
CSE 331

Software Design & Implementation

Andrew Gies & Hal Perkins

Winter 2021

HTML + TypeScript Overview

Administrivia

- Quiz 3 out now – finish by 7 pm tonight
- TypeScript code demo video on 331 course canvas panopto page
 - Must watch this before Wednesday's class (or else that will be *really* confusing/mysterious)
 - Extended TypeScript Q&A zoomathon with Andrew tomorrow afternoon 4-6 pm. Zoom link on the course canvas calendar
- And keep going on hw7 – due Thursday

The Weeks Ahead

- We're building an application that can find walking paths on the campus (hw7)
- We'd like to add a graphical user interface front-end (hw9)
 - The web is a common way to build/distribute apps
 - Web programming uses the same concepts we're learning
 - So: We're going to make a webapp for this.
 - Therefore: Let's learn how to do this!
 - Note: There are *many* ways to approach web programming. We're doing just one...

Our Approach

- We're going to be using several different pieces:
 - HTML
 - The language that web browsers render
 - Describes the structure and content of the page
 - TypeScript (TS)
 - A version of JavaScript that adds type-safety
 - Used to create the bulk of our application
 - Adds interactivity to the webpage
 - React
 - A UI library – handles the interactions between TS and HTML, makes UI programming easier

Our Approach (2)

- We're going to learn just enough to display a map, allow users to select endpoints, and draw a path
 - So we we'll focus on the basics, particularly key differences between what we're doing and Java
 - But also realize our goal isn't to exhaustively cover everything – don't have time, so core ideas only
- Will probably be outside your comfort zone – this is new stuff!
 - Remember to ask questions 😊
- Last two assignments this quarter:
 - HW8 draw dots and lines on an image (using TS/React)
 - HW9: use HW8 framework to build campus path GUI, use your existing Java graph/pathfinding

Credits

- CSE 331 JS/TS project due to Andrew Gies and Avi Bhagat
- Slides due to Andrew Gies, Hal Perkins & Kevin Zatloukal
- Thanks to Lauren Bricker and CSE 154 crew for some recent notes (even if you took 154 recently this stuff probably will look different)
- And from wherever we can find useful things...
- Notes: JS = JavaScript. ECMAScript is the official standard version (we're using v6/2015) so you'll also see ES or ES6 or ES2015, etc.

A little history

In the beginning was the web page

- It was displayed in a browser
- It had links
- But it was static
- There was no way to update or compute content dynamically or interact with users
- Solution: add a scripting language to the browser
 - Users (page developers) should be able to write code
 - Code should be able to interact with the browser's data structures to read / update / modify the page contents

World Wide Web

The WorldWideWeb (W3) is a wide-area [hypermedia](#) information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an [executive summary](#) of the project, [Mailing lists](#), [Policy](#), November's [W3 news](#), [Frequently Asked Questions](#).

[What's out there?](#) Pointers to the world's online information, [subjects](#), [W3 servers](#), etc.

[Help](#) on the browser you are using

[Software Products](#) A list of W3 project components and their current state. (e.g. [Line Mode](#), X11 [Viola](#), [NeXTStep](#), [Servers](#), [Tools](#), [Mail robot](#), [Library](#))

[Technical](#) Details of protocols, formats, program internals etc

[Bibliography](#) Paper documentation on W3 and references.

[People](#) A list of some people involved in the project.

[History](#) A summary of the history of the project.

[How can I help](#) ? If you would like to support the web..

[Getting code](#) Getting the code by [anonymous FTP](#), etc.

Enter JavaScript

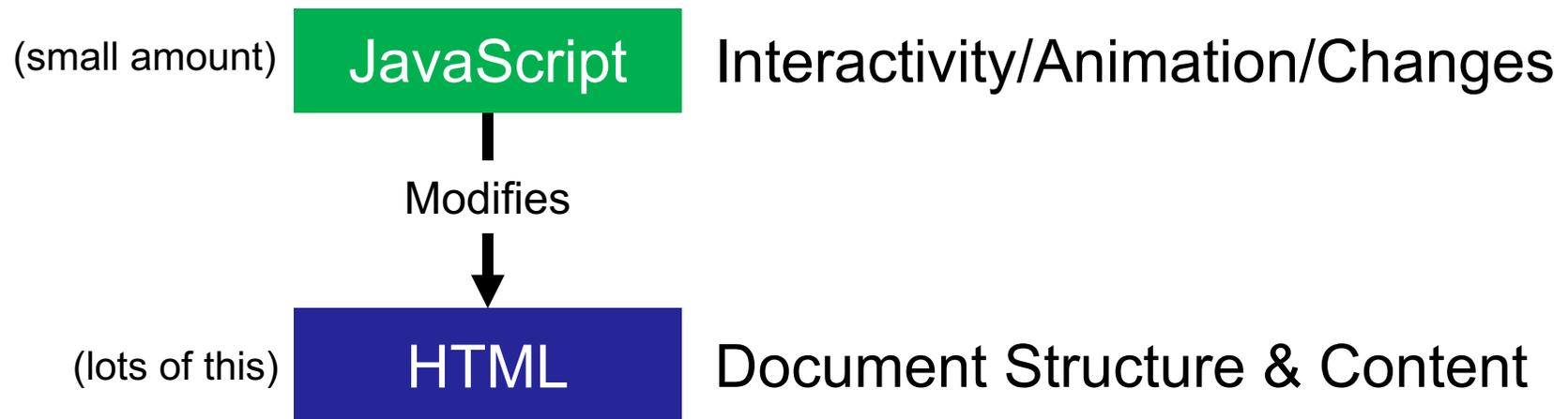
- Created in 1995 by Brenden Eich as a “scripting language” for Mozilla’s browser
 - Done in 10 days!
- Used to make web pages interactive:
 - Change the content/structure in HTML
 - React to events (page load, user clicks)
 - Discover info about local computer
 - Do local calculations
- No relation to Java other than trying to piggyback on all the Java hype at that time

Why JavaScript now?

- JavaScript is a web standard & ships in every browser
 - But not supported identically by all of them ☹️
- De facto execution engine for dynamic code on web
 - If a website is doing something interesting, there's probably JavaScript inside
- We will try to stick to portable, generic stuff
 - We use tooling that "smooths out" the difference between browsers as much as possible (it's the wild west out there)
 - But for hw8/hw9 we're only supporting Chrome (at least this time around) to avoid cross-platform grief
 - Install and update to current version please

In Context...

The "Original" Model of (Dynamic) Web Development

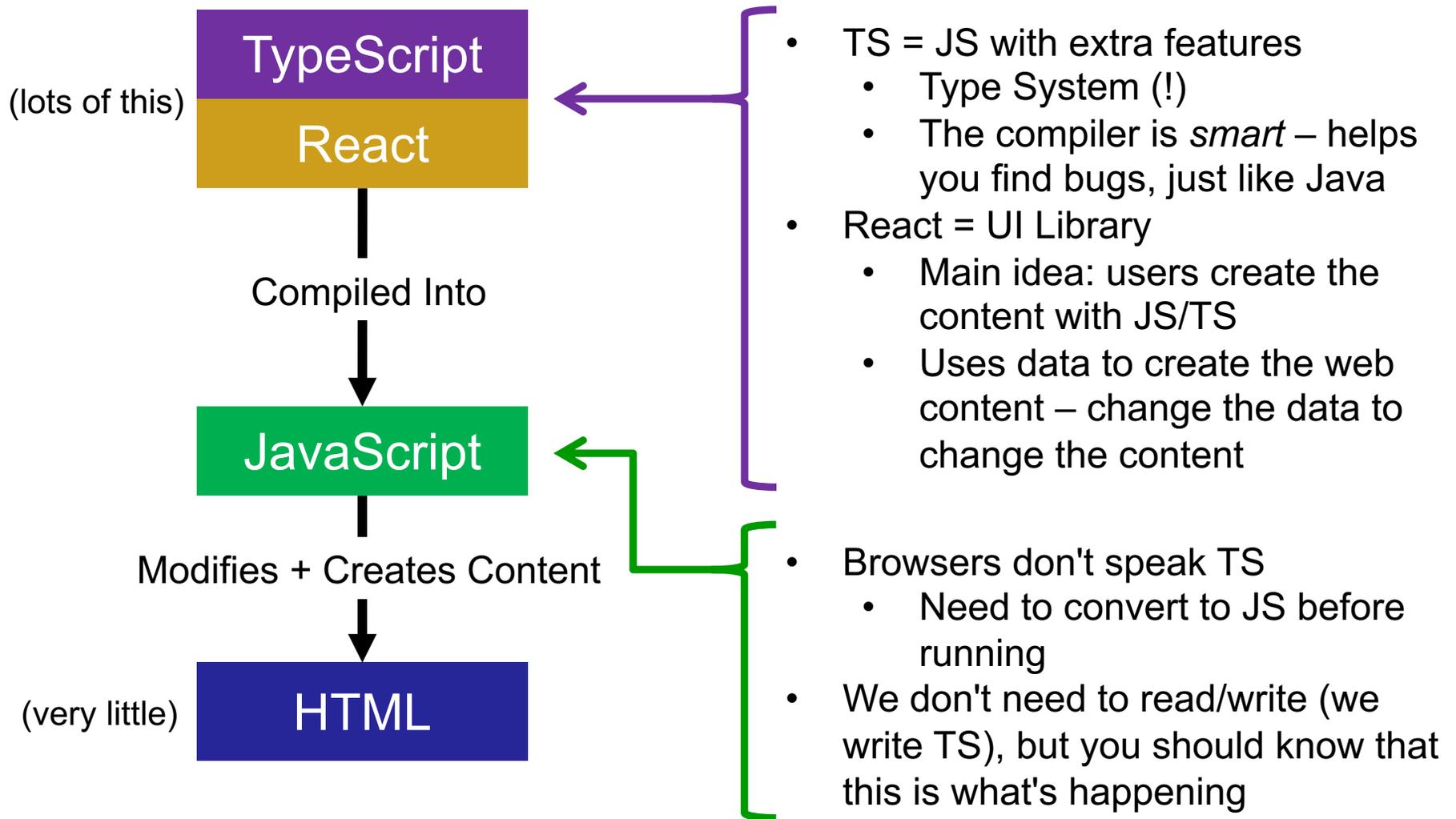


So that's what we're doing, right?

- Not quite...
- The original model was meant for simple things
 - click a button to submit a form, change a color, etc..
- The modern web now hosts full-fledged *applications* entirely using web technology
 - JS + HTML were never designed for this
- The "old" way:
 - Language + tooling doesn't help much, difficult to write big programs correctly/safely/efficiently
 - Managing large parts of the webpage with pure JS is difficult to get right

* There are a lot of ways to do things in modern web dev

One* Modern Alternative



Resources

- Lectures will (try to) point out key things
- TypeScript is *mostly* JavaScript – only big difference is types
 - Wondering how to do something? Look for JavaScript answers
 - Wondering how to type something? Look for TypeScript answers
- For more...
 - Mozilla (MDN) tutorials are good
 - CodeAcademy JavaScript basics
 - React documentation – small doses, way more info than we need
 - TypeScript documentation – focused on the "new stuff" in TS vs JS
- **Be very careful about web searches**
 - There are 1000 ways to do anything, many of which may be different than what we're doing
 - Code snippets from the web may lead you *way* off.
 - When in doubt, make an Ed post!

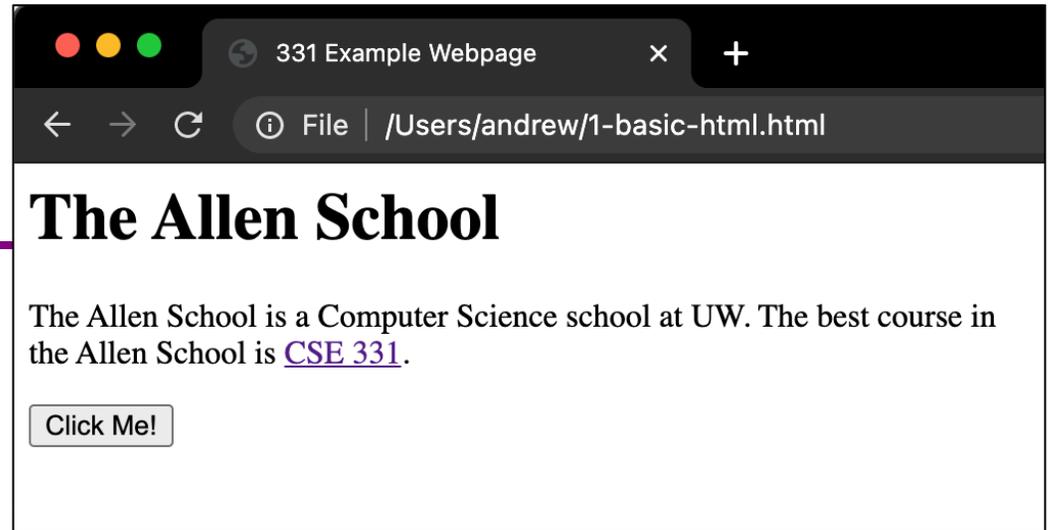
Our plan...

- First, look at basic HTML on its own
 - No scripting, no dynamic content
 - Just how content/structure is communicated to the browser
- Second, look at basic TypeScript on its own
 - No browser, no HTML, just the language
 - Get a feel for what's different from Java
- Third, use TypeScript with React with HTML
 - Write TypeScript code, using the React library
 - Generates the page content using HTML-like syntax

HTML, Formally

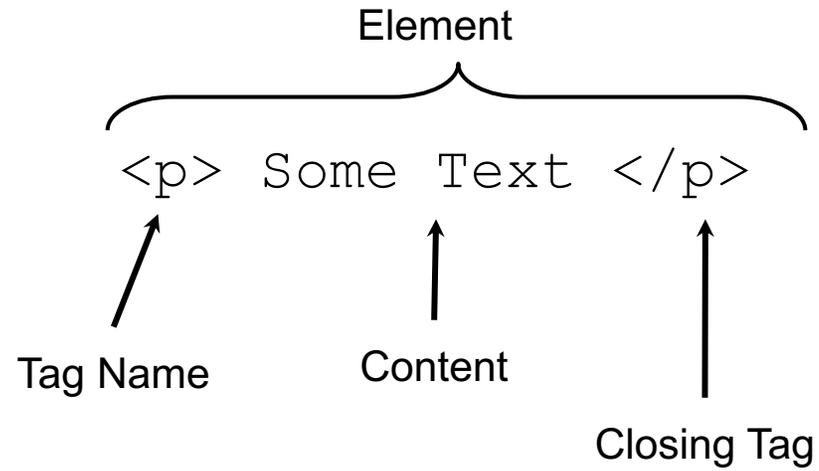
- HTML - HyperText Markup Language
- Consists of *tags* and their contents
 - Each tag has a different meaning
 - button, paragraph, link, etc...
 - Each one has a beginning and end.
 - Can contain text (content) and other tags. Optional attributes (organized as key-value pairs)
 - Can think of them like “constructor parameters”: pieces of data that specify extra info about the tag.
- Define document *structure and content*

Demo

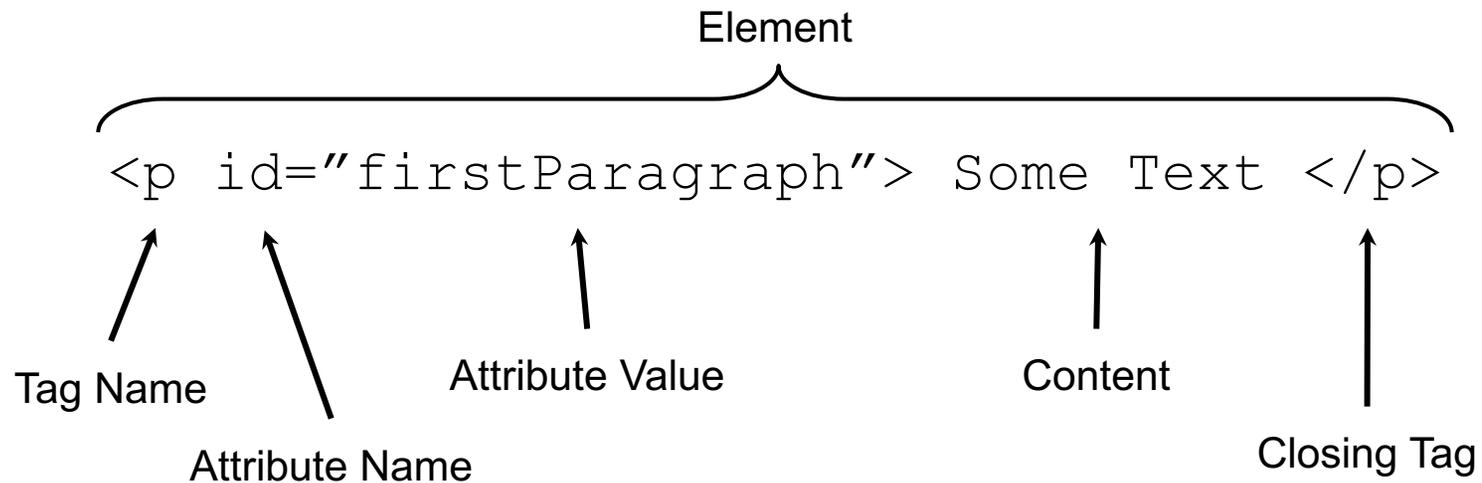


```
<html lang="en">
  <head>
    <title>331 Example Webpage</title>
  </head>
  <body>
    <h1>The Allen School</h1>
    <div>
      <p>
        The Allen School is a Computer Science school at
        UW. The best course in <br/> the Allen School is
        <a href="https://cs.uw.edu/331">CSE 331</a>.
      </p>
      <button>Click Me!</button>
    </div>
  </body>
</html>
```

Anatomy of a Tag



Anatomy of a Tag



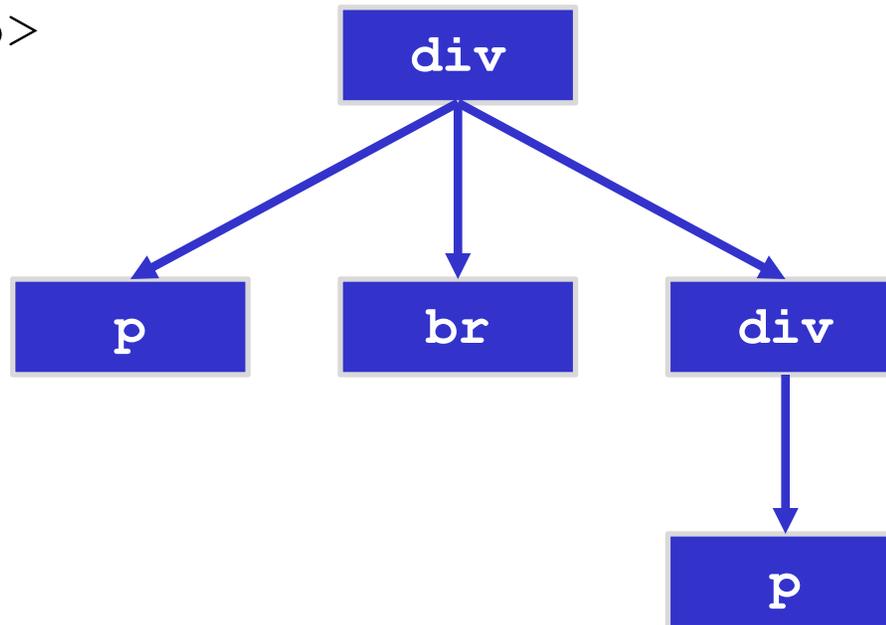
Self-Closing Tag (No Content)

`
`

*We'll see what `<p>` and `
` mean soon...*

Tags form a Tree

```
<div>
  <p id="firstParagraph"> Some Text </p>
  <br />
  <div>
    <p>Hello</p>
  </div>
</div>
```

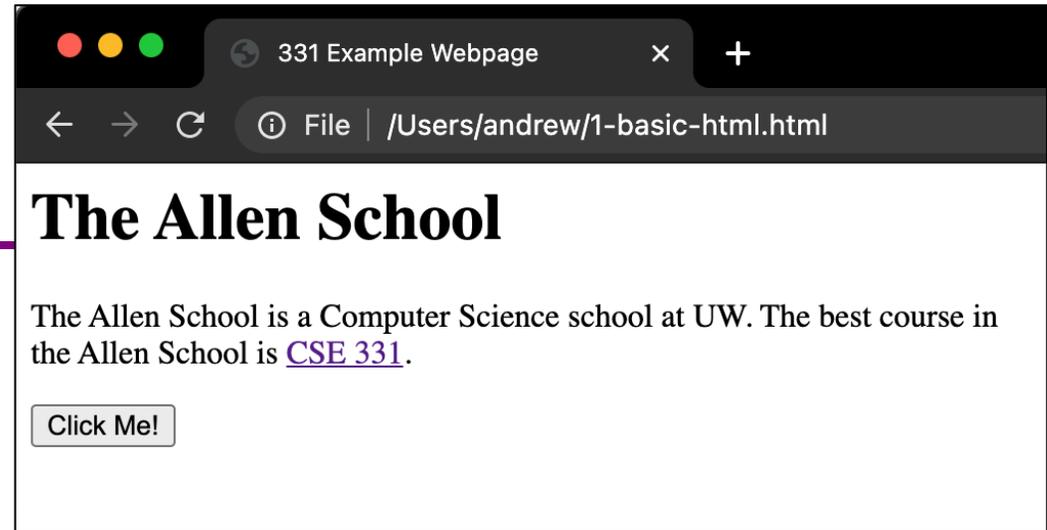


This tree data structure, which lives in the browser, is often called the "DOM" – *Document Object Model*

A Few Useful Tags

- A few worth mentioning here:
 - `<html>` and `<head>` and `<body>` - Used to organize a basic HTML document.
 - `<title>` - Sets the title of the webpage
 - `<p>` - Paragraph tag, surrounds text with whitespace/line breaks.
 - `<a>` - Link tag – links to another webpage.
 - `<div>` - “The curly braces of HTML” - used for grouping other tags. Surrounds its content with whitespace/line breaks.
 - `` - Like `<div>`, but no whitespace/line breaks.
 - `
` - Forces a new line (like “\n”). Has no content.
 - `<button>` - Create a clickable button on the screen
- See the W3Schools HTML reference for a complete list, along with all their supported attributes.

Demo Again



```
<html lang="en">
  <head>
    <title>331 Example Webpage</title>
  </head>
  <body>
    <h1>The Allen School</h1>
    <div>
      <p>
        The Allen School is a Computer Science school at
        UW. The best course in <br/> the Allen School is
        <a href="https://cs.uw.edu/331">CSE 331</a>.
      </p>
      <button>Click Me!</button>
    </div>
  </body>
</html>
```

What's next?

Done:

- First, look at basic HTML on its own
 - No scripting, no dynamic content
 - Just how content/structure is communicated to the browser

Now:

- Second, look at basic TypeScript on its own
 - No browser, no HTML, just the language
 - Get a feel for what's different from Java
- Third, use TypeScript with React with HTML
 - Write TypeScript code, using the React library
 - Generates the page content using HTML-like syntax

JavaScript (1)

Like Java in many ways:

- Variables:
 - `let` allows rebinding
 - `const` is like Java's `final` – can't change after creation

```
let something = "hello, world";  
const pi = 3.1415;
```

- Types of values:
 - `number` – floating point only, no integer type
 - `boolean` – `true/false`
 - `string` – similar to Java's strings
 - `undefined` – "unset" values
 - `object` (includes `null`) – more info later

JavaScript (2)

- `if/else if/else` statements
 - Structurally identical to Java
 - *Any value* can be used as a boolean:
 - `false`, `0`, `""`, `null`, `undefined`, `NaN` behave as `false`
 - Everything else (!) behaves as `true`
 - Values are described as "falsey" and "truthy"
- Loops
 - `for` & `while` – same as Java
 - `for-in` and `for-of` are like Java's `for-each`
 - Be careful with `for-in` and `for-of`, they're tricky
- Arrays
 - Can mix types in the array – `[123, "hello", false]`
 - No bounds checks, possible to access after the end
 - Versatile: behave as stacks/queues/lists

JavaScript (3)

- Functions
 - Can exist outside of classes/objects
 - Functions are *values*
 - Put them in variables
 - Pass them to functions
 - (more in demo)
- Objects
 - Similar to a Java HashMap
 - key/value pairs
 - The values can be functions
 - This is how we get methods!
 - Written using { and }
 - (more in demo)

```
let mul = function(x, y) {  
  return x * y;  
}
```

```
let add = function(x, y) {  
  return x + y;  
}
```

```
add(2, 3); // result is 5  
add = mul;  
add(2, 3); // result is 6
```

```
let simpleObj = {  
  x: 8,  
  y: "abc",  
  z: true  
};  
simpleObj.x; // result is 8
```

Why TypeScript?

- JS variables are *dynamically typed*
 - The type of a variable can change based on its value
 - JS will attempt to convert values where it can
 - This leads to tricky bugs

```
let x = 5;    // x holds a number
x = "35";    // x now holds a string
x += 7;      // x = "357"
```

- TS = Mostly JS, but adds *static* types (like Java)
 - Can declare type when creating a variable
 - TypeScript compiler will enforce this – prevents bugs!

```
let x: number = 5;
x = "35";    // TypeScript error!
```

Up Next

- Finish the current quiz before 7pm tonight
- Watch the TS Demo video before Wednesday
 - On Canvas under "Panopto Recordings"
 - Details on how the language works
 - Then zoom Q&A tomorrow on TS, 4-6 pm, optional but likely really useful
- Wednesday class: Using React + TS to create websites
- Thursday sections: HW8, TS + React Practice, Q+A