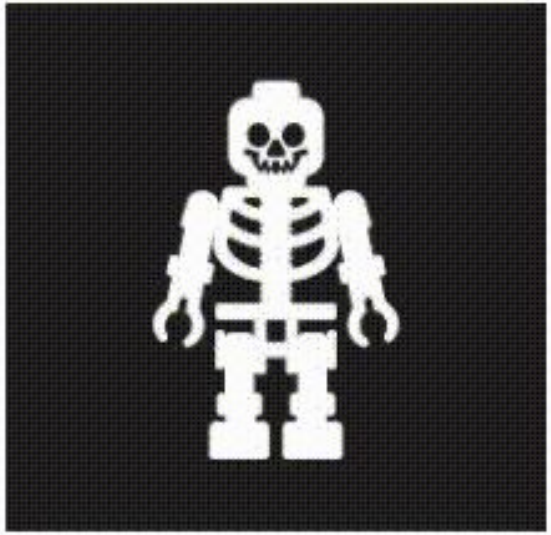
Front-end development

The Basics - Lesson 1

adapt

HTML
structure



CSS
presentation/appearance

JavaScript
dynamism/action

HTML

Hypertext markup language

adapt

Elements

```
<h1>I  am an Header 1</h1>

<p>I am Paragrapgh element</p>

<span>Hello World</span>

<div>
  <p>Paragrapgh inside division</p>
</div>

<br/>
```

Tags

```
<h2>Header two</h2>

<p>I am a piece of text and I am
<b>bold</b> as well<p>

<ul>
  <li>I am a list element entry</li>
  <li>I am second one</li>
</ul>
```

Attributes

```
<div class="my-name"
id="uniqueDiv"><div>

<a href="http://google.com"
target="_blank" title="click to go to
Google">Google link</a>

<img src="path/to/image.jpg"
width="200" height="200"/>

<p title="I am a tooltip on
hover">This is some text</p>
```

Common HTML Tags

a - "anchor" used for hyperlinks

blockquote - for large quotes

body - visible part of your site

br - line break

cite - a citation

div - content divider

DOCTYPE - document type

h1 - most important header

h2 - 2nd most important

h3-h6 - 3-6th most important

head - invisible part of your site

html - "what follows is HTML"

img - image

li - list item

link - to attach CSS stylesheets

ol - ordered list

p - paragraph

span - inline content divider

strong - strong text emphasis

style - for inline CSS styling

title - title of the page

ul - unordered list

The standard HTML document

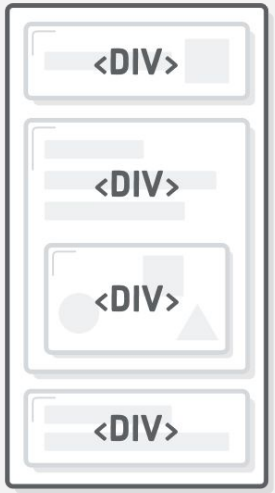
```
<!DOCTYPE html>
<html>
  <head>
   <meta charset="UTF-8">
   <title>Title of the document</title>
  </head>

  <body>
   <h1>This is a Heading</h1>
   <p>This is a paragraph</p>
  </body>
</html>
```

adapt

Semantics - why is it important?

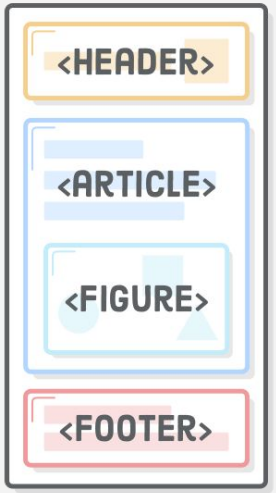
Using these as an alternative to ** ** elements is an important aspect of modern web development because it makes it easier for search engines, screen readers, and other machines to identify the different parts of your website. It also helps you as a developer keep your site organized, which, in turn, makes it easier to maintain.

```
<header>
<footer>
<article>
<figure>
<section>
<aside>
<time>
<address>
```



AMBIGUOUS STRUCTURE

(AKA "<DIV> SOUP")



IDENTIFIABLE SECTIONS

(AKA "SEMANTIC MARKUP")

<header>

The <header> element is a new piece of semantic markup, not to be confused with head*ings* (the <h1>-<h6> elements). It denotes introductory content for a section, article, or entire web page. “Introductory content” can be anything from your company’s logo to navigational aids or author information.

```
<header>
  <h1>Interneting Is Easy!</h1>
  <nav>
    <ul>
      <li><a href='#'>Home</a></li>
      <li><a href='#'>About</a></li>
      <li><a href='#'>Blog</a></li>
      <li><a href='#'>Sign Up</a></li>
    </ul>
  </nav>
</header>
----------------------
<article>
  <header>
    <h2>Semantic HTML</h2>
    <p>by Author</p>
  </header>

  <p>This is an example web page explaining HTML5
semantic markup.</p>
</article>
```

<footer>

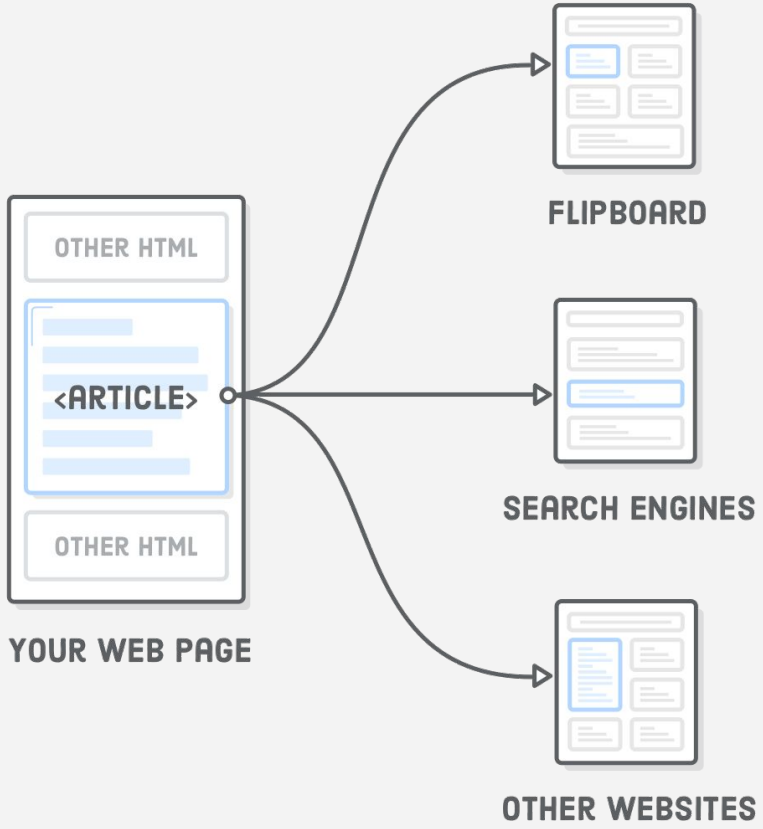
Conceptually, footers are basically the same as headers, except they generally come at end of an article/website opposed to the beginning. Common use cases include things like copyright notices, footer navigation, and author bios at the end of blog posts.

```
<header>Site title</header>

<article>
  <header>
    <h2>Semantic HTML</h2>
  </header>

  <p>This is an example web page explaining HTML5
semantic markup.</p>

  <footer>
    <p>This fake article was written by somebody<p>
  </footer>
</article>

<footer>
  <p>This fake article was written by somebody<p>
</footer>
```

adapt

<article>

The <article> element represents an independent article in a web page. It should only wrap content that can be plucked out of your page and distributed in a completely different context. For instance, an app like Flipboard should be able to grab an <article> element from your site, display it in its own app, and have it make perfect sense to its readers.



<section>

Think of <section> as an *explicit* way to define the sections in a document outline. Why would we want this instead of letting the heading levels do it for us? Often times, you need a container to wrap a section for layout purposes, and it makes sense to use the more descriptive <section>element over a generic .

```
<section>
  <article>...</article>
  <article>
    <section>...</section>
    <section>...</section>
  </article>
  <aside>...</aside>
</section>
```

adapt

<nav>

The <nav> element lets you mark up the various navigation sections of your website. This goes for the main site navigation, links to related pages in a sidebar, tables of content, and pretty much any group of links. For example, we should stick our site-wide navigation menu in a <nav> element

```
<nav>
  <ul>
    <li><a href='#'>Home</a></li>
    <li><a href='#'>About</a></li>
    <li><a href='#'>Blog</a></li>
    <li><a href='#'>Sign Up</a></li>
  </ul>
</nav>
```

<aside>

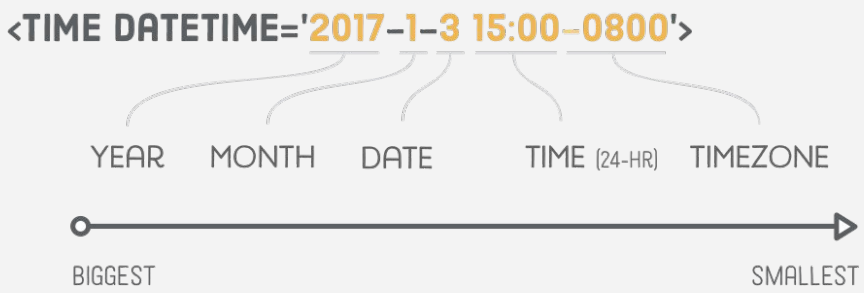
Headers and footers are ways to add extra information to an article, but sometimes we want to *remove* information from an article. For example, a sponsored blog post might contain an advertisement about the sponsoring company; however, we probably don't want to make it part of the article text. This is what the <aside> element is for.

```
<article>
  <header>
    <h2>Semantic HTML</h2>
    <p>By Troy McClure. Published January 3rd</p>
  </header>
  <!-- advertisement -->
  <aside class='advert'>
    <img src='some-ad.png'/>
  </aside>

  <p>This is an example web page explaining HTML5
semantic markup.</p>

</article>
```

<time> and date

The <time> element represents either a time of day or a calendar date. Providing a machine-readable date makes it possible for browsers to automatically link it to users' calendars and helps search engines clearly identify specific dates.

```
<p>By a good dude. Published
<time datetime='2017-1-22'>January
      22nd</time>
</p>
```



<address>

The HTML **<address>** element indicates that the enclosed HTML provides contact information for a person or people, or for an organization.

```
<footer>
  <p>This fake article was written by
somebody.</p>
  <address>
    Please contact <a
href='mailto:troymcclure@example.com'>Troy
    McClure</a> for questions about this
article.
  </address>
</footer>
```

adapt

<figure> <figcaption>

Last, but certainly not least, are the <figure> and <figcaption> elements. The former represents a self-contained “figure”, like a diagram, illustration, or even a code snippet. The latter is optional, and it associates a caption with its parent <figure> element.

A common use case for both of these is to add visible descriptions to the<img/> elements in an article, like so:

```
 <figure>
    <img src='semantic-elements.png'
         alt='Diagram showing <article>, <section>,
and <nav> elements'/>
    <figcaption>New HTML5 semantic
elements</figcaption>
  </figure>
```

adapt

HTML also contains a lot of form element

HTML Forms are one of the main points of interaction between a user and a web site or application. They allow users to send data to the web site. Most of the time that data is sent to the web server, but the web page can also intercept it to use it on its own.

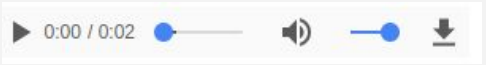
An HTML Form is made of one or more widgets. Those widgets can be text fields (single line or multiline), select boxes, buttons, checkboxes, or radio buttons. Most of the time those widgets are paired with a label that describes their purpose — properly implemented labels are able to clearly instruct both sighted and blind users on what to enter into a form input.

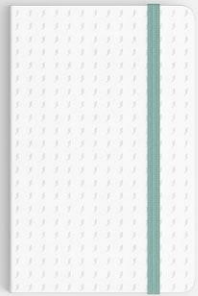

```
<input type="radio"/>
```

And it also allows for some media functionality

The **HTML Video element** (**<video>**) embeds a media player which supports video playback into the document. You can use <video> for audio content as well, but the <audio> element may provide a more appropriate user experience.

```
<video controls muted width="300" height="200">
 <source src="myVideo.mp4" type="video/mp4">
 <source src="myVideo.webm" type="video/webm">
 Sorry, your browser doesn't support embedded videos
</video>

<audio controls>
  <source src="myAudio.mp3" type="audio/mp3">
  <source src="myAudio.ogg" type="audio/ogg">
  <p>Your browser doesn't support HTML5
audio.</p>
</audio>
```



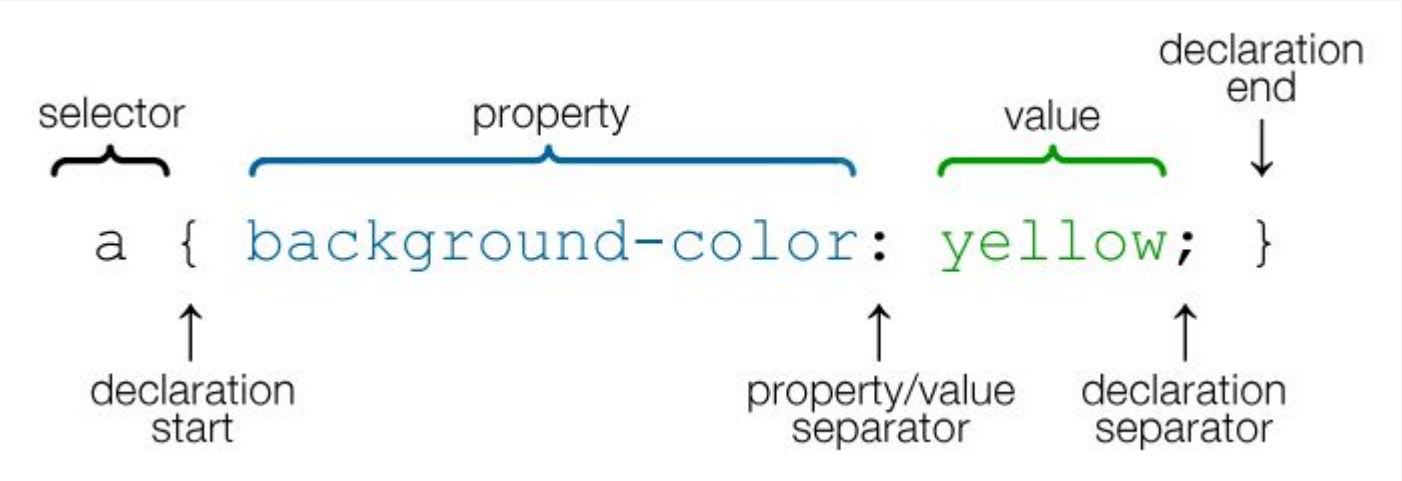


CSS

Cascading style sheet

adapt

Syntax

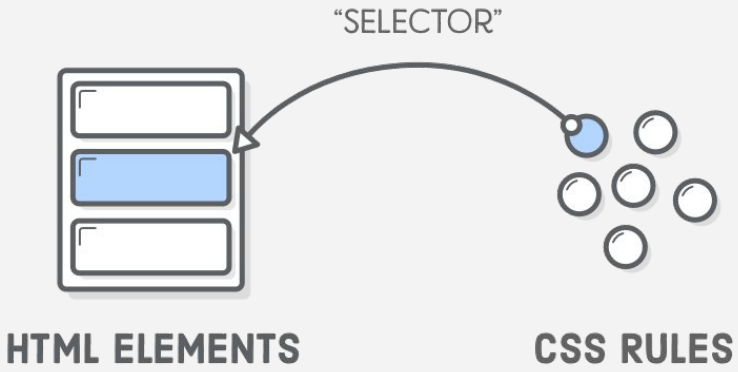



```
h1 { width: 200px; }  |  div { display: none; } | a { color: red; font-size: 30px; }
```

```
.myRuleset {
  display: block;
  background-color: red;
  position: relative;
}
```

Selectors

In CSS, selectors are used to target the HTML elements on our web pages that we want to style. There are a wide variety of CSS selectors available, allowing for fine grained precision when selecting elements to style.



Simple selectors: class, or id. > `.myClassName`

Attribute selectors > `[class="myClassName"]`

Pseudo-classes > `div:hover`

Pseudo-elements > `div::before`

Combinators > `.div + div`

Multiple selectors > `div.myClassname:hover a {...}`

Pseudo-classes

A CSS pseudo-class is a keyword added to the end of a selector, preceded by a colon (:), which is used to specify that you want to style the selected element but only when it is in a certain state.

```
a:visited {
  color: blue;
}

a:hover,
a:focus {
  color: darkred;
  text-decoration: none;
}

a:focus {
  color: darkred;
```

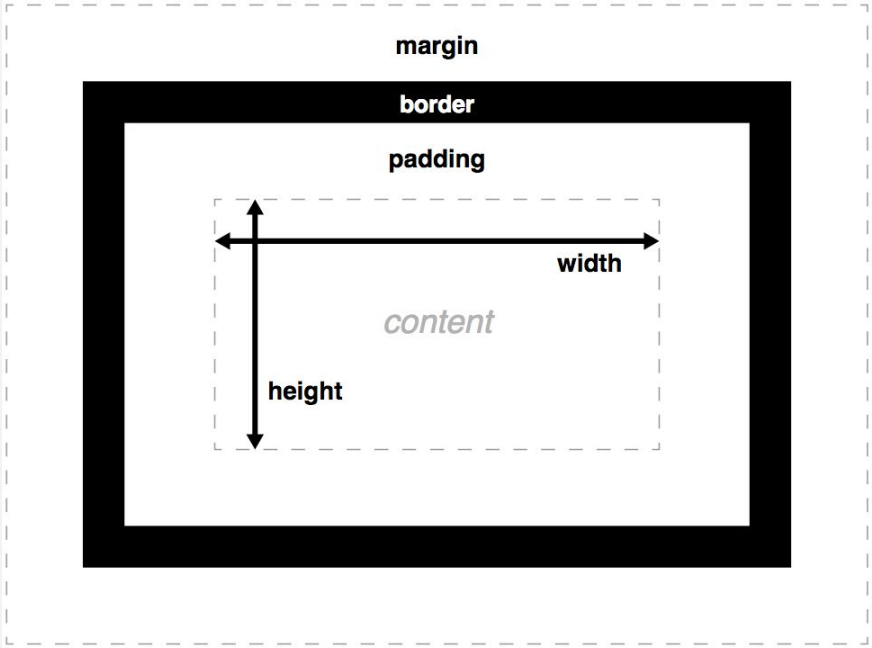
Pseudo-elements

Pseudo-elements are very much like pseudo-classes, but they have differences. They are keywords, this time preceded by two colons (::), that can be added to the end of selectors to select a certain part of an element.

```
::after ::before ::first-letter ::first-line
::selection ::backdrop

[href^=http]::after {
  content: '↗';
}
```

Box-model

The CSS box model is the foundation of layout on the Web — each element is represented as a rectangular box, with the box's content, padding, border, and margin built up around one another like the layers of an onion. As a browser renders a web page layout, it works out what styles are applied to the content of each box, how big the surrounding onion layers are, and where the boxes sit in relation to one another. Before understanding how to create CSS layouts, you need to understand the box model.

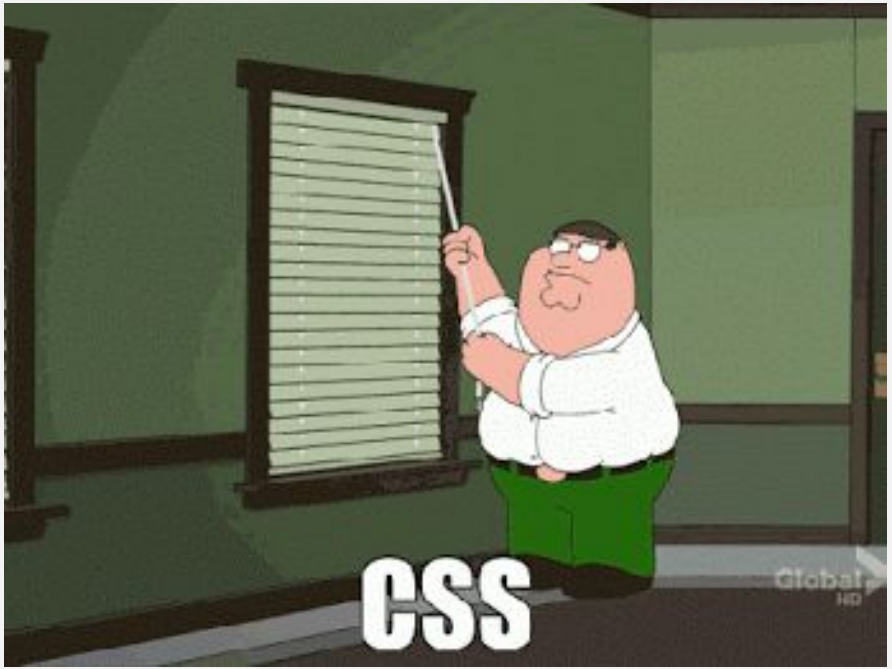


How CSS works - The Cascade

The cascade takes a unordered list of declared values for a given property on a given element, sorts them by their declaration's precedence, and outputs a single cascaded value.

Here are the attributes that the CSS Cascade algorithm checks, listed in order from highest weight to least weight.

1. Origin & Importance
2. Selector Specificity
3. Order of Appearance
4. Initial & Inherited Properties (default values



Origin & importance

User-Agent: These are the default styles provided for the element by the browser. This is why inputs can look slightly different on different browsers, and it's also one of the reasons that people like to use CSS resets, to make sure that user-agent styles are overridden.

User: These are defined and controlled by the user of the browser. Not everyone will have one, but when people do add one, it's usually for overriding styles & adding accessibility to websites.

Author: This is CSS declared by the HTML document. When we're writing stuff as front-end developers this is really the only origin that we have in our control.

Browsers apply the following sorting logic:

- Find all declarations that apply to a given element/property combination, for the target media type.
- Sort declarations according to their importance (normal or important) and origin (author, user, or user agent). From highest to lowest precedence:
    1. **user** !important declarations
    2. **author** !important declarations
    3. **author** normal declarations
    4. **user normal** declarations
    5. **user agent** declarations
- If declarations have the same importance and source, sort them by selector specificity.
- Finally, if declarations have the same importance, source, and specificity, sort them by the order they are specified in the CSS. The last declaration wins.

Selector specificity

Specificity determines, which CSS rule is applied by the browsers.

```
<div id="someID">
  <label>Input name</label>
  <input class="classname" type="text"/>
</div>
```

Measuring specificity - (0, 0, 0, 0)

add 1 for each element name or pseudo-element.
add 10 for each attribute, class or pseudo-class
add 100 for each ID
add 1000 for style attribute

```
input = 0, 0, 0, 1
input.classname = 0, 0, 1, 1
#someID .classname = 0, 1, 1, 0
#someID input.classname = 0, 1, 1, 1
div#someID input.classname = 0, 1, 1, 2
label + input.classname[type="text"] = 0, 0, 2, 2
input.classname + !important rule = 1, 0, 1, 1
```

Some practicalities regarding specificity

Use LVHA for link styling. “To ensure that you see your various link styles, you’re best off putting your styles in the order “link-visited-hover-active”, or “LVHA” for short.”

Never use !important. “If you’re having specificity issues, there’s some quick ways to solve it. First, avoid !important.” “The !important declaration overrides normal declarations, but is unstructured and rarely required in an author’s style sheet.”

Use id to make a rule more specific. Replacing a.highlight with ul#blogroll a.highlight changes the specificity from 0, 0, 1, 1 to 0, 1, 1, 2.

Minimize the number of selectors. Use the least number of selectors required to style an element.

How every developer sees the end user

Order of Appearance

The last main tier of the CSS cascade algorithm is resolution by source order. When two selectors have the same specificity, the declaration that comes last in the source code wins.

Initial & inherited properties

While initial & inherited values aren't truly part of the CSS cascade, they do determine what happens if there are *no CSS declarations* targeting the element. In this way, they determine default values for an element.

Inherited properties will trickle down from parent elements to child elements. For example, the font-family & color properties are inherited. This behavior is what most people think of when they see the word "cascade" because styles will trickle down to their children.

In the following example, the <p> tag will render with a monospace font & red text, since its parent node contains those styles.

```
<div style="font-family: monospace; color: red;">
 <p>inheritance can be super useful!</p>
</div>
```

Reading list

HTML (all sections)

Understanding how Cascade works

CSS Selector game

CSS Specificity calculator

Float vs Inline-blocks

Basics of blend modes

Introduction to CSS