4. Find an online version of the Ruby API and read about the Hash method **merge**. What is the value of the following expression?

```
 { "a" => 100, "b" => 200 }.merge({ "b" => 300 })
```

4.3.4 CSS revisited

It's time now to revisit the line from Listing 4.1 used in the layout to include the Cascading Style Sheets:

```
<%= stylesheet_link_tag 'application', media: 'all',
                                       'data-turbolinks-track': 'reload' %>
```

We are now nearly in a position to understand this. As mentioned briefly in Section 4.1, Rails defines a special function to include stylesheets, and

```
stylesheet_link_tag 'application', media: 'all',
                                   'data-turbolinks-track': 'reload'
```

is a call to this function. But there are several mysteries. First, where are the parentheses? In Ruby, they are optional, so these two are equivalent:

```
# Parentheses on function calls are optional.
# This:
stylesheet_link_tag('application', media: 'all',
                                   'data-turbolinks-track': 'reload')
# is the same as this:
stylesheet_link_tag 'application', media: 'all',
                                   'data-turbolinks-track': 'reload'
```

Second, the **media** argument sure looks like a hash, but where are the curly braces? When hashes are the *last* argument in a function call, the curly braces are optional, so these two are equivalent:

```
# Curly braces on final hash arguments are optional.
# This:
stylesheet_link_tag 'application', { media: 'all',
                                     'data-turbolinks-track': 'reload' }
# is the same as this:
stylesheet_link_tag 'application', media: 'all',
                                   'data-turbolinks-track': 'reload'
```

Finally, why does Ruby correctly interpret the lines

```
stylesheet_link_tag 'application', media: 'all',
                                   'data-turbolinks-track': 'reload'
```

even with a line break between the final elements? The answer is that Ruby doesn't distinguish between newlines and other whitespace in this context.[17] The *reason* I chose to break the code into pieces is that I prefer to keep lines of source code under 80 characters for legibility.[18]

So, we see now that the line

```
stylesheet_link_tag 'application', media: 'all',
                                   'data-turbolinks-track': 'reload'
```

calls the **`stylesheet_link_tag`** function with two arguments: a string, indicating the path to the stylesheet, and a hash with two elements, indicating the media type and telling Rails to use the turbolinks feature added in Rails 4.0. Because of the **`<%= ... %>`** brackets, the results are inserted into the template by ERb, and if you view the source of the page in your browser you should see the HTML needed to include a stylesheet (Listing 4.14). (The extra stuff in Listing 4.14, like **`?body=1`** and the long string of hexadecimal digits are,

[17]A newline is what comes at the end of a line, thereby starting a new line. As noted in Section 4.2.1, it is typically represented by the character **`\n`**.

[18]Constantly having to check the column number is rather inconvenient, so many text editors have a visual aid to help you. For example, if you take a look back at Figure 1.12, you may be able to make out the small vertical line on the right side of the screen, which is designed to help keep code under 80 characters. (It's very subtle, so you may not be able to see it in the screenshot.) The cloud IDE (Section 1.1.1) includes such a line by default. In Sublime Text, you can use `View > Ruler > 78` or `View > Ruler > 80`.

inserted by Rails to ensure that browsers reload the CSS when it changes on the server. Because the hex string is by design unique, your exact version of Listing 4.14 will differ.)

Listing 4.14: The HTML source produced by the CSS includes.

```
<link rel="stylesheet" media="all" href="/assets/application.self-
f0d704deea029cf000697e2c0181ec173a1b474645466ed843eb5ee7bb215794.css?body=1"
data-turbolinks-track="reload" />
```

4.4 Ruby classes

We've said before that everything in Ruby is an object, and in this section we'll finally get to define some of our own. Ruby, like many object-oriented languages, uses *classes* to organize methods; these classes are then *instantiated* to create objects. If you're new to object-oriented programming, this may sound like gibberish, so let's look at some concrete examples.

4.4.1 Constructors

We've seen lots of examples of using classes to instantiate objects, but we have yet to do so explicitly. For example, we instantiated a string using the double quote characters, which is a *literal constructor* for strings:

```
>> s = "foobar"       # A literal constructor for strings using double quotes
=> "foobar"
>> s.class
=> String
```

We see here that strings respond to the method **`class`**, and simply return the class they belong to.

Instead of using a literal constructor, we can use the equivalent *named constructor*, which involves calling the **`new`** method on the class name:[19]

[19]These results will vary based on the version of Ruby you are using. This example assumes you are using Ruby 1.9.3 or later.