HTML Overview

HTML puts a lot of emphasis on structure — headings, lists, tables, etc. Why? It has its roots in library science (databases, the cataloging of things) as well as the online exchange of papers between academics, which are highly structured.

HTML evolves as various working groups consider changes to the specification and then ratify them. The open nature of the language, along with competition among browser makers, means that not every HTML tag or CSS property is supported by every browser.

On the other hand, browsers are built to try to render code even when something is missing or invalid. *Inconsistent rendering is and will always be a part of web design.*

The way we used to write HTML

```
<TABLE WIDTH="800" BORDER="0" CELLSPACING="0">
<TR>
<TD WIDTH="600" VALIGN="TOP">
<P FONT="HELVETICA">Here is a <B>very</B> short paragraph.</P>
</TD>
</TR>
</TABLE>
```

The way we write it now

```
<p>Here is a <strong>very</strong> short paragraph.</p>
```

(plus a CSS file....)

Arguments for the new way

History: HTML was always supposed to work the web standards way, but proprietary browsers and the pace of development encouraged shortcuts.

Accessibility: The visually-impaired use special browsers (“screen readers”) that interpret structural markup in particular ways, allowing the user to understand the structure of the content without relying on the formatting.

Flexibility: Separating the content from its style and layout allows RSS readers, search engines to parse, index, and reformat the content.

Convenience: Style sheets also allow us to make global changes across hundreds or thousands of pages.

Tables are for tabular data...

Tables are legitimate HTML when used for tabular data:

```
<h1>Lemonade Stand Earnings</h1>
<table>
<tr>
<td>Mon</td><td>
<td>Tue</td></tr>
<tr><td>$1</td>
<td><td>$2</td></tr>
</table>
```

LEMONADE STAND EARNINGS

MON	TUE
$1	$2

For the same reasons you wouldn’t put a paragraph of text in an `<h1>`, you shouldn’t put non-tabular data in tables. CSS does a much better job with layout than tables.

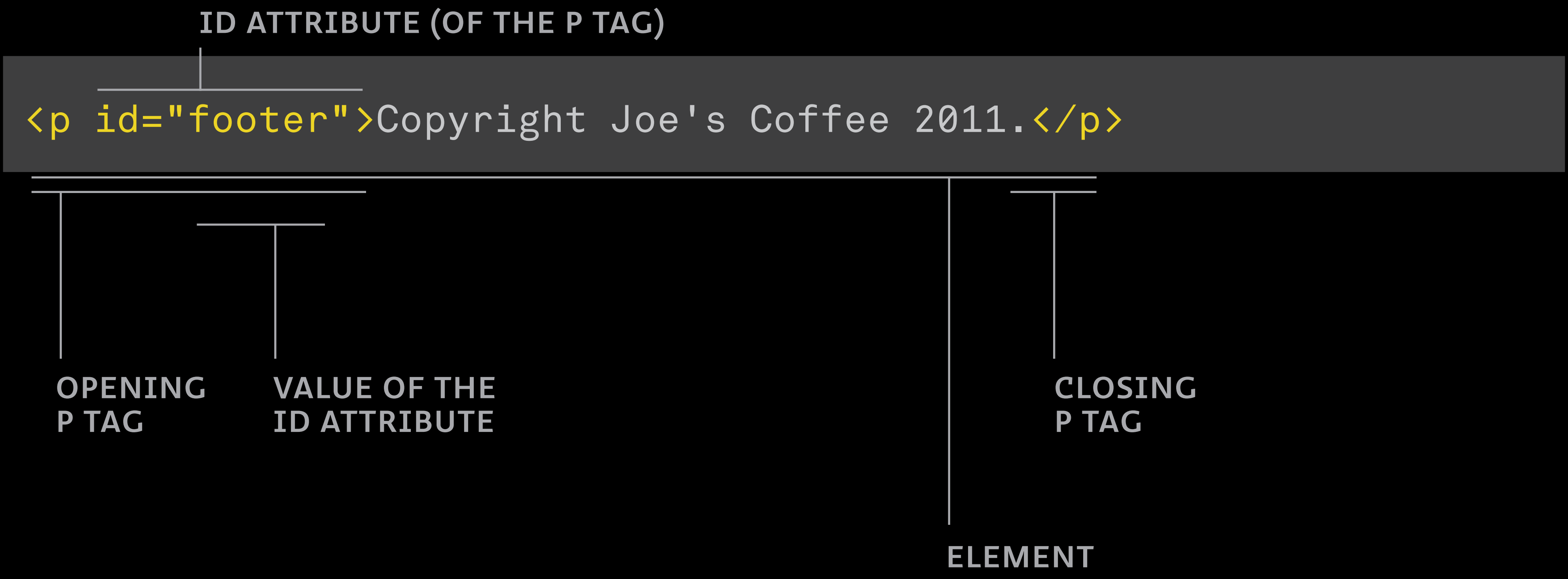
HTML:
Hypertext Mark-up Language

```
<img src="coffee-logo.png" />

<h1>Welcome to Joe's Coffee</h1>

<p>We sell the best coffee!</p>

<!-- News section starts here -->

<h2>News</h2>
<p>We now have <a href="locations">three locations</a> in
Nashville.</p>

<p id="footer">Copyright Joe's Coffee 2011.</p>
```

Paragraphs need a pair of "p" tags, one opening and one closing:

```
<p>We sell the best coffee!</p>
```

The footer paragraph has an "id" attribute that will let us refer to it specifically in our CSS.



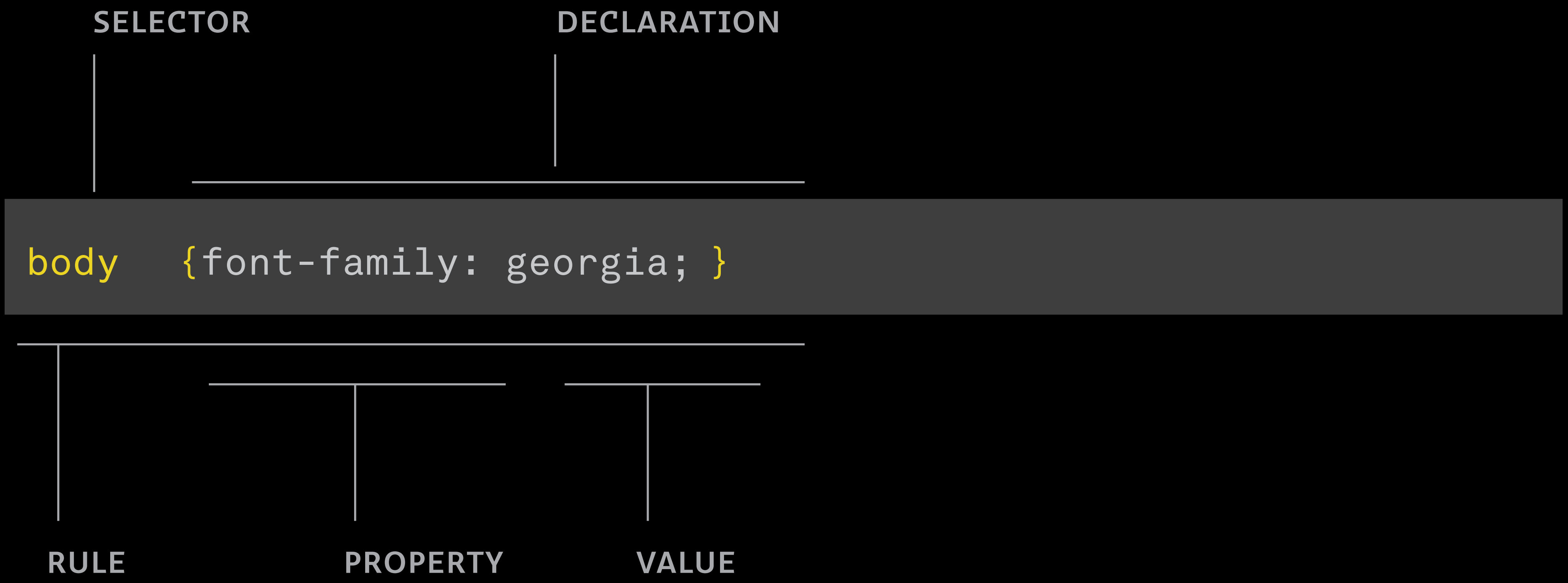
```
<p id="footer">Copyright Joe's Coffee 2011.</p>
```

Lists can be *ordered* or *unordered*. Notice how the `ul` or `ol` tags *surround* the `li` tags.

```
<ul>
<li>Some list item</li>
<li>Some other list item</li>
</ul>

<ol>
<li>First, pre-heat the oven.</li>
<li>Second, chop the onions.</li>
</ol>
```

CSS:
Cascading Stylesheets

```
body    {
        /* I love serifs */
        font-family: georgia;
        font-size: 20px;
        }


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



```
body   {font-family: georgia; }
```

Some More Tags

```html
<h1>Some top-level header</h1>
<p>Some paragraph.</p>

<div id="main">

<h2>Some second-level header</h2>
<ul>
<li>A list item</li>
<li>A list item</li>
</ul>

<p>Check out <a href="http://somesite.com">this site</a>.</p>

<p>I love <span class="caps">HTML</span>.</p>

</div> <!-- this closes the main div -->
```

Inline vs. Block-level Tags

block-level

- exists within a box that fills the full width available to it
- "interrupts" the flow of content
- heights, margins can be manipulated with CSS
- starts on a new line by default
- examples: `p`, `ul`, `ol`, `h1`, `h2`, `blockquote`, `li`

inline

- only takes up as much space as it needs
- part of the flow of the content
- heights, margins, etc. *can't* be manipulated with CSS
- can't be placed "loose" within the body tag; must be wrapped by a block-level element
- examples: `a`, `strong`, `em`, `img`

div vs. span

`div` (“division”)

- block-level by default
- semantically neutral
- use it to group a bunch of related content
- most likely you’ll attach an id, class, or both

`span`

- inline by default
- semantically neutral
- use it to group small amounts of content within a larger tag
- most likely you’ll attach a class or id, but sometimes the tag by itself is enough

class vs. id

`id`

- on any given page, a particular id can only appear once: if your main navigation is contained in `<div id="nav">`, you can't use `<ul id="nav">` on that page.
- a tag can have only one id.
- id's also serve as markers of locations within pages: you could link to `<div id="nav">`from anywhere on the page by writing `<a href="#nav">Scroll to the navigation</a>`.

`class`

- classes work like id's but can be re-used multiple times on a single page: it might make sense to have `<ul class="nav">` appear several times in a column if your navigation is split into several lists.
- you can put two classes on a single tag by separating them with spaces: `<ul class="nav primary">` vs. `<ul class="nav secondary">`

How to code a web page

1. Mark up the content as HTML to convey its structure, organization, and to embed things like hyperlinks, forms, images, etc.

2. Style and lay out as much of the page as possible using CSS.

3. *Where necessary*, go back to the HTML and add **classes**, **id**s, and non-semantic groupings (**divs** and **spans**) to the HTML so that they can be styled to achieve the design you want. These will give you the "hooks" you need to target different parts of the page in the CSS.