HTML5 & CSS3

COMP 126: Practical Web Design & Development for Everyone

SETUP

Download & install

- a text editor: I recommend Visual Studio Code, but if you already use something else and prefer to continue doing so, that's . ne (details here)
- Chrome and Firefox (if you don't have them), and find/turn on if needed their developer tools

Codepen

There are lots of online platforms that allow you to quickly get your code up in a browser for testing. In this class, I'll use Codepen for lecture demos and code sharing. Please create an account at https://codepen.io/ and follow my account by searching for username

tkjn.

Anatomy of a website

your content in a .html file,

+ HTML tags for structure,

+ CSS declarations for styling,

+ JavaScript or other scripting language/s for behavior & interactivity,

+ a browser to display it in

= website

HTML

HyperText Markup Language: used to describe and differentiate the content and structure of a web page

Syntax: Elements, Tags

Element: Any individual component of an html document

Tag: "Opening" and "closing" tags are used to enclose each element (sometimes no closing tag is needed: e.g., <img> and <br>)

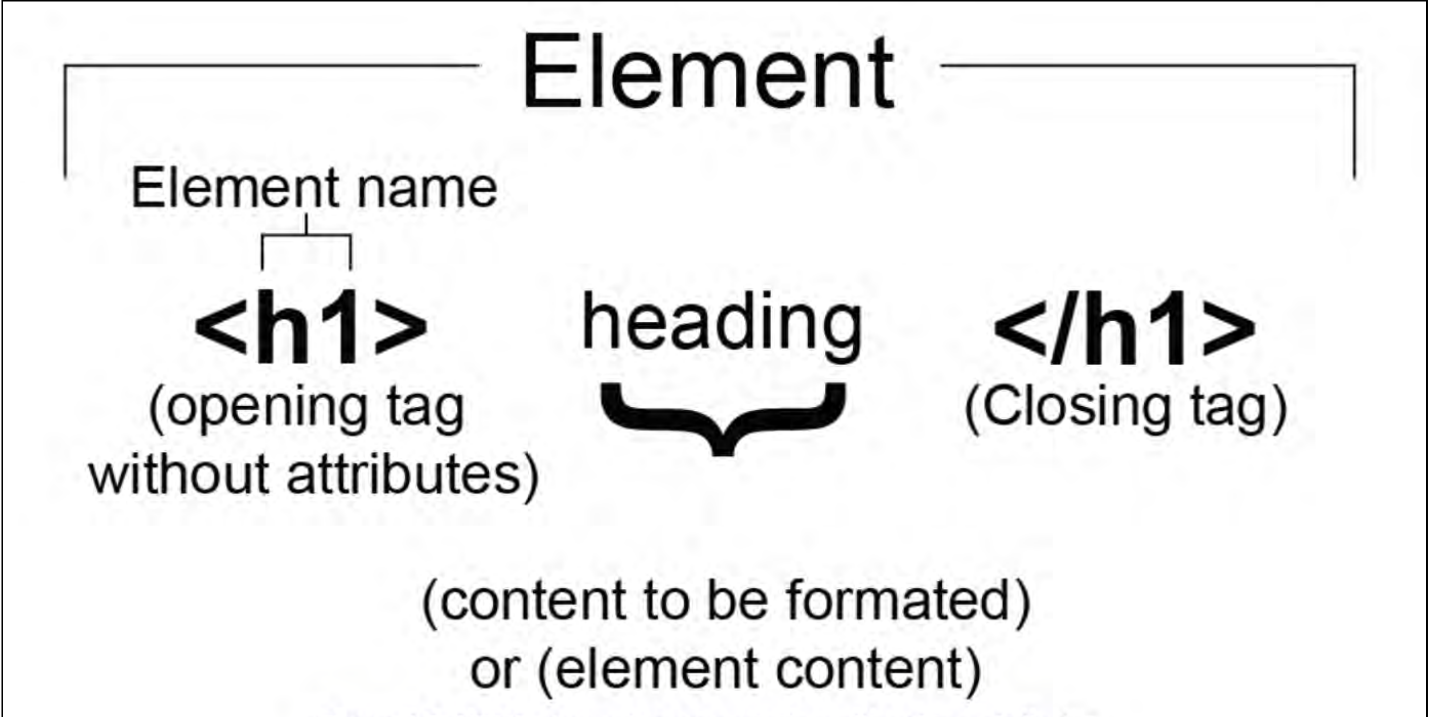

Element
Element name
<h1>
(opening tag without attributes)
heading
</h1>
(Closing tag)
(content to be formated) or (element content)

Figure 1.3: HTML element representation without tag attributes

Syntax: Attributes

<u>Attribute</u>: adds more detail about how the browser should handle an element

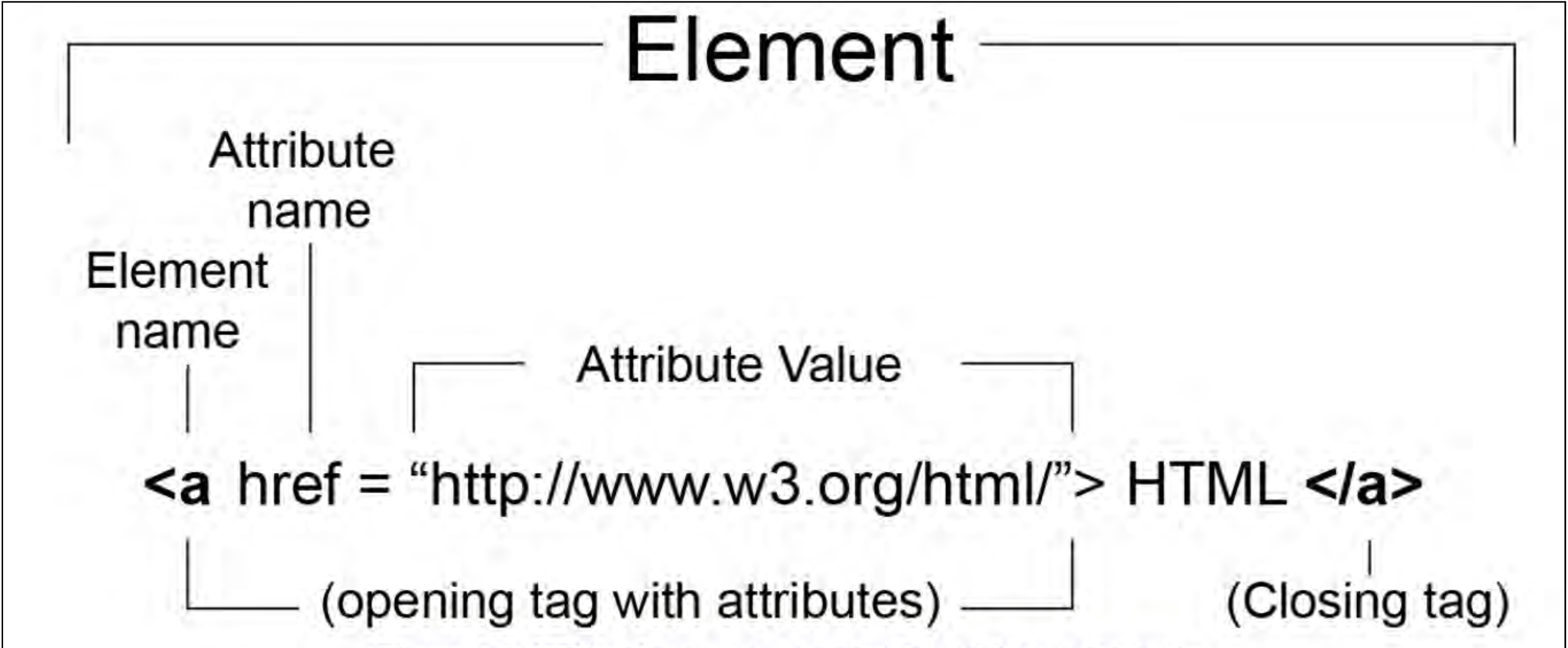

Element
Attribute name
Element name
Attribute Value
<a href = "http://www.w3.org/html/"> HTML </a>
(opening tag with attributes)
(Closing tag)

Figure 1.4: HTML element representation with tag attributes

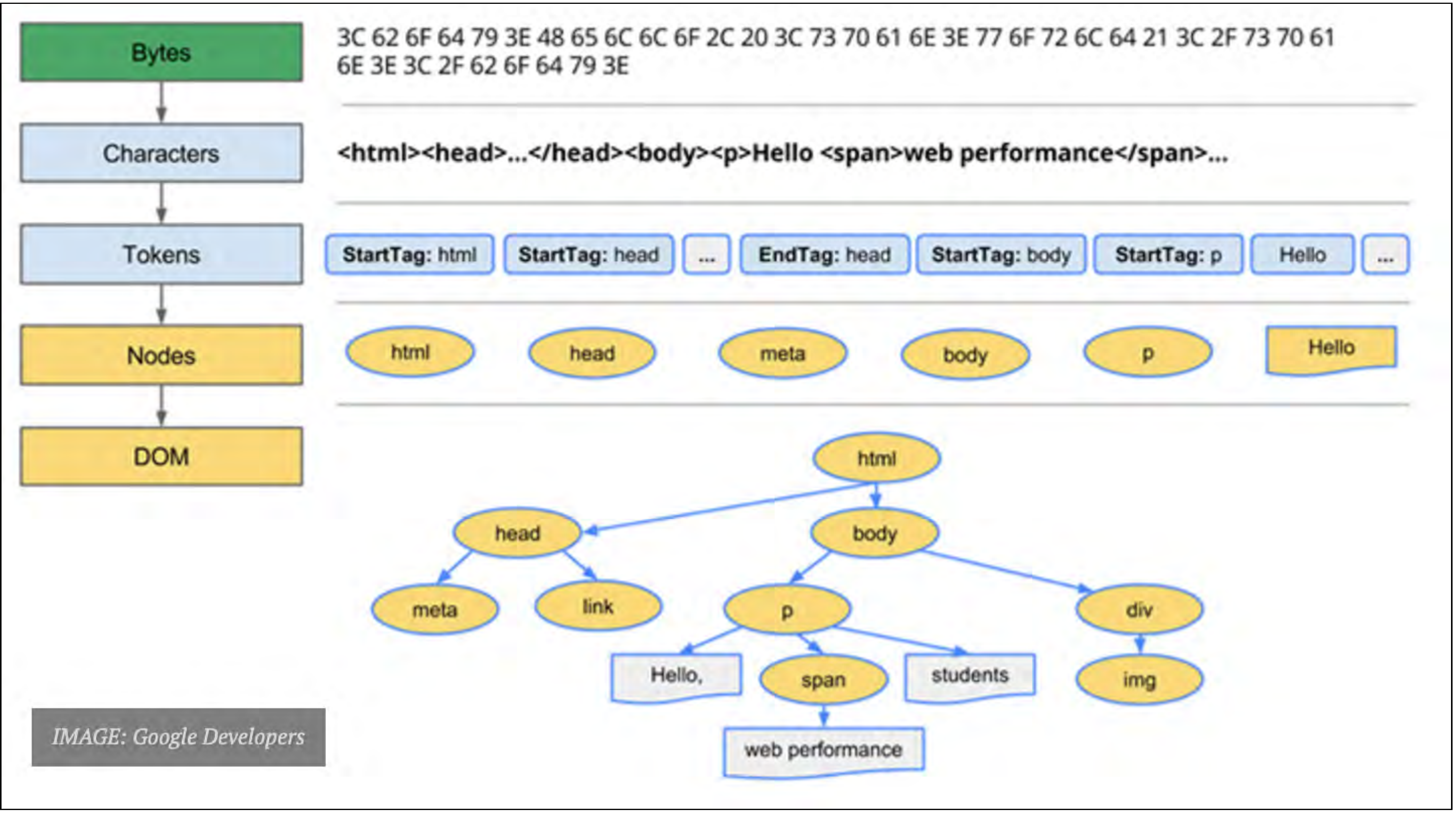
Syntax: Comments

Comment: the parser/browser ignores any content formatted as a comment; use comments to organize your code and add clarifications and notes for your future self and/or future developers

```
<!-- the browser/parser ignores any HTML
you place between these symbols -->
```

Get in the habit of commenting your code *frequently* and *often*

The HTML DOM

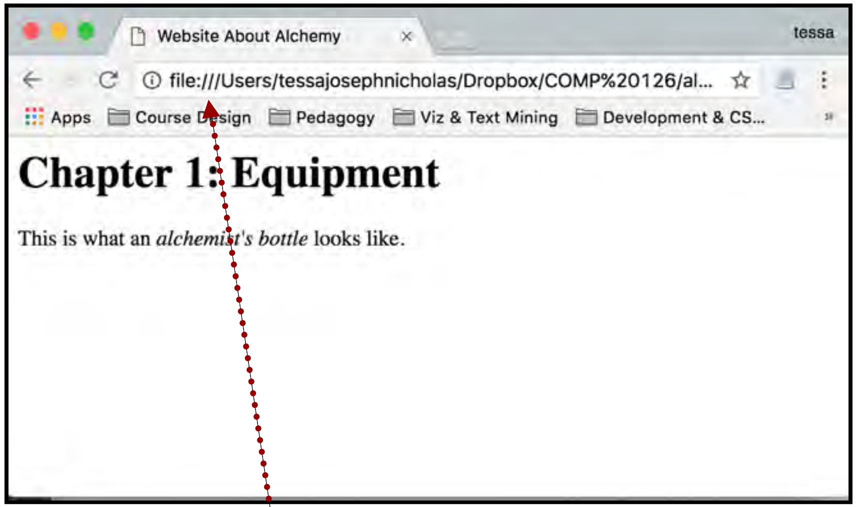

Bytes
Characters
Tokens
Nodes
DOM
3C 62 6F 64 79 3E 48 65 6C 6C 6F 2C 20 3C 73 70 61 6E 3E 77 6F 72 6C 64 21 3C 2F 73 70 61 6E 3E 3C 2F 62 6F 64 79 3E
<html><head>...</head><body><p>Hello <span>web performance</span>...
StartTag: html
StartTag: head
...
EndTag: head
StartTag: body
StartTag: p
Hello
...
html
head
meta
body
p
Hello
html
head
body
meta
link
p
div
Hello,
span
students
img
web performance
IMAGE: Google Developers

HTML document -> DOM

the HTML

```
<html>
<head>
  <title>
     Website About Alchemy
  </title>
</head>

<body>
  <h1>Chapter 1: Equipment</h1>
      <p>This is what an <em>alchemist's bottle</em>
      looks like.</p>
</body>
</html>
```

viewed in a browser



Website About Alchemy ×

tessa

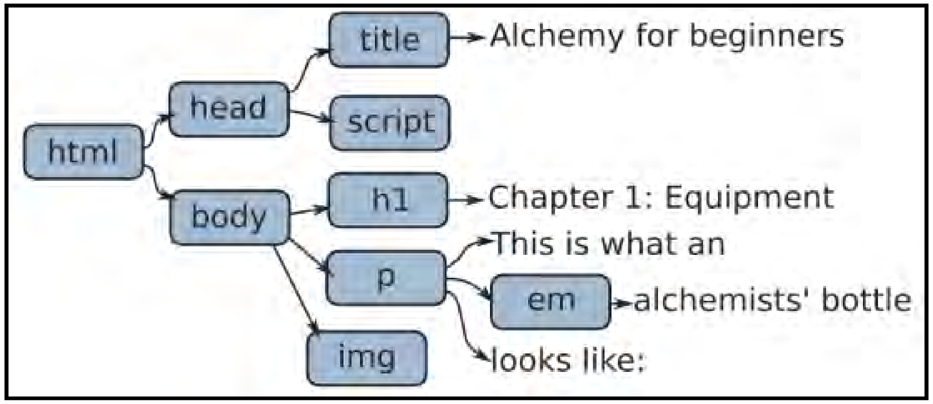
file:///Users/tessajosephnicholas/Dropbox/COMP%20126/al...

Apps | Course Design | Pedagogy | Viz & Text Mining | Development & CS...

Chapter 1: Equipment

This is what an *alchemist's bottle* looks like.

the DOM



(Reminder: this is not a URL! This site is not online! URLs start with http:// or https://, not . le:///. This path *only* means something on the computer in which these files are saved. This is also not an acceptable format for a relative path within a web project.)

Sample HTML5 Page Structure

```
<!DOCTYPE HTML>
<html lang="en">
<head>
  <meta http-equiv="Content-Type"
  content="text/html; charset=UTF-8" />
  <title>Title for browser tab</title>
</head>

<body>
        <header>
          <nav>
            <ul>
              <li>Menu item</li>
          <li>Menu item</li>
            </ul>
          </nav>
        </header>

        <section>
        <h1>Article title</h1>
      <article>
        <p>Article text</p>
      </article>
        </section>

        <aside>
          <h2>Sidebar header</h2>
        <p>Sidebar content</p>
    </aside>

        <footer>
          <p>Copyright, etc</p>
        </footer>
</body>
</html>
```

HTML5 elements you'll use a lot

```html
<h1>top level header</h1>
<h2>header level 2</h2>
<h3>header level 3</h3>
<h4>header level 4</h4>
<h5>header level 5</h5>
<h6>header level 6</h6>

<p>paragraph text</p>

<a href="some url">link</a>

<ul>
  <li>list item</li>
  <li>list item</li>
  <li>list item</li>
</ul>

<div>generic blockcontent container</div>

<span>generic inline text container</span>

<img src="path to image"
     alt="text describing image">
```

```html
<header>the content of
  your header</header>

<nav>your navbar</nav>

<main>the main content
  of the page</main>

<section>a group of
  related content</section>

<aside>a sidebar</aside>

<article>a textual article
  </article>

<footer>the conten of
  your footer</footer>
```

Types of links

```
ABSOLUTE URL

<a href="https://www.google.com/">Link to Google</a>

RELATIVE URL

<a href="contact.html">Link to another page
  in your website</a>

<a href="pages/contact.html">Link to another page
  in your website inside a folder called "pages"</a>

Note that this approach also applies to linking to images,
CSS stylesheets, script files, and other stuff. For example:

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

<img src="img/elephant.png" alt="graphic of elephant">

```

links & file paths

Absolute file paths...

point to specific locations on the Internet but outside your project and usually start with http:// or https://. In web dev, relative paths are more flexible (and thus more stable), so we use them whenever possible. But we use absolute paths when we need to link to, for example...

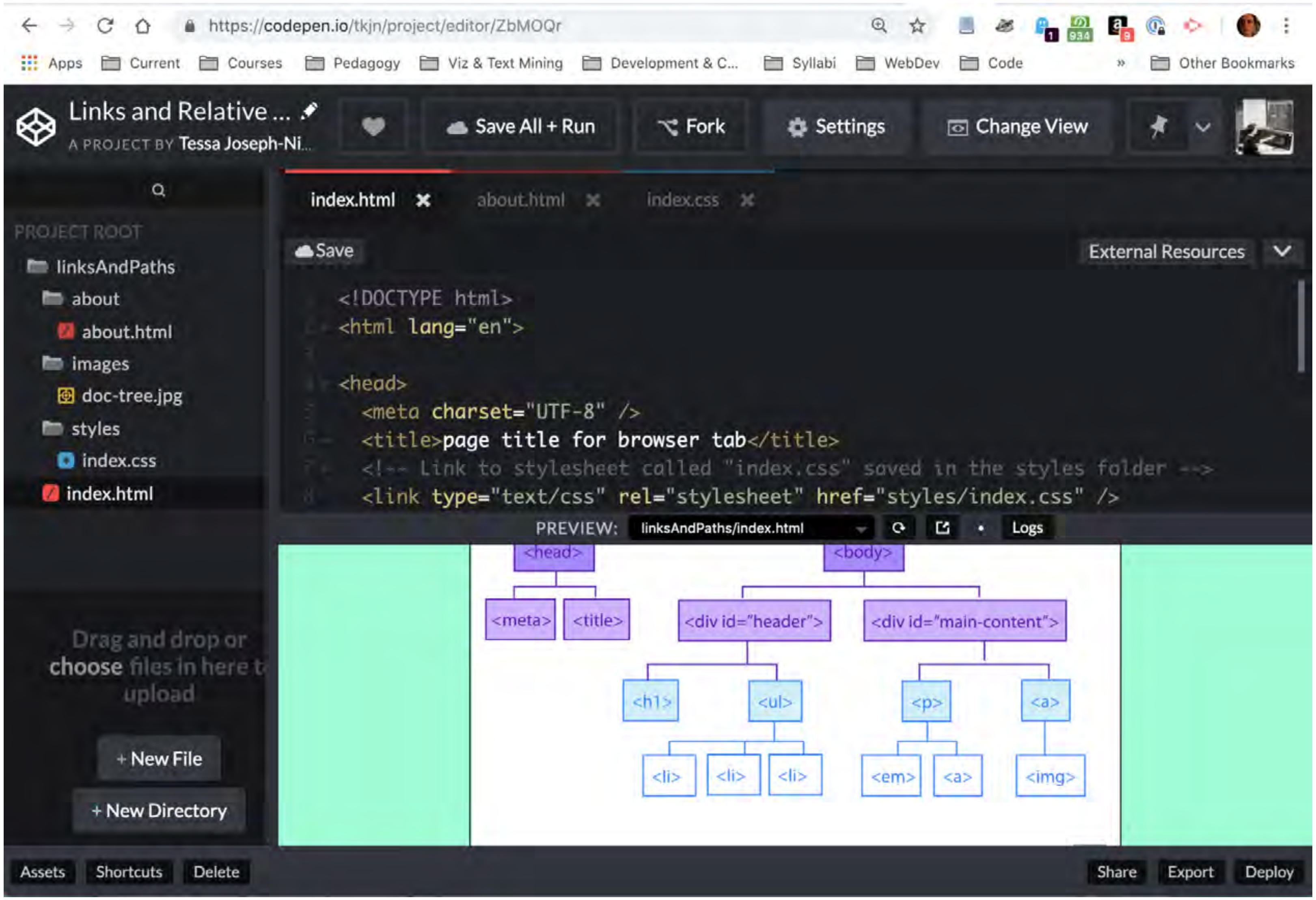
- external web sites
- CDNs (content delivery networks): stylesheets, fonts, icons, and other assets hosted publicly by trusted sources (e.g., Google, Bootstrap)
- Media hosted elsewhere, not in your project's file structure

Relative file paths...

point to documents or locations within your project's file structure, relative to the location of the html page you are viewing. We usually use relative paths to link to

- stylesheets located within your project's file structure (inside the same root directory)
- images and media files located within your project's file structure (inside the same root directory)
- other html documents in your file structure--i.e., between pages in your website

click on the image to explore some absolute & relative links/paths



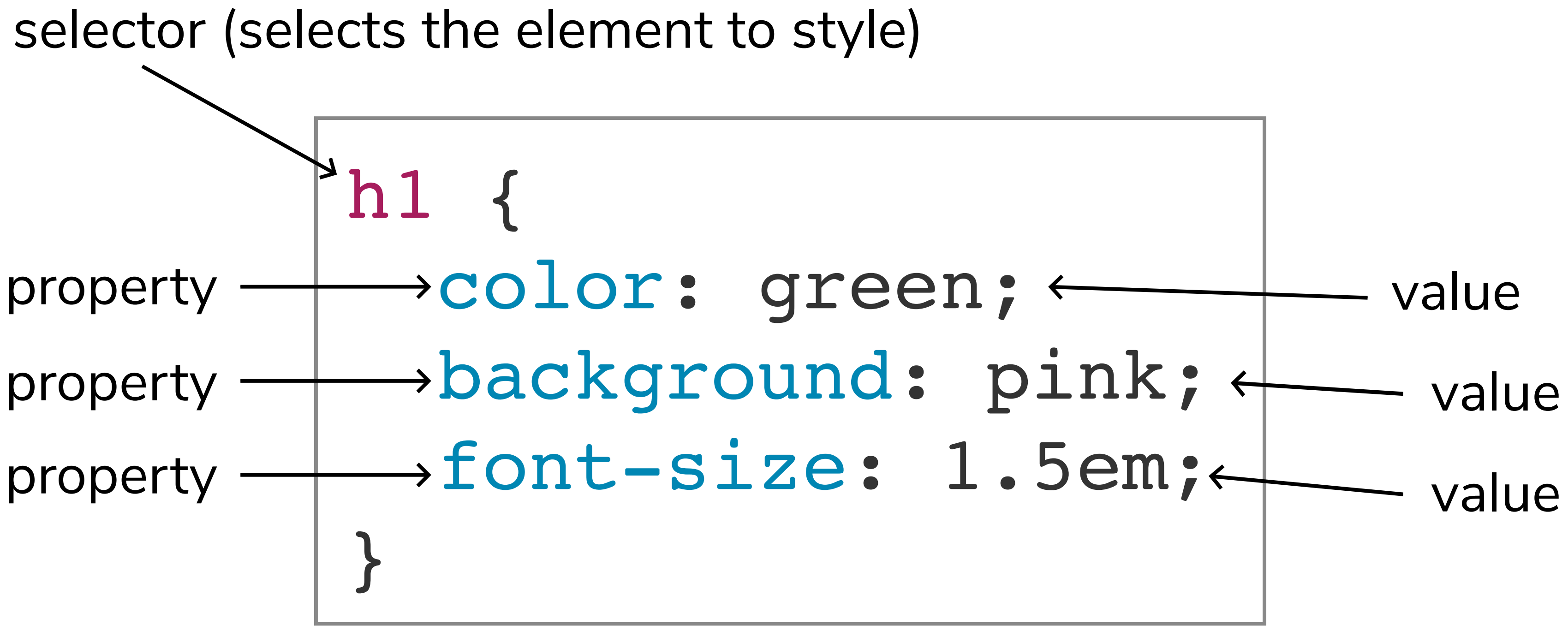
```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <title>page title for browser tab</title>
  <!-- Link to stylesheet called "index.css" saved in the styles folder -->
  <link type="text/css" rel="stylesheet" href="styles/index.css" />
```

CSS

"Cascading Style Sheet". CSS handles the presentation and styling of a website, allowing us to separate a site's styling from its structure and content (which are handled by HTML)

CSS Style Rules: Syntax

CSS consists of style rules, or "declarations". Each style rule consists of a *selector* and one or more *property-value pairs*, as follows:



```
h1 {
  color: green;
  background: pink;
  font-size: 1.5em;
}
```

note that all punctuation must be exactly as above!

CSS Comment Syntax

```
/* h1 {
   color: green;
   background: pink;
   font-size: 1.5em;
}
*/


//single lines
```

COMMENT YOUR CSS! IT HELPS!

CSS !important rule

```css
p {
   color: purple !important;
}
```

...more on this in a moment...

adding CSS to HTML

Adding CSS to HTML: inline (do not do)

- Uses the HTML <*selector* style=""> attribute
- Only applies to one element at a time
- Old school; NOT a preferred method

//codepen.io/tkjn/embed/aPxEOB/?height=265&theme-id=0&default-tab=html,result

Adding CSS to HTML: in the <head> element (do sparingly)

//codepen.io/tkjn/embed/pqBpbo/?height=265&theme-id=0&default-tab=html,result

- Declarations placed between <style></style> tags within the <head> element
- Best used only with demos or single-page sites that require very few declarations

Adding CSS to HTML: External stylesheet! Do this!

```
<head>
  <link href="styles.css"
    rel="stylesheet"
    type="text/css" >
</head>
```

//codepen.io/tkjn/embed/GPLyrX/?height=265&theme-id=0&default-tab=html,result

- **This is the preferred method!** Add a .css stylesheet to the project's file structure and link your .html documents to it via a <link rel> element in the <head>. (In a multi-page site, you must do this in the <head> of every single .html document.)
- You can (and frequently do) link more than one stylesheet to a single html document to help organize styles.

Types of CSS Selectors: the Standard HTML Element

When you want to make sure that every single instance of a standard HTML element on a page receives a certain style, use its standard HTML element/tag name as your selector.

```
p {
  property: value;
}
```

selects *all* <p> elements

```
img {
  property: value;
}
```

selects *all* <img> elements

ID & Class Selectors

IDs

IDs are for elements that you only need *once*. Can be used only once per page.

The id name is preceded by a # in the CSS.

Classes

Can be reused many times across a project.

The class name is preceded by a . in the CSS.

//codepen.io/tkjn/embed/yGrqJR/?height=265&theme-id=0&default-tab=css,result

DOM Position Selectors

- Select elements by DOM position when you want to specify styling for an element according to its relationship with another element
- Elements selected this way are more specific and will override elements selected with single HTML element selectors

Here's an example:

This declaration selects all <em> elements nested *inside* <p> elements.

```
p em {
  color: red;
}
```

List elements in descending order, with a space between each.

This <em> element will be selected by the above declaration:

```
<p>This is <em>important.</em></p>
```

This <em> element will not:

```
<h1>This is <em>important.</em></h1>
```

DOM Position & Attribute Selectors

These selectors combine DOM position with specific element attributes (qualities)

all h1 elements with a footer parent element

```
footer > h1 {
  color: red;
}
```

all elements with a .frame class that appear *immediately* after an h1 element

```
h1 + .frame {
  color: red;
}
```

all elements with .call class that are descendents of the element with id #contact

```
#contact .call {
  color: red;
}
```

all p elements anywhere after a h3 element

```
h3 ~ p {
  color: red;
}
```

all a elements with href starting "https"

```
a[href^="https"] {
  color: red;
}
```

all a elements that link to PDFs

```
a[href$=".pdf"] {
  color: red;
}
```

Pseudo-classes

Pseudo-classes style elements based on their current states.

links that are active

```
a:active {
  color: red;
}
```

links when hovered over

```
a:hover {
  color: yellow;
}
```

links that haven't been clicked

```
a:link {
  color: purple;
}
```

links that have been selected with the keyboard

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

links that have been clicked/visited

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

elements that are the "first child" of a parent element--there are variations for other child statuses

```
p:first-child {
  color: red;
}
```

Notes

1. most any element can take pseudo-classes, not just <a>

2. :hover must come after :link and/or :visited and before a:active in your CSS (you don't have to use them all, though)

3. all :hover styles should also be :focus styles: e.g., a:hover, a:focus { color: purple; } because :focus *is* :hover for people who're using a keyboard or clicker to navigate instead of a mouse

Types of CSS Selectors: Pseudo-elements

Pseudo-elements style part of an element or insert content before or after an element

first line of text of element

```
p::first-line {
  color: red;
}
```

first letter of element

```
p::first-letter {
  color: yellow;
}
```

inserts something before element

```
p::before {
    content: url(bird.jpg);
}
```

inserts something after element

```
h1::after {
  content: "hi!";
  color: red;
}
```

changes element selected by user

```
::selection {
    color: orange;
}
```

first letter of elements of a specific class

```
p.warning::first-letter {
  color: red;
}
```

- ::first-line, etc., only work on block elements
- more cool things you can do with pseudo-elements are here

Important CSS Concept: Specificity

1. inline styles (styles directly in the HTML; e.g., `<h1 style="color:blue;">`): highest specificity, highest priority
2. id selectors
3. class selectors, attribute selectors, pseudo-class selectors
4. normal element and pseudo-element selectors
5. the * selector (all elements): lowest specificity, lowest priority

CSS Selectors: Multiple ID/Class Selectors

You can combine classes and IDs in selectors.

Regular rules of specificity apply.

elements with both an id and a class

```
#contact.call {
  color: blue;
}
```

```
<div id="contact" class="call"></div>
```

elements with a mix of ids & classes

```
#call.first.contact {
    background-color: aqua;
}
```

```
<div id="call" class="first contact"></div>
```

elements with two classes

```
.first.second {
  color: yellow;
}
```

```
<div class="first second"></div>
```

Cascading Within Stylesheets

When rules conflict, who wins?

https://codepen.io/tkjn/embed/MWoyWpZ?default-tab=html%2Cresult

Cascading Across Style Sheets

1. !important declarations in user style sheet: highest priority
2. !important declarations in site author style sheet
3. normal declarations in site author style sheet
4. normal declarations in user style sheet
5. default declarations of web browser (user agent style sheet): lowest priority

Priority Factors

1. Importance

```
.style1 {
  background-color: aqua;
  color: black;
  border: none !important;
}
```

2. Specificity

```
p {
  background-color: black;
  color: white;
  padding: 10px;
}

.style1 {
  background-color: aqua;
  color: black;
  border: none !important;
}
```

3. Source order

```
p {
  background-color: green;
  color: white;
  padding: 10px;
}

p {
  background-color: gray;
}
```

More Selectors, and More About Selectors

- list of all pseudo-classes, pseudo-elements, and details of their methods is available here
- list of every single CSS selector is available here
- CSS selector testing tool available here

the display property

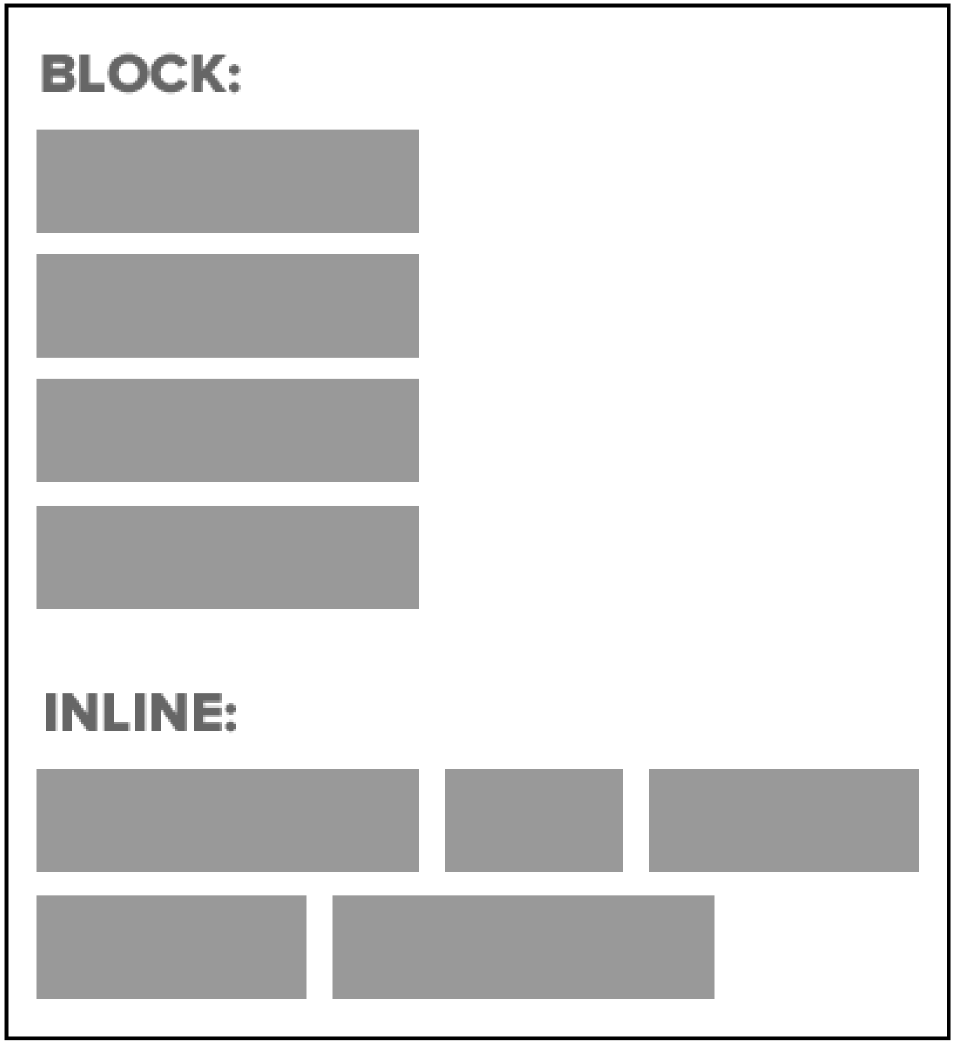
inline vs. block elements

- in CSS, there are two types of element: inline and block, and each has different default behaviors
- *inline* and *block* are values of the *display* property: e.g., {display: inline;}
- inline elements: img, a, br, em, strong, span
- block elements: p, h1-h6, ul, li, div, HTML5 structural-semantic elements, almost everything else

inline vs. block: default behaviors

Block elements stack top to bottom. The browser always inserts a line break after the end of a block element.

Inline elements flow L to R, on the same line that they are written on. The browser does not insert a space or line break.

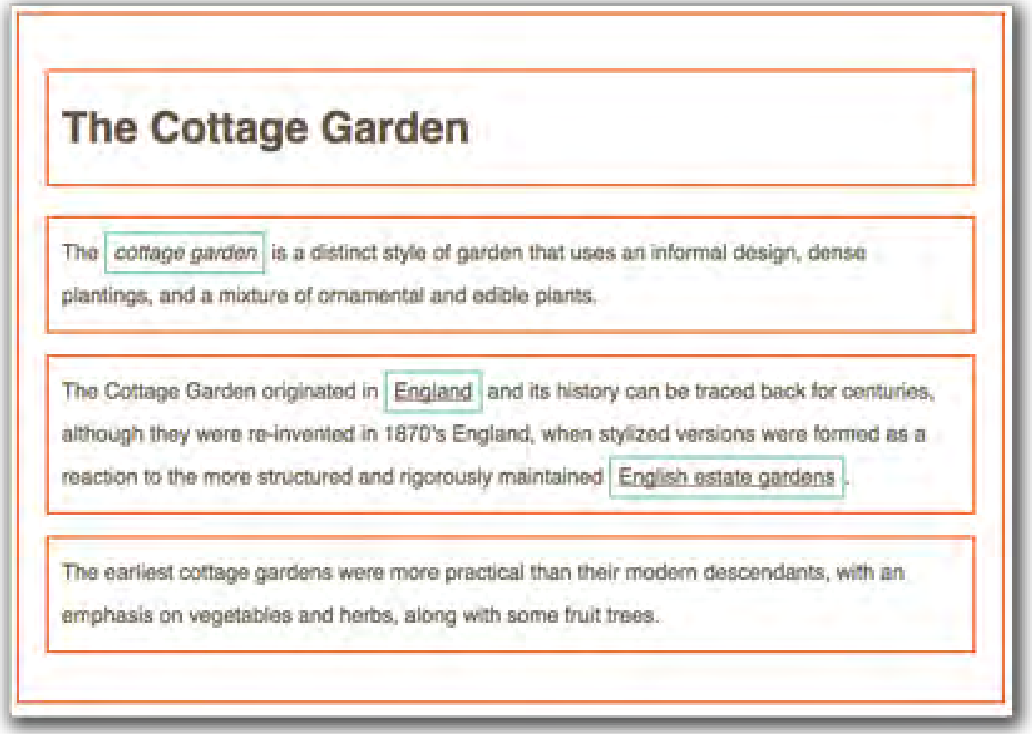


the *display* property

- {display: inline;} turns element into an inline element: can't assign width & height
- {display: block;} turns element into a block element
- {display: inline-block;} makes element a block element (can assign width & height) that behaves like an inline element (L to R, no line break)
- {display: contents;} makes the element's container disappear; contents become the child of the element the next level up in the DOM
- there are lots more display values, (including {display: flex;} and {display: grid;} which we'll get to later); full list here

the CSS Box Model

CSS is boxes inside boxes!



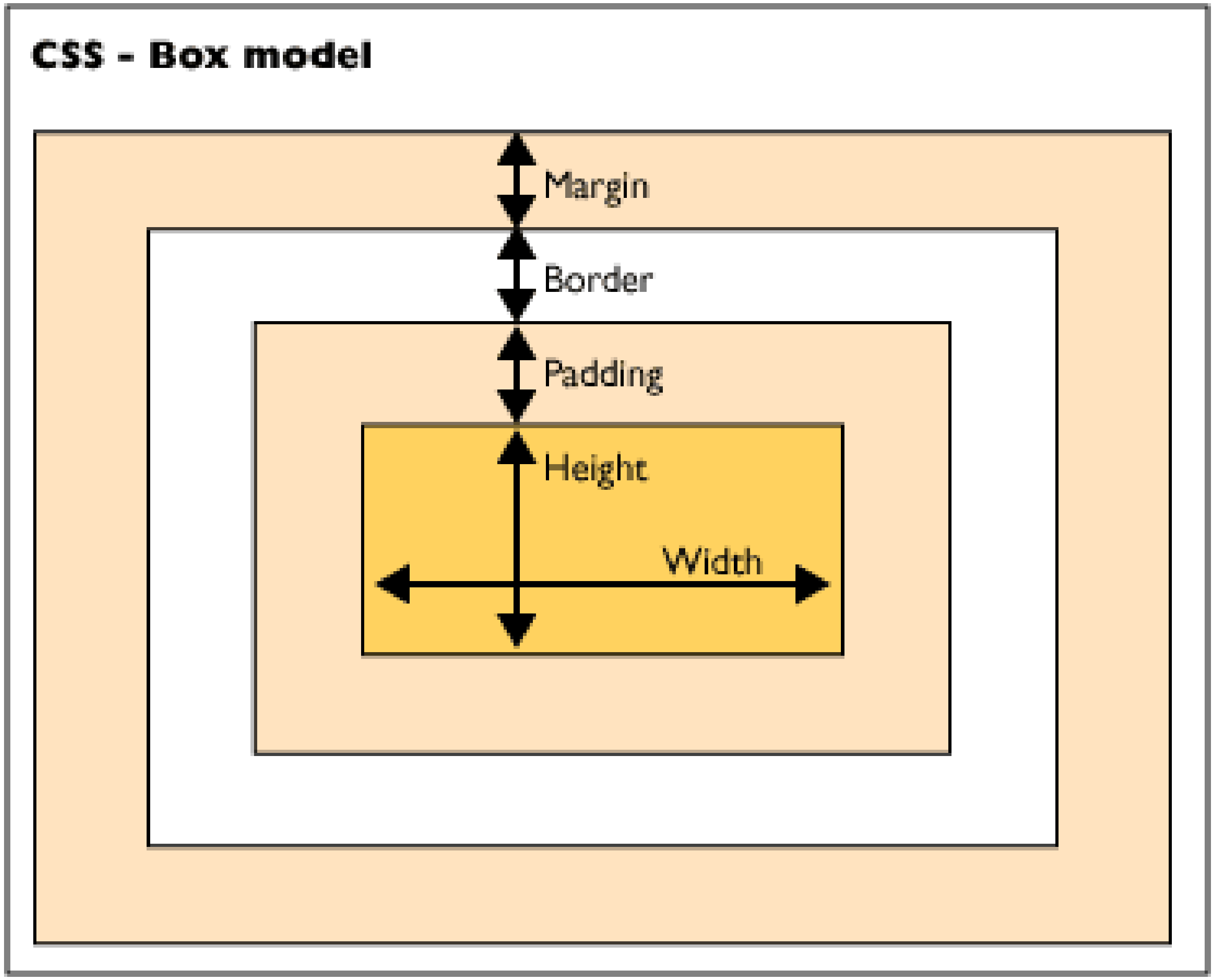
The Cottage Garden

The *cottage garden* is a distinct style of garden that uses an informal design, dense plantings, and a mixture of ornamental and edible plants.

The Cottage Garden originated in England and its history can be traced back for centuries, although they were re-invented in 1870's England, when stylized versions were formed as a reaction to the more structured and rigorously maintained English estate gardens.

The earliest cottage gardens were more practical than their modern descendants, with an emphasis on vegetables and herbs, along with some fruit trees.

The CSS Box Model



Components of the Box Model

Content area

The box that contains the content. Usually block-level. Width must be set by you; height is determined by the content unless you set it.

Padding area

The space between the content itself and the border/outer edge of the box.

Border area

Can be set to any width. The border adds to the overall width & height of the content unless you use {box-sizing: border-box;}.

Margin area

The space between the outer edge/border of an element and the outer edge/border of the next closest element/box. Btw, {margin: 0 auto;} applied to an element is a nice trick to center it in the browser.

Box Model Basics

//codepen.io/tkjn/embed/gqarWw/?height=265&theme-id=0&default-tab=css,result

box-sizing example

//codepen.io/tkjn/embed/BOdZQa/?height=387&theme-id=0&default-tab=css,result&embed-version=2

Measurement Units

- **1px:** fixed; size in pixels
- **1pt**: fixed; 1/72 of an inch, used in print media, but not much online anymore
- **1em: relative;** 1x text size of the *parent element*
- **1rem: relative;** 1x text size of *root element* (i.e., <html> or the user's browser preferences; often 16px)
- **1%: fluid;** if used on a font, it's 1% of the element's text size; if on width/height, it's 1% of the parent element's width/height
- **1vh & 1vw: fluid:** 1/100th of the viewport's height or width
- SO WHICH ONE DO I USE? For responsive design, think fluid measurement units for containers and element widths, relative for fonts, fixed for small details. But there's room for exceptions and your preferences.