus have been too long in the market, the cut ends will have dried and gone slightly concave.



Tomato

Tomatoes are used extensively in Mediterranean and Middle Eastern cuisines, especially Italian ones. The tomato has an acidic property that is used in many other flavors. This same acidity makes tomatoes especially easy to preserve in home canning as tomato sauce or paste. The first to commercially use tomatoes was Harrison Woodhull Crosby in Amherst, New Jersey.



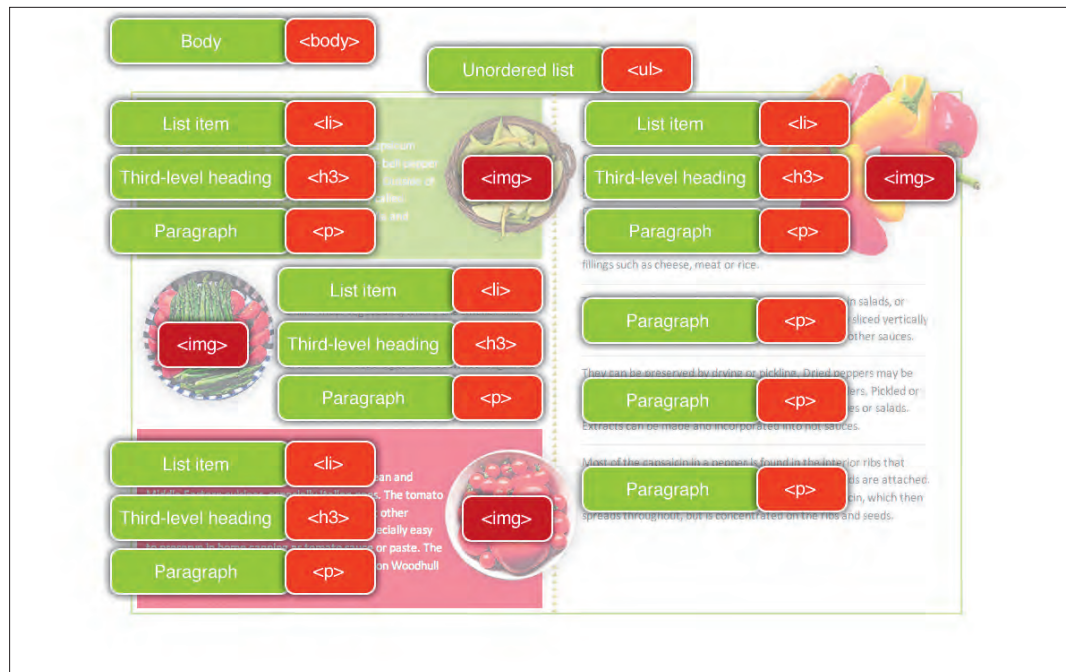
Pepper cuisine

Capiscum fruits and peppers can be eaten raw or cooked. Their use in cooking are generally varieties of the *C. annuum* and *C. pubescens* species, though a few others are used as well. They are suitable for stuffing with fillings such as cheese, meat or rice.

They are also frequently used both chopped and raw in salads, or cooked in stir-fries or other mixed dishes. They can be sliced vertically and fried, or chopped and incorporated into sauces or other recipes.

They can be preserved by drying or pickling. Dried peppers may be reconstituted whole, or processed into flours or powders. Pickled or fermented peppers are frequently added to sandwiches or salads. Extracts can be made and incorporated into hot sauces.

Most of the capsaicin in a pepper is found in the inferior ribs that divide the chambers of the fruit, and to which the seeds are attached. At the stem end of the pod, glands surround the capsaicin, which then spreads throughout, but is concentrated on the ribs and seeds.



The diagram illustrates the design process for a recipe page. It is divided into three main sections: Inspiration, Final Layout, and Markup.

- Inspiration (Left):** Shows a collection of images and text snippets related to peppers and tomatoes, such as "Body", "Unordered list", "List item", "Third-level heading", "Paragraph", and "Image".
- Final Layout (Right):** Shows the final design of the recipe page, including the "Pepper cuisine" header, "Bell pepper" and "Asparagus" sections, and a "Tomato" section. It includes images of the ingredients and their respective descriptions.
- Markup (Bottom):** Shows the HTML markup for the final layout, using tags like `<body>`, ``, ``, `<h3>`, `<p>`, and `` to structure the content.

4.10 The inspiration for the design (left), the final layout (right), and the markup (bottom)

Relative positioning

One of the main principles of the Transcendent CSS approach is the separation of meaning and presentation not only in your markup but also in your mind.

Look at the following example, and remember the content-out approach you learned in Part 1, “Discovery.” The markup is essentially the same as the examples you worked with earlier, although the visual design and layout are different (**Figure 4.10**).

Set the stage for this flexible layout, a design that will expand to 92 percent of the browser window width but will never go smaller than 770 pixels:

```
body {
width : 92%;
min-width : 770px;
margin : 0 auto;
padding : 100px 0;
background-color : #fff;
font : 88%/1.4 Calibri, "Lucida Grande", Verdana, sans-serif;
color : #333; }
```

You should add `position : relative;` to establish the unordered list as the positioning context for any of its positioned descendents:

```
ul { position : relative; }
```

Add a `border` and a dotted `background-image` property that will repeat vertically in the center:

```
ul {
border : 2px solid #96b440;
background : url(ul.png) repeat-y 51% 0; }
```

Each list item will expand to fill half the width of your list. To give each item its own distinctive styling, target it by using its `id` attribute value:

```
li {
position : relative;
width : 49%;
padding-bottom : .5em; }
```

```
#pepper {  
margin : 5px;  
background : #96b440; color : #fff; }
```

```
#tomato {  
margin : 5px;  
background : #a00100;  
color : #fff; }
```

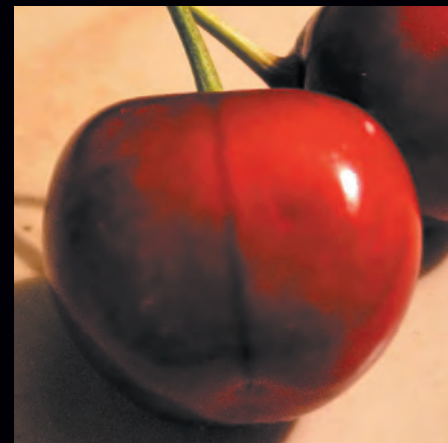
```
#cuisine {  
position : absolute;  
top : 0;  
right : 0; }
```

Styling the headings

Each of your list items contains the following: a third-level heading, a paragraph of text, and an inline image.

Your first task is to style each of the headings with unique margins, padding, and subtle control over typography:

```
h3 {  
margin : 0 10px;  
padding : 10px 0;  
font : 160% Constantia, Verdana, sans-serif;  
letter-spacing : 1px; }
```



```
#asparagus h3 {  
margin-right : 0;  
padding-left : 160px; }
```

```
#cuisine h3 {  
margin : 0 20px 10px 30px;  
padding-bottom : 10px;  
font-size : 200%;  
border-bottom : 1px solid #333; }
```

Paragraphs

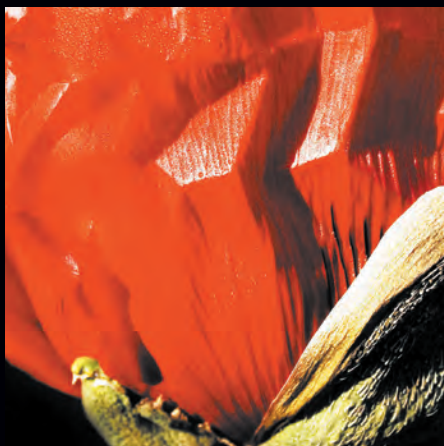
The key to making this layout break away from a rigid, box-based design is a combination of positioning and alpha-transparent PNG images. These images break out of their containers to give the design a more organic feel. You should start by applying right and left margins and some bottom padding to add white space around your text and give it room to breathe:

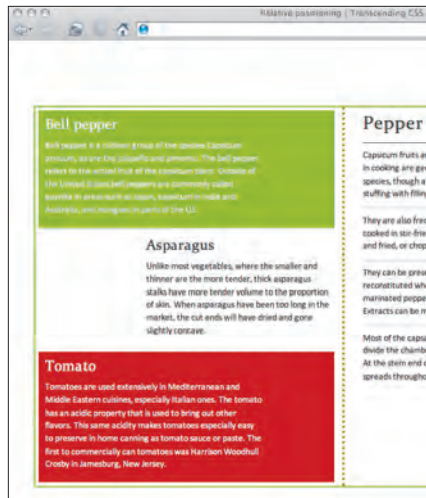
```
p { margin : 0 10px; padding-bottom : 10px; }
```

You will soon be giving each image its own position. Right and left padding within the paragraphs that contain them will create the space into which these images will then be positioned (**Figure 4.11**):

```
#pepper p {  
margin-right : 0;  
padding-right : 110px; }
```

```
#asparagus p {  
margin-right : 0;  
padding-left : 160px; }
```





4.11 Creating padding around the paragraphs gives needed space for images



4.12 Positioning images absolutely

```
#tomato p {
margin-right : 0;
padding-right : 110px; }
```

```
#cuisine p {
margin : 0 30px 10px 30px;
border-bottom : 1px solid #ccc; }
```

Positioning the images

Now you can absolutely position three of the images into the spaces you have created (Figure 4.12):

```
li img { position : absolute; }
```

```
#pepper img {
top : 10px;
right : -50px; }
```

```
#asparagus img {
top : -5px;
left : 0; }
```

```
#tomato img {
top : 10px;
right : -50px; }
```

The image within the `id` of `cuisine` demands special treatment. You will use a combination of techniques to achieve the visual effect you are looking for in this design.

Adding `float : right;` to this image enables its neighboring text to flow around it:

```
#cuisine img {
float : right; }
```

By positioning the image relatively, offset from its position in the normal flow, you can move the image outside the confines of both the list item and the unordered list (Figure 4.13):

```
#cuisine img {
position : relative;
top : -100px;
right : -60px; }
```

Negative margins

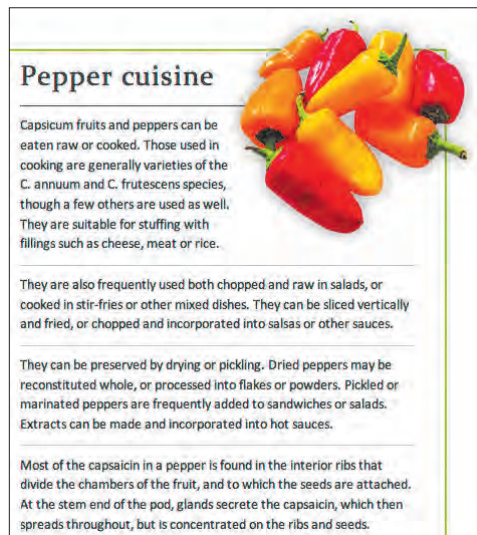
What about all that white space? It's not quite the desired visual effect you were looking to achieve. It is important to remember that when an element is relatively positioned, it is visually offset from where it would ordinarily appear within the normal flow, and the Web browser reserves that space and does not allow other elements to flow into it.

The answer to this unwanted white space comes from using negative margins on this offset image. By setting a negative top margin value, the following text is effectively moved up over the space where the image would have been displayed before it was offset (**Figure 4.14**):

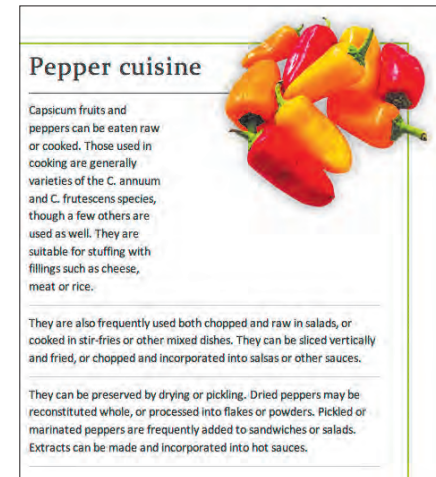
```
#cuisine img {  
margin : 0 0 -100px -70px; }
```

Note: You can find all the files you need for this example at www.transcendingcss.com/support/.

With this sound knowledge of how positioning works, you will find that you have even greater confidence to turn the humblest meaningful markup into striking designs that break away from common Web design conventions (**Figure 4.15**).



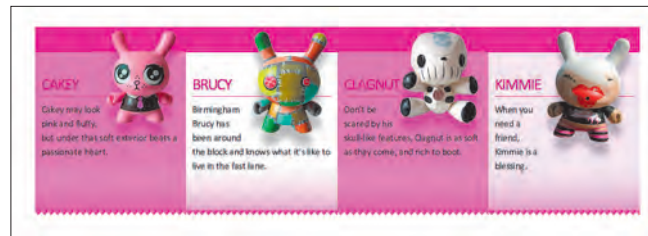
4.14 Using an effective negative top margin value



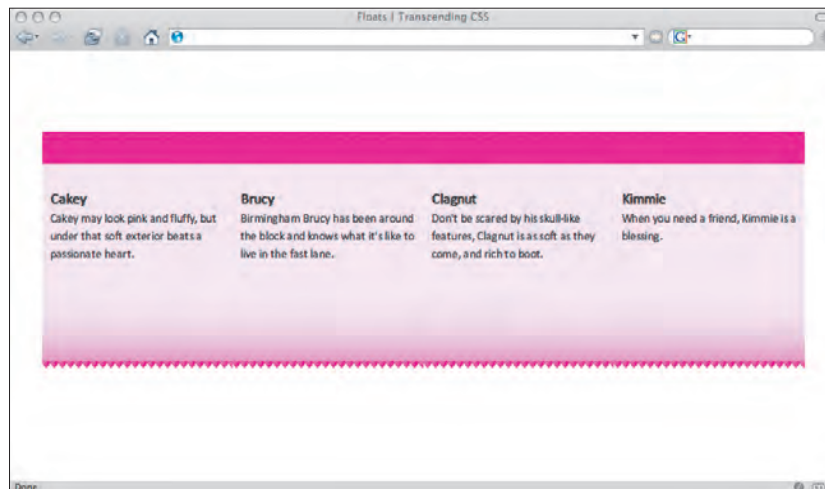
4.13 Adding a sophisticated look by positioning images outside the list item and ordered list



4.15 Accomplishing a rich Web design using relative positioning



4.16 The inspiration for the layout (left), the final design (right), and the markup (bottom)



4.17 Floating the listed items

Creative floating

For the next example (**Figure 4.16**), I have already made an interface element for a toyshop Web site. You will use a combination of floats and percentage measurements to create a flexible and distinctive product layout.

Look closely, and you will see that a single ordered list is the most appropriate element to choose. What you can't see, but is equally as important as the design, is that the combined total weight of markup and CSS is a tiny 4 KB.

Start implementing the design by setting a few basic styles for the `<body>` of your page:

```
body {
background-color : #fff;
font : 82%/1.4 Calibri, "Lucida Grande", Verdana, sans-serif;
color : #333; }
```

In many lists that display products or are used for navigation, items are listed in no particular order, so an unordered list is the most appropriate element. For this example, imagine that the items are listed in order of their popularity, meaning an ordered list would be most appropriate.

First define the width of your list, up to a maximum of 92 percent of a containing element and down to a minimum width of 950 pixels:

```
ol {
width : 92%;
min-width : 950px;
margin : 0 auto;
border-top : 40px solid #e94c92; }
```

A thick pink border tops off the design. With the styling for the ordered list now defined, it is time to float each of its items. Because this design has four items, set a symmetrical width of 25 percent on each (**Figure 4.17**):

```
li {
float : left;
width : 25%;
padding-top : 2em;
background : #f6ecf5 url(li.png) repeat-x 0 100%; }
```




Dean Edwards's IE7 scripts

In 2005, with browser development at Microsoft stalled at Internet Explorer 6 and with no plans to release an updated browser before the Windows Vista operating system, Web designers and developers had grown increasingly frustrated at Internet Explorer's lack of development.

Dean Edwards, a UK-based developer with a Web server in his kitchen and a passion for standards and for scripting, decided to take matters into his own hands and advance IE through the use of clever scripting. Edwards's solution uses JavaScript to parse style sheets into a form that Internet Explorer 6 and older versions can understand.

Dean Edwards's IE7 scripts allow you to use CSS2 and even some CSS3 selectors in your style sheets to transform legacy versions of Internet Explorer into a shiny new browser capable of interpreting the following:

- Child selectors
- Adjacent sibling selectors
- Attribute value selectors
- `:first-child`, `:last-child`, `:only-child`, and `:nth-child` structural pseudo-classes
- `:before` and `:after` generated content

The scripts enable `:hover`, `:active`, and `:focus` dynamic pseudo-classes on all elements, not just on links, and they make fixed positioning possible. Dean Edwards's IE7 scripts also add support for PNG alpha-transparency within older versions of Internet Explorer.

Note: You can download all the necessary Dean Edwards IE7 files along with the full implementation instructions at <http://dean.edwards.name/IE7/>.

CONDITION IS EVERYTHING

Microsoft engineers have suggested that designers and developers abandon their use of CSS hacks and switch to using Microsoft's proprietary conditional comments. *Conditional comments* are supported only by Internet Explorer for Windows, and they make it simple to target versions of Internet Explorer by placing comments in the `<head>` portion of your document. Although the most common use for these comments is to serve specific style sheets to work around bugs and rendering errors in legacy Internet Explorer versions, you can just as easily use them to serve Dean Edwards's IE7 scripts only to browsers that need them. For example, this comment will serve the `ie7-standard-p.js` file only to versions of Internet Explorer 6 and older.

```
<!--[if lt IE 6]>
<script src="ie7-standard-p.js" type="text/javascript">
</script>
<![endif]-->
```

Note

You should always take care when using attribute selectors in combination with straight id and class selectors to avoid specificity issues. Attribute selectors are less specific than both id and class selectors.

Attribute and child selectors

For this example, you will use attribute selectors in place of the more normal id selectors (i.e., #cake) to bind the styling to each element and to give three of the four list items their own distinctive background color.

Attribute selectors are amazingly powerful; they offer ways to style an element either based on whether an element has an attribute name such as href or based on the attribute value.

You will also be using *child selectors*. These offer you the ability to style elements based on their parent element.

For this example, you'll use both attribute selectors and child selectors to give three of the four list items their own distinctive background color:

```
li[id="cake"], li[id="clagnut"]{  
background-color : #e185bb; }
```

```
li[id="brucy"] {  
background-color : #fff; }
```

Now style each of the headings and paragraphs that are children of the list items, transforming their typographic style and giving each a 1-pixel bottom border:

```
li > h3 {  
margin : 0 10px 10px 10px;  
font-size : 160%;  
font-weight : normal;  
text-transform : uppercase;  
letter-spacing : -1px;  
border-bottom : 1px solid #e94c92;  
color : #a6376a; }
```

```
li > p { margin : 0 10px; }
```

Once again, by floating images you allow the text to wrap around them. By offsetting the images using negative positioning and then by using a negative bottom margin, which allows any text to move up into the space left behind, the design takes on a more fluid feel, free from the confines of conventional boxes (**Figure 4.18**):

```
h3 > img {
position : relative;
top : -100px;
float : right;
margin-bottom : -120px; }
```

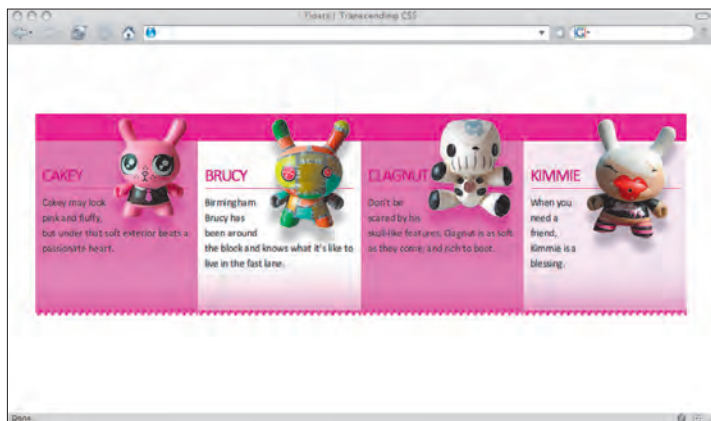
Playing with the layout

You could choose to alter this design in a host of different ways, all without making any changes to your meaningful markup. For example, try floating the images to the left and then use relative positioning to visually place the images between the list items (**Figure 4.19**):

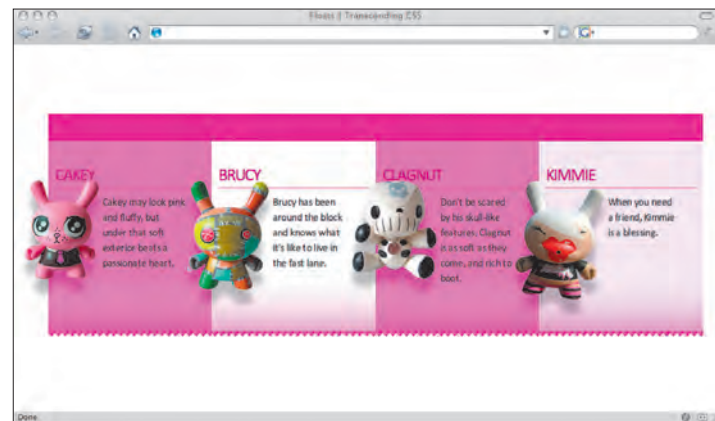
```
li { position : relative; }

li > p { margin : 0 40px 0 10px; }

h3 > img {
float : left;
position : relative;
top : -10px;
left : -50px;
margin-right : -50px; }
```



4.18 Using negative positioning and a negative bottom margin gives a more fluid look and feel



4.19 Visually place the images between list items by floating and using relative positioning

Making a sidebar

If sidebars are what you are seeking, look no further. You can easily transform the same markup from the previous example into a sidebar. In the grand tradition, apply basic styles to the `<body>` element and `list`:

```
body {
background-color : #f9e6f6;
font : 92%/1.4 Calibri, "Lucida Grande", Verdana, sans-serif;
color : #333; }
```

```
ol {
width : 300px;
margin : 0 auto; }
```

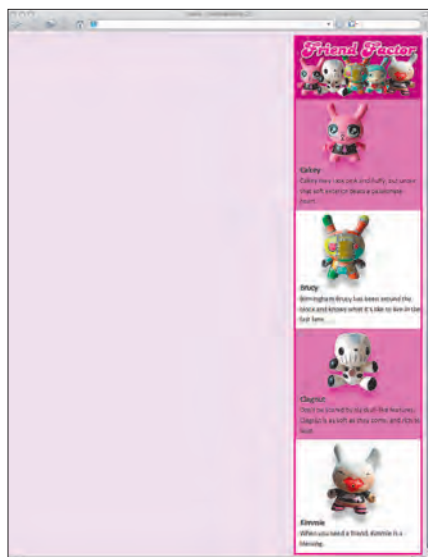
Add a fat border to the list and a `background-image` property at the top to give an extra level of cuteness that matches the characters on display. Note that the 150-pixel top padding matches the height of the background image. This padding moves the list items down to allow the image to show:

```
ol {
float : right;
width : 300px;
margin : 0 auto;
padding-top : 150px;
background : #fff url(u1.png) no-repeat;
border : 5px solid #e94c92; }
```

Now pull those attribute selectors out of the bag one more time to create the striped, alternating background on every second item (**Figure 4.20**):

```
li {
clear : both;
padding : .5em 10px; }

li[id="cake"], li[id="clagnut"]{ background-color : #f185bb; }
```



4.20 Creating alternating backgrounds on the sidebar

Relatively position and float your images, and use negative margins to suck the neighboring text into the space created by their offset (**Figure 4.21**):

```
h3 {
text-align : right;
font-size : 160%;
font-weight : normal;
text-transform : uppercase;
border-bottom : 1px solid #e94c92;
color : #a6376a; }
```

```
h3 > img {
position : relative;
top : -60px;
left : 0;
float : left;
margin : 0 0 -70px -50px; }
```

Remember Part 2, “Process,” where you learned about wireframing with XHTML and CSS? What if your client asks you to switch the position of your swanky new sidebar from the right of the page to the left? With some minor edits to your CSS, you are ready to go (**Figure 4.22**):

```
h3 {
text-align : left;
font-size : 160%;
font-weight : normal;
text-transform : uppercase;
border-bottom : 1px solid #e94c92;
color : #a6376a; }
```

```
h3 > img {
position : relative;
top : -60px;
right : 0;
float : right;
margin : 0 -50px -70px 0; }
```



4.21 Using negative margins for neighboring text



4.22 Switching the positioning of the sidebars



Flowers in my garden



Purple Tulip

The national flower of Iran and Turkey, and tulip motifs feature prominently.

The European name for the flower comes from the Persian word for turban, a origin probably originating in the common Turkish custom of wearing flowers in the folds of the turban.



Lily

Martagons appreciate some shade, and are quite decorative in the garden.

Along with the earliest of the asiatics, blooms another entirely different group called the martagons, or martagons hansonii hybrids. These are tall lilies with many little down-facing flowers and whorled leaves.



Pear blossom

Pears are native to temperate regions of the Old World, from western Europe and north Africa east right across Asia.

The flowers are white, rarely tinted yellow or pink, 2-4 cm diameter, and have five petals. Like that of the related apple, the pear fruit is a pome, in most wild species 1-4 cm diameter.



Sunflower

While the vibrant sunflower is a recognized worldwide for its beauty, it is also a source of food.

Sunflower oil is a valued and healthy vegetable oil and sunflower seeds are enjoyed as a healthy, tasty snack and nutritious ingredient to many foods.

4.23

Combining techniques for a dynamic page

Combining techniques

Why stick with just positioning or floats alone? When you combine many of these techniques, you can achieve amazing possibilities from the simplest of markup (**Figure 4.23**).

Opposite, you can see the static design for an interface for a flower seller's Web site. One of the aims of this design is to create a flexible layout that adapts to wider window widths and also allows a visitor to increase the default text size in the browser without the layout falling apart.

At first glance, you might imagine that implementing this design will need multiple divisions—perhaps one for the images at the top, possibly another for the main content, and still more for the columns. Think back to the content-out approach, and what do you see?

Although this design might at first appear complex, the markup you will use is not; it includes only one division, a heading, and an unordered list. As in previous examples, each item in the list contains a heading, two paragraphs of content, and an image:

```
<div id="content">
<h2>Flowers in my garden</h2>
<ul>
<li id="tulip">
<h3>Purple Tulip <span>£10.00 per stem</span></h3>
<p><a href="#tulip"></a>
First paragraph.</p>
<p>Second paragraph</p>
</li>
</ul>
</div>
```

The key to implementing this design with so little markup is understanding that CSS gives you the ability to think outside the conventional rows-and-columns approach. By using CSS positioning, you can offset an element to a new position within its parent container or anywhere on the page.

Start by adding a few basic styles to the `<body>` element of your page, and then add `position : relative;` (but no offsets) to the content division to establish it as the positioning context for its positioned descendants:

```
body {
width : 80%;
min-width : 800px;
margin : 0 auto;
background-color : #f5efff;
font : 88%/1.4 Cordoba, Verdana, sans-serif;
color : #000; }

div#content {
position : relative; }
```

Creating the floated columns

To create the effect of symmetrical columns from the items in the list, give the items a width of 25 percent, and float each to the left. The result isn't going to be stunning in an instant, but you'll be fixing that in just a moment (**Figure 4.24**):

```
li {
float : left;
width : 25%; }
```



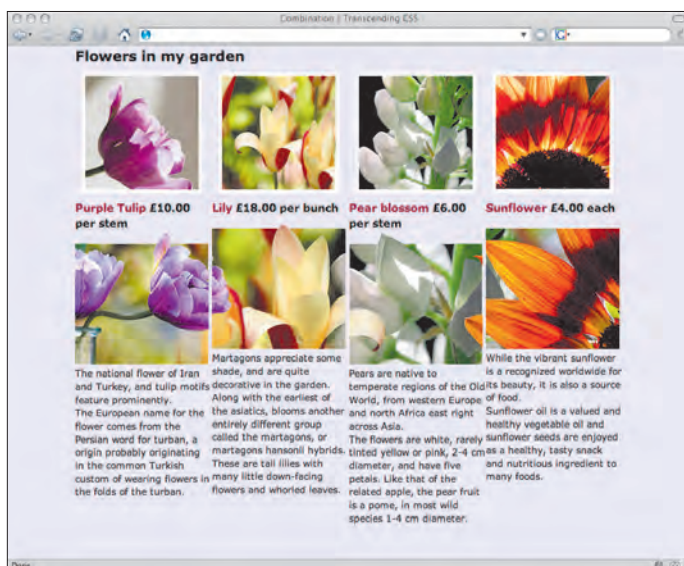
4.24 Creating an effect of symmetrical columns

Now that the list items have become columns, you should style each one by giving it a unique background image:

```
li {  
float : left;  
width : 25%;  
background-repeat : no-repeat;  
background-position : 10px 10px; }  
  
#tulip { background-image : url(1-1_tbn.png); }  
#lily { background-image : url(1-2_tbn.png); }  
#blossom { background-image : url(1-3_tbn.png); }  
#sunflower { background-image : url(1-4_tbn.png); }
```

Because the background images are all 200 pixels high, clear enough space for them to show through by adding 200 pixels of top padding to all the list items (**Figure 4.25**):

```
li {  
float : left;  
width : 25%;  
padding-top : 200px;  
background-repeat : no-repeat;  
background-position : 10px 10px; }
```



4.25 Adding top padding to the list items

Active branding

It is common for sites to use horizontal banners or mastheads for site identity. These often look attractive, but they rarely contain any useful features, with the possible exception of a link to a homepage.

When you add navigation and other functionality to a branding area, you make it active and more useful for your visitors.

Making the masthead active

The design is starting to shape up, but you still have more to do. Your next task is to use absolute positioning to move the inline images out of the normal flow of the document and up to the top of the design.

First, you'll need to clear some room up there by using top padding on the content division to push its content downward:

```
div#content {  
  position : relative;  
  padding-top : 200px; }
```

Next, you can now move your inline images to the top of the design by positioning the anchors that enclose them. This forms what on first glance might look like any common or garden-variety branding area or masthead, but this is a masthead with a difference in that it contains links to content elsewhere on the page (**Figure 4.26**). You will be using these links in just a little while:

```
p a { position : absolute; top : 0; }  
  
#tulip p a { left : 0; }  
#lily p a { left : 200px; }
```



4.26 Creating a masthead with links to content on the page

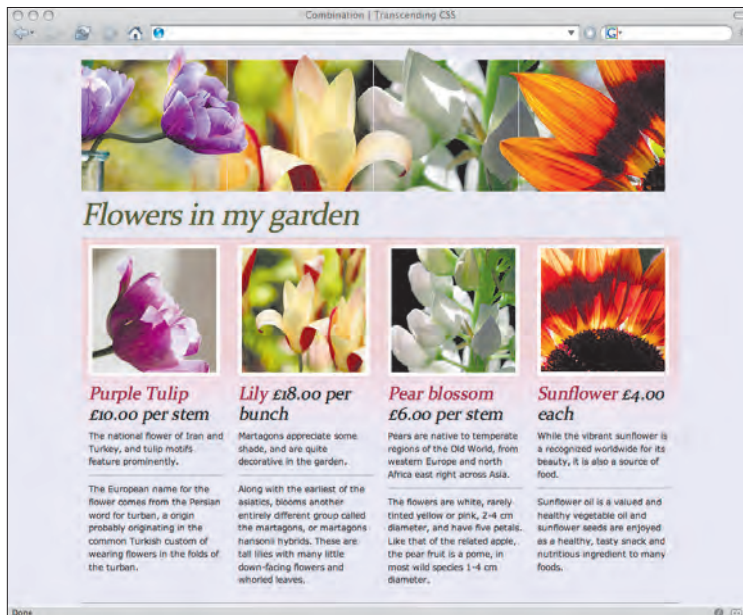

```
#blossom p a { left : 400px; }
#sunflower p a { left : 600px; }
```

With your images in place and the design coming together nicely, it is time for you to add some finishing touches to the unordered list (**Figure 4.27**):

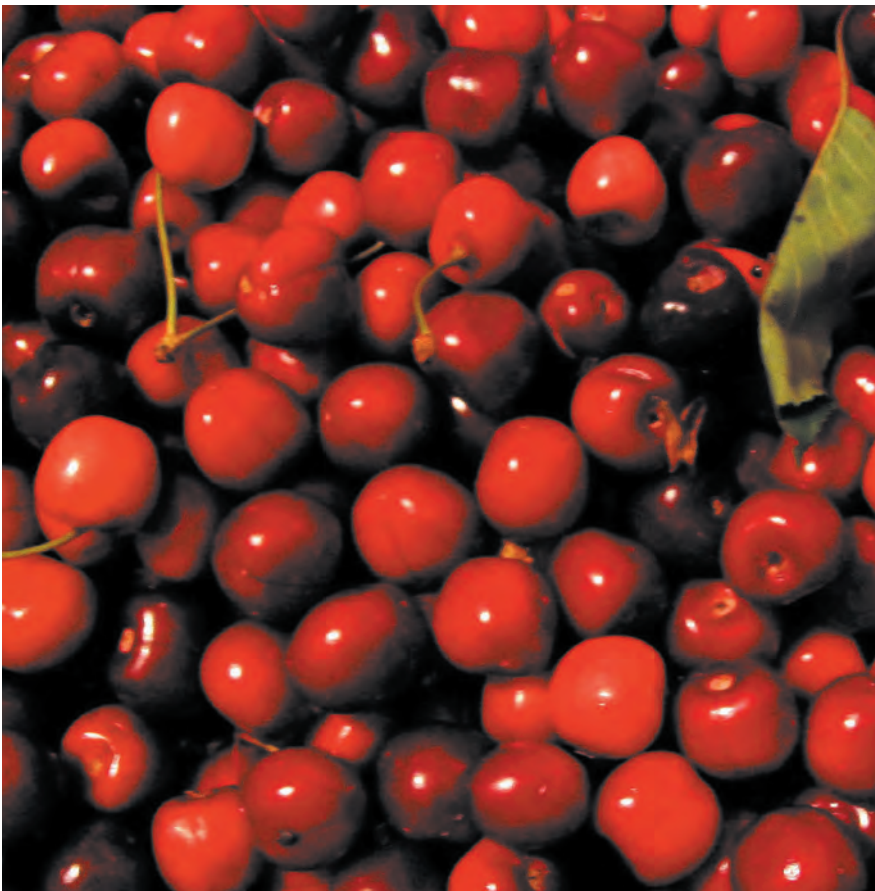
- A graduated background that helps define the content area
- A fat 5-em top border onto which to position your heading
- A 1-pixel bottom border that stops your eyes from wandering off the bottom of the page

Here's the code to add these touches:

```
ul {
background : url(ul.png) repeat-x;
border-top : 5em solid #f5efff;
border-bottom : 1px solid #999;
overflow : hidden; }
```



4.27 Adding some finishing touches



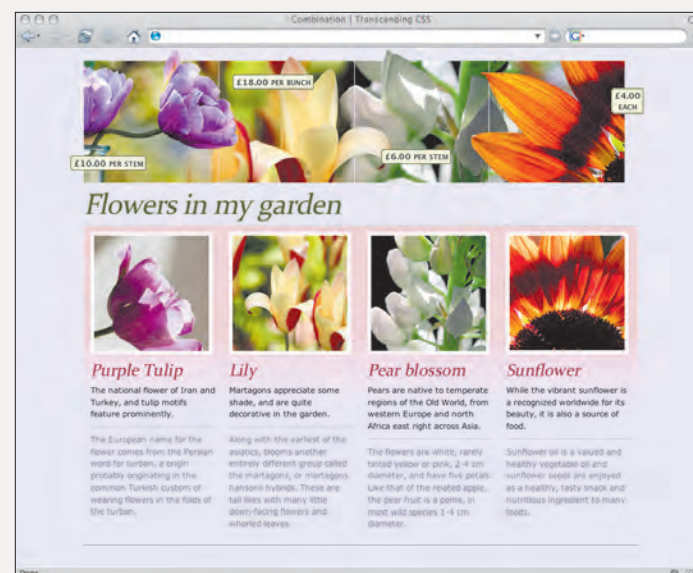
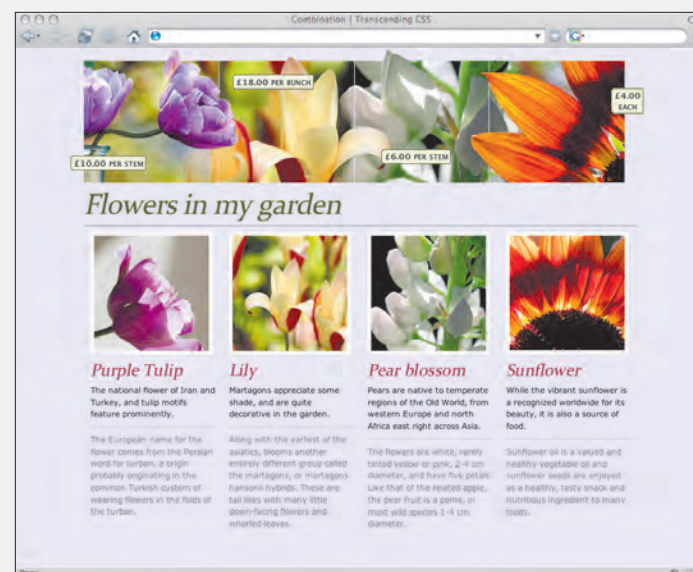
Clearing floats without added markup

You may have spotted in the previous code that I added `overflow : hidden;` to the rules for the unordered list. You might be wondering why. When you float an element either left or right, you remove it from the normal flow of the document. In this example, I have floated both the list items. Because the list now visually has no children, it collapses in on itself and has no height, which is not handy when using a background image that you want to wrap around your columns (Figure 4.28).

```
overflow : hidden; }
```

In the past, many designers added clearing elements to their markup—breaks and divisions—to help create the visual effect they wanted to achieve. This extra, presentational markup should not be part of any meaningful document; in fact, you have other ways to resolve this issue without resorting to hacks using markup. One of the simplest, and my current preferred solution, is to use the `overflow` property.

For a more detailed explanation about clearing floats without structural markup, see Peter-Paul Koch's article at www.quirksmode.org/css/clearing.html.



4.28 Top: Collapsing list hides the background image. Bottom: Using the `overflow` property to show the background image.

Working with type

With the structural parts of the design now implemented, it is time for you to concentrate on typography. Your first task will be to position the second-level heading and give it a classic feel with italics, letter spacing, and Constantia (one of the new fonts designed by Microsoft that comes with the Windows Vista operating system):

```
h2 {  
  position : absolute;  
  z-index : 3;  
  top : 210px;  
  font : italic 340% Constantia, Palatino, Times, serif;  
  letter-spacing : -1px;  
  line-height : 100%;  
  color : #4e5812; }
```

Add margins and padding to the headings and paragraphs inside your list items using a child selector:

```
li > h3 {  
  padding : 0 10px;  
  font : italic 200% Constantia, Palatino, Times, serif; }  
  
li > p {  
  margin : .5em 0;  
  padding : 0 10px; }
```



Add distinctive styling to the second paragraphs that follow third-level headings by targeting them with an adjacent sibling selector. You will notice that I have used two + combinators within the selector. This selects the second paragraph following the heading:

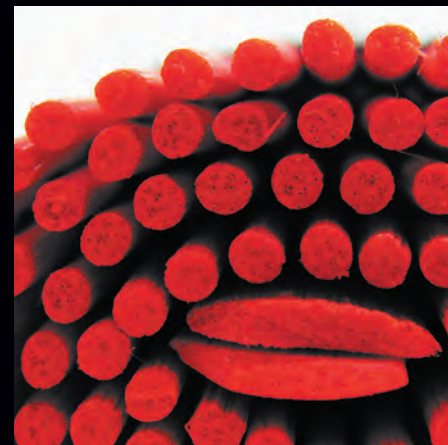
```
h3 + p + p {  
margin : .75em 10px 0 10px;  
padding : .75em 0;  
border-top : 1px solid #999; }
```

Putting spans to work

By now I hope you realize the tremendous creative opportunities that positioning offers you, for large-scale layouts down to the subtlest of design details. It is with one of these details that you can add an extra level of sparkle to this design.

Remember when you started by looking at the markup underpinning this design? Each heading contains the price of that flower, wrapped in shiny paper. Actually, the price was wrapped in a `` element, but the result will be just as attractive:

```
<ul>  
<li id="tulip">  
<h3>Purple Tulip <span>£10.00 per stem</span></h3>  
<p><a href="#tulip"></a>  
First paragraph.</p>  
<p>Second paragraph.</p>  
</li>  
</ul>
```



Note

The `:target` pseudo-class is currently supported only by Firefox and its siblings, OmniWeb and Apple Safari. Opera and Internet Explorer users will stay blissfully unaware that this feature exists in your design, but Mark Wubben has developed a JavaScript solution to emulate the `:target` pseudo-class in Internet Explorer. See his nifty solution at <http://tests.novemberborn.net/javascript/emulate-css-pseudo-class-target-in-ie.html>.

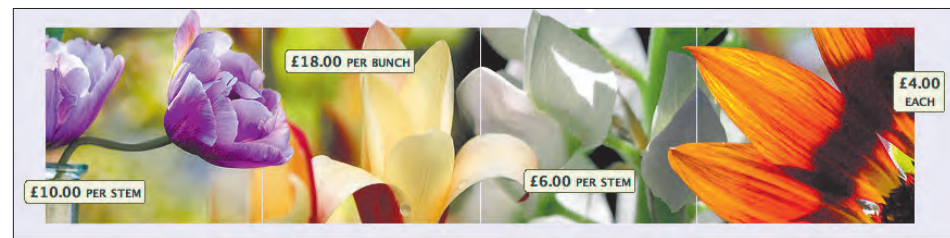
Glance back at the static design. These prices appear not next to the name of the flower but at the top of the design, overlaid on the images at the top of the page.

You will give these `` elements a style that will set them apart from the images behind them by using a background color, a border, and fonts that all work well at smaller sizes:

```
h3 span {
  position : absolute;
  z-index : 2;
  padding : .15em .3em;
  background-color : #f3f4e4;
  border : 1px solid #4e5812;
  color : #333;
  font : bold 52% "Lucida Grande","Lucida Sans Unicode", Verdana, sans-serif;
  font-variant : small-caps;
  text-align : center; }
```

Next, you should give each `` element a unique position, placing them complementary to the images behind them. You can experiment with different compositions and even place some of the `` elements outside their parent containers (**Figure 4.29**):

```
#tulip h3 span { top : 160px; left : -20px; }
#lily h3 span { top : 40px; left : 220px; }
#blossom h3 span { top : 150px; left : 440px; }
#sunflower h3 span { top : 60px; left : 780px; }
```



4.29 Placing the `` elements inside and outside their parent containers

Hearing the fat lady warming up

The design layout is complete, but you don't want to stop there, do you? One of the reasons why the Web is such an exciting medium to design for is that, unlike a printed page, a Web page can and often should include interactive features to help visitors accomplish their goals on your site.

CSS can provide many interactive effects without needing to resort to scripting techniques. For the final part of this example, you'll add some subtle interactivity to this design.

The inline images you previously positioned at the top of this design are each wrapped in an anchor that contains a fragment identifier that points to its parent, a named list item:

```
<a href="#tulip"></a>
```

Now it is time for you to put those fragment identifiers to good use by employing the CSS3 `:target` pseudo-class. This handy pseudo-class enables you to alter the styling of any element that is the target of a link. For this design, you will reverse the contrast of a targeted list item by changing its background and text color (**Figure 4.30**):

```
:target {  
background-color : #4e5812;  
color : #fff; }
```

Ride of the Valkyries

So now that the lights have dimmed and the orchestra is playing, the fat lady is on her way to the stage. If the prospect of listening to an opera that lasts several days has little appeal to you, don't run for cover. CSS3 is about to take you on a ride that will be far more exciting than *Ride of the Valkyries*.

In the next section, you'll learn about CSS3 and many of the cool features it will bring to your design for the Web. "Ah!" you might be thinking, "I can stop reading here, because CSS3 will be a long time coming." Well, put that those thoughts out of your mind because not only will you learn how some of the most interesting new design possibilities will work with CSS3 but you'll also see solutions to emulate them in your work today.



4.30 Using the pseudo-class to change the styling of an element



CSS3 (Third Time Lucky)

As Web designers turn their backs on old-fashioned, presentational layout methods and see the advantages of minimal, meaningful markup and CSS, they recognize that to achieve more complex, rich interfaces, they need more from CSS. Not only should the new CSS specification build on what has gone before, making it easy to learn and more backward compatible, but it should also provide new features for designers to solve their everyday problems. The journey to improved CSS, however, has not been an easy one.

The first CSS specification, CSS1, was published in 1996. Its successor, CSS2, was published less than two years later, and an updated CSS2.1 followed to address a number of errors and inconsistencies. CSS2.1 still remains, at the time of writing, a candidate recommendation despite that many browsers now support most of its features, and standards-savvy designers and developers have long been using CSS2.1 in their daily work.

Work on CSS3 started in 2000, but the progress of the World Wide Web Consortium (W3C) has seemed painfully slow. For Web designers and developers who have realized that CSS2.1 cannot easily accomplish the visually rich, complex interfaces and layouts that modern Web sites need, watching this slow process has been maddening. This is something that, as a recently invited expert to the CSS Working Group, I hope to influence for the better.

The sum of its parts

One of the major differences between CSS3 and earlier versions is that CSS3 is a modular specification. Because so many new features have been requested, the W3C's CSS Working Group decided to break down work on CSS3 into a number of separate modules:

Module name	Description
Selectors Module (www.w3.org/TR/css3-selectors/)	New, refined selectors will make it easier to target an element based on its attributes and position in the document flow. New pseudo-classes and pseudo-elements will make it possible to achieve more typographic effects without adding presentational elements to your markup.

continues

Module name	Description
Paged Media module for printed publications (www.w3.org/TR/css3-page/)	CSS was always intended to do more than simply style the appearance of a document in a Web browser. Generated content for paged media focuses on styling documents for paper publishing and uses generated content to add notes, leaders, markers, and footnotes. Although many modern browsers support generated content, it is missing from Internet Explorer 7 and may not be a priority for implementation in a future version 8.
Backgrounds and Borders module (www.w3.org/TR/css3-background/)	The Backgrounds and Borders module offers designers new ways to style any box's background or borders. It includes a new way to attach more than one background image to an element and to use images to create borders.
Multi-column Layout Module (www.w3.org/TR/css3-multicol/)	Flowing text into multiple columns is a technique more familiar in print than on the Web. The Multi-column Layout Module is designed to make it simpler to create columns without additional markup by using column counts, gaps, and rules.
Advanced Layout Module (www.w3.org/TR/css3-layout/)	The Advanced Layout Module is designed to solve many of the common layout problems that Web designers and developers face. It also aims to fully separate the visual layout order from a document's content.
Media Queries module (www.w3.org/TR/css3-mediaqueries/)	Screen, print, and handheld are three media types that should already be familiar to Web designers and developers. Media queries extend the functionality of these media types when used in combination with other information, such as the width or height of a browser and even the aspect ratio of a screen; this is useful for developing for sites that will be viewed on a TV.

These modules are currently being worked on individually and are at different stages of completion. It is the Working Group's intention that browser makers will be able to choose which modules they will support and when they will implement them.

Getting involved in making new standards

Not only is progress on CSS3 slow, but another problem for those who are interested in helping shape the future of CSS is that the working drafts and documents for CSS are almost impenetrable to Web designers.

Web designers are mostly visual thinkers, and they focus on what they can achieve when using CSS, rather than on the intricate technicalities of the specifications. Much of the language used in specifications, and other W3C documentation, is scientific and complicated and does not lend itself to being easily understood by those without a background in science or academia.

The layout algorithm distinguishes the case of an element of a-priori known width and a shrink-wrapped element. In the former case, the target width of the template is the width of the element itself; in the latter case, the target width is the width of the initial containing block (often the viewport).

—The Advanced Layout Module working draft (www.w3.org/TR/css3-layout/#colwidth)

Hmmm. Answers on a postcard to...

Another problem with the specifications is that without clear examples of the visual effects that CSS3 is designed to accomplish, Web designers have a difficult job in visualizing how new selectors and properties will apply in their work. Many of the current examples used throughout the various modules have little in common with practical realities.

In fairness to the W3C, specifications are technically designed to be read and understood by browser makers and other implementers, rather than to act as training manuals for Web designers or developers. However, strong visual examples would not only help implementers understand how a feature should be implemented, but they would also serve as visual aids for designers and developers who may struggle to understand much of the specifications' complex terminology.

The W3C should not develop ideas for the specifications without input from working designers and developers who use their tools every day. Specification development should be a three-way partnership between designers, implementers, and the W3C. But if more Web designers and developers are going to help the W3C's CSS Working Group create tools that will be useful in their everyday work, the W3C must start working hard to write documentation that can be more widely understood.

Note

You can read more about the new CSS3 selectors in my article “A Tribute to Selectors” at www.stuffandnonsense.co.uk/archives/css_a_tribute_to_selectors.html and in Roger Johansson’s excellent article “CSS3 Selectors Explained” at www.456bereastreet.com/archive/200601/css_3_selectors_explained/.

Back to the future

CSS3 offers many tools to enable Web designers and developers to create highly visual designs without needing presentational markup.

It is understandable that browser makers have been reluctant to implement much of CSS3 until either the specifications have been finalized or there is widespread demand for them to do so. However, some parts of CSS3 are already supported, albeit in a limited capacity, by certain browsers. Where it is relevant and possible, you should take advantage of these tools now, if only so you can become familiar with how they work.

Not enough pages are available in this book to cover all the exciting developments in CSS3, so I have chosen to concentrate on some of the most interesting design opportunities in CSS3: selectors, background images, columns, and finally the Advanced Layout Module, one of the most exciting and intriguing developments to come out of the CSS Working Group.

Designing with the CSS3 Selectors Module

Web designers and developers have been asking for more efficient ways to target either an element or a node in the document tree for styling, and the CSS3 Selectors Module certainly does not disappoint. CSS3 offers so many new and powerful selectors that understanding how and when to use them can be daunting. The new selectors include the following:

- New attribute selectors that will enable you to target an element based on only part of its attribute, including `href`, `src`, `alt`, and `title`
- New dynamic pseudo-classes, including `:target` and `:lang`
- New structural pseudo-classes, including `:nth-child`, `:last-child`, `:only-child`, and even `:first-of-type` and `:last-of-type`

I can see ways that almost all the new CSS3 selectors will improve the lives of Web designers and developers in the future. In the following section, you’ll learn how to work with one of the most helpful new selectors to solve an alarming common problem. No, it’s not what to do when your Apple iPod runs out of battery; it’s how to make zebra-striped tables and lists.

Improving readability with zebra stripes

When you leave tables for layout behind and use them only for presenting tabular information, you emphasize their meaning. Many designers choose to help the readability of this tabular information by giving different styling to alternate rows, often simply by changing the color of the background. In the past, this simple design device would have required you to add a presentational `class` attribute to every other table row:

```
<table class="discography">
<thead>
<tr>
<td>Album</td>
<td>Year</td>
<td>Chart position (<abbr title="United Kingdom">UK</abbr>)</td>
</tr>
</thead>

<tbody>
<tr class="odd">
<td>Paul Weller</td>
<td>1992</td>
<td>8</td>
</tr>

<tr class="even">
<td>Wild Wood</td>
<td>1993</td>
<td>2</td>
</tr>

<tr class="odd">
<td>Live Wood</td>
<td>1994</td>
<td>13</td>
</tr>
</tbody>
</table>
```

And the CSS would be:

```
tbody > .odd { background-color : #fff; }
tbody > .even { background-color : #000; color : #fff; }
```

Plugging the holes in `:nth-child` support with JavaScript and the DOM

With browser support for `:nth-child` structural pseudo-classes still rare (at the time of this writing, only Konqueror 3.52 for Linux supports them), many Web designers and developers have turned to DOM scripting to emulate them.

Using JavaScript, you can insert a `class` attribute of `even` on even-numbered items in a series and style that element using a simple CSS class selector:

```
li.even {
background-color : #d0d0b0; }

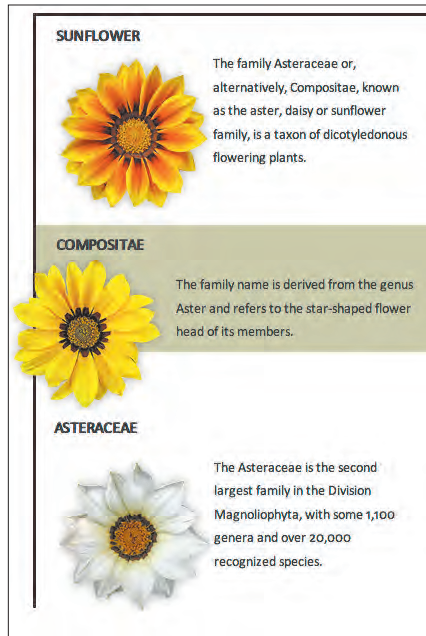
li.even img {
position : relative;
left : -50px;
margin-right : -40px; }
```

Until more browsers support the `:nth-child` pseudo-classes, DOM scripting can be an effective method of creating striping and other visual effects without resorting to presentational markup.

Note: Aaron Gustafson has written a flexible and easily updatable script for striping table rows and list items. Download the script and all supporting files exclusively at <http://easy-designs.net/code/stripey/>.

Album	Year	Chart position (UK)
Paul Weller	1992	8
Wild Wood	1993	2
Live Wood	1994	13

4.31 Helping readability of tabular information



4.32 Alternating-color list items

Not only can this quickly become tedious to edit manually, it can also be a difficult effect to achieve when the information in the table is dynamically generated.

This effect is not appropriate only for table rows. Perhaps you would like to style every other product in an online store's product page or even every second link in a sidebar (Figure 4.31).

The :nth-child() pseudo-class

CSS3 provides a way for you to target odd and even table rows, list items, or other elements in a series by using `:nth-child` pseudo-classes. These are highly useful when you are designing for a series of items on a page and you need to provide a way to separate them visually.

For this example, you'll create an attractive sidebar with alternating-color list items (Figure 4.32). By thinking first about content and meaning, you can keep the markup for this sidebar minimal and meaningful. Because the items in the sidebar list will be ordered by their popularity, an ordered list is the most appropriate element to choose. Each list item contains the name of the featured item, a short summary of it, and an image:

```
<ol id="nav_sub">
<li>
<h3>Sunflower</h3>
<p>
The family Asteraceae or, alternatively, Compositae, known as the aster, daisy
or sunflower family, is a taxon of dicotyledonous flowering plants.</p>
</li>
</ol>
```

Simple top and left borders on this ordered list separate the sidebar content from other elements on the page, and they also visually emphasize the effect of the images breaking free from their containers:

```
ol {
list-style-type : none;
width : 400px;
border-top : 3px solid #3f080a;
border-left : 3px solid #3f080a; }
```

Floating and setting a right margin on each image will enable the text within each of the list items to wrap around them:

```
li { clear : both; }

li img {
float : left;
border : none; }
```

Adding styles for the headings and paragraphs in the list will complete the basic sidebar:

```
h3, p {
margin : 0;
padding : .5em 20px; }

h3 {
font-size : 110%;
text-transform : uppercase; }
```

CSS3 `:nth-child` structural pseudo-classes enable you to style odd and even items without attaching any presentational class attributes in your markup. As is often the case with CSS, you have more than one way to accomplish the same goal, but by far the simplest method is to use `:nth-child(odd)` and `:nth-child(even)` selectors.

The following rule applies a background color only to even-numbered list items, creating the strong horizontal bands that are important in this design:

```
li:nth-child(even) {
background-color : #d0d0b0; }
```

Finally, you can create the effect of the images breaking out of their boxes by using relative positioning to move every image in an “even-numbered” list item 50 pixels to the left. Adding a negative right margin will suck the surrounding text into the space that this image would have occupied:

```
li:nth-child(even) img {
position : relative;
left : -50px;
top : -20px;
margin-right : -40px; }
```

CSS computation

If, after reading this, you feel a strong urge to dive into the specifications for structural pseudo-classes, you might soon be wondering if you have stumbled across a lesson in mathematics by mistake. As well as the more self-explanatory `:first-child`, `:last-child`, and `:nth-child` pseudo-classes, you can target specific elements based on the number of siblings that have come before them.

Imagine you have a long table of data with more than fifty rows; it will be difficult for a visitor to scan this table to find the specific piece of information he needs.

`tr:nth-child(10n-1)` will count the number of rows in increments of 10 (10, 20, 30, and so on) and target the rows that come immediately before (-1), enabling you to style the 9th, 19th, 29th, and so on, rows. You might choose to add a thick bottom border and extra white space to break the table into sections to help readability.

Designing with the Backgrounds and Borders Module

I've been working hard, all hours of the day and night. My eyes are bloodshot, and my fingers are raw, all because I want to invent a time machine—not your average time machine, no twinkling lights or shiny buttons for me (although a time machine that is shaped like a blue police box would be cool). I want a CSS time machine.

Where or when would I go? To the future perhaps, to find out the results of next year's racing results or the winner of the 2010 World Cup (actually I know that already)? No, I would return to 1996 and insist that the CSS Working Group add multiple background images to CSS1.

Attaching more than one background image to any element is on the wish list of almost every CSS-savvy designer I know. Alas, until CSS3 it has been possible to use only a single [image](#) per element.

Web designers and developers have worn their fingers to the bone concocting ways to implement even the simplest design element that requires more than one image. For such a simple effect, these solutions have become increasingly complex.

Like three-button jackets with side vents, fishtail parkas, and Fred Perry shirts before them, rounded corners are the fashionable must-have for many a “mod”ern Web site or application.



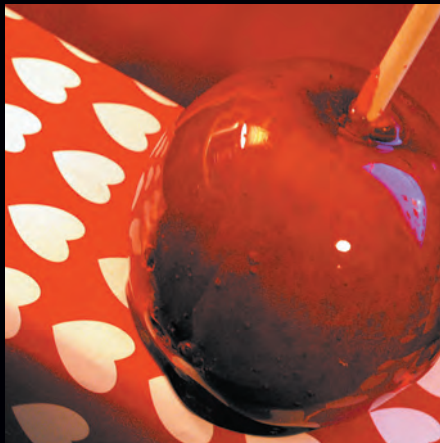
Adding rounded corners to a fixed-pixel-width element using CSS1 and CSS2.1 has been relatively straightforward; you do it usually by attaching a top image to one element and a bottom image to another. Creating a resizable box with rounded corners, custom borders, or drop shadows has always been a more complex affair, usually requiring additional `<div>` or `` elements to be added to your markup.

For example:

```
<div class="content_introduction">
<div class="bi">
<div class="bt">
<div></div>
</div>
<p>A flexible box with rounded corners</p>
<div class="bb">
<div></div>
</div>
</div>
```

This is hardly the most semantic use of divisions and one where the calorie count of your markup can easily weigh down the amount of content. CSS3 puts this markup on a diet by enabling you to attach more than one image to the background of an element. Your new, slim markup is as follows:

```
<p class="content_introduction">A flexible box with rounded corners</p>
```



Plugging the holes in multiple background images using JavaScript and the DOM

Rounded corners have become almost a standard design element in Web sites and application design; square corners just don't cut it anymore.

Rounded corners are not the only use for multiple background images, and to make them a possibility in a wider range of browsers, designers and developers have been turning to JavaScript to simulate multiple background images.

Among the many JavaScript solutions, one developed by Roger Johansson solves many problems by inserting additional elements into a document via the DOM. You can find Johansson's solution at www.456bereastreet.com/archive/200505/transparent_custom_corners_and_borders/.

That's it! It's lighter, healthier, and with almost 100 fewer characters. The CSS, on the other hand, is a little fatty. You can add several background images to your division, separating each image with a comma (**Figure 4.33**):

```
div.content_introduction { background-image :  
url("top_left.png"),  
url("top_right.png"),  
url("bottom_right.png"),  
url("bottom_left.png"),  
url("top_center.png"),  
url("middle_right.png"),  
url("bottom_center.png"),  
url("middle_left.png"); }
```

You can also set the repeat properties, taking care to use the same order for the repeat as for your list of images:

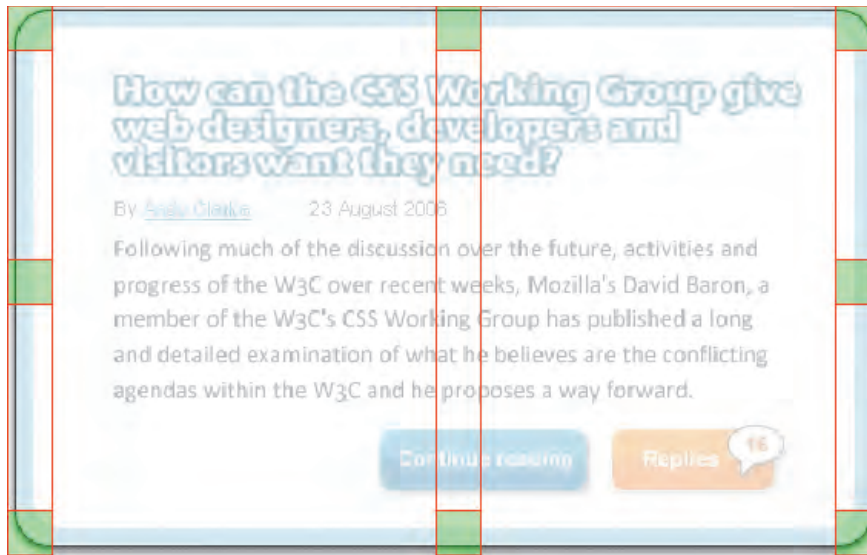
```
div.content_introduction { background-repeat :  
no-repeat, no-repeat, no-repeat, no-repeat, repeat-x,  
repeat-y, repeat-x, repeat-y }
```

Finally, you can position each background image to create the effect of a flexible, resizable box (**Figure 4.34**):

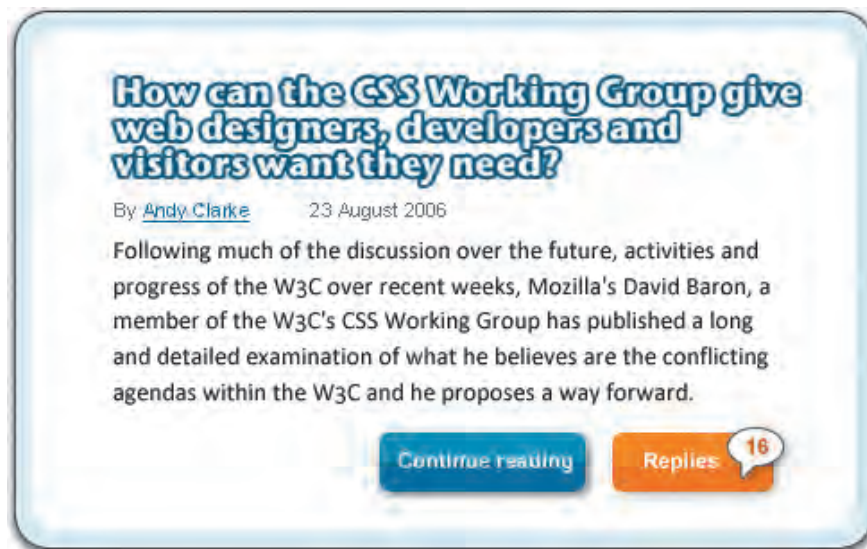
```
div.content_introduction { background-position:  
top left,  
top right,  
bottom right,  
bottom left,  
top left,  
top right,  
bottom right,  
bottom left; }
```

Phew! It's not quite the slimmest CSS, but it keeps the presentation information where it belongs, in a style sheet rather than in your markup, bulking it up with high-calorie "divness."

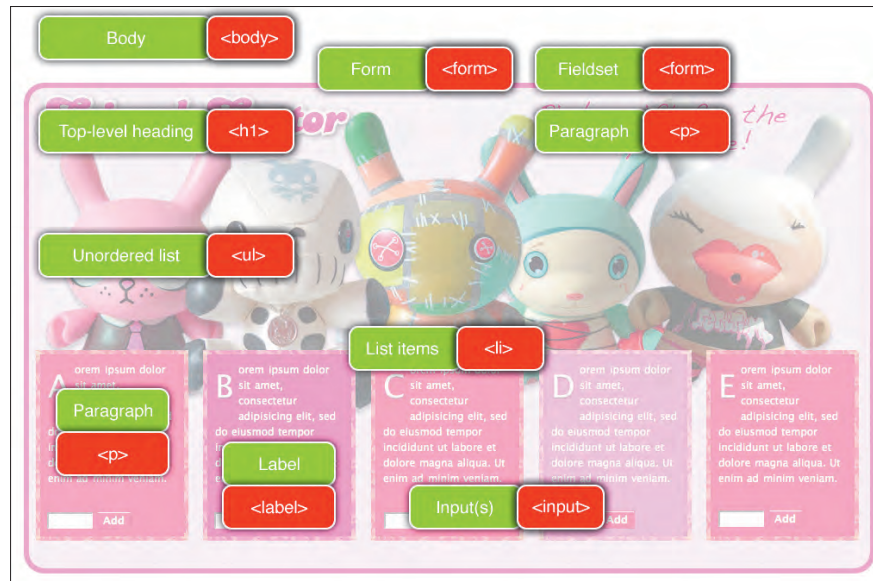
Note: At the time of writing, only those health-conscious folks at Apple have made multiple backgrounds available in Safari and other browsers based on the WebKit engine.



4.33 Adding multiple background images



4.34 Creating a flexible box



4.35 The final layout (top) and the markup (bottom)

Designing with multiple background images

Whatever platform you are working on—Windows, OS X, or Linux—you will be able to work along with most of the techniques in the next example (**Figure 4.35**). However, as this book goes to press, only Mac users running Safari and other browsers based on WebKit will be able to see the full benefits of multiple background images. If that’s not a good enough reason to switch to a Mac, I don’t know what is.

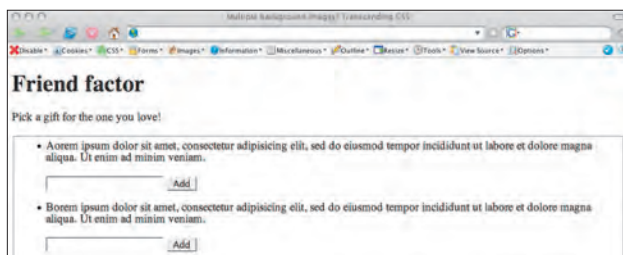
For this example, you’re designing an interface element for a gift site that has been inspired by a teen magazine. You will use percentage and em measurements to create a flexible design and use positioning, image replacement, and multiple background images to create a different type of e-commerce layout.

On the opposite page is the static design you are aiming to achieve, plus the meaningful elements you will use to mark up your content. Look closely, and you will see you need headings, paragraphs, a form, and a single unordered list. Look Ma, no divisions! What you can’t see, but is important for e-commerce stores, is that the total combined “weight” of the markup and CSS is only 8Kb.

Before you get carried away with this design, you’ll preview the naked document in a browser (**Figure 4.36**).

You’ll set the stage for the design by applying some simple rules to both the root `<html>` and `<body>` elements. Because this design will be flexible, all the measurements you’ll use are based on `em`:

```
html {
padding : 2em 0;
background-color : #fff;
color : #333; }
```



4.36 Previewing the naked document first

Note

Windows users, you can test your own examples in Safari and just about any other browser by using BrowserCam, a subscription service that takes screen shots of your pages in just about any browser and operating system combination. Find out more at www.browsercam.com.

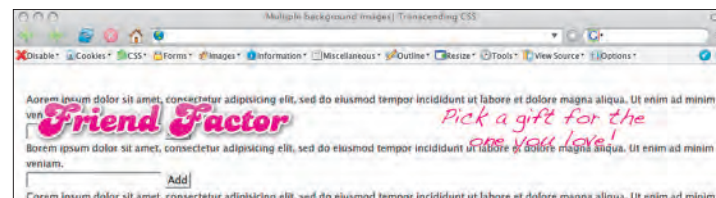
```
body {
font : 78%/1.5 "Lucida Grande","Lucida Sans Unicode", Verdana, sans-serif;
width : 66em;
min-width : 710px;
margin : 0 auto; }
```

Because you'll use absolute positioning to place the top-level heading and tagline, you will need to add `position : relative;` to the `<body>` element to establish it as the first positioning context:

```
body {
position : relative; }
```

Positioning and z-index

Positioning and `z-index` are two of the main techniques making this design possible. The first part of the positioning process is to place the top-level heading and tagline in position. You will then use the Phark image replacement technique discussed in Part 2, "Process," to replace the browser text with graphic images (**Figure 4.37**):



4.37

Placing the top-level heading and tagline before replacing the browser text with images



```
h1 {  
  position : absolute;  
  z-index : 2;  
  top : 10px;  
  left : 10px;  
  width : 375px;  
  height : 65px;  
  background : url(h1.png) no-repeat 0 0; }
```

```
h1 + p {  
  position : absolute;  
  z-index : 2;  
  top : 20px;  
  right : 0;  
  width : 395px;  
  height : 70px;  
  background : url(p.png) no-repeat 0 0; }
```

```
h1, h1 + p {  
  text-indent : -9999px; }
```

Because you will position many of the elements within the form, the form must also become a positioning context by adding `position : relative;` but no offsets:

```
form {  
  position : relative;  
  z-index : 1;  
  padding : 0 1em;  
  min-height : 38em;  
  background-color : #f9e6f6; }
```



Z's not dead, baby; Z's not dead

In combination with alpha-transparency in PNG images, stacking elements with `z-index` is a powerful creative tool. Remember geometry at school? The *x*-axis represents the horizontal; the *y*-axis represents the vertical. In CSS, the *z*-axis represents depth. Elements that are stacked using `z-index` are arranged from front to back. It is also important to understand that `z-index` is applied to only those elements with a `position` property and a value—in other words, no positioning property and value, no `z-index`.

`z-index` values can be either negative or positive, and the element with the highest value appears closest to the viewer, regardless of its order in the source. If more than one element has the same `z-index`, the element that comes last in the source comes out on top of the pile.

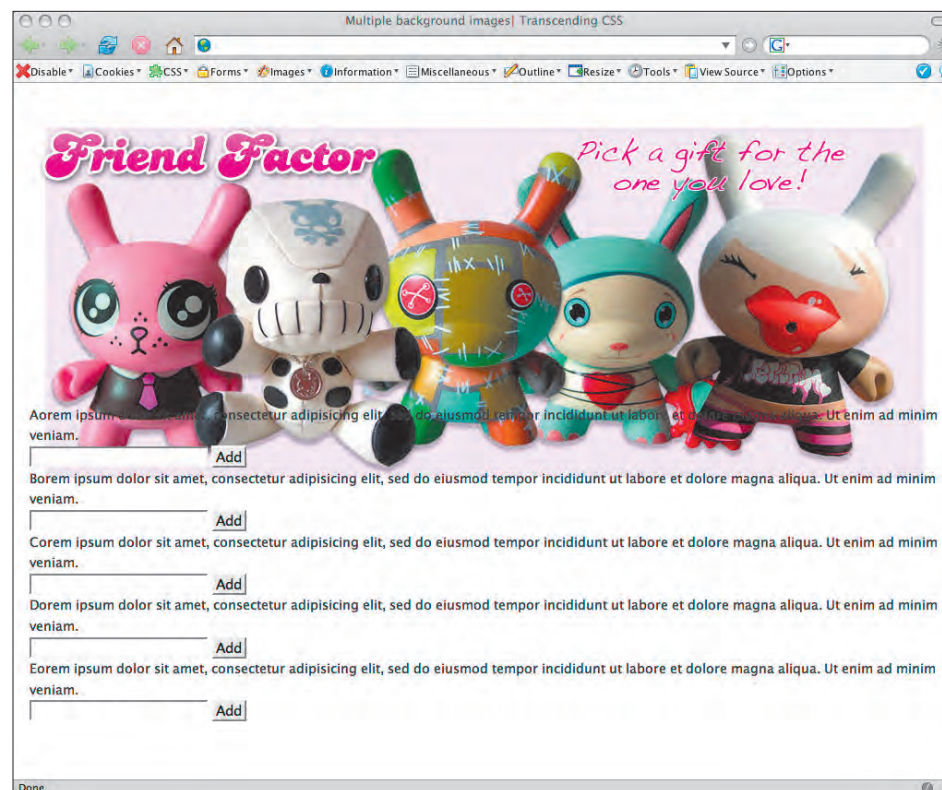
Note: For a more detailed examination of `z-index`, read my article at <http://24ways.org/advent/zs-not-dead-baby-zs-not-dead>. More about complex `z-index` relationships from Aleksandar Vacic at <http://aplus.co.yu/css/z-pos/>.

You will now position the unordered list 10 pixels from the top of the form and use a combination of minimum height and padding to allow room for the montage of characters:

```
ul {  
  position : relative;  
  top : 10px;  
  min-height : 80px;  
  padding-top : 300px; }
```

It's time to complete the unordered list by attaching the characters image to its background. Because this image sits behind the positioned list items, the design takes on an unusual out-of-the-box appeal (**Figure 4.38**):

```
ul {  
  background : url(ul.jpg) no-repeat 50% 0; }
```

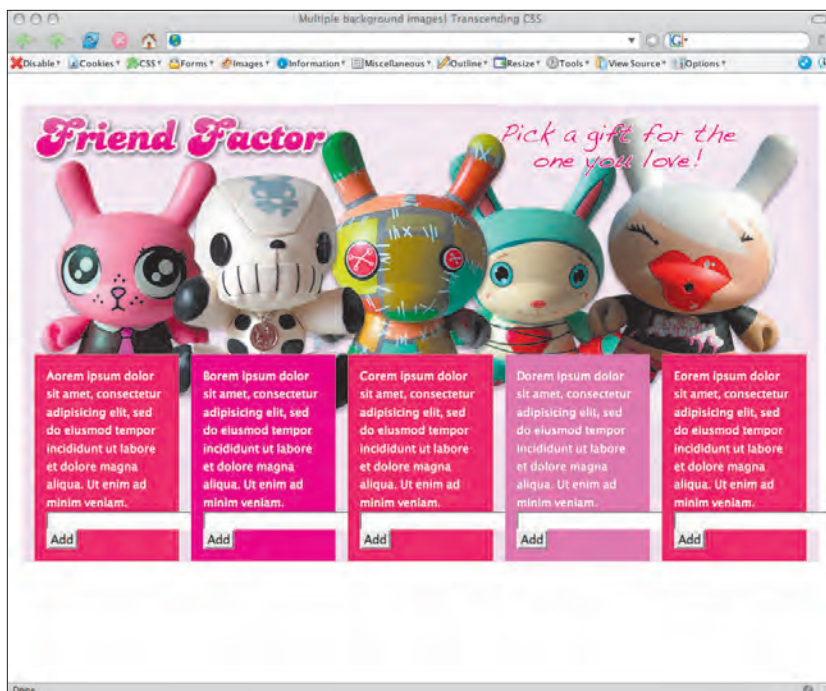


4.38 Attaching the characters image to its background

You'll now position each of the five list items to create the appearance of five columns. I have calculated their positioning to add a small gutter between them:

```
li {  
  position : absolute;  
  width : 10em;  
  padding : 1em;  
  background-color : #f9496b;  
  color : #fff; }  
  
li[id="a"] { left : 0; }  
li[id="b"] { left : 13em; background-color : #f185bb; }  
li[id="c"] { left : 26em; }  
li[id="d"] { left : 39em; background-color : #f185bb; }  
li[id="e"] { left : 52em; }
```

The layout of this design is beginning to take shape, so press on and preview the design in your browser (**Figure 4.39**).



4.39 Previewing the design

Note

For more on styling form buttons, see Aaron Gustafson's "Push My Button" article for *Digital Web Magazine* at www.digital-web.com/articles/push_my_button/.

Form element styling

To add a little flair to each of the form elements, it is time to pull out one of the most flexible selector types in the CSS designer's toolbox, an attribute selector. Peeking into the markup, you will see that each of the list items contains a text input for the quantity and an "Add to cart" button:

```
<label for="a_qty"><span>Add to cart</span>
<input type="text" id="a_qty" />
<input type="submit" value="Add" />
```

Without attribute selectors, you would need to target each form element either by using its `id` (requiring multiple CSS selectors) or by adding a `class` attribute such as this:

```
<label for="a_qty"><span>Add to cart</span>
<input type="text" id="a_qty" class="qty" />
<input type="submit" value="Add" class="submit" />
```

Attribute selectors remove the need for this extra markup. They target inputs based on their `type` and their `value` (**Figure 4.40**):

```
input[type="text"] { width : 4em;
font : 82% "Lucida Grande", "Lucida Sans Unicode", sans-serif; }

input[value="Add"] {
padding : 0 .25em;
color : #fff;
border : 2px double #9c2f45;
border-top-color : #fff;
background-color : #f9496b;
font : bold 82% "Lucida Grande", "Lucida Sans Unicode", sans-serif; }
```

Typographic flair

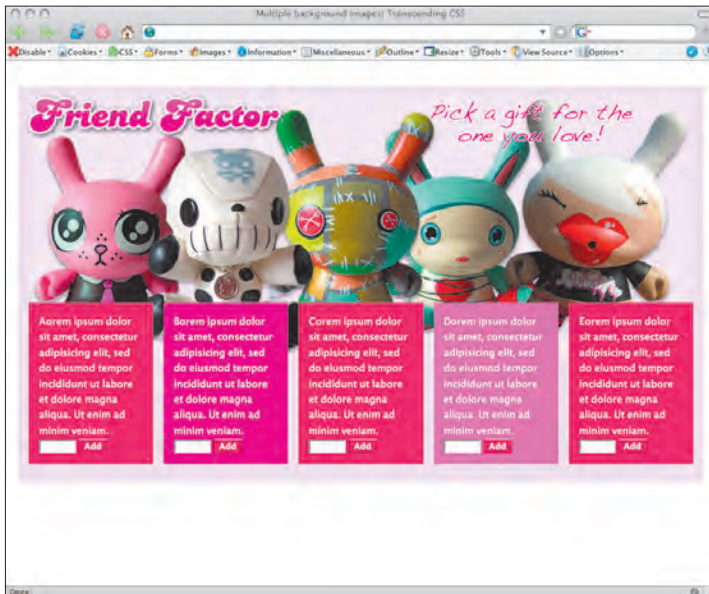
One of the aspects from the inspirational magazine design you are aiming to replicate in this interface is a drop cap. The `:first-letter` pseudo-element will style this first letter, enlarging the text size and floating it to enable the neighboring text to wrap:

```
li > p {
min-height : 14em;
font-size : 82%; }
```

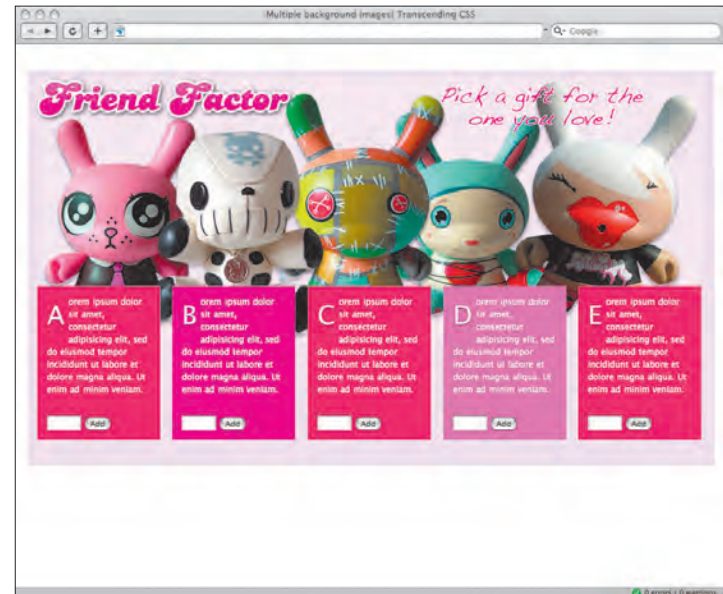
```
li > p:first-letter {
float : left;
margin : 0 .15em 0 0;
font-size : 300%;
text-transform : uppercase; }
```

Text shadow is a CSS3 property that you can use to add a little more typographical flair to your designs. Sadly, at the time this book was rolling off the presses, Apple Safari and Konqueror are the only browsers that support this useful property (**Figure 4.41**):

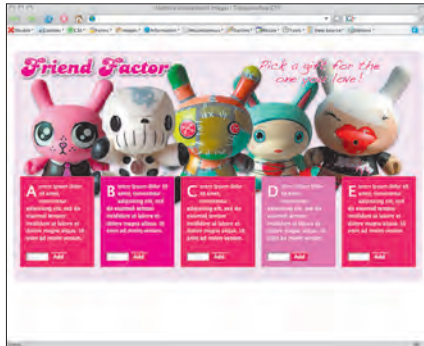
```
li > p:first-letter {
text-shadow: #333 0 1px 2px; }
```



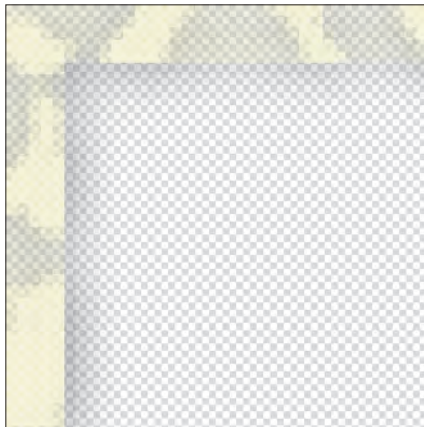
4.40 Using attribute selectors



4.41 Using text shadow, a CSS3 property, for more flair



4.42 Viewing the design before adding multiple background images



4.43 Keeping your images to a minimum using PNG alpha-transparency

Multiple background images

This section is the part for which you've probably been waiting. But before you move on, it's a good idea to preview your design in a browser, and, yes, I mean any standards-aware browser. Until support for multiple background images extends beyond Safari, this is how your design will appear to the multi-background challenged (**Figure 4.42**).

Speed is of the essence when developing any site and can be particularly important when developing sites that sell. You can improve speed and performance in many ways, such as by minimizing document size, image sizes, and the quantity of HTTP requests.

Using multiple background images solves many of the problems that designers making flexible boxes have encountered, but you must take care to keep the number of images you use to a minimum. For this design, you have only two sets of background images required, one set for the form and another that is common to all the list items (**Figure 4.43**).

To reduce the number of images, you will create a single set of semitransparent background images. These alpha-transparent PNG images allow 50 percent of the list item's `background-color` property to show through, creating the effect of using several sets of images.

You will start by adding the eight images making up the form's pink, rounded, and friendly border (**Figure 4.44**):

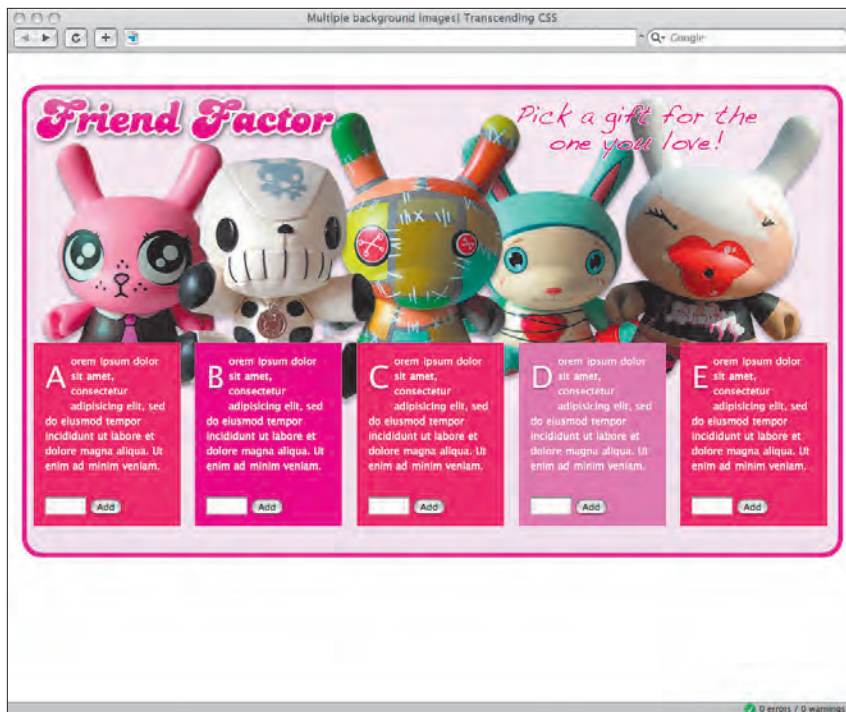
```
form { background-image :
  url("top_left.png"),
  url("top_right.png"),
  url("bottom_right.png"),
  url("bottom_left.png"),
  url("top_center.png"),
  url("middle_right.png"),
  url("bottom_center.png"),
  url("middle_left.png");
```

```
background-repeat :
  no-repeat,
  no-repeat,
  no-repeat,
```



```
no-repeat,  
repeat-x,  
repeat-y,  
repeat-x,  
repeat-y;
```

```
background-position:  
top left,  
top right,  
bottom right,  
bottom left,  
top left,  
top right,  
bottom right,  
bottom left; }
```



4.44 Adding multiple images to the form

Note

If you haven't had the time yet to create this design for yourself, don't worry; I have saved you the trouble. You can find all the files you need for this example at www.transcendingcss.com/support/.

Follow this with the same syntax for all the list items:

```
li { background-image :  
  url("li_top_left.png"),  
  url("li_top_right.png"),  
  url("li_bottom_right.png"),  
  url("li_bottom_left.png"),  
  url("li_top_center.png"),  
  url("li_middle_right.png"),  
  url("li_bottom_center.png"),  
  url("li_middle_left.png");
```

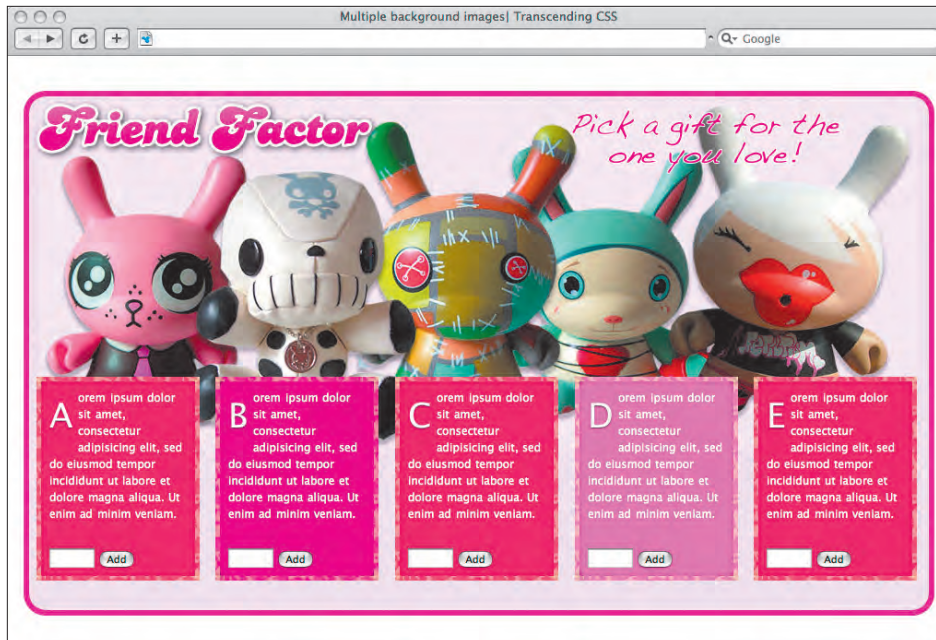
```
background-repeat :  
no-repeat,  
no-repeat,  
no-repeat,  
no-repeat,  
repeat-x,  
repeat-y,  
repeat-x,  
repeat-y;
```

```
background-position:  
top left,  
top right,  
bottom right,  
bottom left,  
top left,  
top right,  
bottom right,  
bottom left; }
```

If you have a Mac handy, it is time to preview your design in Safari (**Figure 4.45**). If you are not a lucky Mac owner, a swift upload to a server and a trip to BrowserCam will show you the results: a fun, rounded form design.

The notion of multiple background images seems so obvious that I sometimes wonder why it has taken so long to find its way first into CSS and then into browsers.

I can only hope that the developers of Internet Explorer, Firefox, Opera, and other browsers follow Apple's example and make support for CSS multiple background images a priority.



4.45 Previewing the results of adding multiple background images

Designing multicolumn layouts

As you saw in Part 3, “Inspiration,” dividing text into columns has been a common technique in many forms of design. It is one that helps the readability of written content by limiting the length of lines of text.

On the Web, splitting large blocks of text into multiple columns has always been problematic and has so far required that Web designers break their text into extra divisions to form visual, rather than semantic, groupings of content:

```
<div id="content_main" class="column">
Main content
</div>
```

```
<div id="content_sub" class="column">
Additional content
</div>
```

Note

Need a reminder of the difference between a block and an inline element? Tommy Olsson has written an excellent tutorial at www.autisticcuckoo.net/archive.php?id=2005/01/11/block-vs-inline-1.

Multicolumn layout considered harmful?

Since the CSS Working Group first announced the Multi-column Layout Module, some designers and accessibility experts have been critical of its potential effects on readability. It is true that when it is used inappropriately, spreading text across multiple columns can force a visitor to repeatedly scroll up and down to follow the flow of an article. Roger Johansson wrote this:

Too many designers value “creativity” above readability, usability, and accessibility. Using multiple columns in a print stylesheet may be useful, but on-screen, for longer articles? No. Face it, the Web is not a printed magazine.

—Roger Johansson (www.456bereastreet.com/archive/200509/css3_multicolumn_layout_considered_harmful/)

```
<div id="content_supp" class="column">
Supplementary content
</div>
```

The CSS3 Multi-column Layout Module is attempting to rectify this problem by making creating columns easier. It introduces a new column box that you can apply to paragraphs, lists, divisions, and other block-level elements.

Column widths and count

You can implement multiple columns within a column box in two ways. The first is by dictating the number of columns you require using the `column-count` property:

```
div#content_main { column-count : 3; }
```

The width of all three columns will expand equally to fill the available horizontal space of the containing element. Alternatively, you can set your desired width for the columns, and the browser will calculate how many columns will fit into the available space:

```
div#content_main { column-width : 15em; }
```

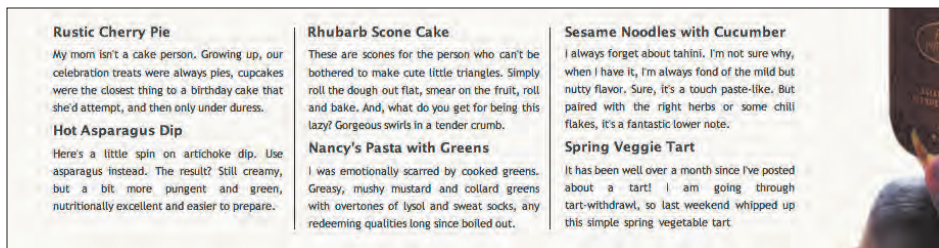
The ability for content to flow from column to column and the ability for new columns to be created as the browser window expands are effects that have not yet been achievable using CSS alone.

Gutters, gaps, and rules

To improve readability and visual balance, the Multi-column Layout Module introduces two new properties, `column-gap` and `column-rule`. To use these features, you insert both *gaps* (gutters) and *rules* (dividers) between columns, and their heights will always be equal to the height of the columns.

You place a column rule in the middle of a column gap. These rules do not take up space. That is, the presence or thickness of a column rule will not alter the placement of either columns or gaps (**Figure 4.46**):

```
div#content_main {
column-gap : 1em;
column-rule : thin solid black; }
```

4.46 Placing a column rule in the middle of the column gap

Note: At the time this book went to press, parts of the Multi-column Layout Module are supported only in the Firefox 1.5+ and its siblings using the standards-compliant prefix `-moz-` prefix. (According to the spec, proprietary CSS properties are allowed, but must be prefaced with a hyphen and an identifier; `-moz-`, in this case. These include `-moz-column-count`, `-moz-column-width`, and `-moz-column-gap`.) Find out more about Multi-column Layout support in these browsers at http://developer.mozilla.org/en/docs/CSS3_Columns.

Multi-column thrills or spills?

I am definitely thrilled by certain aspects of the Multi-column Layout Module but am distinctly underwhelmed by others. The ability for content to flow from column to column and the ability for new columns to be created as the browser window expands make Multi-column Layout an exciting prospect when used under the right circumstances.

However, many parts of the current working draft are less than ideal, and a great deal of work needs to be done on the specification before it should become a recommendation.

From both the design and usability perspectives, the results of content reflowing can be unsatisfactory; for example, content can easily become separated from its associated headers, and images can be separated from their descriptions or other associated text (**Figure 4.47**).

Fortunately, a group of new properties determine where column breaks occur:

```
h1 { column-break-before : always; }
h2 { column-break-after  : avoid; }
h1, h2 { column-break-inside : avoid; }
```

A gap in the proposals?

The current Multi-column Layout working draft allows only for the basic styling of `column-rule` including `dotted` and `dashed` styles and other values from the CSS2.1 border style list, including the unusable, ugly `ridge` and `groove` styles.

The CSS Working Group doesn't seem to be planning to implement a feature I imagine many designers would like to use: image rules. I can think of many instances where images would be the perfect choice for column rules. Perhaps the following *fictional* syntax might be appropriate:

```
div#content_main {
  column-rule-image : url(rule.
  png); }
```

Even better would be the possibility of positioning these images either at the top, bottom, or vertical center of the column gap:

```
div#content_main {
  column-rule-image : url(rule.
  png);
  column-rule-align : middle; }
```

Plugging the holes in the Multi-column Layout Module using JavaScript and the DOM

Designers and developers have turned to JavaScript to help them simulate some parts of CSS3, including the Multi-column Layout Module.

Developer and author of “Introducing the CSS3 Multi-Column Module” at A List Apart (www.alistapart.com/articles/css3multicolumn/), Cédric Savarese has developed an experimental JavaScript implementation of the CSS3 Multi-column Layout Module.

Find out more about Savarese’s ingenious solution at www.cssscripting.com.

Rustic Cherry Pie

My mom isn't a cake person. Growing up, our celebration treats were always pies, cupcakes were the closest thing to a birthday cake that she'd attempt, and then only under duress.

Hot Asparagus Dip

Here's a little spin on artichoke dip. Use asparagus instead. The result? Still creamy, but a bit more pungent and green, nutritionally excellent and easier to prepare.

Rhubarb Scone Cake

These are scones for the person who can't be bothered to make cute little triangles. Simply roll the dough out flat, smear on the fruit, roll and bake. And, what do you get for being this lazy? Gorgeous swirls in a tender crumb.

Nancy's Pasta with Greens

I was emotionally scarred by cooked greens. Greasy, mushy mustard and collard greens with overtones of lysol and sweat socks, any redeeming qualities long since boiled out.

Sesame Noodles with Cucumber

I always forget about tahini. I'm not sure why, when I have it, I'm always fond of the mild but nutty flavor. Sure, it's a touch paste-like. But paired with the right herbs or some chili flakes, it's a fantastic lower note.

Spring Veggie Tart

It has been well over a month since I've posted about a tart! I am going through tart-withdrawal, so last weekend whipped up this simple spring vegetable tart, with fresh, organic local asparagus, garlic shoots and morels that I picked up at Pike Place market.

4.47 Column reflowing can result in undesirable results

Rustic Cherry Pie

My mom isn't a cake person. Growing up, our celebration treats were always pies, cupcakes were the closest thing to a birthday cake that she'd attempt, and then only under duress.

Hot Asparagus Dip

Here's a little spin on artichoke dip. Use asparagus instead. The result? Still creamy, but a bit more pungent and green, nutritionally excellent and easier to prepare.

Rhubarb Scone Cake

These are scones for the person who can't be bothered to make cute little triangles. Simply roll the dough out flat, smear on the fruit, roll and bake. And, what do you get for being this lazy? Gorgeous swirls in a tender crumb.

Nancy's Pasta with Greens

I was emotionally scarred by cooked greens. Greasy, mushy mustard and collard greens with overtones of lysol and sweat socks, any redeeming qualities long since boiled out.

Sesame Noodles with Cucumber

I always forget about tahini. I'm not sure why, when I have it, I'm always fond of the mild but nutty flavor. Sure, it's a touch paste-like. But paired with the right herbs or some chili flakes, it's a fantastic lower note.

Spring Veggie Tart

It has been well over a month since I've posted about a tart! I am going through tart-withdrawal, so last weekend whipped up this simple spring vegetable tart, with fresh, organic local asparagus, garlic shoots and morels that I picked up at Pike Place market.

4.48 Using column break properties can result in additional problems

Unfortunately, these too can introduce problems, including large amounts of unnecessary white space or ugly and uneven column lengths (**Figure 4.48**).

Earlier working drafts of the Multi-column Layout Module included the now missing in action `column-span` property that would have enabled elements, such as headings, to span across a designated number of columns to create the effect regularly seen in newspaper design. Later drafts are now lacking this way to stop and restart columns.

The Multi-column Layout Module is an exciting prospect for designers, but to be completely useful, the CSS Working Group must talk to working designers about the features they think are missing and how they would put these features into everyday use.

Too many designers value “creativity” above readability, usability, and accessibility. Using multiple columns in a print stylesheet may be useful, but onscreen, for longer articles? No. Face it, the Web is not a printed magazine.

—ROGER JOHANSSON

www.456bereastreet.com/archive/200509/css3_multicolumn_layout_considered_harmful/



4.49 The design inspiration (top), the final layout (bottom), and the markup (right)

Designing with the Multi-column Layout Module

For the next example (**Figure 4.49**), you'll design an interface design for a cookery site. You will use percentage and `em` measurements to create a flexible layout, use a little `background-image` trickery, and use the Multi-column Layout Module to split the text content and create an attractive result.

On the opposite page is the static design you are aiming to achieve plus the meaningful elements you will use to mark up your content. Look closely, and you will see that you require headings, paragraphs, a single unordered list, and only two divisions. What you can't see is that the total combined weight of the markup and CSS is only 4Kb, hardly a heavyweight.

You should begin by setting the stage for the design by applying some simple rules to both the root `<html>` and `<body>` elements, allowing the design to fill 92 percent of the browser's width, down to a minimum of 640 pixels:

```
html {
background-color : #fff; }

body {
font : 78%/1.5 "Trebuchet MS", "Lucida Grande", "Lucida Sans Unicode", Verdana,
sans-serif;
width : 92%;
min-width : 640px;
margin : 0 auto;
color : #333; }
```

To allow room for the large product images at the top of the design, set `padding-top` to 360 pixels on the `<body>` element, and apply the image to the background. Setting its horizontal position at 50 percent ensures that it will always stay horizontally centered, no matter what the browser window size:

```
body {
padding-top : 360px;
background : url(body.png) no-repeat 50% 0; }
```

The design needs only two divisions: an outer container for the top-level heading and introduction text and a second for the main content that includes the all-important unordered list.

Note

If you haven't had the time yet to create this design for yourself, don't worry; I have saved you the trouble. You can find all the files you need for this example at www.transcendingcss.com/support/.

To make room for the image that runs down the right side of the page, set a right margin on the `#main-content` division to prevent its contents from overlapping this `background-image`:

```
div#content {  
width : 100%;  
background : url(content.png) no-repeat 100% 0; }
```

```
div#content_main {  
margin-right : 320px; }
```

Now, with your layout complete, a little sprinkling of typographic style will complete the design (**Figure 4.50**).



4.50 Finishing up the design

What about the multi-columns? Oops! I almost forgot. Creating these columns can't be much simpler. You will now apply the multi-column layout styles to the unordered list only so as not to affect any of the content that comes either before or after it in the document flow.

Because Gecko-based browsers have already implemented the new multi-column layout properties using the proprietary `-moz-` prefix, you will add two sets of style rules, one for today's Gecko browsers and one for other browsers when they begin supporting the Multi-column Layout Module:

```
ul {
padding: 1em 0;
column-width: 18em;
column-gap: 25px;

-moz-column-width: 18em;
-moz-column-gap: 25px; }
```

Now you have a flexible design with `em`-based column widths. Open the final result in a Gecko-based browser, and play with both the browser width and the text size. Watch as the list switches from a one-, two-, and three-column layout (Figure 4.51).

4.51 Going from a one-, to two-, to three-column layout





Advanced Layout

It has been ten years since the launch of CSS1, and visual designers have done more with CSS layouts than the early adopters ever would have thought possible. Compared to the earliest layout examples from The Noodle Incident and Blue Robot's Layout Reservoir, the designs featured every week on CSS gallery sites or planted in the CSS Zen Garden show just how far designers have taken CSS. In the hands of creative designers and Web developers, CSS is capable of producing stunning results.

Full CSS layouts have always been a compromise. The current CSS specifications were never designed to create the visually rich and complex interface layouts that the modern Web demands. The current methods—floats and positioning—were never intended as layout tools. The problems with using floats and positioning for layout go far beyond the fragility of floating an element to create a column and the fact that absolutely positioned are removed from the normal flow of a document. For a large part, both floated and positioned layouts depend on a document's source order. Although many designers have worked hard to create any-order columns, these types of layouts have almost always required a mass of additional divisions or hacks and filters to make them work reliably across different browsers.

Note: You can find many solutions to any-ordered columns at www.positioniseverything.net.

When a visual layout depends on source order, making larger changes to a design has always required changes to the underlying markup—hardly a true separation of content and style as promised by CSS. A full separation of content order from visual layout is not only desirable, but it is essential if designers are ever going to be able to move away from the familiar CSS layout with either two or three columns and create rich and inspiring designs without resorting to presentational markup.

Advanced Layout is one of the modules forming the CSS3 specification. It is being designed specifically to enable designers to break free of these conventions and many of the limitations of the past. It is perhaps one of the most interesting developments in CSS 3.

Acknowledging César Acebal

On a personal note, I would like to thank my co-contributor, César Acebal, and the University of Oviedo, Spain. Acebal is a member of the W3C's CSS Working Group and the author of the ALMCSS Advanced Layout proof of concept that made this section possible.

Back to the grid

The Advanced Layout Module builds on the concepts of grid design discussed in Part 3, “Inspiration.” It establishes a new visual grid model that will enable designers to determine the layout of forms, navigation, content divisions, or even an entire page. Advanced Layout divides these elements into slots and uses a simple set of letters to position any child element inside a slot in the grid.

Advanced Layout introduces a new `display-model` property that defines the number of horizontal fields within the areas of the grid with strings of letters.

For example, the following code:

```
display:
"abc"
"def";
```

will create two horizontal grid fields, each with three vertical slots. Two other values (`@` and `.`) define whether a slot is the default or contains only white space:

- **Slot letter.** This identifies the slot within the grid for any content that will be positioned within it.
- **@ (at symbol).** This identifies a default slot into which content that has not been situated can flow.
- **. (period).** This identifies a slot that can have no content inserted into it.

For more creative control over the grid, using the same letter more than once will force two or more slots to combine to form supercolumns. Any content placed inside them will span multiple columns, a design device that is common in newspaper layouts (**Figure 4.52**):

```
{ display:
"a a a"
"d e f"; }
```

In the next example, using floats to construct even this simple grid layout would require the elements to be ordered largely according to their visual layout. Using positioning would be a more reliable but more complex solution. Sadly, this solution could easily break if a visitor resized the browser window or changed the default size of the text.



4.52 Spanning multiple columns

Using the Advanced Layout Module, you can keep the markup needed for this example lean and structural (**Figure 4.53, next page**):

```
<div id="biscotti">
<p><a href="#"></a>
Moroccan Biscotti</p>
</div>
```

```
<div id="waffles">
<p><a href="#"></a>
Yeast Waffles</p>
</div>
```



4.53 Seeing only the browser default styles

```
<div id="muffins">
<p><a href="#"></a>
Oat Bran Carrot and Orange Muffins</p>
</div>
```

```
<div id="jelly">
<p><a href="#"></a>
White peach jelly</p>
</div>
```

```
<div id="bread">
<p><a href="#"></a>
Zucchini-walnut protein bread</p>
</div>
```

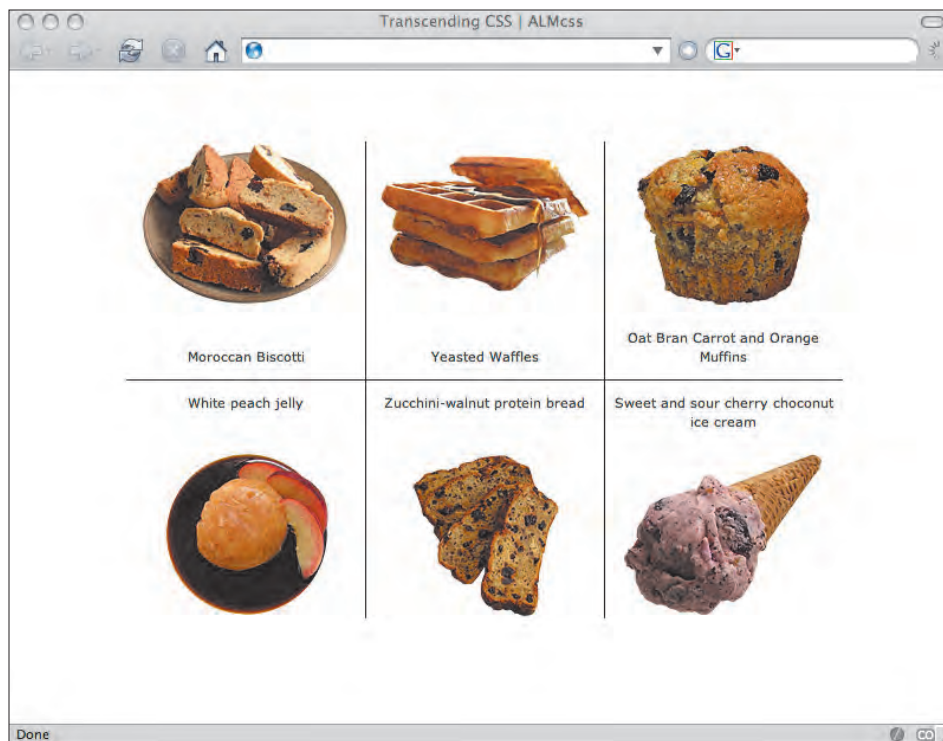
```
<div id="icecream">
<p><a href="#"></a>
Sweet and sour cherry choconut ice cream</p>
</div>
```

Also using Advanced Layout, you can define a grid either on a division or, as in this example, on the `<body>` element. Two strings will divide the `<body>` element into two fields, each containing three slots. You will also be able to dictate how the height of each slot will be defined: either by the height of the content inside it, termed *intrinsic height*, or by setting an explicit height in pixels or ems:

```
body { display:
"a b c (intrinsic)"
"d e f (intrinsic); }
```

With this simple CSS in place, each of the page elements will be situated in their slots by referencing each slot's identifying letter. The CSS is far less complex than any floating or positioning methods (**Figure 4.54**):

```
div#biscotti { position : a; }
div#waffles { position : b; }
div#muffins { position : c; }
div#jelly { position : d; }
div#bread { position : e; }
div#icecream { position : f; }
```

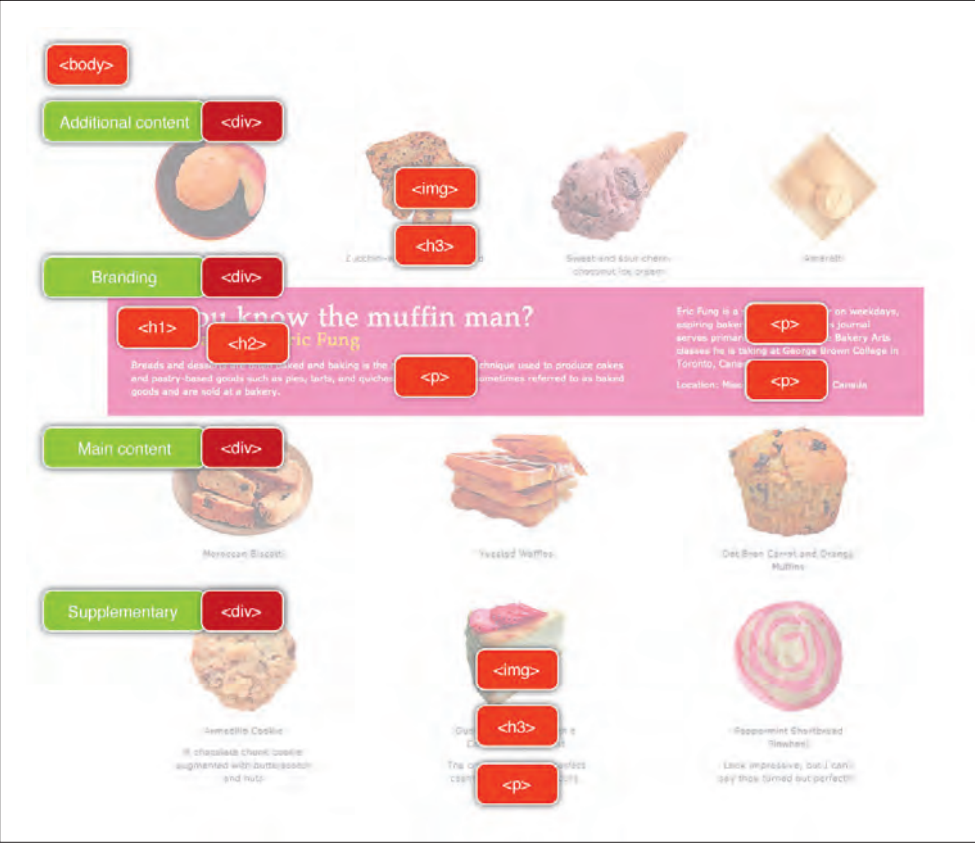



4.54 Situating content into slots

That's it! You don't have any floating worries or absolute positioning-clearing woes, just letters. It doesn't get much simpler than that.

The Advanced Layout working draft calls its grid-based approach *template-based positioning*, an unfortunate choice of words. Templating is rarely a concept that appeals to creative designers, but you should forgive the choice of name. Advanced Layout will allow a whole new realm of creative potential, not limit designers' creative options.

If you're hungry to see how easy it will be in the future to create layouts, check out the next section, where you'll see just how flexible the Advanced Layout Module will be.



4.55 The inspiration for the layout (top), the finished design (bottom), and the markup (right)

Designing with the Advanced Layout Module

On the opposite page (**Figure 4.51**) you can see the static design you are aiming to achieve using the Advanced Layout Module, plus the meaningful elements you will use to mark up your content.

Developing this layout using floats should, on the face of it, present few challenges; you would give each column an explicit width and float it inside its container division.

But what would happen if you needed to switch the position of any of the columns or perhaps even the vertical position of the fields? Most likely this would involve diving back into your markup and moving the elements so you could achieve the visual design. The Advanced Layout Module removes this necessity because it finally breaks any relationship between the visual design and the order of the content.

First, set up the four horizontal fields by creating four divisions. The order of these divisions should make sense to any visitor who is reading the content of the page without styles, and you should preview your document in a browser without styles to ensure that the order is logical:

```
<div id="branding">
Branding and introduction
</div>
```

```
<div id="content_main">
Main content
</div>
```

```
<div id="content_sub">
Additional content
</div>
```

```
<div id="content_supp">
Supplementary content
</div>
```

Note: César Acebal has made ALMCSS publicly available, and you can find links to all the files and Acebal's own examples at www.transcendingcss.com/support/.

Advanced layout in action using ALMCSS

The Advanced Layout Module is an exciting prospect, and many Web designers and developers (including me) have been aching to try it. Experimenting with new CSS features before the CSS Working Group has finalized a specification has been impossible until now.

As you might imagine, currently no browsers support any of the CSS for the Advanced Layout Module. This should come as no surprise because the module is, at the time of this writing, only a working draft. Fortunately, you can explore many parts of Advanced Layout by using ALMCSS, which is a proof of concept that uses JavaScript and CSS positioning to mimic how Advanced Layout will work.

As you will see in the following examples, to work around certain issues with CSS parsers in some browsers, ALMCSS uses a slightly different syntax than the proposals in the Advanced Layout Module: `display` becomes `display-model` and `position` becomes `situated`.

Note that the proposed syntax for the Advanced Layout CSS has been modified to work in this section's examples.



4.56 Seeing only the default browser styles

Second, in each of these divisions, create the columns using nested divisions that, in this example, contain a paragraph and an image but could just as easily contain any structural elements. For simplicity (some might say laziness), I have named these divisions *one*, *two*, *three*, and so on (**Figure 4.56**):

```
<div id="content_main">
  <div class="one">
    
    <p>Moroccan Biscotti</p>
  </div>
```

```
<div class="two">
  
  <p>Yeasted Waffles</p>
</div>
```

```
<div class="three">
  
  <p>Oat Bran Carrot and Orange Muffins</p>
</div>
</div>
```

Whereas using floats or positioning to accomplish the design layout might be convoluted, with the Advanced Layout Module the CSS is simple. Next, define the `display-model` property on the `<body>` element:

```
body { display-model :
  "a (intrinsic)"
  "b (intrinsic)"
  "c (intrinsic)"
  "d (intrinsic)"; }
```

You can also create a microgrid by giving each field its own `display-model` property to divide it into varying numbers of vertical columns:

```
div#branding { situated : a; display-model : "112 (intrinsic)";
div#content_main { situated : b; display-model : "123 (intrinsic)";
div#content_sub { situated : c; display-model : "1234 (intrinsic)";
div#content_supp { situated : d; display-model : "123 (intrinsic)";
```


If you are watching closely, you may notice that the `#branding` division uses the same numbered identifier twice (122). This will allow its content to span across two of the three columns.

Lastly, you can position each of the nested divisions into any of the slots you have defined, but only inside their containing division:

```
div#branding div.one { situated : 1; }
div#branding div.two { situated : 2; }

div#content_main div.one { situated : 1; }
div#content_main div.two { situated : 2; }
div#content_main div.three { situated : 3; }

div#content_sub div.one { situated : 1; }
div#content_sub div.two { situated : 2; }
div#content_sub div.three { situated : 3; }
div#content_sub div.four { situated : 4; }

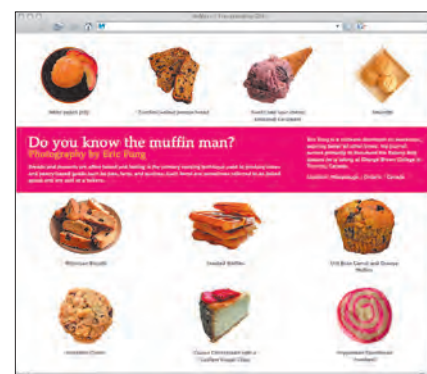
div#content_supp div.one { situated : 1; }
div#content_supp div.two { situated : 2; }
div#content_supp div.three { situated : 3; }
```

The result is a robust, flexible grid layout that can get smaller and bigger with the browser window and will accommodate changes in text size without breaking (**Figure 4.57**).

Have your cake, and eat it too

You have seen how simple it will be to create a complex grid design with the Advanced Layout Module. The flexibility of placing content into any slot without changing the source order of the content makes the Advanced Layout Module impressive, and it will liberate designers from presentational thinking about markup and CSS.

To demonstrate the layout flexibility of Advanced Layout, the next examples are all variations of the same markup. The Advanced Layout CSS appears with each variation (**next pages**).



4.57 Developing a layout that is flexible and robust

```

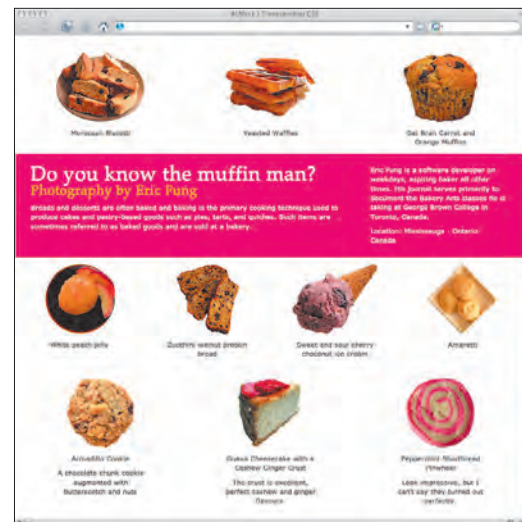
div#branding { situated : b; display-model : "112 (intrinsic)"; }
div#branding div.one { situated : 1; }
div#branding div.two { situated : 2; }

div#content_main { situated : a; display-model : "123 (intrinsic)"; }
div#content_main div.one { situated : 1; }
div#content_main div.two { situated : 2; }
div#content_main div.three { situated : 3; }

div#content_sub { situated : c; display-model : "1234 (intrinsic)"; }
div#content_sub div.one { situated : 1; }
div#content_sub div.two { situated : 2; }
div#content_sub div.three { situated : 3; }
div#content_sub div.four { situated : 4; }

div#content_supp { situated : d; display-model : "123 (intrinsic)"; }
div#content_supp div.one { situated : 1; }
div#content_supp div.two { situated : 2; }
div#content_supp div.three { situated : 3; }

```



```

div#branding { situated : a; display-model : "112 (intrinsic)"; }
div#branding div.one { situated : 1; }
div#branding div.two { situated : 2; }

div#content_main { situated : b; display-model : "123 (intrinsic)"; }
div#content_main div.one { situated : 1; }
div#content_main div.two { situated : 2; }
div#content_main div.three { situated : 3; }

div#content_sub { situated : c; display-model : "1234 (intrinsic)"; }
div#content_sub div.one { situated : 1; }
div#content_sub div.two { situated : 2; }
div#content_sub div.three { situated : 3; }
div#content_sub div.four { situated : 4; }

div#content_supp { situated : d; display-model : "123 (intrinsic)"; }
div#content_supp div.one { situated : 1; }
div#content_supp div.two { situated : 2; }
div#content_supp div.three { situated : 3; }

```





```
div#branding { situated : b; display-model : "112 (intrinsic)"; }
div#branding div.one { situated : 1; }
div#branding div.two { situated : 2; }
```

```
div#content_main { situated : c; display-model : "123 (intrinsic)"; }
div#content_main div.one { situated : 1; }
div#content_main div.two { situated : 2; }
div#content_main div.three { situated : 3; }
```

```
div#content_sub { situated : d; display-model : "1234 (intrinsic)"; }
div#content_sub div.one { situated : 1; }
div#content_sub div.two { situated : 2; }
div#content_sub div.three { situated : 3; }
div#content_sub div.four { situated : 4; }
```

```
div#content_supp { situated : a; display-model : "123 (intrinsic)"; }
div#content_supp div.one { situated : 1; }
div#content_supp div.two { situated : 2; }
div#content_supp div.three { situated : 3; }
```



```
div#branding { situated : d; display-model : "112 (intrinsic)"; }
div#branding div.one { situated : 1; }
div#branding div.two { situated : 2; }
```

```
div#content_main { situated : a; display-model : "123 (intrinsic)"; }
div#content_main div.one { situated : 1; }
div#content_main div.two { situated : 2; }
div#content_main div.three { situated : 3; }
```

```
div#content_sub { situated : b; display-model : "1234 (intrinsic)"; }
div#content_sub div.one { situated : 1; }
div#content_sub div.two { situated : 2; }
div#content_sub div.three { situated : 3; }
div#content_sub div.four { situated : 4; }
```

```
div#content_supp { situated : c; display-model : "123 (intrinsic)"; }
div#content_supp div.one { situated : 1; }
div#content_supp div.two { situated : 2; }
div#content_supp div.three { situated : 3; }
```

```
div#branding { situated : b; display-model : "112 (intrinsic)"; }
div#branding div.one { situated : 1; }
div#branding div.two { situated : 2; }
```

```
div#content_main { situated : d; display-model : "123 (intrinsic)"; }
div#content_main div.one { situated : 1; }
div#content_main div.two { situated : 2; }
div#content_main div.three { situated : 3; }
```

```
div#content_sub { situated : a; display-model : "1234 (intrinsic)"; }
div#content_sub div.one { situated : 1; }
div#content_sub div.two { situated : 2; }
div#content_sub div.three { situated : 3; }
div#content_sub div.four { situated : 4; }
```

```
div#content_supp { situated : c; display-model : "123 (intrinsic)"; }
div#content_supp div.one { situated : 1; }
div#content_supp div.two { situated : 2; }
div#content_supp div.three { situated : 3; }
```

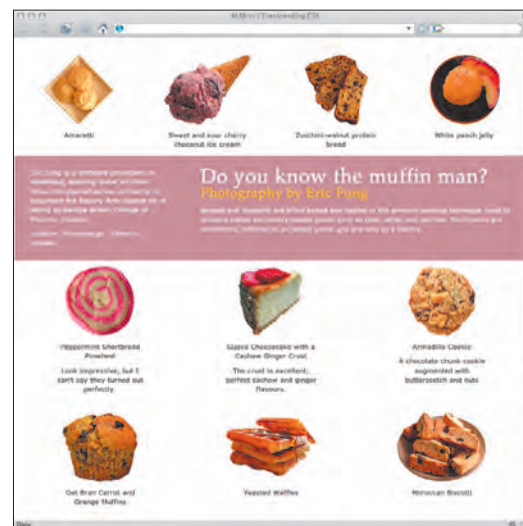


```
div#branding { situated : b; display-model : "122 (intrinsic)"; }
div#branding div.one { situated : 2; }
div#branding div.two { situated : 1; }
```

```
div#content_main { situated : d; display-model : "123 (intrinsic)"; }
div#content_main div.one { situated : 3; }
div#content_main div.two { situated : 2; }
div#content_main div.three { situated : 1; }
```

```
div#content_sub { situated : a; display-model : "1234 (intrinsic)"; }
div#content_sub div.one { situated : 4; }
div#content_sub div.two { situated : 3; }
div#content_sub div.three { situated : 2; }
div#content_sub div.four { situated : 1; }
```

```
div#content_supp { situated : c; display-model : "123 (intrinsic)"; }
div#content_supp div.one { situated : 3; }
div#content_supp div.two { situated : 2; }
div#content_supp div.three { situated : 1; }
```





```
div#branding { situated : c; display-model : "122 (intrinsic)"; }
div#branding div.one { situated : 2; }
div#branding div.two { situated : 1; }
```

```
div#content_main { situated : a; display-model : "123 (intrinsic)"; }
div#content_main div.one { situated : 3; }
div#content_main div.two { situated : 1; }
div#content_main div.three { situated : 2; }
```

```
div#content_sub { situated : d; display-model : "1234 (intrinsic)"; }
div#content_sub div.one { situated : 3; }
div#content_sub div.two { situated : 2; }
div#content_sub div.three { situated : 4; }
div#content_sub div.four { situated : 1; }
```

```
div#content_supp { situated : b; display-model : "123 (intrinsic)"; }
div#content_supp div.one { situated : 2; }
div#content_supp div.two { situated : 3; }
div#content_supp div.three { situated : 1; }
```



```
div#branding { situated : a; display-model : "122 (intrinsic)"; }
div#branding div.one { situated : 2; }
div#branding div.two { situated : 1; }
```

```
div#content_main { situated : c; display-model : "123 (intrinsic)"; }
div#content_main div.one { situated : 1; }
div#content_main div.two { situated : 3; }
div#content_main div.three { situated : 2; }
```

```
div#content_sub { situated : d; display-model : "1234 (intrinsic)"; }
div#content_sub div.one { situated : 1; }
div#content_sub div.two { situated : 2; }
div#content_sub div.three { situated : 4; }
div#content_sub div.four { situated : 3; }
```

```
div#content_supp { situated : b; display-model : "123 (intrinsic)"; }
div#content_supp div.one { situated : 1; }
div#content_supp div.two { situated : 3; }
div#content_supp div.three { situated : 2; }
```



Concluding Remarks

So, here you are at the end of the book. What's next?

The future.

If you enjoy designing for the Web as much as I do, you'll find that there will be very exciting times ahead of you. You will have many opportunities—ones that only a few short years ago were unimaginable—to create great-looking sites that people will love to use.

In just a few years since CSS was born, the Web design industry has changed significantly: Accessibility, meaningful markup, and presentation with CSS are now a reality in the Web professional's daily life. A true professional never stops learning and CSS would not be used so widely today had it not been for the hard work and generosity of the many men and women who struggled to find solutions and who then shared their work with the rest of us. I hope that as CSS develops in the future and even more exiting solutions are reached, you will follow in their footsteps and share what you have learned.

Designing and developing for the Web should be a process filled with creativity and whether you write code, create databases, or are a visual designer, there is creativity in everything that you do. As part of my role as an invited expert to the W3C's CSS Working Group, I help to bring a creative designer's voice to the table and I welcome your thoughts. If future versions of CSS are to meet the needs of visual designers and developers then it is up to all of us to help shape its future by becoming involved in the discussions and debates over how and where it is to go next. So I urge you to become involved in whatever capacity you can, no matter how big or how small your contribution.

Now it's time to stop listening to me and time to start designing the future.