Front-End Coding: HTML and CSS

Bo Zhao

The Department of Geography

Outline

HTML

- Element
- Attribute
- Anatomy of an HTML document
- Debugging
- Active Learning: Html document structure

What is HTML?

HTML (Hypertext Markup Language) is **not a programming language**. It is a **markup language** that tells web browsers how to structure the web pages you visit. It can be as complicated or as simple as the web developer wants it to be.

HTML consists of a series of elements, which you use to enclose, wrap, or mark up different parts of content to make it appear or act in a certain way. The enclosing tags can make content into a hyperlink to connect to another page, italicize words, and so on.
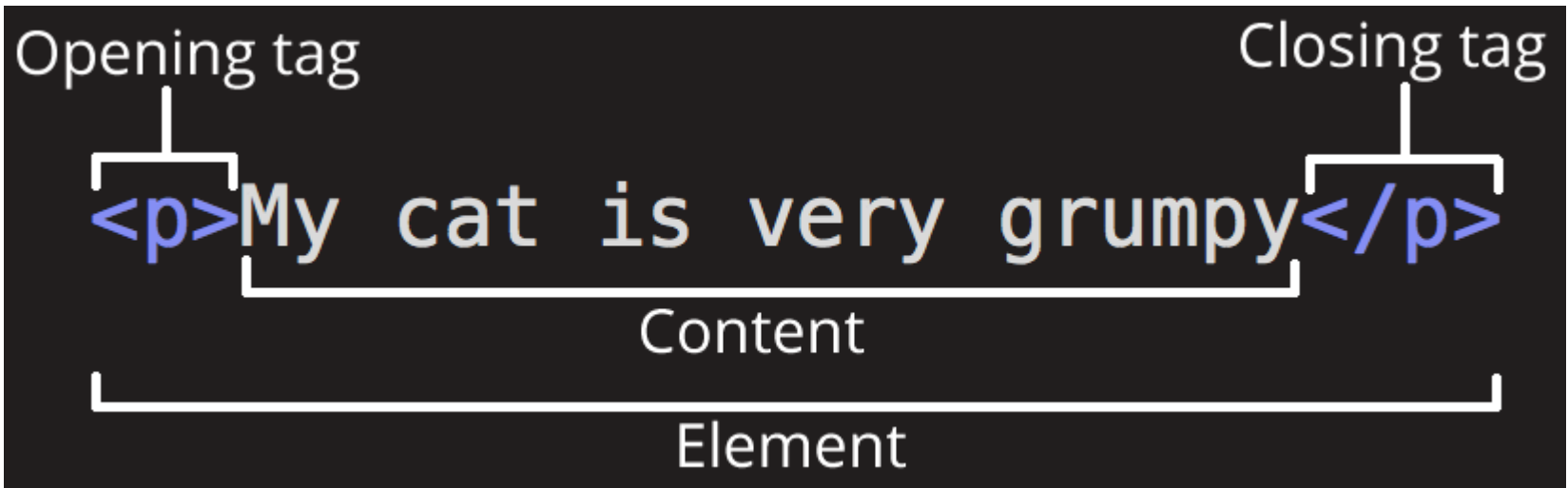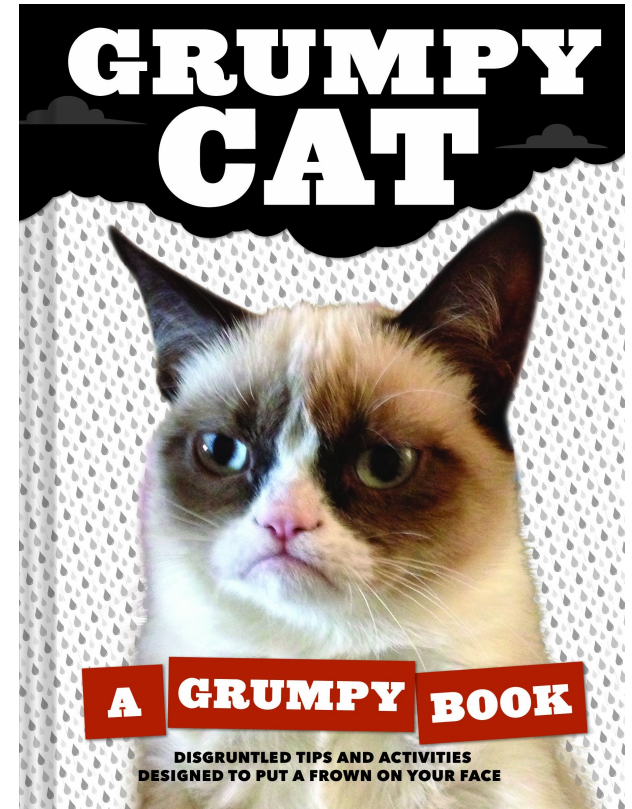
Nesting elements

Elements can be placed within other elements. This is called **nesting**. If we wanted to state that our cat is very grumpy, we could wrap the word very in a <strong> element, which means that the word is to have strong(er) text formatting:

```
<p>My cat is <strong>very</strong> grumpy.</p>
```

```
<p>My cat is <strong>very grumpy.</p></strong>
```

Block v.s. inline elements

Two important categories of elements to know in HTML: block-level elements and inline elements.

- **Block-level elements** form a visible block on a page. **A block-level element appears on a new line following the content that precedes it**. Any content that follows a block-level element also appears on a new line. Block-level elements are usually structural elements on the page. For example, a block-level element might represent headings, paragraphs, lists, navigation menus, or footers. A block-level element wouldn't be nested inside an inline element, but it might be nested inside another block-level element.

- **Inline elements** are contained **within block-level elements and surround only small parts of the document's content** (not entire paragraphs or groupings of content). An inline element will not cause a new line to appear in the document. It is typically used with text, for example an <a> element creates a hyperlink, and elements such as <em> or <strong> create emphasis.
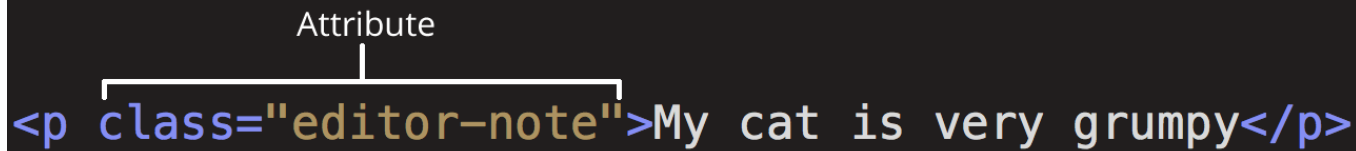
Empty elements

- Not all elements follow the pattern of an opening tag, content, and a closing tag.

- Some elements consist of a single tag, which is typically used to insert/embed something in the document. For example, the <img> element embeds an image file onto a page:

```
<img src="https://raw.githubusercontent.com/mdn/firefox-icon.png">
```

Attribute

Attributes contain extra information about the element that won't appear in the content. In this example, the class attribute is an identifying name used to target the element with style information.

```
               Attribute
         ┌──────────┴──────────┐
<p class="editor-note">My cat is very grumpy</p>
```

An attribute should have:

- A space between it and the element name. (For an element with more than one attribute, the attributes should be separated by spaces too.)
- The attribute name, followed by an equal sign.
- An attribute value, wrapped with opening and closing quote marks.

Single or double quotes?

In this article you will also notice that the attributes are wrapped in double quotes. However, you might see single quotes in some HTML code. This is a matter of style. You can feel free to choose which one you prefer. Both of these lines are equivalent:

```
<a href="https://www.example.com">A link to my example.</a>


<a href='https://www.example.com'>A link to my example.</a>
```

if you use one type of quote, you can include the other type of quote inside your attribute values:

```
<a href="https://www.example.com" title="Isn't this fun?">
    A link to my example.
</a>
```

Single or double quotes?

To use quote marks inside other quote marks of the same type (single quote or double quote), use HTML entities. For example, this will break:

```
<a href='https://www.example.com' title='Isn't this fun?'>A link to my example.</a>
```

Instead, you need to do this:

```
<a href='https://www.example.com' title='Isn&apos;t this fun?'>A link to my example.</a>
```

Anatomy of an HTML document

```
<!DOCTYPE html>

<html>

 <head>

   <meta charset="utf-8">

   <title>My test page</title>

 </head>

 <body>

   <p>This is my page</p>

 </body>

</html>
```

- <!DOCTYPE html>: The doctype. When HTML was young (1991-1992), doctypes were meant to act as links to a set of rules that the HTML page had to follow to be considered good HTML. More recently, the doctype is a historical artifact that needs to be included for everything else to work right. <!DOCTYPE html> is the shortest string of characters that counts as a valid doctype.

- <html></html>: The <html> element. This element wraps all the content on the page. It is sometimes known as the root element.

- <head></head>: The <head> element. This element acts as a container for everything you want to include on the HTML page, that isn't the content the page will show to viewers. This includes keywords and a page description that would appear in search results, CSS to style content, character set declarations, and more. You'll learn more about this in the next article of the series.

Anatomy of an HTML document

```
<!DOCTYPE html>
<html>
 <head>
   <meta charset="utf-8">
   <title>My test page</title>
 </head>
 <body>
   <p>This is my page</p>
 </body>
</html>
```

- <meta charset="utf-8">: This element specifies the character set for your document to UTF-8, which includes most characters from the vast majority of human written languages. With this setting, the page can now handle any textual content it might contain. There is no reason not to set this, and it can help avoid some problems later.

- <title></title>: The <title> element. This sets the title of the page, which is the title that appears in the browser tab the page is loaded in. The page title is also used to describe the page when it is bookmarked.

- <body></body>: The <body> element. This contains all the content that displays on the page, including text, images, videos, games, playable audio tracks, or whatever else.

Whitespace in HTML

In the example shown below, you may have noticed that a lot of whitespace is included in the code. This is optional. These two code snippets are equivalent:

```
<p>This is an example.</p>


<p>This    is an

          example.</p>
```

No matter how much whitespace you use inside HTML element content (which can include one or more space character, but also line breaks), the HTML parser reduces each sequence of whitespace to a single space when rendering the code. So why use so much whitespace? The answer is readability.

Including special characters in HTML

- In HTML, the characters <, >,",' and & are special characters. They are parts of the HTML syntax itself. So how do you include one of these special characters in your text? For example, if you want to use an ampersand or less-than sign, and not have it interpreted as code.

- You do this with character references. These are special codes that represent characters, to be used in these exact circumstances. Each character reference starts with an ampersand (&) and ends with a semicolon (;).

- The character reference equivalent could be easily remembered because the text it uses can be seen as less than for '<' , quotation for ' " ' and similarly for others.

- List of XML and HTML character entity references.

```
<p>In HTML, you define a paragraph using the <p> element.</p>
<p>In HTML, you define a paragraph using the &lt;p&gt; element.</p>
```

HTML Comments

HTML has a mechanism to write comments in the code. Browsers ignore comments, effectively making comments invisible to the user. The purpose of comments is to allow you to include notes in the code to explain your logic or coding. This is very useful if you return to a code base after being away for long enough that you don't completely remember it. Likewise, comments are invaluable as different people are making changes and updates.

To write an HTML comment, wrap it in the special markers <!-- and -->. For example:

```
<p>I'm not inside a comment</p>
<!-- <p> I am ! </p> -->
```

The head in HTML

The head of an HTML document is the part that is not displayed in the web browser when the page is loaded.

It contains information such as the page <title>, links to CSS (if you choose to style your HTML content with CSS), links to custom favicons, and other metadata (data about the HTML, such as the author, and important keywords that describe the document

- Adding a title: the title tag
- Metadata: the <meta> element, charset, user, description, keyword, etc.
- Adding custom icons to your site: favicon
- Applying CSS and JavaScript to HTML: link and script in the head element
- Setting the primary language of the document: <html lang="en-US">

Custom icons to your html page

- To further enrich your site design, you can add references to custom icons in your metadata, and these will be displayed in certain contexts. The most commonly used of these is the favicon (short for "favorites icon", referring to its use in the "favorites" or "bookmarks" lists in browsers).

```
<link rel="icon" href="favicon.ico" type="image/x-icon">
```

- The humble favicon has been around for many years. It is the first icon of this type: a 16-pixel square icon used in multiple places. You may see (depending on the browser) favicons displayed in the browser tab containing each open page, and next to bookmarked pages in the bookmarks panel.

- Try out this online favicon creator at https://favicon.io/

Custom icons to your html page

```html
<!-- third-generation iPad with high-resolution Retina display: -->
<link rel="apple-touch-icon-precomposed" sizes="144x144"
href="https://developer.mozilla.org/static/img/favicon144.png">
<!-- iPhone with high-resolution Retina display: -->
<link rel="apple-touch-icon-precomposed" sizes="114x114"
href="https://developer.mozilla.org/static/img/favicon114.png">
<!-- first- and second-generation iPad: -->
<link rel="apple-touch-icon-precomposed" sizes="72x72"
href="https://developer.mozilla.org/static/img/favicon72.png">
<!-- non-Retina iPhone, iPod Touch, and Android 2.1+ devices: -->
<link rel="apple-touch-icon-precomposed"
href="https://developer.mozilla.org/static/img/favicon57.png">
<!-- basic favicon --> <link rel="icon"
href="https://developer.mozilla.org/static/img/favicon32.png">
```

The comments explain what each icon is used for — these elements cover things like providing a nice high-resolution icon to use when the website is saved to an iPad's home screen.

Don't worry too much about implementing all these types of icon right now — this is a fairly advanced feature, and you won't be expected to have knowledge of this to progress through the course. The main purpose here is to let you know what such things are, in case you come across them while browsing other websites' source code.

Debugging HTML

- From the Chrome menu: Open the Chrome menu and go to "More Tools" > "Developer Tools." Finally, you can right-click (Windows) or Ctrl-click (Mac) anything on a web page and select "Inspect Element" to open Developer Tools. The Developer Tools panel will open in whatever web page you're on.

- [An overview to Google Chrome DevTools](#)
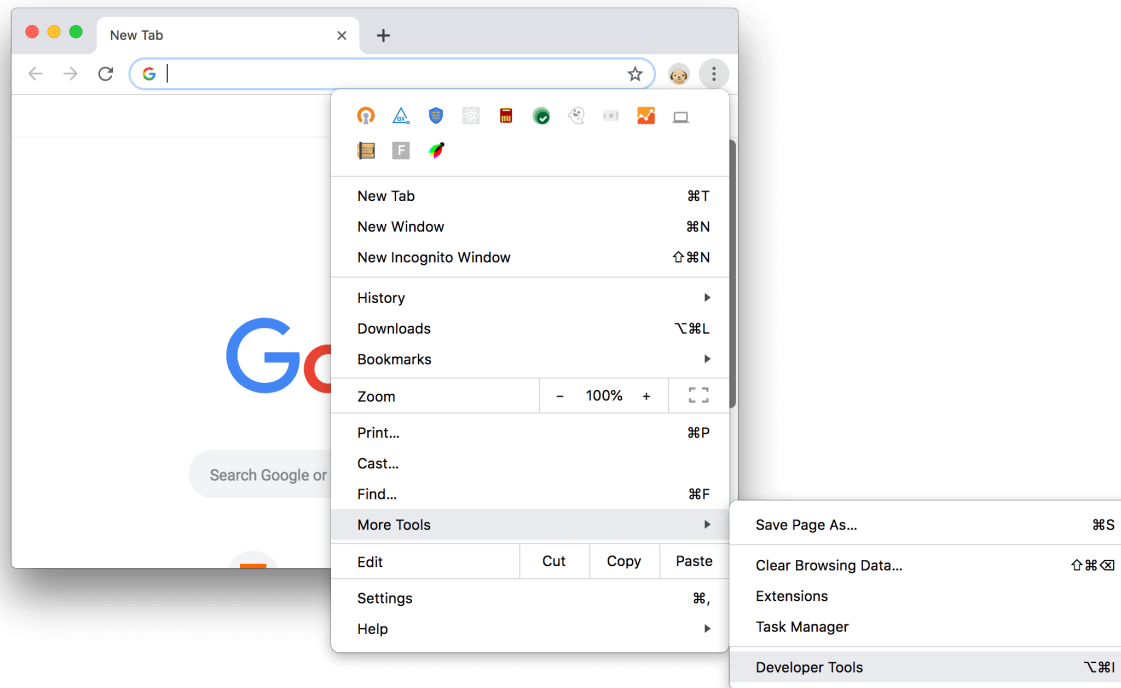
Document and website structure

Webpages can and will look different from one another, but they all tend to share similar standard components, unless the page is displaying a Fullscreen video or game, is part of some kind of art project, or is just badly structured:

- **header:** Usually a big strip across the top with a big heading, logo, and perhaps a tagline. This usually stays the same from one webpage to another.

- **navigation bar:** Links to the site's main sections; usually represented by menu buttons, links, or tabs. Like the header, this content usually remains consistent from one webpage to another — having inconsistent navigation on your website will just lead to confused, frustrated users. Many web designers consider the navigation bar to be part of the header rather than an individual component, but that's not a requirement; in fact, some also argue that having the two separate is better for accessibility, as screen readers can read the two features better if they are separate.

- **main content:** A big area in the center that contains most of the unique content of a given webpage, for example, the video you want to watch, or the main story you're reading, or the map you want to view, or the news headlines, etc. This is the one part of the website that definitely will vary from page to page!

- **sidebar:** Some peripheral info, links, quotes, ads, etc. Usually, this is contextual to what is contained in the main content (for example on a news article page, the sidebar might contain the author's bio, or links to related articles) but there are also cases where you'll find some recurring elements like a secondary navigation system.

- **footer:** A strip across the bottom of the page that generally contains fine print, copyright notices, or contact info. It's a place to put common information (like the header) but usually, that information is not critical or secondary to the website itself. The footer is also sometimes used for SEO purposes, by providing links for quick access to popular content.

Document and website structure
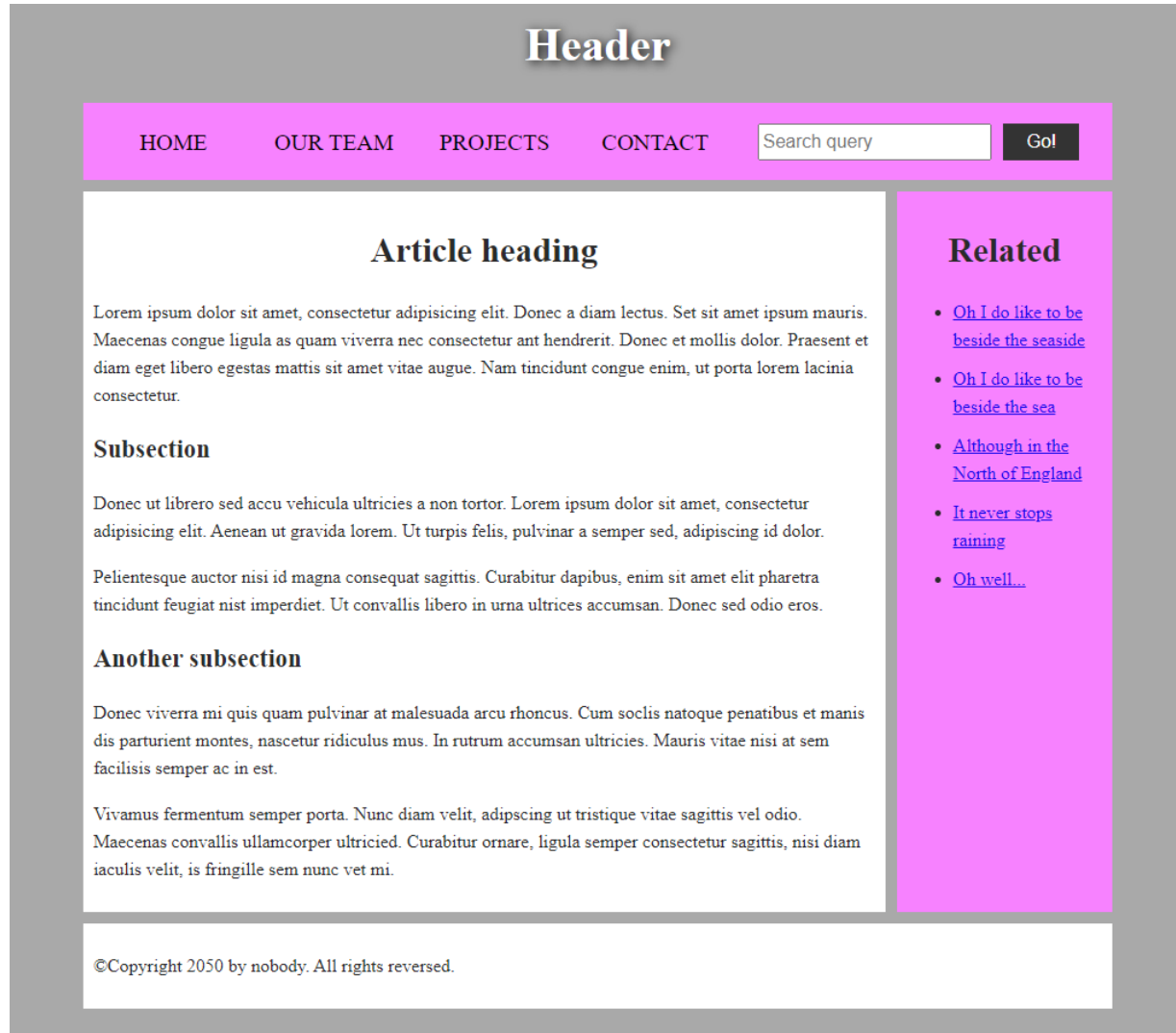
HTML for structuring content

To implement such semantic mark up, HTML provides dedicated tags that you can use to represent such sections, for example:

- header: <header>.
- navigation bar: <nav>.
- main content: <main>, with various content subsections represented by <article>, <section>, and <div> elements.
- sidebar: <aside>; often placed inside <main>.
- footer: <footer>.

CSS - Cascading Style Sheets

CSS (Cascading Style Sheets) is used to style and lay out web pages — for example, to alter the font, color, size, and spacing of your content, split it into multiple columns, or add animations and other decorative features. This module provides a gentle beginning to your path towards CSS mastery with the basics of how it works, what the syntax looks like, and how you can start using it to add styling to HTML.

Outline

- CSS Syntax

- CSS Specifications

- The work process

- Integrate CSS to HTML

- Selectors

- CSS Layout

- Active Learning: CSS Debugging

CSS syntax

CSS is a rule-based language — you define rules specifying groups of styles that should be applied to particular elements or groups of elements on your web page. For example "I want the main heading on my page to be shown as large red text."

```css
h1 {
        color: red;
        font-size: 5em;
}
```

The rule opens with a selector . This selects the HTML element that we are going to style. In this case we are styling level one headings (<h1>).

We then have a set of curly braces { }. Inside those will be one or more declarations, which take the form of property and value pairs. Each pair specifies a property of the element(s) we are selecting, then a value that we'd like to give the property.

Before the colon, we have the property, and after the colon, the value. CSS properties have different allowable values, depending on which property is being specified. In our example, we have the color property, which can take various color values. We also have the font-size property. This property can take various size units as a value.

CSS Specifications

- All web standards technologies (HTML, CSS, JavaScript, etc.) are defined in giant documents called specifications (or "specs"), which are published by standards organizations (such as the W3C, WHATWG, ECMA, or Khronos) and define precisely how those technologies are supposed to behave.

- CSS is no different — it is developed by a group within the W3C called the CSS Working Group. This group is made of representatives of browser vendors and other companies who have an interest in CSS. There are also other people, known as invited experts, who act as independent voices; they are not linked to a member organization.

How does CSS actually work?

- When a browser displays a document, it must combine the document's content with its style information. It processes the document in a number of stages, which we've listed below.

- Bear in mind that this is a very simplified version of what happens when a browser loads a webpage, and that different browsers will handle the process in different ways. But this is roughly what happens.

The working process

1. The browser loads the HTML (e.g., receives it from the network).

2. It converts the HTML into a DOM (Document Object Model). The DOM represents the document in the computer's memory. The DOM is explained in a bit more detail in the next section.

3. The browser then fetches most of the resources that are linked to by the HTML document, such as embedded images and videos ... and linked CSS! JavaScript is handled a bit later on in the process, and we won't talk about it here to keep things simpler.

4. The browser parses the fetched CSS and sorts the different rules **by their selector types into different "buckets"**, e.g., element, class, ID, and so on. Based on the selectors it finds, it works out which rules should be applied to which nodes in the DOM and attaches style to them as required (this intermediate step is called a render tree).

5. The render tree is laid out in the structure it should appear in after the rules have been applied to it.

6. The visual display of the page is shown on the screen (this stage is called painting).

The working process

The following diagram also offers a simple view of the process.
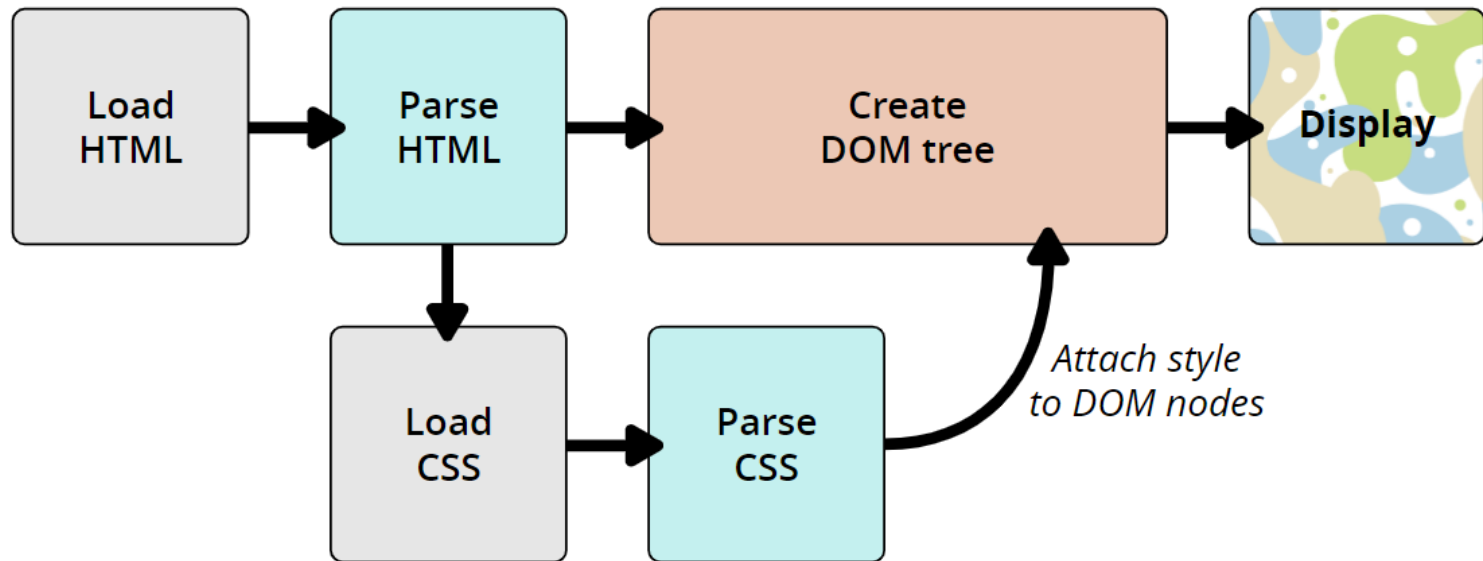
What happens if a browser encounters CSS it doesn't understand?

- Given that CSS is being developed all the time, and is therefore ahead of what browsers can recognize, you might wonder what happens if a browser encounters a CSS selector or declaration it doesn't recognize.

- The answer is that it does nothing, and just moves on to the next bit of CSS.

- If a browser is parsing your rules and encounters a property or value that it doesn't understand, it ignores it and moves on to the next declaration. It will do this if you have made an error and misspelled a property or value, or if the property or value is just too new and the browser doesn't yet support it.

Applying CSS to HTML

The very first thing we need to do is to tell the HTML document that we have some CSS rules we want it to use. There are three different ways to apply CSS to an HTML document that you'll commonly come across, however, for now, we will look at the most usual and useful way of doing so — linking CSS from the head of your document.

Create a file in the same folder as your HTML document and save it as styles.css. The .css extension shows that this is a CSS file.

To link styles.css to index.html add the following line somewhere inside the <head> of the HTML document:

```
<link rel="stylesheet" href="styles.css">
```

Applying CSS to HTML: External Stylesheet

An external stylesheet contains CSS in a separate file with a .css extension. This is the most common and useful method of bringing CSS to a document. You can link a single CSS file to multiple web pages, styling all of them with the same CSS stylesheet. In the Getting started with CSS, we linked an external stylesheet to our web page.

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My CSS experiment</title>
    <link rel="stylesheet" href="styles.css">
  </head>
  <body>
    <h1>Hello World!</h1>
    <p>This is my first CSS example</p>
  </body>
</html>
```

Applying CSS to HTML: Internal Stylesheet

An internal stylesheet resides within an HTML document. To create an internal stylesheet, you place CSS inside a <style> element contained inside the HTML <head>.

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My CSS experiment</title>
    <style>
      h1 {
        color: blue;
        background-color: yellow;
        border: 1px solid black;
      }

      p {
        color: red;
      }
    </style>
  </head>
  <body>
    <h1>Hello World!</h1>
    <p>This is my first CSS example</p>
  </body>
</html>
```

Applying CSS to HTML: Inline Style

Inline styles

Inline styles are CSS declarations that affect a single HTML element, contained within a style attribute. The implementation of an inline style in an HTML document might look like this:

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My CSS experiment</title>
  </head>
  <body>
    <h1 style="color: blue;background-color: yellow;border: 1px solid black;">Hello World!</h1>
    <p style="color:red;">This is my first CSS example</p>
  </body>
</html>
```

Avoid using CSS in this way, when possible. It is the opposite of a best practice.

Selectors

By making our heading red we have already demonstrated that we can target and style an HTML element. We do this by targeting an element selector — this is a selector that directly matches an HTML element name. To target all paragraphs in the document you would use the selector p. To turn all paragraphs green you would use:

```
p {
    color: green;
}
```

You can target multiple selectors at once, by separating the selectors with a comma. If I want all paragraphs and all list items to be green my rule looks like this:

```
p, li {
    color: green;
}
```

Selectors: Changing the default style

We can see how the browser is making the HTML readable by adding some default styling. Headings are large and bold and our list has bullets. This happens because browsers have internal stylesheets containing default styles, which they apply to all pages by default; without them all of the text would run together in a clump and we would have to style everything from scratch. All modern browsers display HTML content by default in pretty much the same way.

However, you will often want something other than the choice the browser has made. This can be done by choosing the HTML element that you want to change, and using a CSS rule to change the way it looks. A good example is our <ul>, an unordered list. It has list bullets, and if I decide I don't want those bullets I can remove them like so:

```css
li {
    list-style-type: none;
}
```

Selectors using class

So far we have styled elements based on their HTML element names. This works as long as you want all of the elements of that type in your document to look the same. Most of the time that isn't the case and so you will need to find a way to select a subset of the elements without changing the others. The most common way to do this is to add a class to your HTML element and target that class.

In your HTML document, add a class attribute to the second list item. Your list will now look like this:

```
<ul>
 <li>Item one</li>
 <li class="special">Item two</li>
 <li>Item <em>three</em></li>
</ul>
```

In your CSS you can target the class of special by creating a selector that starts with a full stop character. Add the following to your CSS file:

```
.special {
    color: orange;
    font-weight: bold;
}
```

Selector based on rules

Sometimes you will see rules with a selector that lists the HTML element selector along with the class:

```css
li.special {
        color: orange;
        font-weight: bold;
}
```

This syntax means "target any li element that has a class of special". If you were to do this then you would no longer be able to apply the class to a <span> or another element by adding the class to it; you would have to add that element to the list of selectors:

```css
li.special,
span.special {
        color: orange;
        font-weight: bold;
}
```

Selector based on locations

To select only an <em> that is nested inside an <li> element I can use a selector called the **descendant combinator**, which takes the form of a space between two other selectors.

```
li em {
     color: rebeccapurple;

}
```

Something else you might like to try is styling a paragraph when it comes directly after a heading at the same hierarchy level in the HTML. To do so place a + (an **adjacent sibling combinator**) between the selectors.

```
h1 + p {
   font-weight: bold;
  color: orange;}
```

Selector based on states

- When we style a link we need to target the <a> (anchor) element. This has different states depending on whether it is unvisited, visited, being hovered over, focused via the keyboard, or in the process of being clicked (activated).

- You can use CSS to target these different states — the CSS below styles unvisited links pink and visited links green.

- You can change the way the link looks when the user hovers over it, for example by removing the underline, which is achieved by the next rule:

```css
a:link { color: pink; }

a:visited { color: green; }

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

More selectors

Selector	Example	Learn CSS tutorial
Type selector	h1 { }	Type selectors
Universal selector	* { }	The universal selector
Class selector	.box { }	Class selectors
id selector	#unique { }	ID selectors
Attribute selector	a[title] { }	Attribute selectors
Pseudo-class selectors	p:first-child { }	Pseudo-classes
Pseudo-element selectors	p::first-line { }	Pseudo-elements
Descendant combinator	article p	Descendant combinator
Child combinator	article > p	Child combinator
Adjacent sibling combinator	h1 + p	Adjacent sibling
General sibling combinator	h1 ~ p	General sibling

https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Selectors

CSS layout

CSS page layout techniques allow us to take elements contained in a web page and control where they're positioned relative to the following factors: their default position in normal layout flow, the other elements around them, their parent container, and the main viewport/window.

- **Normal flow:** HTML is displayed in the exact order in which it appears in the source code, with elements stacked on top of one another

- **The display property:** Standard values such as block, inline or inline-block can change how elements behave in normal flow, for example, by making a block-level element behave like an inline-level element (see Types of CSS boxes for more information). We also have entire layout methods that are enabled via specific display values, for example, CSS Grid and Flexbox, which alter how child elements are laid out inside their parents.

- **Floats:** Applying a float value such as left can cause block-level elements to wrap along one side of an element, like the way images sometimes have text floating around them in magazine layouts.

- **The position property:** Allows you to precisely control the placement of boxes inside other boxes. static positioning is the default in normal flow, but you can cause elements to be laid out differently using other values, for example, as fixed to the top of the browser viewport.

Active Learning
CSS Debugging