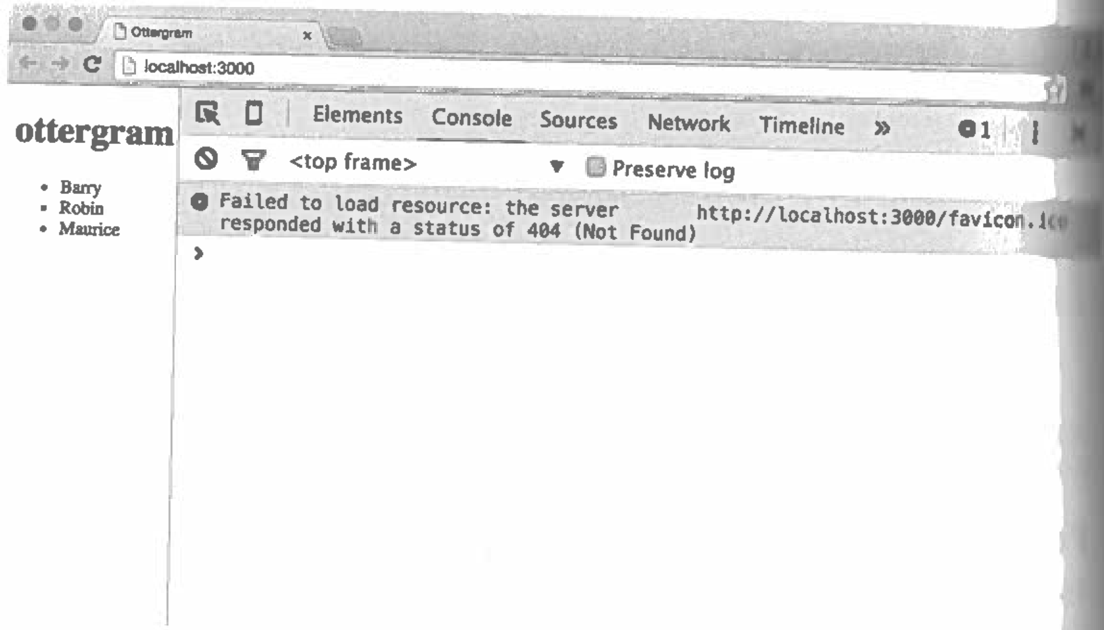
That is the favicon.ico image file. Many sites have one, and browsers request one by default. Because Ottergram does not have one, you may see an error like the one in Figure 2.20 in the DevTools.

Figure 2.20 Error about missing `favicon.ico`



```
Failed to load resource: the server      http://localhost:3000/favicon.ico
responded with a status of 404 (Not Found)
```

Do not worry about this error if it appears. It will not affect your project. However, you can easily add a `favicon.ico` image – and that is your first challenge.

Silver Challenge: Adding a favicon.ico

You have decided that you like otters more than you like seeing the `favicon.ico` error message. You are going to create a `favicon.ico` file using one of the otter images.

Do a web search for "favicon generator" and you should see a list of websites that will do a file conversion for you. Most will let you upload an image and then provide you with a `favicon.ico` version.

Choose one and upload any one of the otter images.

Save the resulting `favicon.ico` file in the same folder as your `index.html` file. Finally, reload your browser. Your browser tab will look something like Figure 2.21.

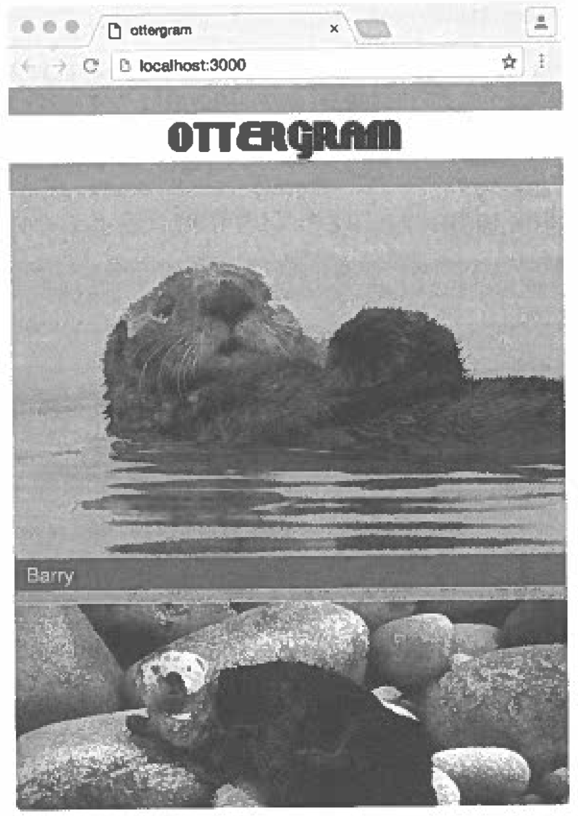
Figure 2.21 Ottergram with a `favicon.ico`

3 Styles

In this chapter, you will design a static version of Ottergram. In the chapters that follow, you will make Ottergram interactive.

When you reach the end of this chapter, your website will look like Figure 3.1.

Figure 3.1 Ottergram: stylish



This chapter introduces a number of concepts and examples. Do not worry if you do not feel that you have mastered all of them when you get to the end. You will be encountering them again and again as you progress through this book, and your work in this chapter will provide a solid foundation on which you will build true understanding.

Of course, we can only introduce you to a tiny fraction of all the styles that are available in CSS. You will want to consult the MDN for information about the full set of properties and their values.

Front-end developers have to choose between two approaches to styling a website: start with the overall layout and work down to the smallest details, or start with the smallest details and work up to the overall layout.

Not only does working from detail to big picture produce cleaner, more reusable code, it also has a cool name: *atomic styling*. You will use this approach as you style the otter thumbnails first, then the thumbnail list layout. In the next chapter, you will work on the layout of the site as a whole.

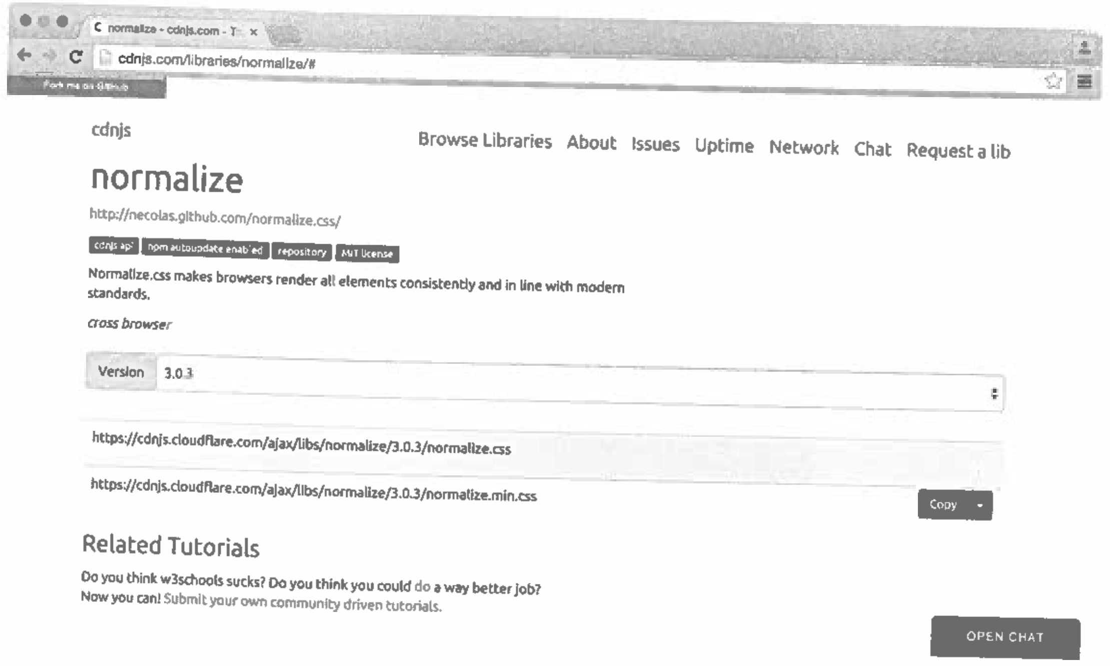
Creating a Styling Baseline

You are going to begin by adding the `normalize.css` file to your project. `normalize.css` helps the CSS you write display consistently across browsers. All browsers come with a set of default styles, but the defaults are different from browser to browser. `normalize.css` gives you a good starting point for developing your own custom CSS for a website or web app.

`normalize.css` is freely available online. You do not need to download it. To add it to Ottergram, you only need to link to it in `index.html`.

To ensure that you are using the most current version of `normalize.css`, you are going to get its address from a content sharing site. Go to `cdnjs.com/libraries/normalize` and find the version of the file ending with `.min.css`. (This version is a smaller download than the others, with the extra whitespace stripped out.) Click the Copy button to copy its address (Figure 3.2).

Figure 3.2 Getting a link to `normalize.css` from `cdnjs.com`



normalize

http://necolas.github.com/normalize.css/

Normalize.css makes browsers render all elements consistently and in line with modern standards.

cross browser

Version 3.0.3

https://cdnjs.cloudflare.com/ajax/libs/normalize/3.0.3/normalize.css

https://cdnjs.cloudflare.com/ajax/libs/normalize/3.0.3/normalize.min.css

Related Tutorials

Do you think w3schools sucks? Do you think you could do a way better job?
Now you can! Submit your own community driven tutorials.

The current version at the time of this writing is 3.0.3, but the version you use may be more recent.

Open your Ottergram folder in Atom, then open `index.html`. Add a new `<link>` tag and paste in the address. (In the code below, the `<link>` has been broken into two lines to fit on the page. You can leave yours on a single line.)

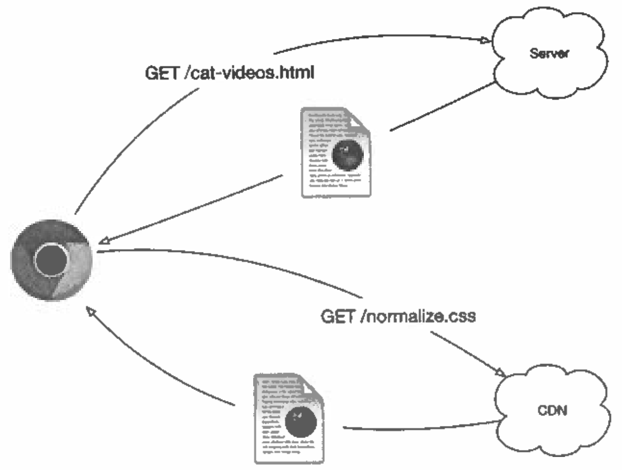
```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>ottergram</title>
    <link rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/normalize/3.0.3/normalize.min.css">
    <link rel="stylesheet" href="stylesheets/styles.css">
  </head>
...
```

Make sure that you add the `<link>` tag for `normalize.css` *before* the `<link>` tag for `styles.css`. The browser needs to read the styles found in `normalize.css` before it reads yours.

And, just like that, your project can take advantage of this useful tool. No other setup is required.

You may be wondering why you are linking to an address on a completely different server. In fact, it is not unusual for an HTML file to specify resources located on different servers (Figure 3.3).

Figure 3.3 Requesting resources from different servers



In this case, `normalize.css` is hosted on `cdnjs.com`, a public server that is part of a *content delivery network*, or CDN. CDNs have servers all around the world, each with copies of the same files. When users request a file, they receive it from a server nearby, cutting down on the load time for that file. `cdnjs.com` hosts many versions of popular front-end libraries and frameworks.

Preparing the HTML for Styling

In the last chapter, you created a stylesheet called `styles.css`, and in this chapter you will add a number of CSS *styling rules* to it. But before you get started adding styles, you need to set up your HTML with targets for your styling rules to refer to.

You are going to add `class` attributes identifying the `span` elements with the otters' names as "thumbnail titles." `class` attributes are a way to identify a group of HTML elements, usually for styling. Your "thumbnail title" `class` will allow you to easily style all the names at once.

In `index.html`, add the class name `thumbnail-title` as an attribute of the `span`s inside the `li` elements, as shown:

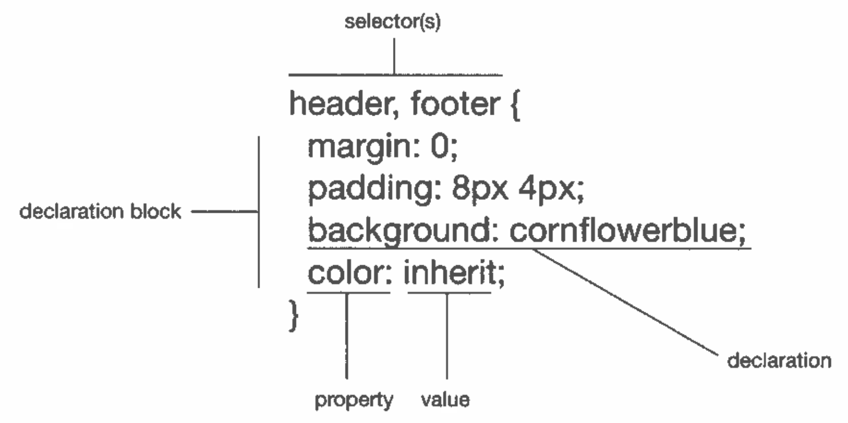
```
...
    <ul>
      <li>
        <a href="#">
          <img src="img/otter1.jpg" alt="Barry the Otter">
          ~~<span>Barry</span>~~
          <span class="thumbnail-title">Barry</span>
        </a>
      </li>
      <li>
        <a href="#">
          <img src="img/otter2.jpg" alt="Robin the Otter">
          ~~<span>Robin</span>~~
          <span class="thumbnail-title">Robin</span>
        </a>
      </li>
      <li>
        <a href="#">
          <img src="img/otter1.jpg" alt="Maurice the Otter">
          ~~<span>Maurice</span>~~
          <span class="thumbnail-title">Maurice</span>
        </a>
      </li>
      <li>
        <a href="#">
          <img src="img/otter4.jpg" alt="Lesley the Otter">
          ~~<span>Lesley</span>~~
          <span class="thumbnail-title">Lesley</span>
        </a>
      </li>
      <li>
        <a href="#">
          <img src="img/otter5.jpg" alt="Barbara the Otter">
          ~~<span>Barbara</span>~~
          <span class="thumbnail-title">Barbara</span>
        </a>
    </ul>
...
```

In a moment, you will use this class name to style all the image titles.

Anatomy of a Style

When you create individual styles, you do so by writing styling rules, which consist of two main parts: *selectors* and *declarations* (Figure 3.4).

Figure 3.4 Anatomy of a styling rule



The first part of a styling rule is one or more selectors. Selectors describe the elements that the style should be applied to, like `h1`, `span`, or `img`. But selectors are not limited to tag names. You can write selectors that apply to a more targeted set of elements by increasing the selector's *specificity*.

For example, you can write selectors based on attributes – such as the `thumbnail-title` class attribute you just added to the `<span>` tags. Selectors based on attributes are more specific than selectors based on element names.

In addition to making sure that styles are only applied to a limited set of elements (e.g., elements with the class name `thumbnail-title` versus all <span> elements), specificity also determines the selector's relative priority. If a stylesheet contains multiple styles that could apply to the same element, the styles with a selector of higher specificity will be used instead styles whose selector has a lower specificity. You can read more about specificity in a For the More Curious section at the end of this chapter.

Throughout this chapter, you will be introduced to a number of different kinds of selectors that vary in their specificity. Though there are often many ways to target the same element for styling, understanding specificity is key to choosing the best selector to use so that your styles are maintainable.

The second part of a styling rule is the declaration block, wrapped in curly braces, which defines the styles to be applied. The individual declarations within the block each include a property name and a value for that property.

In your first styling rule, you will use the `class` attribute you just added as a selector to apply styles around the otters' names.

Your First Styling Rule

To use a `class` as a selector in a styling rule, you prefix the class name with a dot (period), as in `.thumbnail-title`. The first styles you are going to add will set the background and foreground colors for the `.thumbnail-title` class.

Open `styles.css` and add your styling rule:

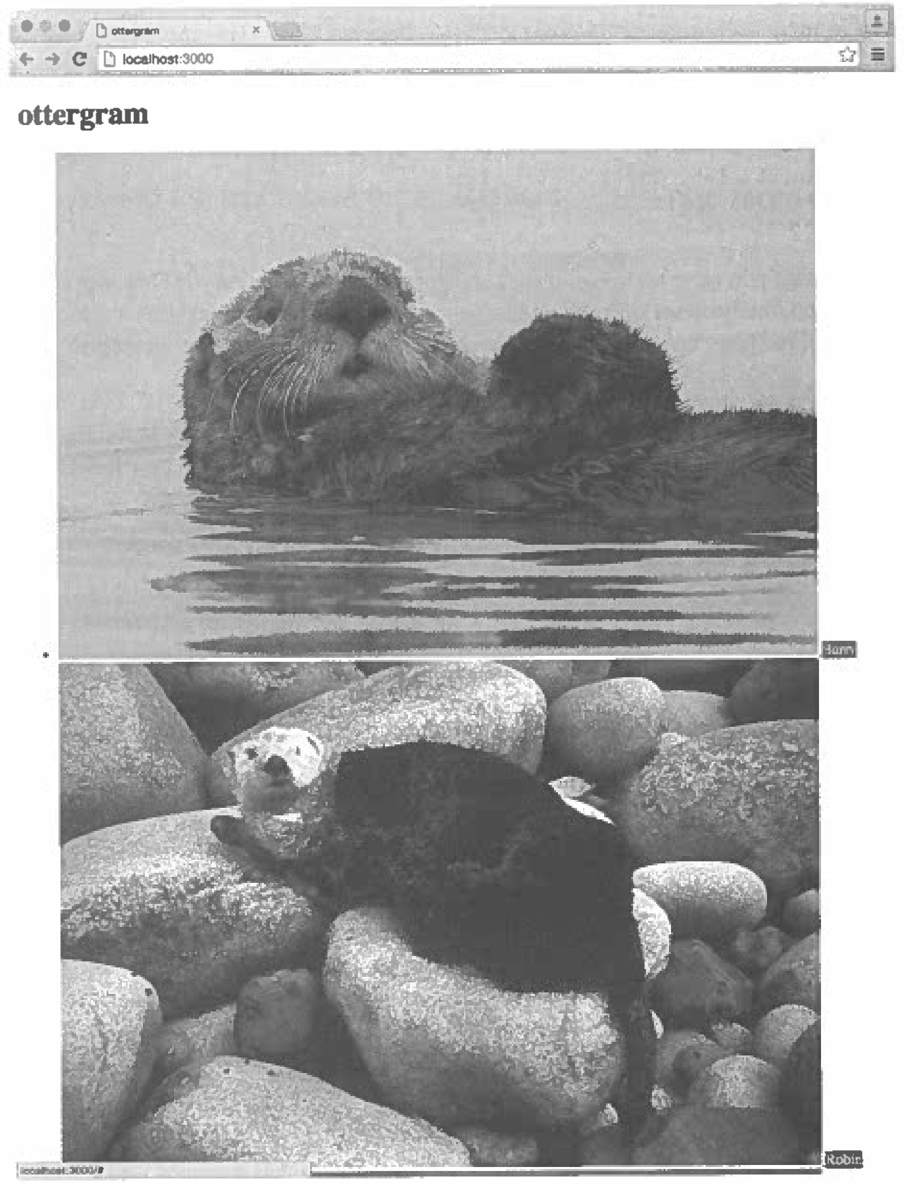
```
.thumbnail-title {
  background: rgb(96, 125, 139);
  color: rgb(202, 238, 255);
}
```

You will learn more about color later in this chapter. For now, just take a look at your changes. Save `styles.css` and make sure `browser-sync` is running. If you need to restart it, the command is:

```
browser-sync start --server --browser "Google Chrome"
                   --files "stylesheets/*.css, *.html"
```

This will open your web page in Chrome (Figure 3.5).

Figure 3.5 A slightly more colorful Ottergram



You can see that you have set the background for the thumbnail titles to a deep gray-blue and the font color to a lighter blue. Nice.

Continue styling the thumbnail titles: Return to `styles.css` and add to your existing styling rule for the `.thumbnail-title` class, as shown:

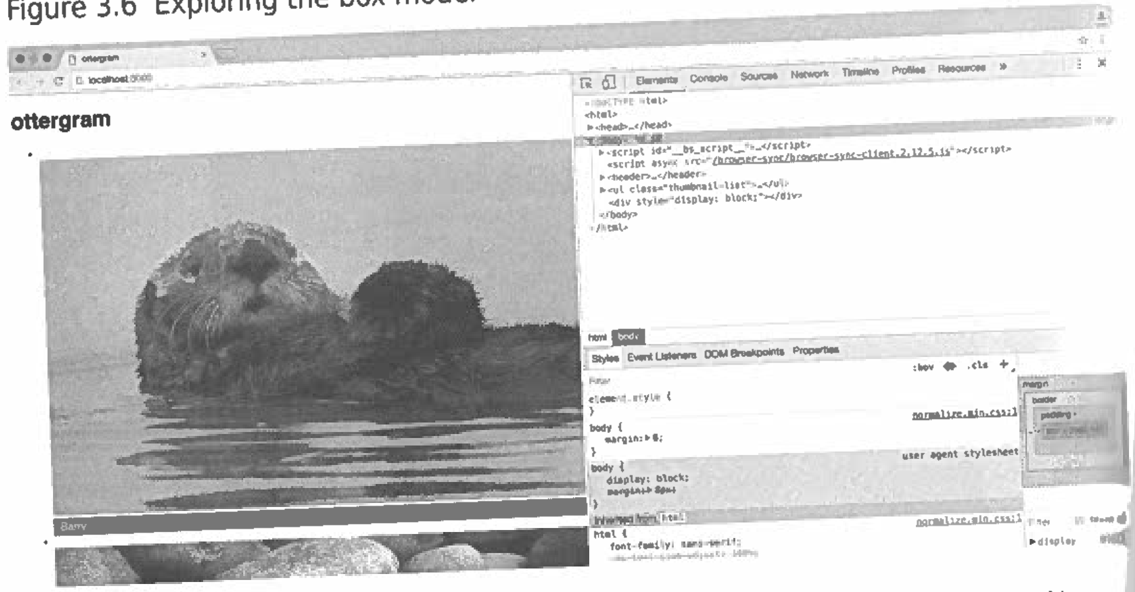
```
.thumbnail-title {
  display: block;
  margin: 0;
  padding: 4px 10px;

  background: rgb(96, 125, 139);
  color: rgb(202, 238, 255);
}
```

The three declarations you have added all affect an element's *box*. For every HTML tag that has a visual representation, the browser draws a rectangle to the page. The browser uses a scheme called the *standard box model* (or just "box model") to determine the dimensions of that rectangle.

The box model

To understand the box model, you are going to look at its representation in the DevTools. Save `styles.css`, switch to Chrome, and make sure the DevTools are open (Figure 3.6).

Figure 3.6 Exploring the box model

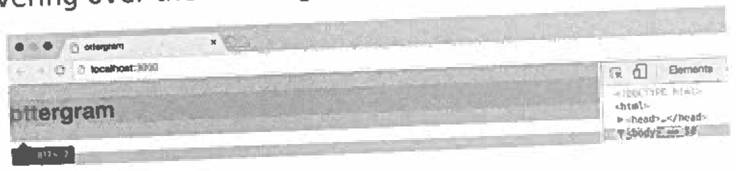


```
<!DOCTYPE html>
<html>
▶<head>…</head>
▼<body> == $0
  ▶<script id="__bs_script__">…</script>
   <script async src="/browser-sync/browser-sync-client.2.12.5.js"></script>
  ▶<header>…</header>
  ▶<ul class="thumbnail-list">…</ul>
   <div style="display: block;"></div>
 </body>
</html>
```

```
element.style {
}
body {                                normalize.min.css:1
  margin:▶0;
}
body {                                user agent stylesheet
  display: block;
  margin:▶8px;
}
Inherited from html
html {                                normalize.min.css:1
  font-family: sans-serif;
```

Click the button in the upper-left of the elements panel. This is the *Inspect Element* button. Now move your cursor over the word "ottergram" on the web page. As you hover over the word, the DevTools surrounds the heading with a blue- and peach-colored rectangle (Figure 3.7).

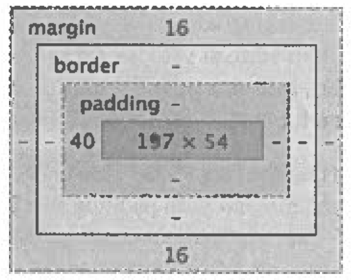
Figure 3.7 Hovering over the heading



Click the word "ottergram" on the web page. Although you no longer see the multicolored overlay, the element is now selected and the DOM tree view in the elements panel will expand to show and highlight the corresponding `<h1>` tag.

The rectangular diagram in the lower-right of the elements panel represents the box model for the `h1` element. You can see that the regions of the diagram have some of the same colors as the rectangle you saw overlaying the heading when you inspected it (Figure 3.8).

Figure 3.8 Viewing the box model for an element



The box model incorporates four aspects of the rectangle drawn for an element (which the DevTools renders in four different colors in the diagram).

content (shown in blue)	the visual content – here, the text
padding (shown in green)	transparent space around the content
border (shown in yellow)	a border, which can be made visible, around the content and padding
margin (shown in peach)	transparent space around the border

The numbers in Figure 3.8 are *pixel* values; a pixel is a unit corresponding to the smallest rectangular area of a computer screen that can display a single color. In the case of the `h1` element, the content area has been allocated an area of 197 pixels by 54 pixels (your values may be different, depending on the size of your browser window). There is padding of 40 pixels on the left side. The border is set at 0, and there is a margin of 16 pixels above and below the element.

Where did that margin value come from? Each browser provides a default stylesheet, called the *user agent stylesheet*, in case an HTML file does not specify one. Styles that you specify override the defaults. Because you have not specified values for the `h1` element's box, the default styles have been applied.

Now you are ready to understand the styling declarations you added:

```
.thumbnail-title {
  display: block;
  margin: 0;
  padding: 4px 10px;

  background: rgb(96, 125, 139);
  color: rgb(202, 238, 255);
```

The `display: block` declaration changes the box for all elements of the class `.thumbnail-title` so that they occupy the entire width allowed by their containing element. (Notice in Figure 3.6 that the background color for the titles now covers a wider area.) Other `display` values, such as the `display: inline` property you will see later, make an element's width fit to its content.

You also set the margin for the thumbnail titles to 0 and the padding to two different values: 4px and 10px (*px* is the abbreviation for "pixels"). This sets the padding to specific pixel values, overriding the default size set by the user agent stylesheet.

Padding, margin, and certain other styles can be written as *shorthand properties*, in which one value is applied to multiple properties. You are taking advantage of this here: When two values are provided for the padding, the first is applied to both vertical values (top and bottom) and the second is applied to both horizontal values (left and right). It is also possible to provide a single value to be applied to all four sides or to specify a separate value for each side.

To sum up, your new declarations say that the box for all elements of the `.thumbnail-title` class will fill the width of its container with no margin and with padding that is 4 pixels at the top and bottom and 10 pixels at the left and right sides.

Style Inheritance

Next, you are going to add styles to change the size and appearance of the text.

Add a new styling rule in `styles.css` to set the font size for the `body` element. To do this, you will use a different type of selector – an *element selector* – by simply using the element's name.

```
body {
  font-size: 10px;
}

.thumbnail-title {
  display: block;
  margin: 0;
  padding: 4px 10px;

  background: rgb(96, 125, 139);
  color: rgb(202, 238, 255);
}
```

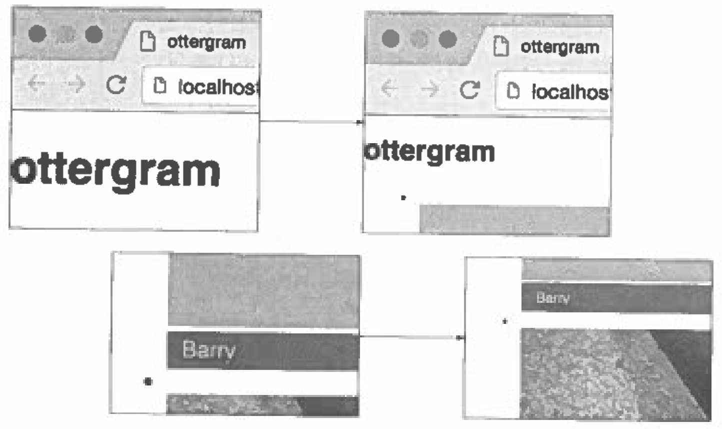
This styling rule sets the `body` element's `font-size` to `10px`.

You will rarely use element selectors in your stylesheets, because you will not often want to apply the exact same styles to every occurrence of a particular tag. Also, element selectors limit your ability to reuse styles; using them means that you may end up retyping the same declarations throughout your stylesheets. This is not great for maintenance if you need to alter those styles.

But, in this case, targeting the `body` element is exactly the right amount of specificity. There can be only one <body> element, and you will not be reusing its styles.

Save `styles.css` and check out your web page in Chrome (Figure 3.9).

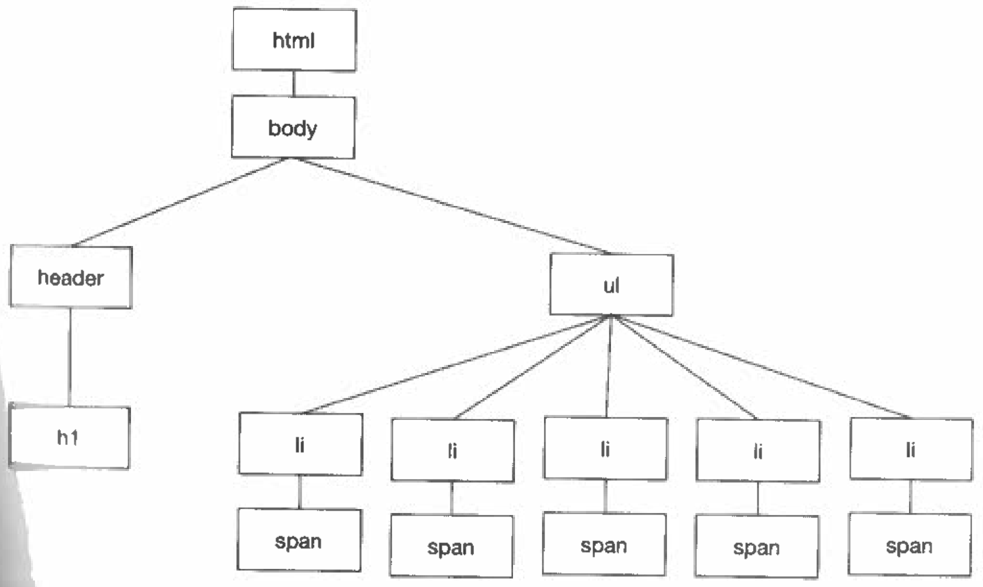
Figure 3.9 After setting the body font size



Your headline and thumbnail titles have gotten smaller. You may – or may not – have expected this. While the headline is directly within the `body` element where you declared the `font-size` property, the thumbnail titles are not. They are nested several levels deep. However, many styles, including font size, are applied to the elements specified by the styling rule as well as the *descendants* of those elements.

The structure of your document can be described using a tree diagram, as in Figure 3.10. Representing your elements as a tree is a good way to visualize the DOM.

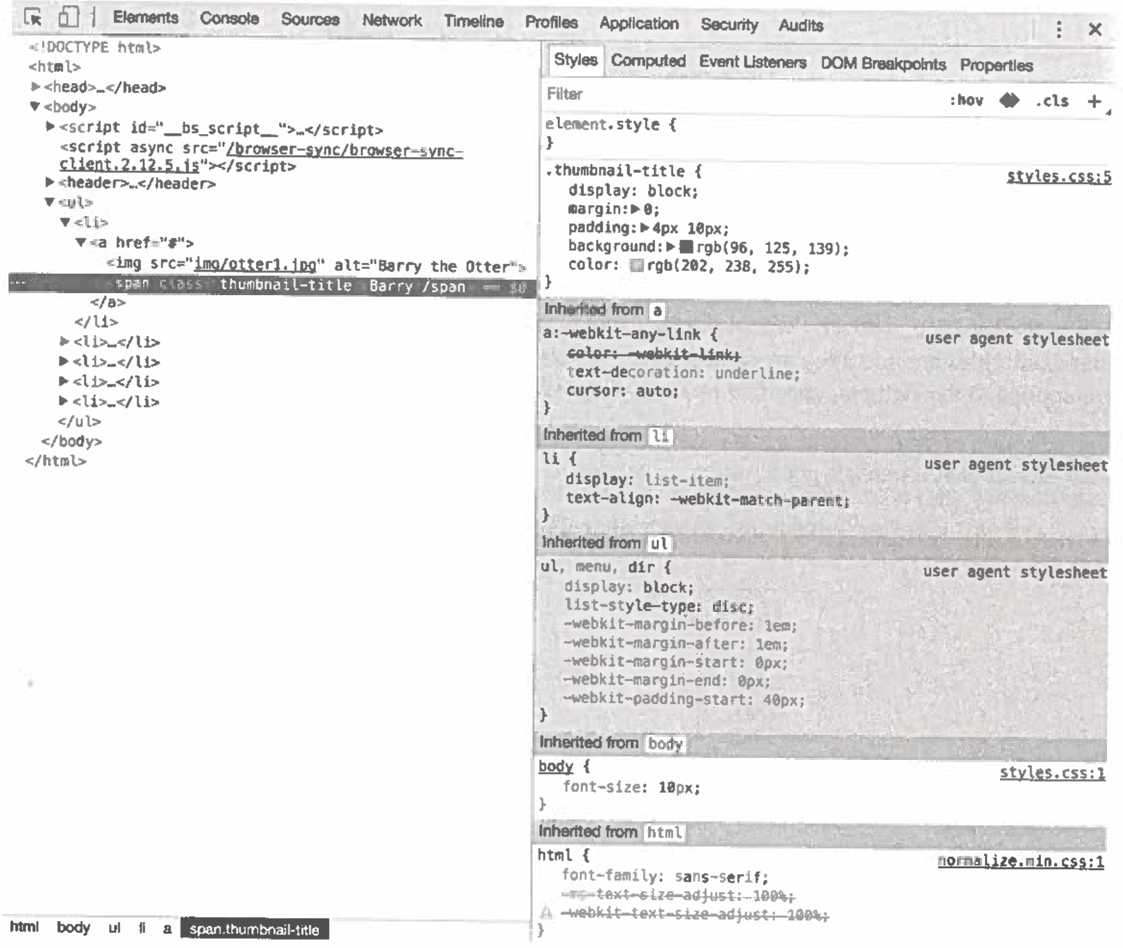
Figure 3.10 Simplified structure of Ottergram



An element contained within another element is said to be its descendent. In this case, your `span`s are all descendents of the `body` (as well as the `ul` and their respective `li`), so they inherit the `body`'s `font-size` style.

In the DevTools' DOM tree view, locate and select one of the span elements. In the styles pane, notice the boxes labeled Inherited from a, Inherited from li, and Inherited from ul. These three areas, as indicated, show styles inherited at each level from the user agent stylesheet. Under Inherited from body, you can see that the `font-size` property has been inherited from the style set for the body element in `styles.css` (Figure 3.11).

Figure 3.11 Styles inherited from ancestor elements



```
<!DOCTYPE html>
<html>
▶<head>…</head>
▼<body>
  ▶<script id="__bs_script__">…</script>
   <script async src="/browser-sync/browser-sync-
   client.2.12.5.js"></script>
  ▶<header>…</header>
  ▼<ul>
    ▼<li>
      ▼<a href="#">
         <img src="img/otter1.jpg" alt="Barry the Otter">
         <span class="thumbnail-title">Barry</span> == $0
       </a>
     </li>
    ▶<li>…</li>
    ▶<li>…</li>
    ▶<li>…</li>
    ▶<li>…</li>
   </ul>
 </body>
</html>
```

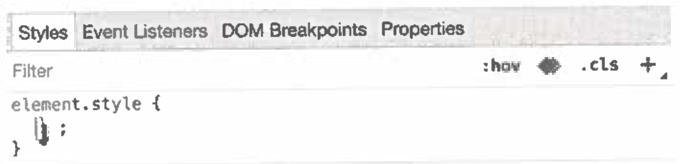
```
element.style {
}
.thumbnail-title {                              styles.css:5
   display: block;
   margin:▶0;
   padding:▶4px 10px;
   background:▶■rgb(96, 125, 139);
   color: ■rgb(202, 238, 255);
}
Inherited from a
a:-webkit-any-link {                     user agent stylesheet
   ~~color: -webkit-link;~~
   text-decoration: underline;
   cursor: auto;
}
Inherited from li
li {                                     user agent stylesheet
   display: list-item;
   text-align: -webkit-match-parent;
}
Inherited from ul
ul, menu, dir {                          user agent stylesheet
   display: block;
   list-style-type: disc;
   -webkit-margin-before: 1em;
   -webkit-margin-after: 1em;
   -webkit-margin-start: 0px;
   -webkit-margin-end: 0px;
   -webkit-padding-start: 40px;
}
Inherited from body
body {                                          styles.css:1
   font-size: 10px;
}
Inherited from html
html {                                   normalize.min.css:1
   font-family: sans-serif;
   ~~-ms-text-size-adjust: 100%;~~
⚠ ~~-webkit-text-size-adjust: 100%;~~
}
```

What if a different font size were set at another level, such as the `ul`? Styles from the closer ancestor take priority, so a font size set in `styles.css` for the `ul` would override one set for the `body` and a font size set for the `span` element itself would override them both.

To see this, click on the `ul` element in the DOM tree view. This will allow you to try out styles on the fly. The styles you add here will be immediately reflected in the web page view, but will not be added to your actual project files.

At the top of the styles pane in the elements panel, you will see a section labeled `elements.style`. Click anywhere in between the curly braces of the `elements.style`, and the DevTools will give you a prompt (Figure 3.12).

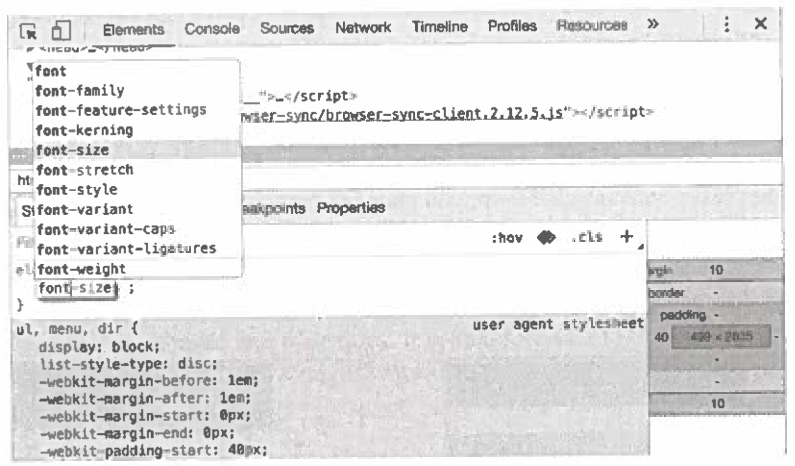
Figure 3.12 Prompting for a style rule



```
element.style {
  ;
}
```

Start typing `font-size`, and the DevTools will suggest possible completions (Figure 3.13).

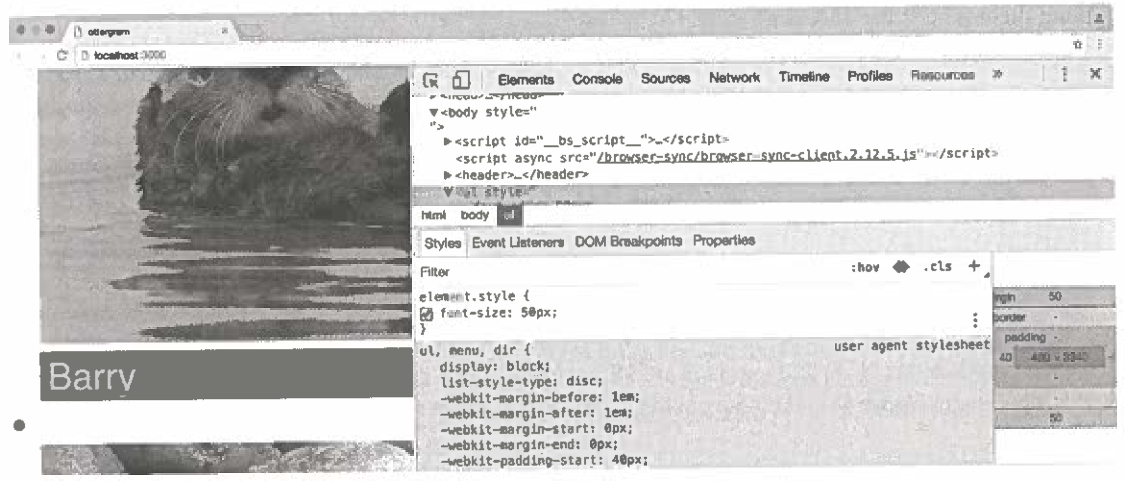
Figure 3.13 Autocompletion options in styles pane



```
font
font-family
font-feature-settings
font-kerning
font-size
font-stretch
font-style
font-variant
font-variant-caps
font-variant-ligatures
font-weight
font-size ;

ul, menu, dir {                          user agent stylesheet
   display: block;
   list-style-type: disc;
   -webkit-margin-before: 1em;
   -webkit-margin-after: 1em;
   -webkit-margin-start: 0px;
   -webkit-margin-end: 0px;
   -webkit-padding-start: 40px;
```

Choose `font-size`, then press the Tab key. Enter a large value, such as `50px`, and press Return. You may need to scroll the page, but you will see that the `ul`'s `font-size` has overridden the `body`'s (Figure 3.14).

Figure 3.14 Giving the `ul` a `font-size` of 50px



```
▼<body style="
">
  ▶<script id="__bs_script__">…</script>
   <script async src="/browser-sync/browser-sync-client.2.12.5.js"></script>
  ▶<header>…</header>
  ▼<ul style="

element.style {
☑ font-size: 50px;
}
ul, menu, dir {                          user agent stylesheet
   display: block;
   list-style-type: disc;
   -webkit-margin-before: 1em;
   -webkit-margin-after: 1em;
   -webkit-margin-start: 0px;
   -webkit-margin-end: 0px;
   -webkit-padding-start: 40px;
```

Not all style properties are inherited – `border`, for example, is not. To find out whether a property is inherited, refer to the property's MDN reference page.

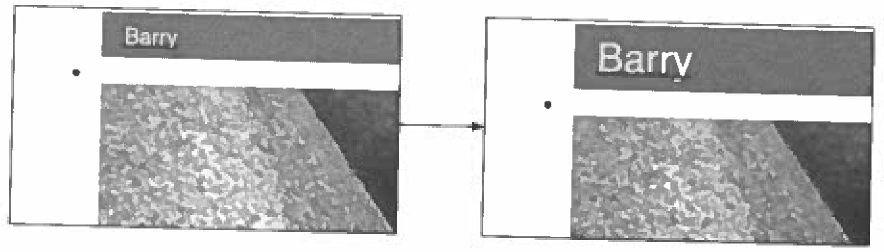
Back in `styles.css`, update your declaration block for the `.thumbnail-title` class to override the body's `font-size` and use a larger font.

```
body {
  font-size: 10px;
}

.thumbnail-title {
  display: block;
  margin: 0;
  padding: 4px 10px;

  background: rgb(96, 125, 139);
  color: rgb(202, 238, 255);

  font-size: 18px;
}
```

For elements of the class `.thumbnail-title`, you changed the font size to 18 pixels.

Save `styles.css` and admire your thumbnail titles in Chrome (Figure 3.15).

Figure 3.15 Styled thumbnail titles



They look good, but the user agent stylesheet is adding underlines to the `.thumbnail-title` elements. This is because you wrapped them (along with the `.thumbnail-image` elements) with an anchor tag, making them inherit the underline style.

You do not need the underlines, so you are going to remove them by changing the `text-decoration` property for the anchor tags in a new styling rule in `styles.css`. What selector should you use for this rule?

If you are confident that you want to remove the underlines from the thumbnail titles *as well as any other anchor elements in Ottergram*, you can simply use an element selector:

```
a {
  /* style declaration */
}
```

(The text between the `/* */` indicators is a CSS *comment*. Code comments are ignored by the browser; they allow the developer to make notes in the code for future reference.)

If you think you might use anchors for another purpose (and will want to style them differently), you can pair the element selector with an *attribute selector*, like this:

```
a[href]{
  /* style declaration */
}
```

This selector would match any anchor element with an `href` attribute. Of course, anchor elements generally do have `href` attributes, so that might not be targeted enough to match only the thumbnail images and titles. To make an attribute selector more precise, you can also specify the value of the attribute, like this:

```
a[href="#"]{
  /* style declaration */
}
```

This selector would match only those anchor elements whose `href` attribute has a value of `#`.

By the way, you can also use attribute selectors, with or without values, on their own, such as:

```
[href]{
  /* style declaration */
}
```

As it happens, Ottergram is a fairly simple project and you will not, in fact, be using anchor tags for anything other than the thumbnails and their titles. It is therefore safe to use an element selector, and you should do so because it is the most straightforward solution with the right amount of specificity.

Add the new style declaration to `styles.css`:

```
body {
  font-size: 10px;
}

a {
  text-decoration: none;
}

.thumbnail-title {
  ...
}
```

Save your file and check your browser. The underlines are gone and your thumbnail titles are nicely styled (Figure 3.16).

Figure 3.16 After setting `text-decoration` to `none`

Note that you should not remove the underlines from links that are in normal text – text that is not an obvious heading, title, or caption. The underlining of linked text is an important visual indicator that users have come to expect. You did it here because the thumbnails do not require the same visual cues. Users will reasonably expect them to be clickable.

In the rest of the chapter, you will use class selectors to style the thumbnail images, the unordered list of images, the list items (which include the thumbnail images and their titles), and, finally, the header. Go ahead and add class names to the `h1`, `ul`, `li`, and `img` elements in `index.html` so they are ready as you need them.

```
...
  </head>
  <body>
    <header>
      ~~<h1>ottergram</h1>~~
      <h1 class="logo-text">ottergram</h1>
    </header>
    ~~<ul>~~
    <ul class="thumbnail-list">
      ~~<li>~~
      <li class="thumbnail-item">
        <a href="#">
          ~~<img src="img/otter1.jpg" alt="Barry the Otter">~~
          <img class="thumbnail-image" src="img/otter1.jpg" alt="Barry the Otter">
          <span class="thumbnail-title">Barry</span>
        </a>
      </li>
      ~~<li>~~
      <li class="thumbnail-item">
        <a href="#">
          ~~<img src="img/otter2.jpg" alt="Robin the Otter">~~
          <img class="thumbnail-image" src="img/otter2.jpg" alt="Robin the Otter">
          <span class="thumbnail-title">Robin</span>
        </a>
      </li>
      ~~<li>~~
      <li class="thumbnail-item">
        <a href="#">
          ~~<img src="img/otter3.jpg" alt="Maurice the Otter">~~
          <img class="thumbnail-image" src="img/otter3.jpg" alt="Maurice the Otter">
          <span class="thumbnail-title">Maurice</span>
        </a>
      </li>
      ~~<li>~~
      <li class="thumbnail-item">
        <a href="#">
          ~~<img src="img/otter4.jpg" alt="Lesley the Otter">~~
          <img class="thumbnail-image" src="img/otter4.jpg" alt="Lesley the Otter">
          <span class="thumbnail-title">Lesley</span>
        </a>
      </li>
      ~~<li>~~
      <li class="thumbnail-item">
        <a href="#">
          ~~<img src="img/otter5.jpg" alt="Barbara the Otter">~~
          <img class="thumbnail-image" src="img/otter5.jpg" alt="Barbara the Otter">
          <span class="thumbnail-title">Barbara</span>
        </a>
      </li>
    </ul>
...
```

By adding class names to these elements, you have given yourself targets for the styles you will be adding.

We favor class selectors over other kinds of selectors, and you should, too. You can write very descriptive class names that make your code easy to develop and maintain. Also, you can add multiple class names to an element, making them a flexible and powerful tool for styling.

Be sure to save `index.html` before moving on.

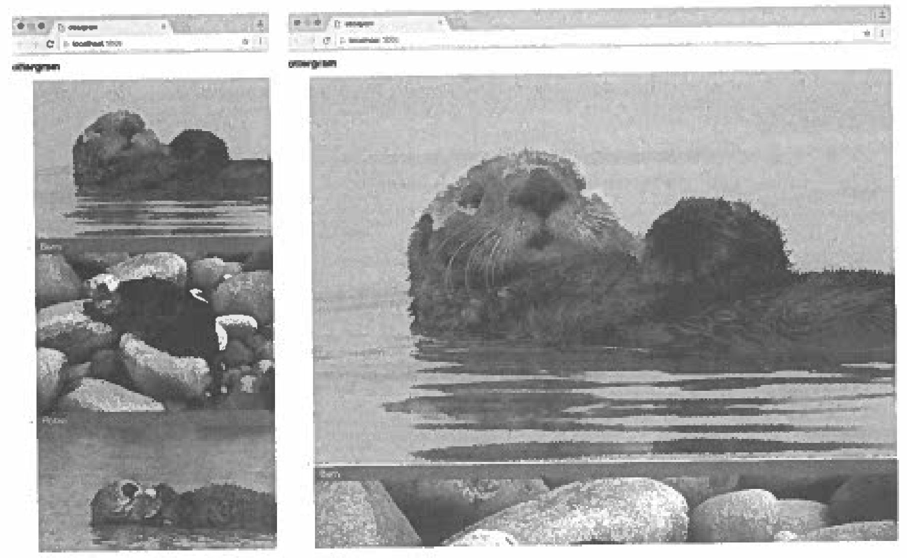
Making Images Fit the Window

Following the atomic styling pattern, the images are next in line for styling. They are so large that they are cut off unless the browser window is also large. Add a styling rule for `.thumbnail-image` in `styles.css` to make the thumbnails fit in the window:

```
...
a {
  text-decoration: none;
}

.thumbnail-image {
  width: 100%;
}

.thumbnail-title {
  ...
}
```

You set the `width` to `100%`, which constrains it to the width of its container. This means that as you widen the browser window, the images get proportionally larger. Check it out: Save `styles.css`, switch to your browser, and make your browser window larger and smaller. The images grow and shrink along with the browser window, always keeping their proportions. Figure 3.17 shows Ottergram in one narrow and one wider browser window.

Figure 3.17 Fitting an image by width

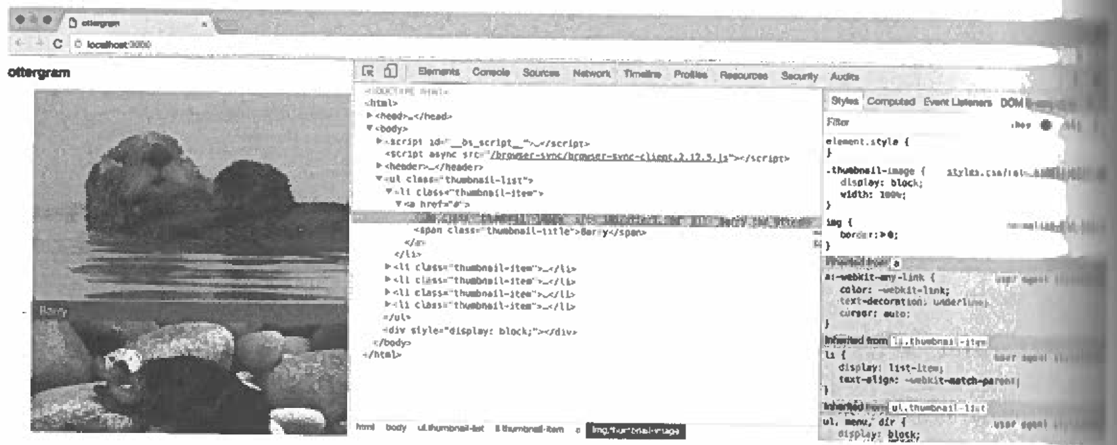


If you look closely, the spacing around the `.thumbnail-titles` is off, so that it appears that the titles go with the images below them. Fix that in `styles.css` by setting the `.thumbnail-image`'s `display` property to `block`.

```
...
.thumbnail-image {
  display: block;
  width: 100%;
}
...
```

Now the space between the image and its title is gone (Figure 3.18).

Figure 3.18 After setting `.thumbnail-image` to `display: block`



```
<!DOCTYPE html>
<html>
▶<head>…</head>
▼<body>
  ▶<script id="__bs_script__">…</script>
   <script async src="/browser-sync/browser-sync-client.2.12.5.js"></script>
  ▶<header>…</header>
  ▼<ul class="thumbnail-list">
    ▼<li class="thumbnail-item">
      ▼<a href="#">
          <span class="thumbnail-title">Barry</span>
        </a>
      </li>
    ▶<li class="thumbnail-item">…</li>
    ▶<li class="thumbnail-item">…</li>
    ▶<li class="thumbnail-item">…</li>
    ▶<li class="thumbnail-item">…</li>
   </ul>
   <div style="display: block;"></div>
 </body>
</html>
```

```
element.style {
}
.thumbnail-image {
  display: block;
  width: 100%;
}
img {
  border:▶0;
}
a:-webkit-any-link {
  color: -webkit-link;
  text-decoration: underline;
  cursor: auto;
}
Inherited from li.thumbnail-item
li {
  display: list-item;
  text-align: -webkit-match-parent;
}
Inherited from ul.thumbnail-list
ul, menu, dir {
  display: block;
```

Why does this work? Images are `display: inline` by default. They are subject to similar rendering rules as text. When text is rendered, the letters are drawn along a common baseline. Some characters, such as p, q, and y, have a *descender* - the tail that drops below this baseline. To accommodate them, there is some whitespace included below the baseline.

Setting the `display` property to `block` removes the whitespace because there is no need to accommodate any text (or any other `display: inline` elements that might be rendered alongside the image).

Color

It is time to explore color a little more deeply. Add the following color styles for the `body` element and the `.thumbnail-item` class in `styles.css`.

```
body {
  font-size: 10px;
  background: rgb(149, 194, 215);
}

a {
  text-decoration: none;
}

.thumbnail-item {
  border: 1px solid rgb(100%, 100%, 100%);
  border: 1px solid rgba(100%, 100%, 100%, 0.8);
}
```

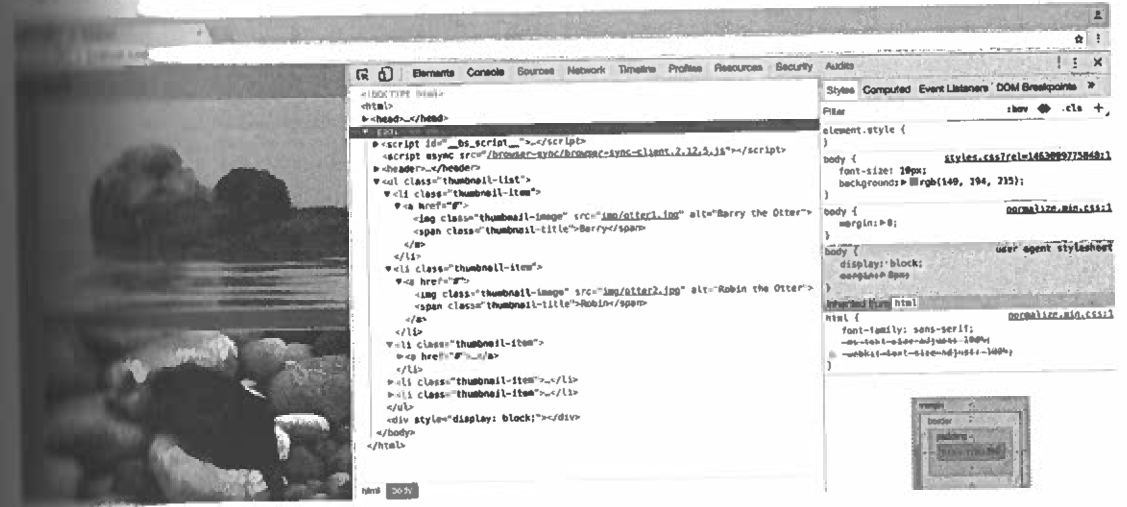
You have declared values for the `.thumbnail-item`'s border twice. Why? Notice that the two declarations use slightly different color functions: `rgb` and `rgba`. The `rgba` color function accepts a fourth argument, which is the opacity. However, some browsers do not support `rgba`, so providing both declarations is a technique that provides a *fallback* value.

All browsers will see the first declaration (`rgb`) and register its value for the `border` property. When browsers that do not support `rgba` see the second declaration, they will not understand it and will simply ignore it, using the value from the first declaration. Browsers that *do* support `rgba` will use the value in the second declaration and discard the value from the first declaration.

(Wondering why the `body`'s background color is defined with integers and the `.thumbnail-item`'s border color is defined with percentages? We will come back to that in just a moment.)

Save `styles.css` and switch to your browser (Figure 3.19).

Figure 3.19 Background color and borders

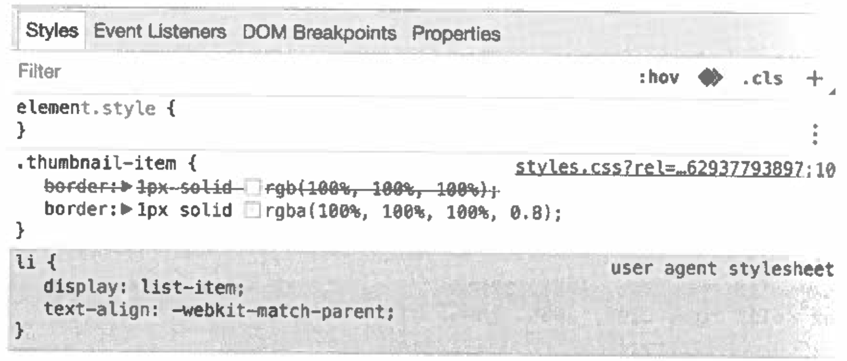


```
<!DOCTYPE html>
<html>
▶<head>…</head>
▼<body>
  ▶<script id="__bs_script__">…</script>
   <script async src="/browser-sync/browser-sync-client.2.12.5.js"></script>
  ▶<header>…</header>
  ▼<ul class="thumbnail-list">
    ▼<li class="thumbnail-item">
      ▼<a href="#">
          <img class="thumbnail-image" src="img/otter1.jpg" alt="Barry the Otter">
          <span class="thumbnail-title">Barry</span>
        </a>
      </li>
    ▼<li class="thumbnail-item">
      ▼<a href="#">
          <img class="thumbnail-image" src="img/otter2.jpg" alt="Robin the Otter">
          <span class="thumbnail-title">Robin</span>
        </a>
      </li>
    ▼<li class="thumbnail-item">
      ▶<a href="#">…</a>
      </li>
    ▶<li class="thumbnail-item">…</li>
    ▶<li class="thumbnail-item">…</li>
   </ul>
   <div style="display: block;"></div>
 </body>
</html>
```

```
element.style {
}
body {                                  styles.css?rel=1463009775840:1
  font-size: 10px;
  background: ▶ rgb(149, 194, 215);
}
body {                                  normalize.min.css:1
  margin:▶0;
}
body {                                  user agent stylesheet
  display: block;
  margin:▶ 8px;
}
Inherited from html
html {                                  normalize.min.css:1
  font-family: sans-serif;
  -ms-text-size-adjust: 100%;
  -webkit-text-size-adjust: 100%;
}
```

In the DevTools, you can see that Chrome supports `rgba`. It denotes that the `rgb` color is not used by striking through the style (Figure 3.20)

Figure 3.20 rgba is used when supported by browser

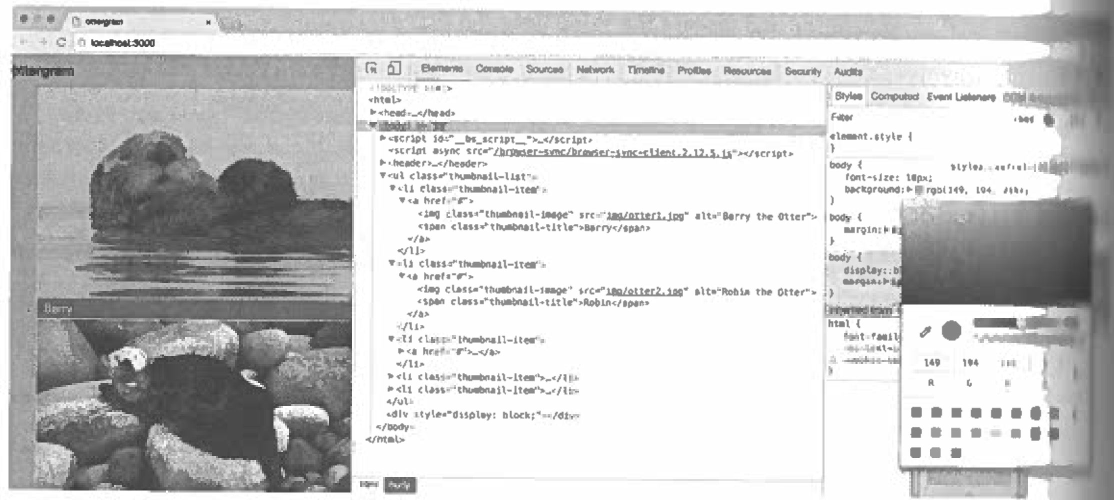


```
Styles | Event Listeners | DOM Breakpoints | Properties
Filter                                            :hov  .cls  +
element.style {
}
.thumbnail-item {                            styles.css?rel=…62937793897:10
  ~~border:▶ 1px solid □ rgb(100%, 100%, 100%);~~
  border:▶ 1px solid □ rgba(100%, 100%, 100%, 0.8);
}
li {                                                user agent stylesheet
  display: list-item;
  text-align: -webkit-match-parent;
}
```

Now, still in the DevTools, select the `body`. In the styles pane, notice the declaration for the background color that you just added. To the left of the RGB value is a small square showing you what the color will look like.

Click that square, and a color picker opens (Figure 3.21). The color picker lets you choose a color and will give you the CSS color value in a variety of different formats.

Figure 3.21 The color picker in the styles pane



```
<!DOCTYPE html>
<html>
▶<head>…</head>
▼<body>
  ▶<script id="__bs_script__">…</script>
   <script async src="/browser-sync/browser-sync-client.2.12.5.js"></script>
  ▶<header>…</header>
  ▼<ul class="thumbnail-list">
    ▼<li class="thumbnail-item">
      ▼<a href="#">
         <img class="thumbnail-image" src="img/otter1.jpg" alt="Barry the Otter">
         <span class="thumbnail-title">Barry</span>
       </a>
     </li>
    ▼<li class="thumbnail-item">
      ▼<a href="#">
         <img class="thumbnail-image" src="img/otter2.jpg" alt="Robin the Otter">
         <span class="thumbnail-title">Robin</span>
       </a>
     </li>
    ▼<li class="thumbnail-item">
      ▶<a href="#">…</a>
     </li>
    ▶<li class="thumbnail-item">…</li>
    ▶<li class="thumbnail-item">…</li>
   </ul>
   <div style="display: block;"></div>
 </body>
</html>
```

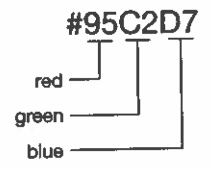
```
element.style {
}
body {
  font-size: 10px;
  background:▶ rgb(149, 194, …);
}
```

To see the background color in different color formats, click the up and down arrows to the right of the RGBA values. You can cycle through HSLA, HEX, and RGBA formats.

The HSLA format (which stands for "hue saturation lightness alpha") is used less frequently than the others, partly because some of the most popular design tools do not provide HSLA values that are accurate for CSS. If you are curious about HSLA, visit the HSLA Explorer at `css-tricks.com/examples/HSLaExplorer`.

Take a look at the HEX value for the background color: `#95C2D7`. HEX, or hexadecimal, is the oldest color specification format. Each digit represents a value from 0 to 15. (If you are not familiar with hexadecimal numbers, this is done by including the characters A through F as digits.) Each pair of digits, then, can represent a value from 0 to 255. From left to right, the pairs of digits correspond to the quantity of red, green, and blue in the color being specified (Figure 3.22).

Figure 3.22 HEX values correspond to red, green, and blue values



Many find HEX colors unintuitive. A modern alternative is to use RGB (red, green, and blue) values. In this model, each color is also assigned a value from 0 to 255, but the values are represented in more familiar decimal numbers and separated by color. As mentioned earlier, for more capable browsers a fourth value can specify the opacity or transparency of the specified color, from 0.0 (fully transparent) to 1.0 (fully opaque). The opacity is officially known as the *alpha* value – hence the A in RGBA. The RGBA value of the body's background color is `(149, 194, 215, 1)`.

As an alternative to declaring integer values for red, green, and blue, you can also use percentages, as you did for the `.thumbnail-item` borders. There is no functional difference between the two options. Just do not mix percentages and integers in the same declaration.

By the way, for help selecting pleasing color palettes, Adobe provides a free online tool at `color.adobe.com`.

Adjusting the Space Between Items

Ottergram now has some nice colors reminiscent of otters' ocean home. But adding the colors has revealed some unwanted whitespace inside the border of the `.thumbnail-item` elements. Also, those pesky bullets are drawing attention away from the glory of the otters.

To get rid of the bullets, set the `.thumbnail-list`'s `list-style` property to `none` in `styles.css`:

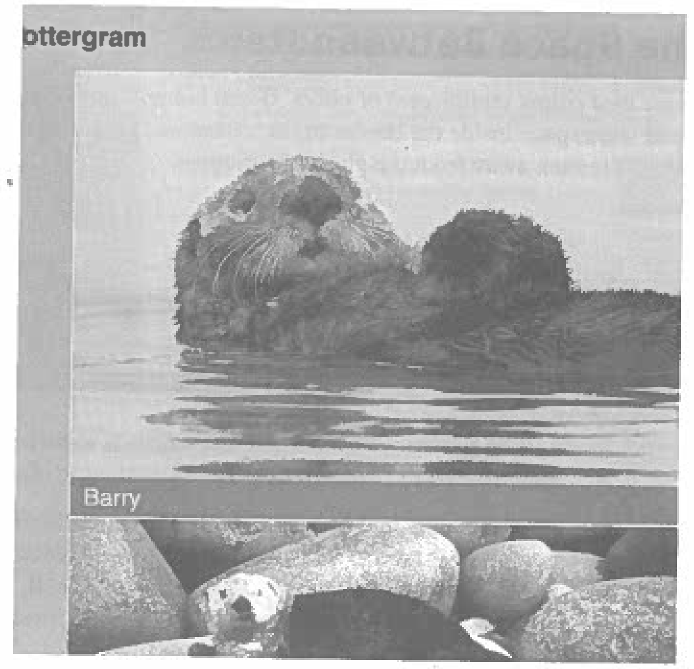
```
...
.thumbnail-item {
  border: 1px solid rgb(100%, 100%, 100%);
  border: 1px solid rgba(100%, 100%, 100%, 0.8);
}

.thumbnail-list {
  list-style: none;
}

.thumbnail-image {
  ...
```

To get rid of the whitespace, you will use the same technique you used with the `.thumbnail-image`. Each `.thumbnail-item` has that whitespace by default to accommodate items in a list, just as the `.thumbnail-image` elements had whitespace to accommodate neighboring text. Add a `display: block` declaration for `.thumbnail-item` to remove it.

```
...
.thumbnail-item {
  display: block;
  border: 1px solid rgb(100%, 100%, 100%);
  border: 1px solid rgba(100%, 100%, 100%, 0.8);
}
...
```

With those additions, the bullets and the excess space above the images disappear, resulting in the more polished layout shown in Figure 3.23.

Figure 3.23 Improved layout



Why use a bullet list if you do not want bullets? It is best to choose HTML tags based on what they are and not how the browser will style them by default. In this case, you want an unordered list of images, so a `ul` is the way to go. The `ul` container for your images will let you style them as a scrolling list when you add a detail image to your project in Chapter 4. The fact that the browser represents `ul`s with bullets by default is not important, as they are easily removed.

Next, you are going to adjust the spacing of the items in the list. The individual `.thumbnail-item` elements currently have no space between them. You are going to add margins between adjacent thumbnails.

However, you do not want to add a margin to *all* of the list items. Why not? Because the heading already has a margin, so the first list item does not need one. This means that you cannot use the `.thumbnail-item` class selector, at least not on its own. Instead, you will use selector syntax that targets elements based on their relationship to other elements.

Relationship selectors

Look again at the diagram of your project in Figure 3.10. It looks much like a family tree, doesn't it? This similarity gives the set of relationship selectors their names: *descendent selectors*, *child selectors*, *sibling selectors*, and *adjacent sibling selectors*.

Relationship selector syntax includes two selectors (like class or element selectors) joined by a symbol called a *combinator* that determines the targeted relationship between them. To understand how relationship selectors work, it is important to keep in mind that the browser reads selector syntax from *right to left*. Let's look at some examples.

A descendent selector targets any element of one specified type that is the descendent of another specified element. For example, to select any `span` element that is the descendent of the `body` element, the syntax would be:

```
body span {
  /* style declarations */
}
```

This syntax uses no combinator. Because it is read from right to left, it targets any `span` descended from a `body`, which in the current code means the thumbnail titles. It would also affect any `span`s that might be added within the `header` or elsewhere within the `body`.

Note that you can also use a class selector (or attribute selector, or indeed any type of selector) within a relationship selector, so the selector above could also be written as:

```
body .thumbnail-title {
  /* style declarations */
}
```

Child selectors target elements of a specified type that are the immediate children of another specified element. Child selector syntax uses the combinator >. To use child selector syntax to target all the `span`s currently in Ottergram, the syntax would be:

```
li > span {
  /* style declarations */
}
```

Reading from right to left, this selector targets any `span` that is the immediate child of a `li` element – again, the thumbnail titles.

Sibling selector syntax uses the combinator ~. As you might expect, this syntax targets elements with the same parent. However, because of the directional nature of relationship selectors, the results might not be exactly as you expect. Take this example:

```
header ~ ul {
  /* style declarations */
}
```

This selector targets any `ul` that is *preceded* by a `header` with the same parent element. This selector would effectively target Ottergram's `ul`, because it has a sibling `header` that precedes it in the code. However, reversing the syntax (`ul ~ header`) would result in no elements being selected, because there is no `header` preceded by a sibling `ul`.

The final relationship selector type is the adjacent sibling selector, which targets elements that are *immediately* preceded by a sibling of the specified type. The adjacent sibling combinator is +:

```
li + li {
  /* style declarations */
}
```

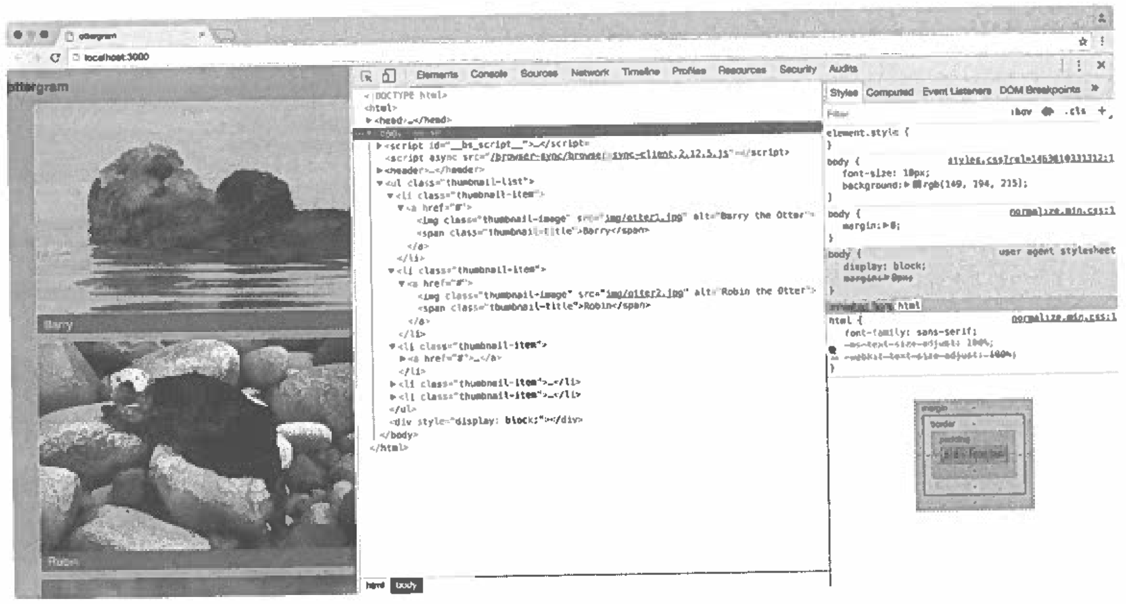
This syntax would select all `li` elements immediately preceded by a sibling `li`. The result is that the declared styles would be applied to the second through fifth `li` – but not the first, because it is not immediately preceded by another `li`. (Note that the general sibling selector and the adjacent sibling selector would work the same way at the moment, due to Ottergram's relatively simple structure.)

Back to the task at hand: adding a margin to the top of each list item except the first. If you used a descendent or child selector to target the `.thumbnail-item` class or the `span` or `li` elements, the margin would be applied to all five thumbnails. Because you want to style all but the first, use the adjacent sibling syntax in `styles.css` to add a top margin to only those thumbnails that are immediately preceded by another thumbnail.

```
...
a {
  text-decoration: none;
}

.thumbnail-item + .thumbnail-item {
  margin-top: 10px;
}

.thumbnail-item {
  ...
```

Save your file and check out the results in your browser (Figure 3.24).

Figure 3.24 Spacing between adjacent `.thumbnail-item` elements

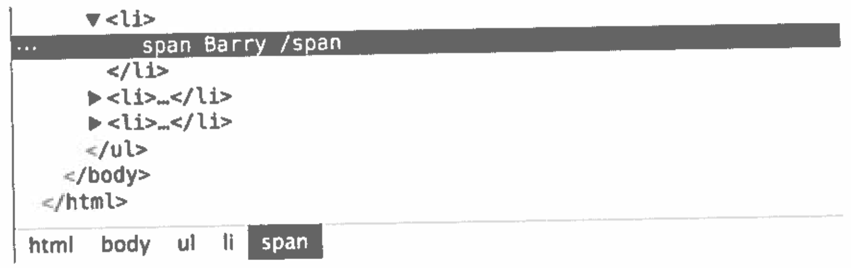


```
<!DOCTYPE html>
<html>
▶<head>…</head>
▼<body>
  ▶<script id="__bs_script__">…</script>
  <script async src="/browser-sync/browser-sync-client.2.12.5.js"></script>
  ▶<header>…</header>
  ▼<ul class="thumbnail-list">
    ▼<li class="thumbnail-item">
      ▼<a href="#">
        <img class="thumbnail-image" src="img/otter1.jpg" alt="Barry the Otter">
        <span class="thumbnail-title">Barry</span>
      </a>
    </li>
    ▼<li class="thumbnail-item">
      ▼<a href="#">
        <img class="thumbnail-image" src="img/otter2.jpg" alt="Robin the Otter">
        <span class="thumbnail-title">Robin</span>
      </a>
    </li>
    ▼<li class="thumbnail-item">
      ▶<a href="#">…</a>
    </li>
    ▶<li class="thumbnail-item">…</li>
    ▶<li class="thumbnail-item">…</li>
  </ul>
  <div style="display: block;"></div>
</body>
</html>
```

```
element.style {
}
body {                                  styles.css?rel=[illegible]:1
  font-size: 10px;
  background: ▶ rgb(149, 194, 215);
}
body {                                  normalize.min.css:1
  margin: ▶ 0;
}
body {                                  user agent stylesheet
  display: block;
  ~~margin: ▶ 8px;~~
}
Inherited from html
html {                                  normalize.min.css:1
  font-family: sans-serif;
  ~~-ms-text-size-adjust: 100%;~~
  ~~-webkit-text-size-adjust: 100%;~~
}
```

Note that the DevTools give you an easy way to find out the nesting path of an element, which can help with writing relationship selectors. If you click one of the `span` elements inside one of the `li` elements, you can see its path at the bottom of the elements panel (Figure 3.25).

Figure 3.25 Nesting path shown by the elements panel



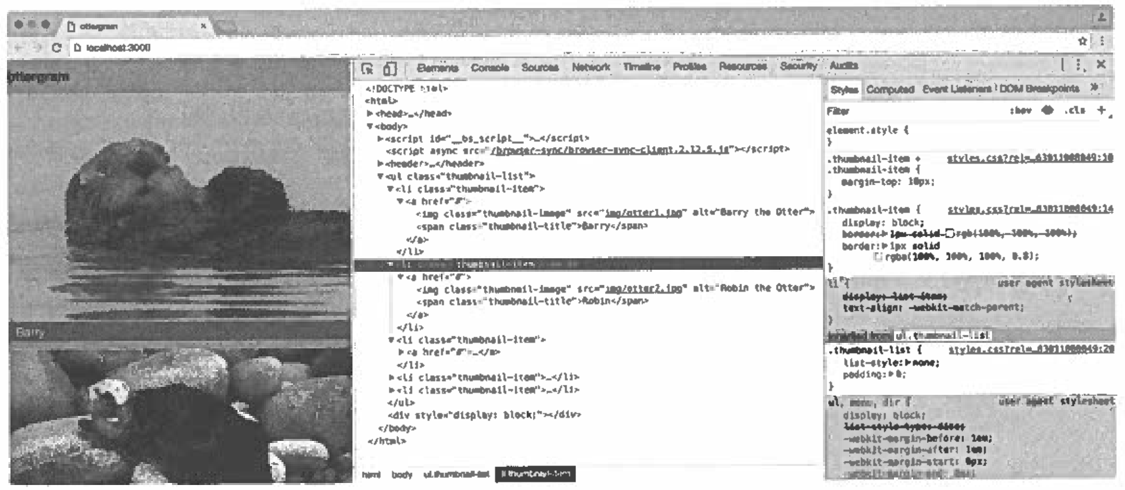
```
      ▼<li>
...        span Barry /span
        </li>
      ▶<li>…</li>
      ▶<li>…</li>
      </ul>
    </body>
  </html>
html  body  ul  li  span
```

For one final tweak to the thumbnail list's appearance, return to `styles.css` and override the padding that the `ul` inherits from the user agent stylesheet so that the images are no longer indented.

```
...
.thumbnail-list {
  list-style: none;
  padding: 0;
}
...
```

As usual, save your file and switch to your browser to see your results (Figure 3.26).

Figure 3.26 `ul` with padding removed


ottergram
localhost:3000
ottergram
Barry
Elements Console Sources Network Timeline Profiles Resources Security Audits
Styles Computed Event Listeners DOM Breakpoints
Filter :hov .cls +
html body ul.thumbnail-list li.thumbnail-item

```
<!DOCTYPE html>
<html>
▶<head>…</head>
▼<body>
  ▶<script id="__bs_script__">…</script>
   <script async src="/browser-sync/browser-sync-client.2.12.5.js"></script>
  ▶<header>…</header>
  ▼<ul class="thumbnail-list">
    ▼<li class="thumbnail-item">
      ▼<a href="#">
         <img class="thumbnail-image" src="img/otter1.jpg" alt="Barry the Otter">
         <span class="thumbnail-title">Barry</span>
       </a>
     </li>
    ▼<li class="thumbnail-item">
      ▼<a href="#">
         <img class="thumbnail-image" src="img/otter2.jpg" alt="Robin the Otter">
         <span class="thumbnail-title">Robin</span>
       </a>
     </li>
    ▼<li class="thumbnail-item">
      ▶<a href="#">…</a>
     </li>
    ▶<li class="thumbnail-item">…</li>
    ▶<li class="thumbnail-item">…</li>
   </ul>
   <div style="display: block;"></div>
  </body>
</html>
```

```
element.style {
}
.thumbnail-item +        styles.css?rel=1430311000049:10
.thumbnail-item {
  margin-top: 10px;
}
.thumbnail-item {        styles.css?rel=1430311000049:14
  display: block;
  border: ▶ 1px solid rgb(100%, 100%, 100%);
  border: ▶ 1px solid
        rgba(100%, 100%, 100%, 0.8);
}
li {                     user agent stylesheet
  display: list-item;
  text-align: -webkit-match-parent;
}
Inherited from ul.thumbnail-list
.thumbnail-list {        styles.css?rel=1430311000049:20
  list-style: ▶ none;
  padding: ▶ 0;
}
ul, menu, dir {          user agent stylesheet
  display: block;
  list-style-type: disc;
  -webkit-margin-before: 1em;
  -webkit-margin-after: 1em;
  -webkit-margin-start: 0px;
```

Ottergram is starting to look polished. With some styling for the header, you will have a nice static web page.

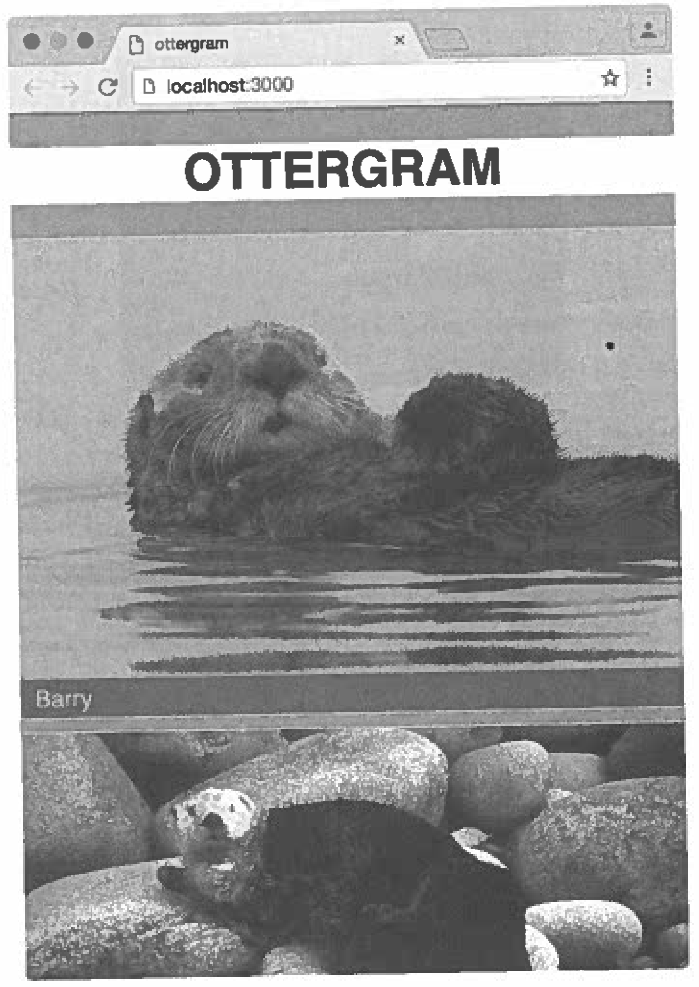
Adding a Font

Earlier, you added the `.logo-text` class to the `h1` element. Use that class as the selector for a new styling rule in `styles.css`. Insert it after the styles for the anchor tag. (In general, the order of your styles only matters when you have multiple rule sets for the same selector. In Ottergram, the styles are arranged in roughly the same order as they appear in the code. This is a matter of preference, and you are free to organize your styles as you see fit.)

```
...
a {
  text-decoration: none;
}

.logo-text {
  background: white;

  text-align: center;
  text-transform: uppercase;
  font-size: 37px;
}

.thumbnail-item + .thumbnail-item {
  ...
```

First, you gave the header a white background. Then you centered the text inside the `.logo-text` element and used the `text-transform` property to format it as uppercase. Finally, you set the font size. Your results will look like Figure 3.27.

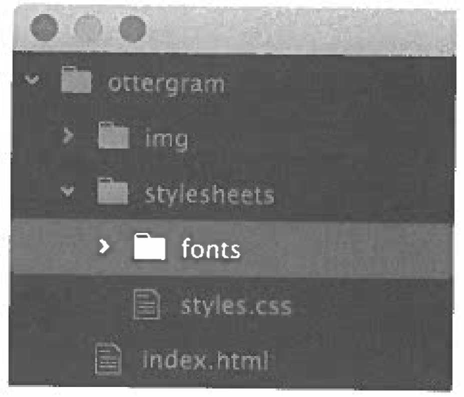
Figure 3.27 Styling the header



Ottergram looks great. Great… but a little plain for a website with *otters*. To add some pizzazz, you can use a font for the header other than the default provided by the user agent stylesheet.

We included some fonts in the resource files you already downloaded and added to your project directory. To use them, you need to copy the `fonts` folder into your project. Place it *inside* your `stylesheets` folder (Figure 3.28).

Figure 3.28 `fonts` folder inside `stylesheets` folder



Now you only need to point some styles to those fonts.

The resource files include many formats of each font. As usual, different browser vendors support different kinds of fonts. To support the widest array of browsers, you need to include all of them in your project. Yes, all of them.

To help you out, the *@font-face* syntax lets you give a custom name to a family of fonts that you can then use in the rest of your styles.

An `@font-face` block is a little different from the declaration blocks you have been using. Inside of the `@font-face` block are three main parts:

- First, the `font-family` property, whose value is a string identifying the custom font name you can use throughout your CSS file.
- Next, several `src` declarations specifying different font files. (Take note – the order is important!)
- Last, declarations that modify the font's presentation, such as the `font-weight` and the `font-style`.

Add an `@font-face` declaration for the `lakeshore` font family to the top of `styles.css` and a style declaration to use the new font for the `.logo-text` class.

```
@font-face {
    font-family: 'lakeshore';
    src: url('fonts/LAKESHOR-webfont.eot');
    src: url('fonts/LAKESHOR-webfont.eot?#iefix') format('embedded-opentype'),
         url('fonts/LAKESHOR-webfont.woff') format('woff'),
         url('fonts/LAKESHOR-webfont.ttf') format('truetype'),
         url('fonts/LAKESHOR-webfont.svg#lakeshore') format('svg');
    font-weight: normal;
    font-style: normal;
}

body {
  font-size: 10px;
  background: rgb(149, 194, 215);
}

a {
  text-decoration: none;
}

.logo-text {
  background: white;

  text-align: center;
  text-transform: uppercase;
  font-family: lakeshore;
  font-size: 37px;
}
...
```

Admittedly, getting the `@font-face` declaration just right can be tricky, because the order of the individual `url` values is important. It is a good idea to keep a copy of the declaration for reference. You can also look into Atom's snippets documentation at `flight-manual.atom.io/using-atom/sections/snippets` to see how to create your own code "snippet," or template.

After declaring the custom `@font-face`, the rest of your CSS has access to the new `lakeshore` value for the `font-family` property. In the `.logo-text` declaration, you set `font-family: lakeshore` to apply the new font.

Save `styles.css`, switch to Chrome, and see how good it feels to have a web page as stylish as an otter (Figure 3.29).

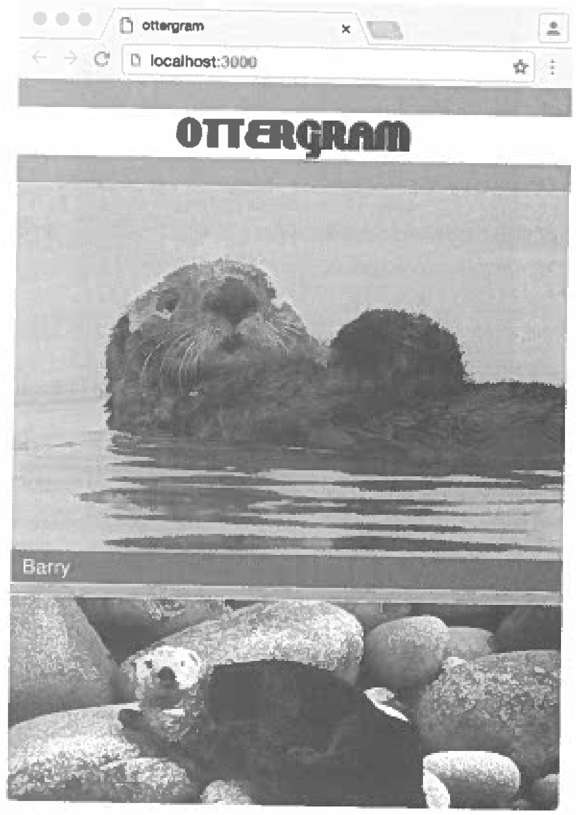


Figure 3.29 Applying a custom font to the header

You did a lot of styling work in this chapter, and Ottergram looks great! In the next chapter you will make it even better by adding interactive functionality.

Bronze Challenge: Color Change

Change the background color styles for body. Use the color picker in the DevTools (Figure 3.21) to help you choose one.

For a more sophisticated color palette, go to `color.adobe.com` and create your own scheme for the body and `.thumbnail-title` background colors.

For the More Curious: Specificity! When Selectors Collide...

You have already seen how you can override styles. You included the link for `normalize.css` before the one for `styles.css`, for example. This made the browser use `normalize.css`'s styles as a baseline, with your styles taking precedence over the baseline styles.

This is the first basic concept of how the browser chooses which styles to apply to the elements on the page, known to front-end developers as *recency*: As the browser processes CSS rules, they can override rules that were processed earlier. You can control the order in which the browser processes CSS by changing the order of the `<link>` tags.

This is simple enough when the rules have the same selector (for example, if your CSS and `normalize.css` were to declare a different `margin` for the `body` element). In this case, the browser chooses the more recent declaration. But what about elements that are matched by more than one selector?

Say you had these two rules in your Ottergram CSS:

```
.thumbnail-item {
  background: blue;
}

li {
  background: red;
}
```

Both of these match your `<li>` elements. What background color will your `<li>` elements have? Even though the `li { background: red; }` rule is more recent, `.thumbnail-item { background: blue; }` will be used. Why? Because it uses a class selector, which is more specific (i.e., assigned a higher specificity value) than the element selector.

Class selectors and attribute selectors have the same degree of specificity, and both have a higher specificity than element selectors. The highest degree of specificity goes to *ID selectors*, which you have not seen yet. If you give an element an `id` attribute, you can write an ID selector that is more specific than any other selector.

ID attributes look like other attributes. For example:

```
<li class="thumbnail-item" id="barry-otter">
```

To use the ID in a selector, you prefix it with #:

```
.thumbnail-item {
  background: blue;
}

#barry-otter {
  background: green;
}

li {
  background: red;
}
```

In this example, the `<li>` is matched by all three selectors, but it will have a green background because the ID selector has the highest specificity. The order of your rulesets makes no difference here, because each has a different specificity.

One note about using ID selectors: It is best to avoid them. ID values must be unique in the document, so you cannot use the `id="barry-otter"` attribute for any other element in your document. Even though ID selectors have the highest specificity, their associated styles cannot be reused, making them a maintenance "worst practice."

To learn more about specificity, go to the MDN page `developer.mozilla.org/en-US/docs/Web/CSS/Specificity`.

The Specificity Calculator at `specificity.keegan.st` is a great tool for comparing the specificty of different selectors. Check it out to get a more precise understanding of how specificity is computed.