HTML & CSS, POSITIONING, ASSETS, STRUCTURE

The first thing we should talk about is structure. What is structure?

Think about a newspaper, or essays, or books, how are they structured?

They have headlines, articles, paragraphs, tables of content, etc.

Essentially, structure is just organization. How are we going to lay everything out? Where will all the pieces fit? That sort of thing.

There are some established conventions on best ways to organize our projects, but also, rules are meant to be broken and you can pretty much do whatever you want.

HTML stands for **HyperText Markup Language**, and it's technically not a programming language--it's a scripting language.

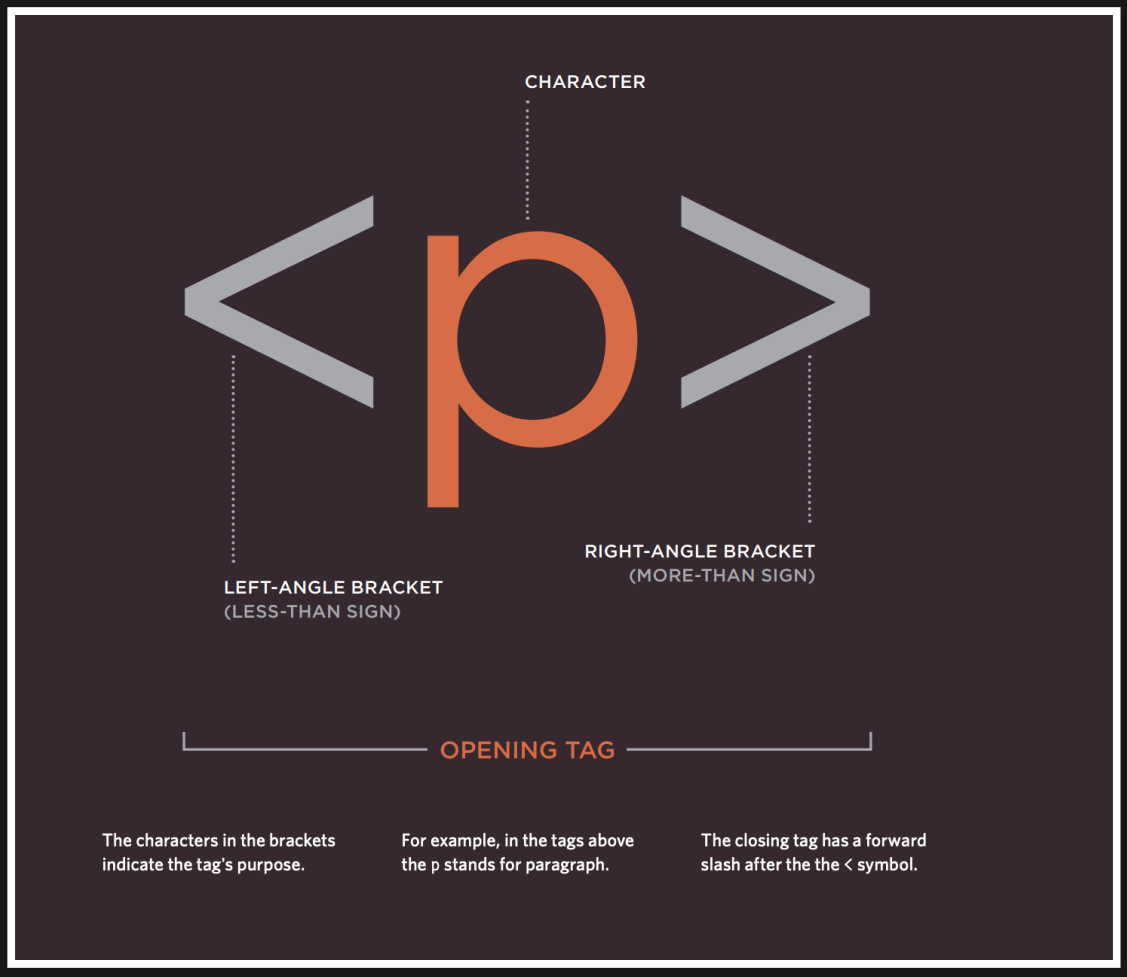
HTML describes the structure of pages.

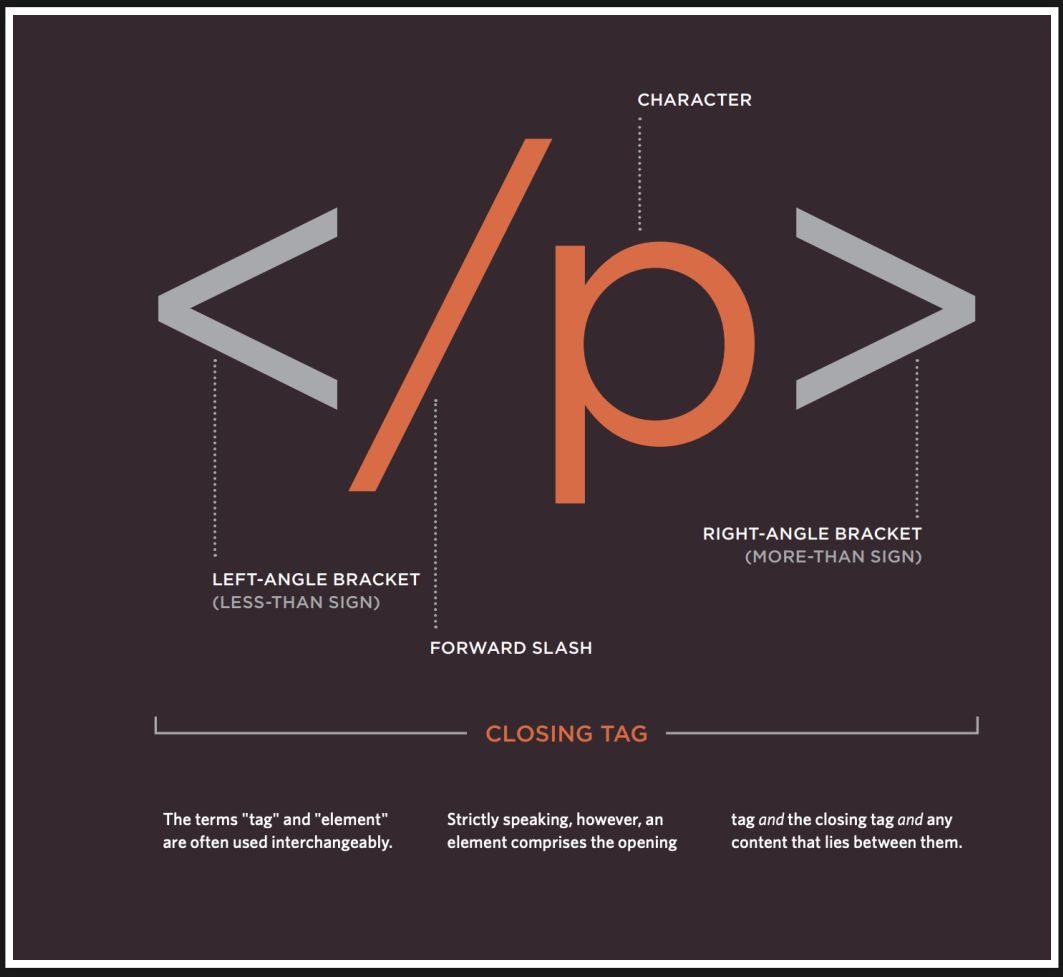
```
<h1>Main Heading</h1>
<p>Paragraph</p>
<h2>Sub-heading</h2>
<p>Another paragraph</p>
```

You can sort of tell that tags are basically containers. There's an opening and corresponding closing tag for each HTML tag. If you don't close your tag, you're going to run into problems.

The tag itself acts as an indication of what the content between the tags should be.

```
<section>
        <article class="post">
                <p>A bunch of content</p>
        </article>
</section>
```


CHARACTER
<p>
LEFT-ANGLE BRACKET
(LESS-THAN SIGN)
RIGHT-ANGLE BRACKET
(MORE-THAN SIGN)
OPENING TAG
The characters in the brackets indicate the tag's purpose.
For example, in the tags above the p stands for paragraph.
The closing tag has a forward slash after the the < symbol.


CHARACTER
</p>
RIGHT-ANGLE BRACKET
(MORE-THAN SIGN)
LEFT-ANGLE BRACKET
(LESS-THAN SIGN)
FORWARD SLASH
CLOSING TAG
The terms "tag" and "element" are often used interchangeably.
Strictly speaking, however, an element comprises the opening
tag *and* the closing tag *and* any content that lies between them.

(images taken from Jon Duckett's HTMl & CSS book)

```
<h2>There are several header tags, h1...h6</h2>
<section>
        <p>We can section off parts of our document.</p>
        <p>
         We can also <b>bold</b>
         text and <i>italicize</i>
        </p>
</section>
```

We can also make lists

```
<p>There are ordered lists</p>
<ol>
        <li>One</li>
        <li>Two</li>
        <li>Three</li>
</ol>
```

```
<p>And unordered lists</p>
<ul>
        <li>One</li>
        <li>Two</li>
        <li>Three</li>
</ul>
```

One of the most useful tags though, is the anchor tag, which is how we link our pages.

```
<a href="https://google.com/">Google</a>
```

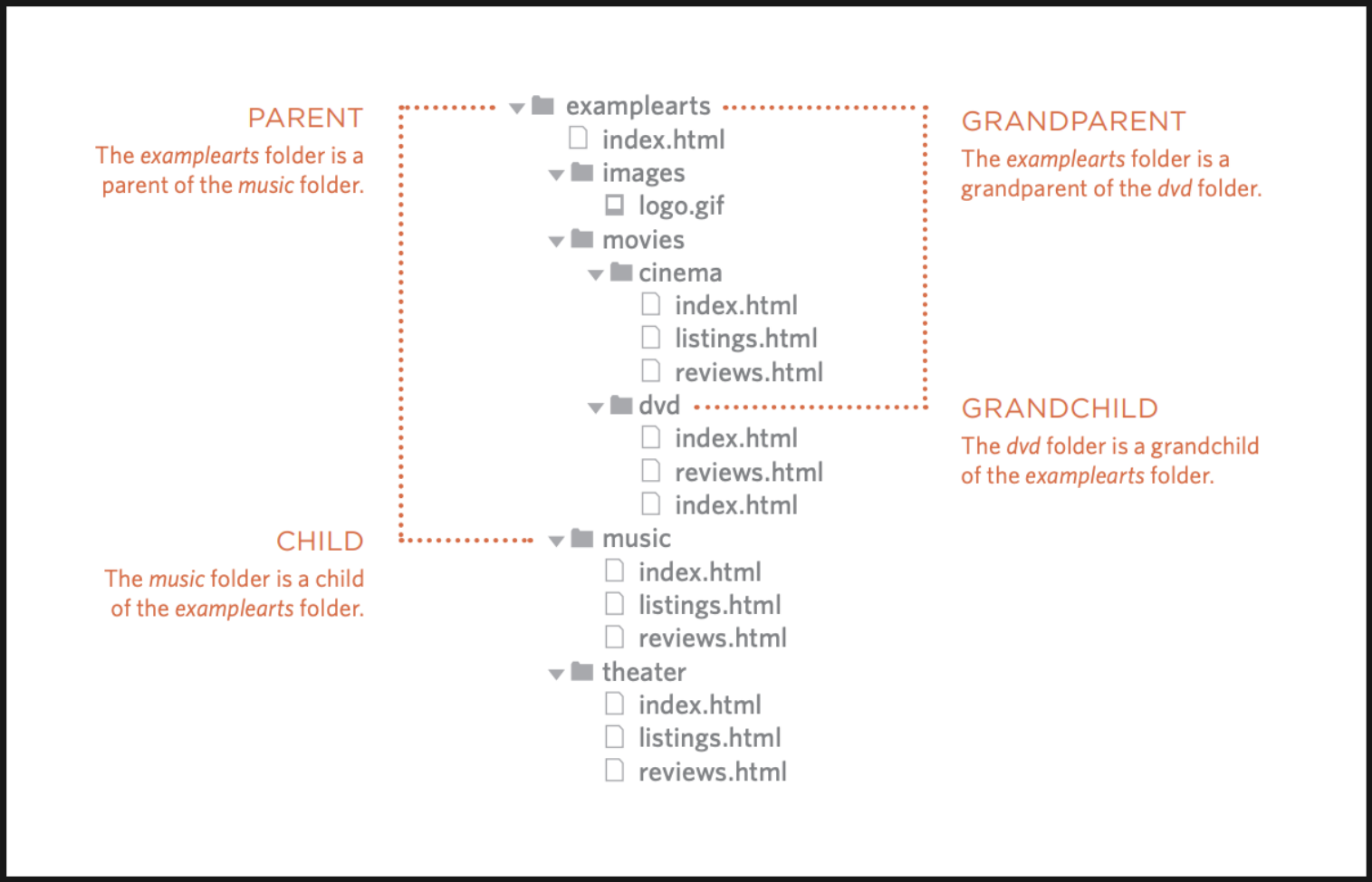

THIS IS THE PAGE THE LINK TAKES YOU TO
THIS IS THE TEXT THE USER CLICKS ON
IMDB
OPENING LINK TAG
CLOSING LINK TAG

Questions before we move on to folder/directory structure?

How we structure and organize our projects is up to us, but we definitely want to emphasize organization.

Whatever you do, stay consistent, and make patterns to help yourself later.

It might not seem terribly important when you have one HTML page, but when you have hundreds of pages and thousands of lines of code... it's very important.


PARENT
The examplearts folder is a parent of the music folder.
CHILD
The music folder is a child of the examplearts folder.
examplearts
index.html
images
logo.gif
movies
cinema
index.html
listings.html
reviews.html
dvd
index.html
reviews.html
index.html
music
index.html
listings.html
reviews.html
theater
index.html
listings.html
reviews.html
GRANDPARENT
The examplearts folder is a grandparent of the dvd folder.
GRANDCHILD
The dvd folder is a grandchild of the examplearts folder.

Every page and iamge on a website has a URL (Uniform Resource Locator). A URL is comprised of a domain name followed by the path to the resources you're looking for.

For example:
https://example.com/cool_directory/my_cool_webpage

Example.com is the domain, cool_directory is the directory (folder), and my_cool_webpage.html is the HTML file.

We also often use **relative URLs** to link our pages. This is for linking resources within our own directory structure.

This is often confusing, especially when you see things like "../" or "../../" to jump up directories.

This is a remnant of old unix systems, before GUIs were so common.

RELATIVE LINK TYPE	EXAMPLE (from diagram on previous page)
SAME FOLDER To link to a file in the same folder, just use the file name. (Nothing else is needed.)	To link to music reviews from the music homepage: `<a href="reviews.html">Reviews</a>`
CHILD FOLDER For a child folder, use the name of the child folder, followed by a forward slash, then the file name.	To link to music listings from the homepage: `<a href="music/listings.html">Listings</a>`

GRANDCHILD FOLDER
Use the name of the child folder, followed by a forward slash, then the name of the grandchild folder, followed by another forward slash, then the file name.
	To link to DVD reviews from the homepage:
	`<a href="movies/dvd/reviews.html">`
	`Reviews</a>`

PARENT FOLDER Use `../` to indicate the folder above the current one, then follow it with the file name.	To link to the homepage from the music reviews: `<a href="../index.html">Home</a>`
GRANDPARENT FOLDER Repeat the `../` to indicate that you want to go up two folders (rather than one), then follow it with the file name.	To link to the homepage from the DVD reviews: `<a href="../../index.html">Home</a>`

You may have also already noted how to add an image:

```
<img src="../images/my_cool_image.jpeg" alt="An image of somet
```

There's a lot to talk about in regards to images, more than we can probably cover today. Just some things to think about though...file formats, size, pixel density, cropping, etc.

In short, you should produce images specifically for your project, to fit within very specific contexts.

Now, moving on to **CSS**, **Cascading Style Sheets**

CSS is essentially a way for us to set rules that dictate how our projects look.

Think of it like this... "Okay CSS, make every paragraph tag that is called 'centered' positioned in the middle of the page."

How exactly we communicate that in code is another story, but that's the end goal and why we need **CSS**.

Note, there's no way for us to cover all of the **CSS** declarations that exist, you'll learn them when you need them. Note the structure below:

```
tag {
        property: value;
}
```

Actual examples:

```
h1, h2, h3 {
        font-family: Helvetica;
        color: red;
}

p {
        line-height: 100%;
}
```

Generally we want to separate all of our languages, so we put our **CSS** in a separate file, like **style.css**

We then link our external style sheet with a **link** tag.

```
<link rel="stylesheet" href="/css/master.css">
```

There's also several ways to **target** or **select** the HTML tag that we want to style.

The two most common are through **classes** and **IDs**.

Both a **class** and an **ID** are arbitrary in the sense that we can name them whatever we want, but we want to be precise in our naming conventions.

It doesn't make much sense to have a **section** with an **ID** of "about" if it really contains contact information.

The primary difference between a **class** and an **ID** is the intention:

IDS ARE UNIQUE, AND CALLED USING A # (POUND/HASHTAG)

CLASSES ARE MULTIPLE, AND CALLED USING A . (PERIOD)

If you label something with an **ID**, it should be the ONLY One in the entire site.

A **class** can be applied to any number of HTML tags, but STAY CONSISTENT.

Sample HTML:

```
<nav id="primary-navigation">
        <ul>
                <li class="navigation-link">
                        <a href="one.html">One</a>
                </li>
                <li class="navigation-link">
                        <a href="two.html">Two</a>
                </li>
        </ul>
</nav>
```

Sample CSS:

```
nav#primary-navigation {
        width: 100%;
        font-size: 2em;
}

li.navigation-link {
        color: red;
        margin: 10px;
}
```

This is essentially the same, just less precise:

```
#primary-navigation {
        width: 100%;
        font-size: 2em;
}

.navigation-link {
        color: red;
        margin: 10px;
}
```

SELECTOR	MEANING	EXAMPLE
UNIVERSAL SELECTOR	Applies to all elements in the document	`* {}` Targets all elements on the page
TYPE SELECTOR	Matches element names	`h1, h2, h3 {}` Targets the <h1>, <h2> and <h3> elements

CLASS SELECTOR	Matches an element whose `class` attribute has a value that matches the one specified after the period (or full stop) symbol	`.note {}`
		Targets any element whose `class` attribute has a value of `note`
		`p.note {}`
		Targets only <p> elements whose `class` attribute has a value of `note`

ID SELECTOR	Matches an element whose `id` attribute has a value that matches the one specified after the pound or hash symbol	`#introduction {}` Targets the element whose `id` attribute has a value of `introduction`

CHILD SELECTOR	Matches an element that is a direct child of another	`li>a {}` Targets any <a> elements that are children of an element (but not other <a> elements in the page)
DESCENDANT SELECTOR	Matches an element that is a descendent of another specified element (not just a direct child of that element)	`p a {}` Targets any <a> elements that sit inside a <p> element, even if there are other elements nested between them
ADJACENT SIBLING SELECTOR	Matches an element that is the next sibling of another	`h1+p {}` Targets the first <p> element after any <h1> element (but not other <p> elements)
GENERAL SIBLING SELECTOR	Matches an element that is a sibling of another, although it does not have to be the directly preceding element	`h1~p {}` If you had two <p> elements that are siblings of an <h1> element, this rule would apply to both

Questions so far?

We can only go far just explaining things, so let's do a few demos to try and put these in practice.