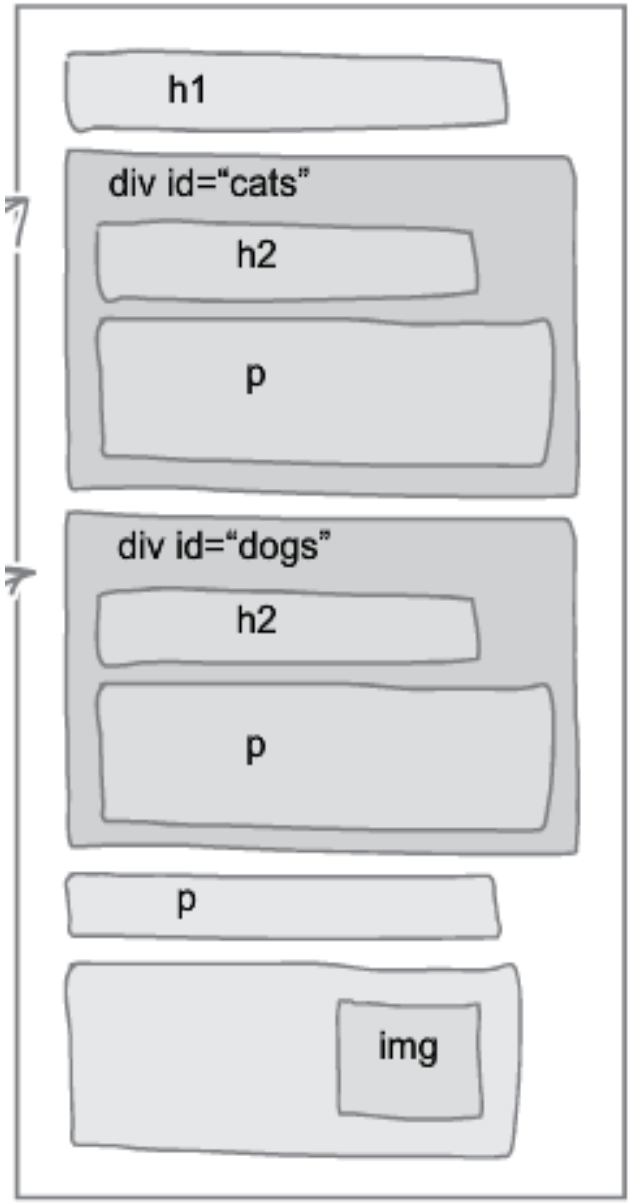
Lecture 10: `Divs`, `Spans`, Descendant Selectors, Pseudo-Classes, and the Cascade

Dr. Jason Schanker

The div element

- When you want to group a part of related content together in order to treat it as a single entity for styling purposes, you can enclose the HTML in `<div>` and `</div>` tags. The `div` elements are used to `div`ide your web page into logical blocks; use the `id` attribute as an identifier for each such section.
- It's a general purpose container; it "contains" all of the related elements. Since we're not going to have elements specific for every type of grouping (e.g., no `cats` element and no `dogs` element), it's the general purpose "catch-all" element.
- HTML5 introduces new elements for common logical groupings for which `div` was used previously (e.g., a `header`; `<header>…</header>` is now used in place of something like `<div id = "header">…</div>`).



Adding some style

Okay, so you've added some logical structure to the PetStorz page, and you've also labeled that structure by giving each `<div>` a unique id. That's all you need to start styling the group of elements contained in the `<div>`.

Here we have two rules, one for each . Each is selected by an id selector.

```
#cats {
   background-image: url(leopard.jpg);
}

#dogs {
   background-image: url(mutt.jpg);
}
```

Each rule sets the background-image property. For cats we have a leopard image, and for dogs we have a mutt image.

Now the s have a little style.

By setting the background on the , it also shows through the elements contained in the .

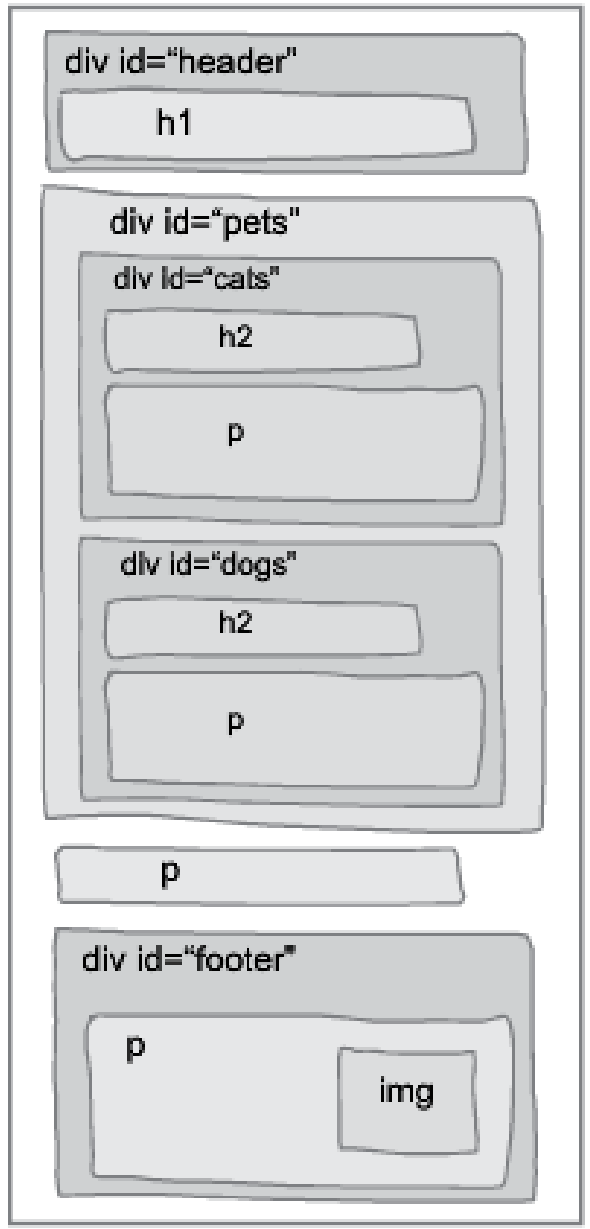
The elements in the will also inherit some properties from the , just as any child element does (like font-size, color, etc).

Nested Divs (Divs inside divs)

Adding structure on structure

And you don't have to stop there. It is common to nest structure, too. For instance, in the PetStorz page, we have a cat section and a dog section, and the two together are logically the "pets" section of the page. So, we could place both the "cats" and "dogs" `<div>`s into a "pets" `<div>`.

Now we've marked up this HTML so that we know there is a logical section in the page with "pets" content in it. Further, that "pets" section has two logical subsections, one for "cats" and one for "dogs".



Exercise: Creating a Stylized Elixir Menu Part I

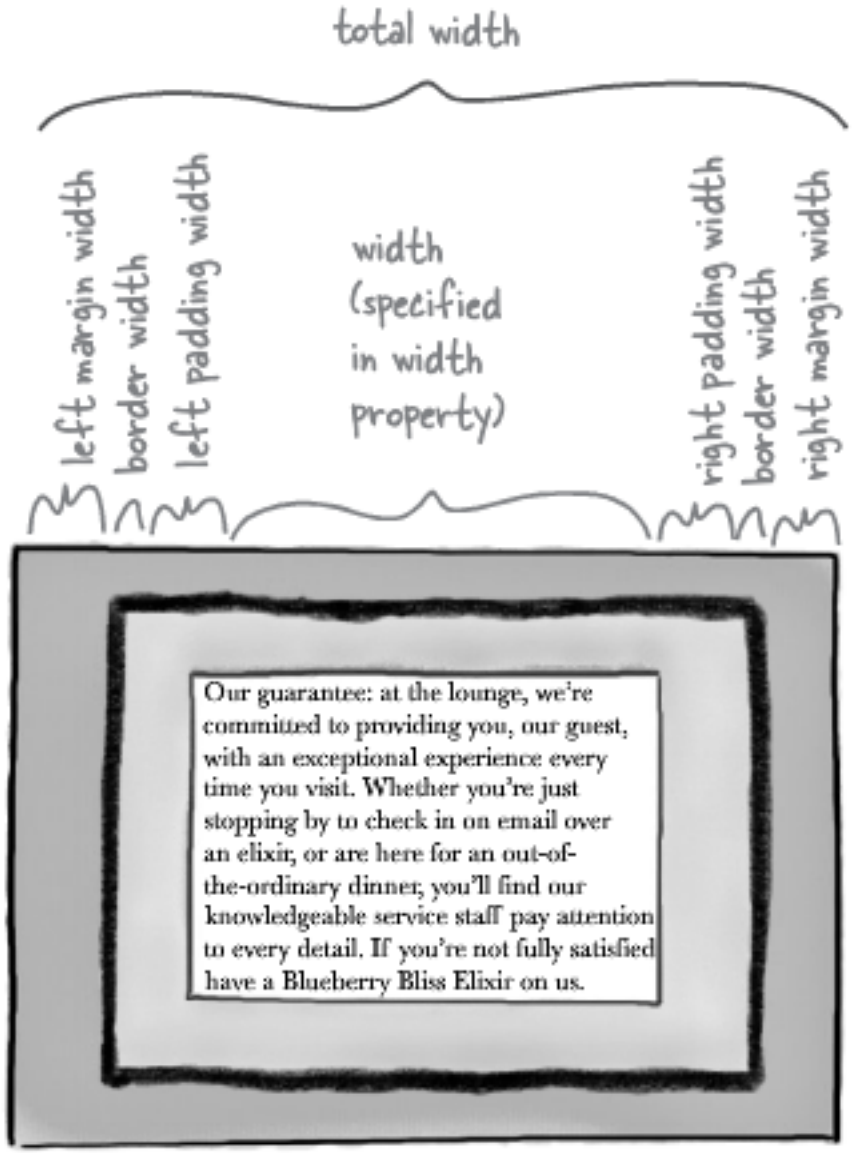
I. Create a logical unit from the elixirs menu by enclosing the HTML in `<div>` and `</div>` tags. Label it (`id` attribute) `elixirs`.

II. Add a CSS rule to give this `div` a thin solid aquamarine (`#007e7e`) border.

III. Set the `width` property to a value of 200px (a width that's about one quarter of a standard browser window).

More on `divs` and Dimension Attributes

- The `div` element follows the box model like any other element (content, padding, border, margin). The width attribute only specifies the width of the content area. The total width will include the padding, borders, and margins on its left and right.
  - Exercise: If the (content) width is 200px, the left and right border are 2px, the left and right padding are 5px, and the left and right margins are 10px, how much is the total width?
- `div` is a `block` element so line breaks are added before and after it. Also, by default, the width of any block element is `auto`, meaning it'll expand horizontally to fill all available space, which in this case will be the entire browser window minus the margins of the `body` element.
- The default `height` of the element will also be auto, but in the case of height, this means the *least* amount of vertical space required to fit the content. In general, you should leave this attribute alone so the content doesn't overflow into the content below it.



Text-align property

- The `text-align` property can be given a value of left, right, or center for *block* elements. Doing so aligns all of its inline content. However, children block elements inherit this property value (unless specified otherwise) from their parents. So if a child block element has text, for example, the text will be aligned according to its parent's block element `text-align` value.
  - Example: Given:

```
<p class = "story">
  And the person uttered the following enlightening words:
    <blockquote>
      Blah blah <em>blah!</em> etc.
    </blockquote>.
  Instantly, <em>everything</em> was okay.
</p>
```

  - If you used the CSS rule:

```
.story {
    text-align:center;
}
```

Then this would tell all story class elements to center their inline content, causing the **text/inline content (including any images)** of the story class that's *not* part of a block element to center. However, also the text of the `blockquote` gets centered too since the `blockquote` inherits the text-align property value so it too tells its inline content to center itself.

Descendant selectors

- Sometimes, you want to specify property values only for elements nested inside a certain element. In this case, you can use descendant selectors by entering a parent (or grandparent, great grandparent, etc.) element followed by a space followed by the element as the selector.
  - Example: Given:

```
<p class = "story">
  And the person uttered the following enlightening words:
    <blockquote>
      Blah blah <em>blah!</em> etc.
    </blockquote>.
  Instantly, <em>everything</em> was okay.
</p>
```

  - You can select the blockquotes that are nested in story elements such as the one above with content, "Blah Blah blah etc." with a selector of for example:

```
.story blockquote {
    font-family: "Time New Roman", serif;
}
```

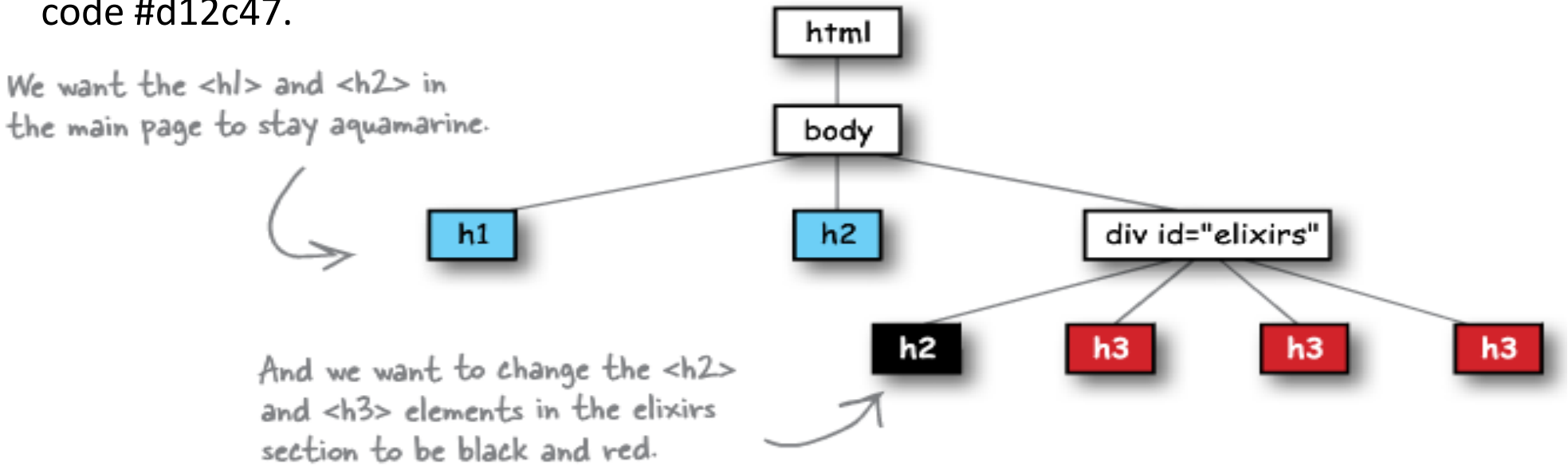
  - You could also chain together descendant selectors: e.g., `.story blockquote em` would specify the `<em>blah!</em>`.
  - To only select direct children, use > instead of space: e.g., `.story em` selects blah! and everything but `.story > em` only selects `<em>everything!</em>`

Exercise: Creating a Stylized Elixir Menu Part II

IV. Add a padding on the left, right, and bottom of the elixirs `div` of 10 pixels each. Then add a margin on the left of 20 pixels. To center all inline content of a block element (the elixirs div in this case), use `text-alignment:center;` . Finally, add the background image with relative path images/cocktail.gif and have it repeat horizontally only.

- Note (Subtle point): The text-alignment property is *not* aligning the elixirs div itself nor the block elements contained in it (e.g., the paragraph elements). Instead, the p (and other contained block) elements are inheriting the center text-alignment making its inline content, the text in this case, center itself within the block which spans the width of the elixirs div element.

V. Color the elixir `div`'s secondary text black and its tertiary text with the color with hex code #d12c47.



Last Adjustment: Fixing the Line Spacing

- If you set line-height to `1em` in a rule selecting the elixirs id (`#elixirs`), then it will be 1 times the height of the `elixirs` element text (i.e., single spaced for this text). However, If you override the text size of a nested element (say an elixirs heading that's 120% the size of this text), then the line spacing will still be with respect to the elixirs element text, *not* the larger heading text. In this case this means that the line spacing for the heading will be less than single spaced causing overlap.
- To fix this, remove the em. A value of a number without a unit for the line-height makes the line spacing be with respect to the text size of the nested element itself, not the elixirs text:

```
#elixirs {
   line-height: 1;
}
```

Shortening Rule bodies

- Some CSS property-value pairs can be combined.
  - Example: Instead of `margin-top: 5px; margin-right:10px; margin-bottom:15px; margin-left: 20px;` you can simply use **`margin: 5px 10px 15px 20px;`** (Order matters here, obviously, to know which number represents which margin.)
  - Example: Instead of `border-width: thin; border-style: solid; border-color: #007e7e;` you can use:

    `border: thin solid #007e7e;` (Order does not matter here.)
  - Example: Instead of `background-color: white; background-image: url(images/cocktail.gif); background-repeat: repeat-x;` you can use:

    `background: white url(images/cocktail.gif) repeat-x;`

    (Order does not matter here.)

Book Exercise: Copyright section

It's time to put all your new knowledge to work. You'll notice that at the bottom of the lounge, there's a small section with copyright information that acts as a footer for the page. Add a to make this into its own logical section. After you've done that, style it with these properties:

```
font-size: 50%;
text-align: center;
line-height: normal;
margin-top: 30px;
```

Let's make the text really small. You know, FINE PRINT.

And let's center the text.

We're also setting the line-height to be "normal", which is a keyword you haven't seen yet. "Normal" allows the browser to pick an appropriate size for the line height, which is typically based on the font.

And let's add some top margin to give the footer a little breathing room.

And while you're at it, have a look over the entire "lounge.css" file. Is there anywhere you might want to simplify things with shorthands? If so, go ahead and make those changes.

The span element

- The span element is like the div element in that it can be used to break up your content into logical parts, but unlike the div, which is a block element that helps organize the overall structure of the page, the span is an **inline** element, which is generally used for fine tuning of the structure within individual blocks.
- can also be thought of as an inline wild card tag (analogous to the block wild card) in that it can appropriately assume the role of any nonexistent HTML tag used to mark up inline content, which can then be styled with CSS accordingly.
  - Example (instead of fictitious "<food>" tag):

```
HTML: <span class = "food">pizza</span>
CSS: .food {
// Food Styling goes here
}
```

Book exercise (enhanced)

- Use appropriate HTML tags and CSS to make the music section have a thin top and bottom (no sides) dotted border. Then have the CDs appear in italics and the artists appear in bold as shown below:

MUSIC SECTION ➔

More on spans

there are no Dumb Questions

Q: When do I use a rather than another inline element like or ?

A: As always, you want to mark up your content with the element that most closely matches the meaning of your content. So, if you are emphasizing words, use ; if you're trying to make a big point, use . But if what you really want is to change the style of certain words—say, the names of albums or music artists on a fan site web page—then you should use a and put your elements into appropriate classes to group them and style them.

Q: Can I set properties like width on elements? Actually, what about inline elements in general?

A: You can set the width of inline elements like , , and , but you won't notice any effect until you position them (which you'll learn how to do in the next chapter). You can also add margin and padding to these elements, as well as a border. Margins and padding on inline elements work a little differently from block elements—if you add a margin on all sides of an inline element, you'll only see space added to the left and right. You can add padding to the top and bottom of an inline element, but the padding doesn't affect the spacing of the other inline elements around it, so the padding will overlap other inline elements.

Images are a little different from other inline elements. The width, padding, and margin properties all behave more like they do for a block element. Remember from Chapter 5: if you set the width of an image using either the width attribute in the <img> element or the width property in CSS, the browser scales the image to fit the width you specify. This can sometimes be handy if you can't edit the image yourself to change the dimensions, and you want the image to appear bigger or smaller on the page. But remember, if you rely on the browser to scale your image, you may be downloading more data than you need (if the image is larger than you need).

Pseudo-classes

- Links can change their styles on the fly depending on whether they've been visited yet, if the user is hovering over it (has mouse over it but not clicking), if it's in focus (hitting the tab button while in the window will likely cycle through the page's links and when it's the selected one, it's considered in focus), or active (clicking on the link without releasing the button).
- In the CSS, you can specify the link style of the unvisited links with `a:link`; the class of visited links with `a:visited`; the class of links being hovered over with `a:hover`, the class of links in focus with `a:focus`; and the class of active links with `a:active`.
  - Links can be in multiple states (e.g., visited and hover). In this case, the styling will go to the rule selecting the state listed *later*. The standard order of the rules (from top to bottom) is `:link`, `:visited`, `:hover`, `:focus`, and then `:active`. With this ordering, the hover style will be the one that's used if the link's only two states are visited and hover.
- `:link`, `:visited`, `:hover`, `:focus`, and `:active` are all examples of pseudo-classes because they behave like ordinary classes in that they group together elements that can be styled together, but are not *actually* classes because e.g., you don't specify classes with names like `:link`. Instead, the browser can change membership on the fly (e.g., When your mouse is placed over a link, it's added as a member of the `:hover` pseudo-class and when it's no longer over the link, it's removed.)
  - **Note**: The use of **:** instead of **.** for ordinary classes (e.g., p.greentea would be selecting paragraphs in the greentea class whereas p:hover would be selecting a paragraph that's currently being hovered over, i.e., a paragraph in the :hover pseudo-class.)

Enhanced Book Exercise

- Set the unvisited links on the page to have an aquamarine color of #007e7e and the visited links to have a gray color of #333333. For links that are currently being clicked, set the color to #f88396, and for links that are in focus, provide them with a thick solid border having color #0d5353. Finally, for hovered over links in the elixir section only, set the background color to #f88396 and the text color to #0d5353.
- Also, do a search on Google and think about how you can emulate the styles of the various link states with CSS.

Accessibility and The Cascade

- **Accessibility**: The browser has default styles that it uses for elements in which no styles are specified (directly or by inheritance). But the people who browse your web pages (your users) can create stylesheets as well for their own styles.
- **Cascade**: "The name "cascade" was chosen because of the way that styles coming from multiple stylesheets can all "cascade" down into the page, with the most specific styling being applied to each element." The browser decides which styles to use by first looking at the stylesheets for your web page (if any). If it can't find any styles to match a given element (directly specified or by inheritance), it then refers to your user's stylesheet for direction (if it exists), and if it can't get any relevant styling direction there, it falls to its defaults. The way the browser decides between which of these rules to use in given instances is called the cascade.
  - **Note**: A property value set by you user can be given precedence if it's proceeded by `!important` . For example, a visually impaired person may make body text bigger than its default:

```
body {
   font-size: 20px !important;
 }
```

- **Exercise**: See if you can figure out how to get your browser to use your own stylesheet for the defaults. Use `!important` in one of your rules and then visit some (safe) pages on the web to see the impact.

More on the Cascade (Tiebreaking)

- When more than one rule specifies a given element's property:

1. First, examining all the spreadsheets (author, user, browser), take all rules that most directly specify the element's property in question (i.e., a rule selecting the element itself is most direct; then IF THE PROPERTY IS INHERITED (color is, border is not), the parent of that element; then the grandparent of that element, etc.) If there's only one such rule, then that's the one that's applied.
2. From the rules left from (1), take only the one(s) from the user having the property value marked as `!important` if there are any; otherwise take only the one(s) from the author if there are any; otherwise take only the rule(s) from the user if there are any; otherwise select the browser's rule(s). If there's only one left, that's the rule that's applied.
3. From the rules left from (2), take only rules with selectors containing the most ids (e.g., `#elixirs #info p` has two) if there are any with at least one id; otherwise take only rules with selectors containing the most classes if there are any (e.g., `p.greentea q` has one class) with at least one id; otherwise take the one with the most element names. If only one remains, that's the rule that's applied.
4. From the rules left from (3), take the one appearing the latest (closest to the bottom if there's only one stylesheet linked in or if there are multiple stylesheets that are linked in, treat them as one single sheet with rules appearing in a stylesheet linked in closer to the bottom appearing afterward).

NOTE: Even if the rule doesn't win the tiebreaker, it still continues to specify properties for which no conflict exists. For example, if it specifies a solid border and a red font color and no rules specify a different type of border or the absence of one, a border will still be present even if a more specific rule calling for a green font color exists.

Exercise: What color is the quote?

```
⋮ Header HTML here
<body>
  <div id = "info" class = "med">
  <p class = "about">
    Blah blah blah...
    <q>To be or not to be.</q>
  </p>
</body>
</html>
```

CASE I:
Two Rules (Both Author):

```
body q
{
  color:blue;
}
#info .about
{
  color:orange;
}
```

CASE II:
Two Rules (Both Author):

```
div.med .about
{
  color:blue;
}
#info p
{
  color:orange;
}
```

CASE III:
Two Rules (Both Author):

```
body #info q
{
  color:blue;
}
#info p q
{
  color:orange;
}
```

CASE IV:
Two Rules (Both Author):

```
body #info .med
{
  color:blue;
}
div
{
  color:orange;
}
```

Notes

- This is primarily a summary of Chapter 10 of *Head First HTML and CSS*, 2nd Edition by Elisabeth Robson and Eric Freeman, 2012. It contains images, exercises, and code from the book.