


---

# CSE 331

## Software Design & Implementation

Topic: HTML and TypeScript

 **Discussion:** What can you do to make a team work smoothly?

# Reminders

---

- I'm out this weekend – make sure to email course staff
- Watch TS Introduction video

# Upcoming Deadlines

---

- Prep. Quiz: HW7                      due Monday (8/01)
- HW7                                        due Thursday (8/04)

## Last Time...

---

- Generic Methods
- Generics and Subtyping
- Arrays
- Type Bounds
- Wildcards
- Type Erasure
- Callbacks

## Today's Agenda

---

- Event-driven Programming
- A Short History of Web
- HTML
- TypeScript

# A sorting example...

---

Consider the following sorting method:

```
public static void sort(List<Integer> lst) {
    for (int i = 0; i != n; i++) {
        for (int j = 0; j != n - 1; j++) {
            if (lst.get(j) > lst.get(j + 1)) {
                swap(lst, j, j + 1);
            }
        }
    }
}
```

What could we improve about this?

# A sorting example...

---

Consider the following sorting method:

```
public static void sort(List<?> lst) {
    for (int i = 0; i != n; i++) {
        for (int j = 0; j != n - 1; j++) {
            if (lst.get(j) > lst.get(j + 1)) {
                swap(lst, j, j + 1);
            }
        }
    }
}
```

But wait - this doesn't compile! Why?

# Achievement unlocked: Callbacks

---

- Even though we are the implementer, we may need the client to help us
  - previously, we have seen clients provide **data** that we can process
  - now, we will see how clients can provide **code** that can be executed

**Callback pattern:** “Code” provided by client to be used by library

- In JS etc., pass a function as an argument
- In Java, pass an object with the “code” in a method

*Synchronous* callbacks:

- Useful when library needs the callback result immediately

*Asynchronous* callbacks (i.e. event-driven programming):

- Useful for performing an action when some interesting event occurs later

# A sorting example...

---

First, we can define:

```
public interface Comparable<T> {  
    public int compareTo(T other);  
}
```

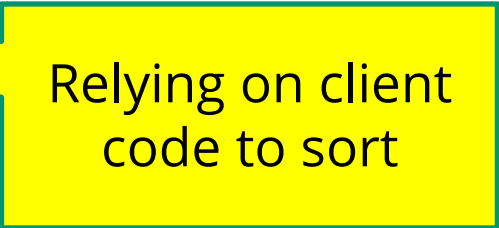
Every object that implements this interface must provide some **code** that informs us which of two objects is bigger.

- returns -1 if this is smaller than other
- returns 0 if this is equal to other
- returns 1 if this is bigger than other

# A sorting example...

---

```
public static <T extends Comparable<T>> void sort(List<T> lst) {  
    for (int i = 0; i != n; i++) {  
        for (int j = 0; j != n - 1; j++) {  
            if (lst.get(j).compareTo(lst.get(j + 1)) > 0) {  
                swap(lst, j, j + 1);  
            }  
        }  
    }  
}
```



Relying on client code to sort

We can use the callback pattern to ask the client how to compare to objects.



# How are callbacks used in practice?

---

- Clients sit around waiting for events like:
  - mouse move/drag/click, button press, button release
  - keyboard: key press or release, sometimes with modifiers like shift/control/alt/etc.
  - finger tap or drag on a touchscreen
  - window resize/minimize/restore/close
  - timer interrupt (including animations)
  - network activity or file I/O (start, done, error)
    - (we will see an example of this shortly)

# Achievement unlocked: Observers

---

This is the *observer pattern*

- Objects can be *observed* via *observers/listeners* that are *notified* via *callbacks* when an *event* (of interest) occurs
- **Pattern**: Something used over-and-over in software, worth recognizing when appropriate and using common terms
- Widely used in public libraries
- Useful for “visual” programs like web applications

More examples of “observers” coming later...

# Event-driven programming

---

An *event-driven* program is designed to wait for events:

- program initializes then enters the *event loop*
- abstractly:

```
do {  
    e = getNextEvent();  
    process event e;  
} while (e != quit);
```

Contrast with most programs we have written so far

- they perform specified steps in order and then exit
- that style is still used, just not as frequently
  - example: computing Page Rank or other Big Data work

# Event-driven programming

---

## Register Event

```
public void myFunction() {  
    System.out.println("I was here");  
}  
button1.addOnClickListener(myFunction);
```

## Event loop:

```
do {  
    e = getNextEvent();  
    process event e;  
} while (e != quit);
```



# Event-driven programming

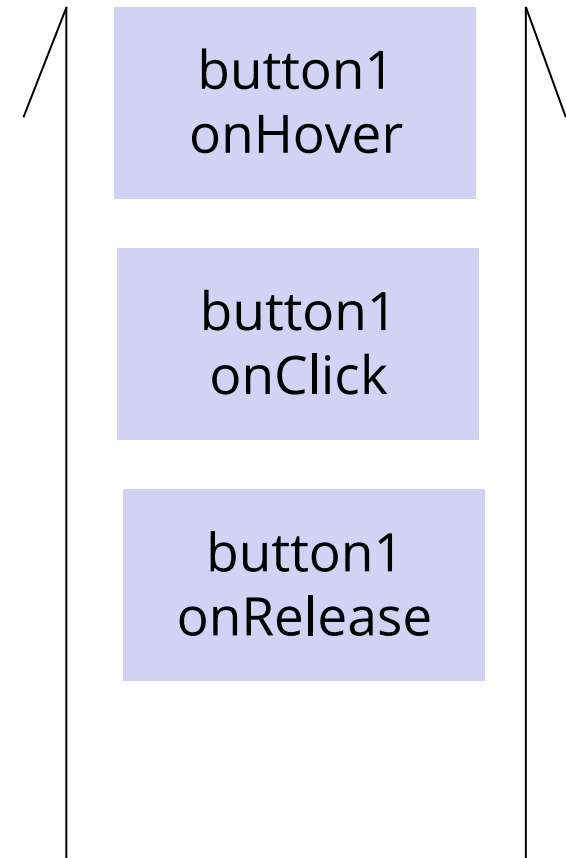
---

## Register Event

```
public void myFunction() {  
    System.out.println("I was here");  
}  
button1.addOnClickListener(myFunction);
```

## Event loop:

```
do {  
    e = getNextEvent();  
    process event e;  
} while (e != quit);
```



# Event-driven programming

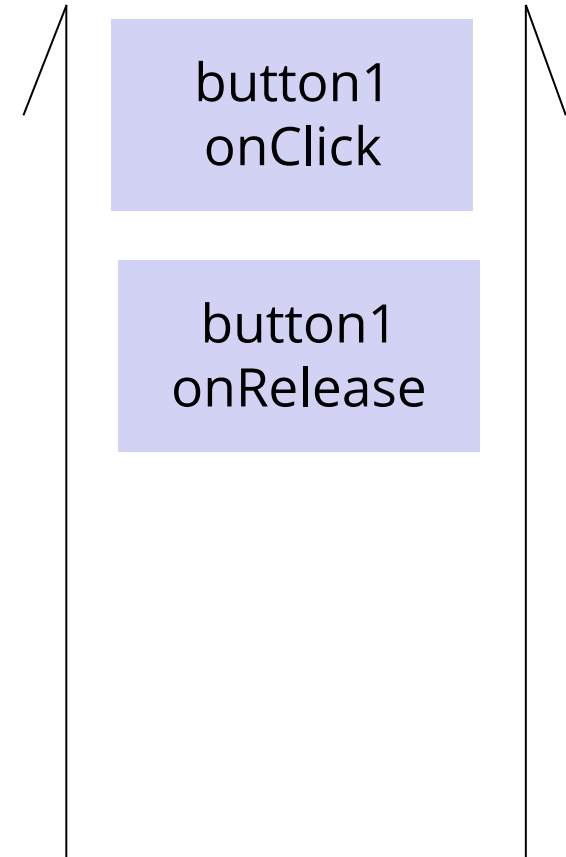
---

## Register Event

```
public void myFunction() {  
    System.out.println("I was here");  
}  
button1.addOnClickListener(myFunction);
```

## Event loop:

```
do {  
    e = getNextEvent();  
    process event e;  
} while (e != quit);
```



# Event-driven programming

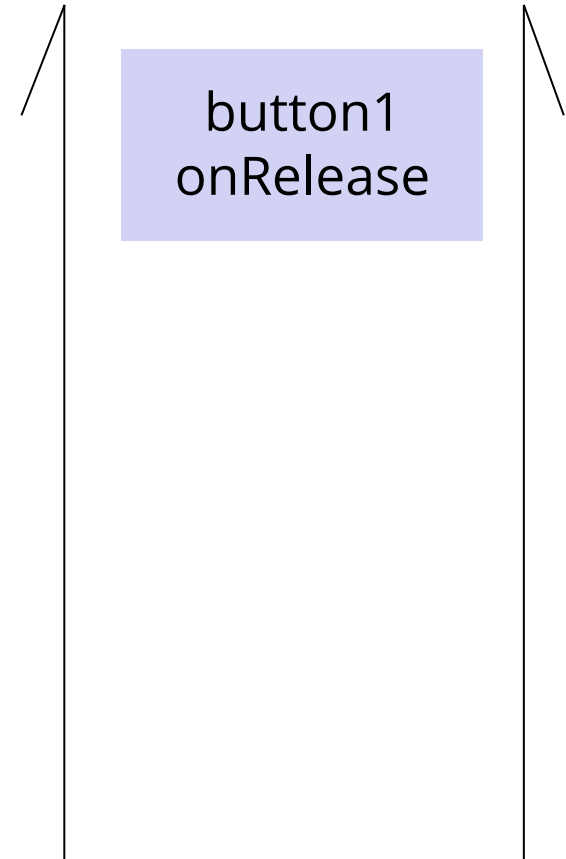
---

## Register Event

```
public void myFunction() {  
    System.out.println("I was here");  
}  
button1.addOnClickListener(myFunction);
```

## Event loop:

```
do {  
    e = getNextEvent();  
    process event e;  
} while (e != quit);
```



# Event-driven programming

---

## Register Event

```
public void myFunction() {  
    System.out.println("I was here");  
}  
button1.addOnClickListener(myFunction);
```

## Event loop:

```
do {  
    e = getNextEvent();  
    process event e;  
} while (e != quit);
```





# Looking Ahead

---

- We're going to build an application that can find walking paths on the campus
- We'd like to add a graphical user interface front-end once that's done
  - The web is a common way to build/distribute apps
  - Web programming uses the same concepts we're learning
- Note: There are *many* ways to approach web programming. We're doing just one...

# Looking Ahead

---

- We're going to need to learn a few 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

# Looking Ahead

---

- We're going to learn just enough to display a map, allow users to select endpoints, and draw a path
  - Focus on the basics, i.e. key differences between what we're doing and Java
  - Our goal isn't to 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 will draw lines on a map image (using TS/React)
  - HW9 connects the HW8 UI to the implementation of Dijkstra's from HW7

# Credits

---

- CSE 331 JS/TS project originally due to **Andrew Gies** and **Avi Bhagat**, new version in 22wi done by **Bryan Lim** and **Ardi Madadi** (& a host of others testing, etc.)
- Slides due to **Andrew Gies**, **Hal Perkins**, and **Kevin Zatloukal**
- Thanks to **Lauren Bricker** and CSE 154 crew for some additional notes (but even if you took 154 recently this stuff probably will look different)
- And from wherever we can find useful things...

# A little history

---

In the beginning, there 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?

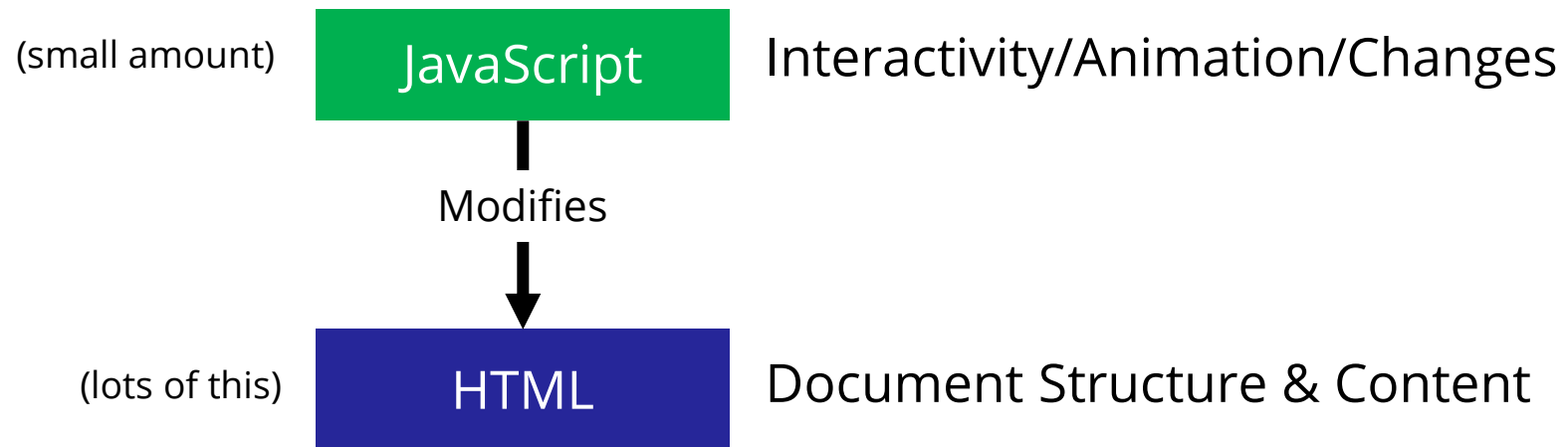
---

- 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
  - 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 issues

# In Context...

---

The "Original" Model of (Dynamic) Web Development





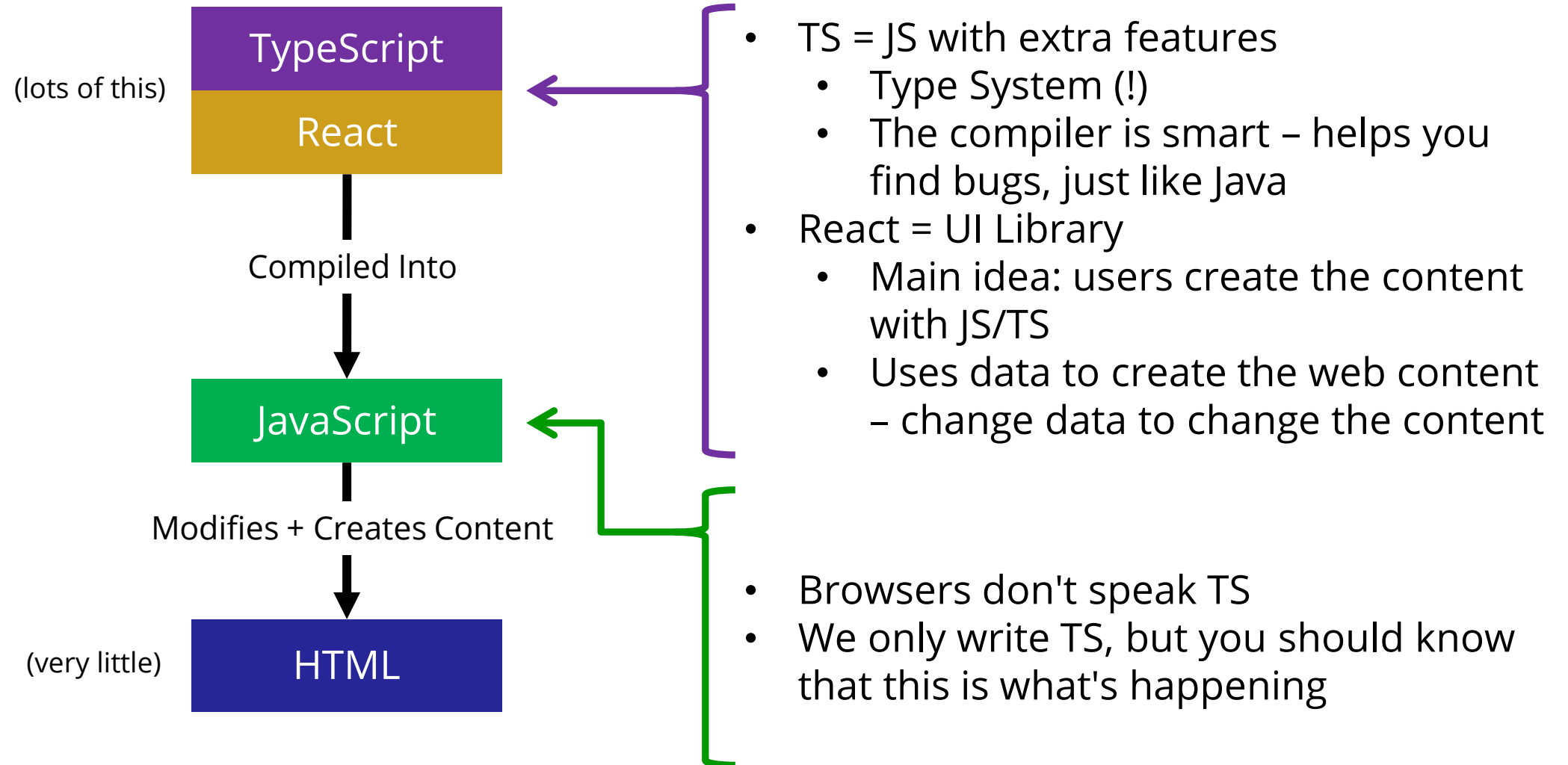
# So that's what we're doing, right?

---

- 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 are 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 (& JavaScript) on its own
  - No browser, no HTML, just the language
  - Get a feel for what's different from Java
- Third, a quick look at very basic user interactions
  - Events, event listeners, and callbacks (just basic ideas now)
- Fourth, use TypeScript with React with HTML
  - Write TypeScript code, using the React library
  - Generates the page content using HTML-like syntax

# HTML, Formally

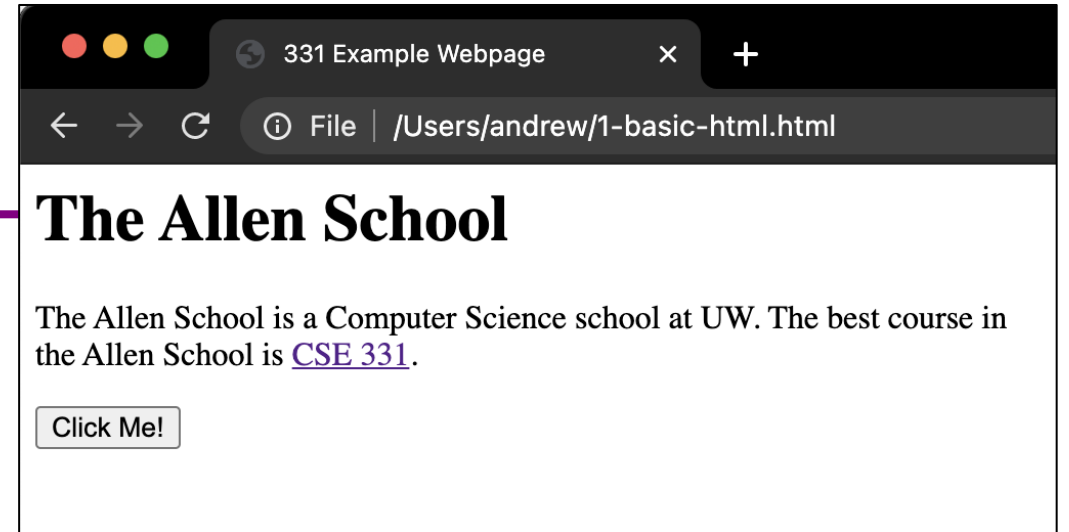
---

- 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

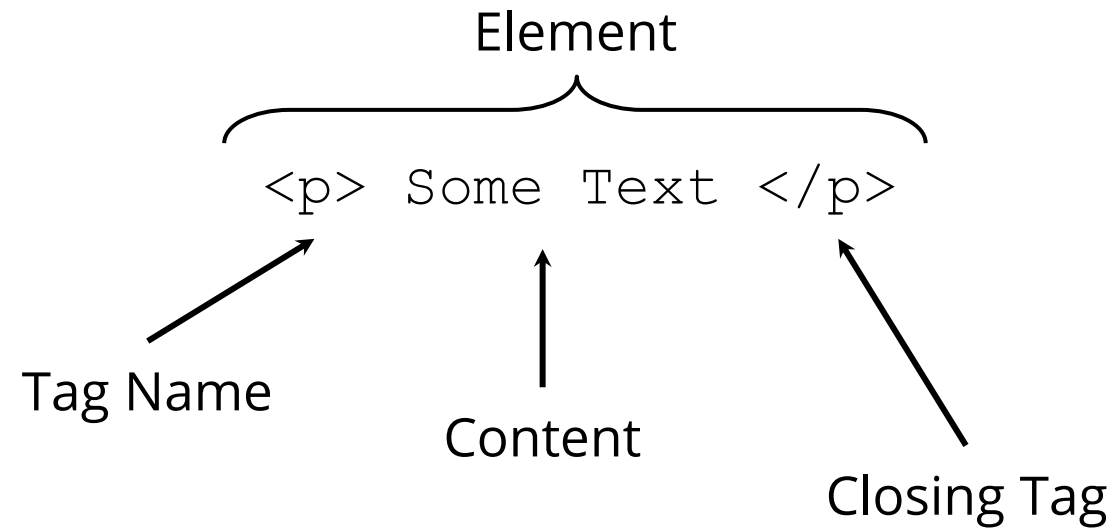
---

```
<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>
```



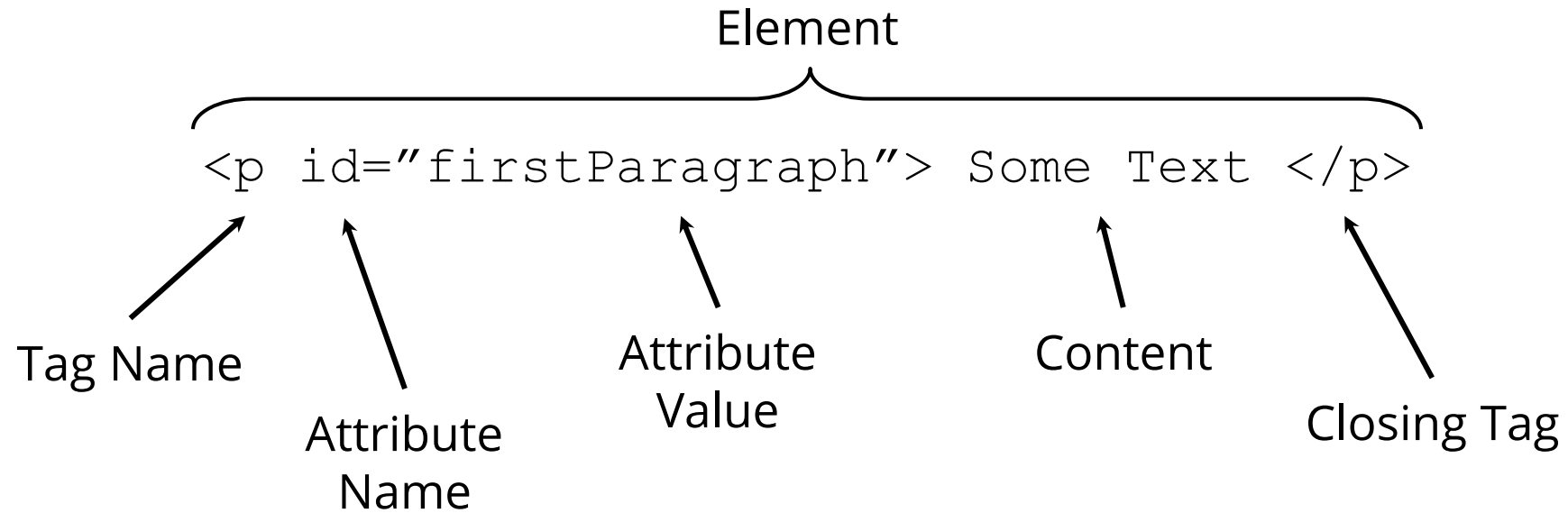
# Anatomy of a Tag

---



# Anatomy of a Tag

---



---

Self-Closing Tag (No Content)

<br />

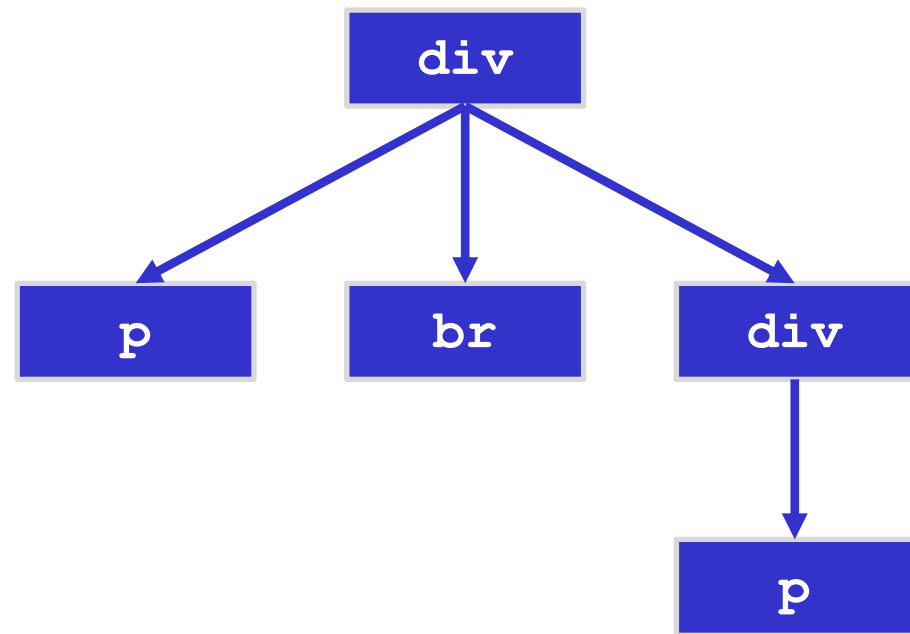


# Tags form a Tree

---

```
<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

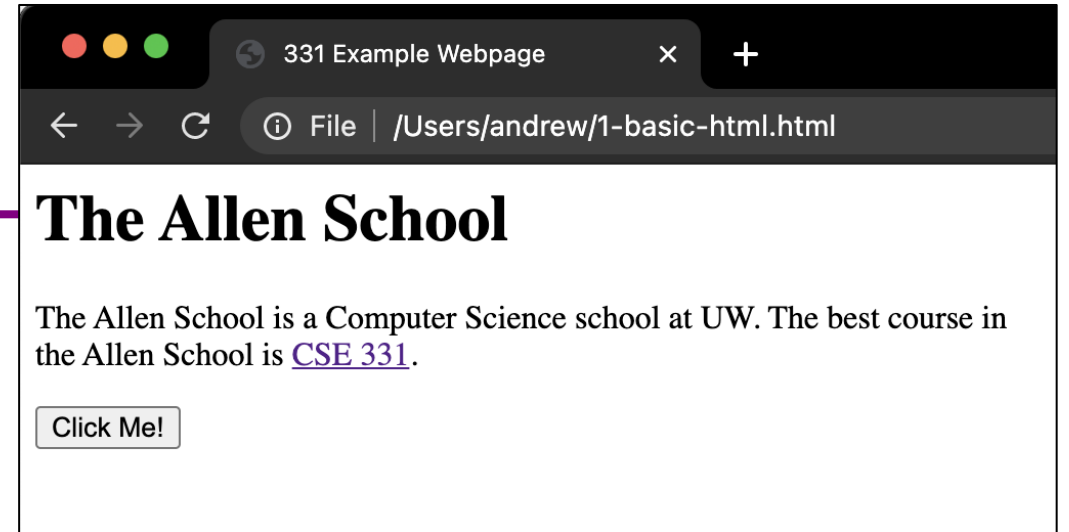
---

- 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.
  - `<span>` - Like `<div>`, but no whitespace/line breaks.
  - `<br />` - 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

---

```
<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?

---

- 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 (& JavaScript) on its own
  - No browser, no HTML, just the language
  - Get a feel for what's different from Java
- Third, a quick look at very basic user interactions
  - Events, event listeners, and callbacks (more depth later)
- Fourth, 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` 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
- Objects
  - Key/Value pairs
    - Similar to a Java HashMap
  - The values can be functions
    - This is how we get methods!
  - Written using { and }
    - Recent JS/ECMAScript adds "class" syntax so it looks more familiar

```
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!
```



# More TypeScript

---

- Longer online video tutorial
  - Please watch before next Monday (otherwise that class won't make much sense)
- Some basic sample files in the TypeScript/ folder accompanying these slides (see calendar for link)

# What's next?

---

- 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 (& JavaScript) on its own
  - No browser, no HTML, just the language
  - Get a feel for what's different from Java
- Third, a quick look at very basic user interactions
  - Events, event listeners, and callbacks (more depth later)
- Fourth, use TypeScript with React with HTML
  - Write TypeScript code, using the React library
  - Generates the page content using HTML-like syntax

# Demo Revisited

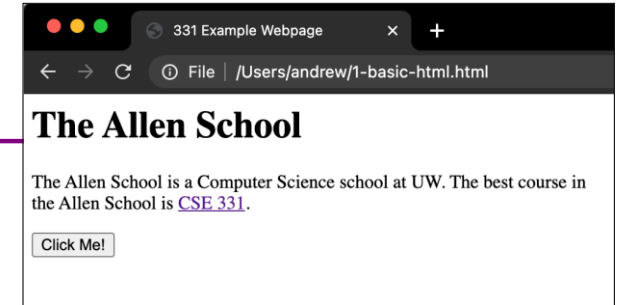
---

- Our first webpage was static
  - It even included a picture of a button, but nothing happened when it was clicked
- How do we add interaction?

## Demo

---

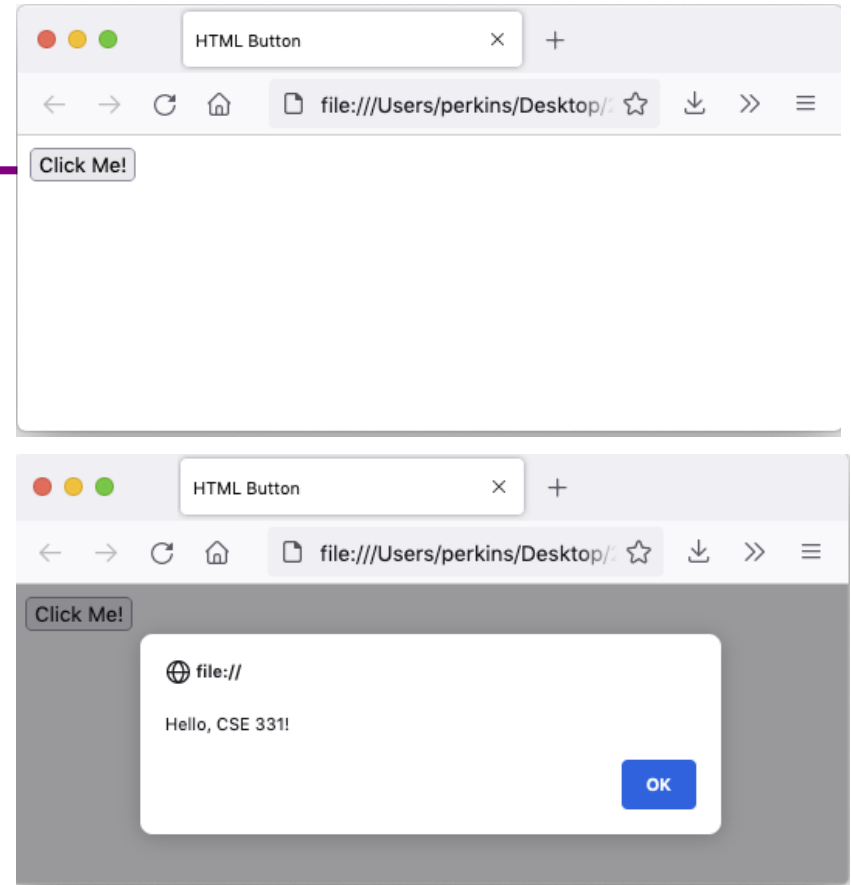
```
<html lang="en">
  <head>
    <title>331 Example Webpage</title>
  </head>
  <body>
    <h1>The Allen School</h1>
    <div>
      <p>
        The Allen School is a Computer Science school at
        UW. The best course in <br/> the Allen School is
        <a href="https://cs.uw.edu/331">CSE 331</a>.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



# Demo 2

---

```
<html lang="en">
  <head>
    <title>HTML Button</title>
  </head>
  <body>
    <script type="text/javascript">
      function sayHello() {
        alert("Hello, CSE 331!");
      }
    </script>
    <button onclick="sayHello()">Click Me!</button>
  </body>
</html>
```



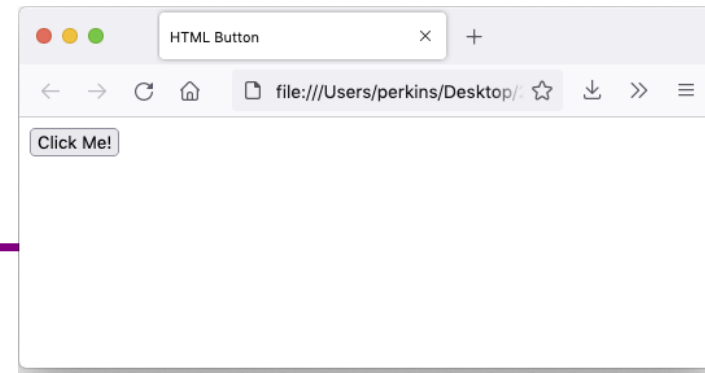
# What happened here?

---

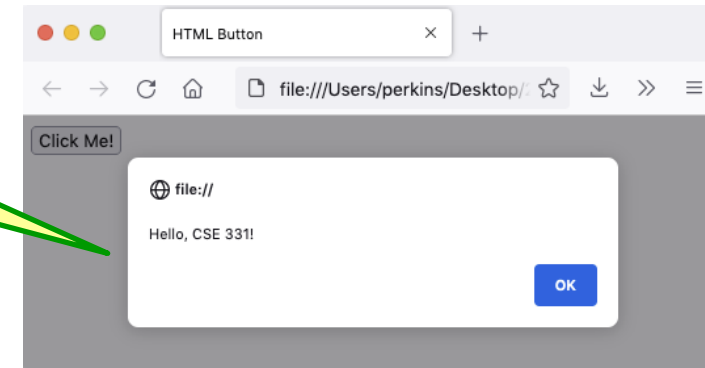
- This is the *callback pattern*
- The webpage is loaded into the web browser and it contains a JavaScript function and a button
- When the button is created, the JS function to be called on a button click is *registered* with the button
  - The function is not called at this time
- When the user clicks the button, it causes a user-interface *event* to happen
  - In response, the button calls the function that was registered to be called (notified) whenever there is a click event
    - This is a *callback*

# Demo 2 rev

0 - web page is loaded into browser



3 - when button is clicked function sayHello() is called and alert box is displayed



```
<html lang="en">
  <head>
    <title>HTML Butt
  </head>
  <body>
    <script type="text/javascript">
      function sayHello() {
        alert("Hello, CSE 331!");
      }
    </script>
    <button onclick="sayHello()">Click Me!</button>
  </body>
</html>
```

1 - JS sayHello function embedded in web page inside <script> tag

2 - Button created on page load; sayHello() function *registered* to be called on click event

# Demo 2 - Takeaway

---

- This demo gives a very simple example using plain JavaScript – details will be different in React, but the core callback idea will be the same
  - On startup, register code to be activated when events happen
    - Multiple ways to do this: options in an html tag (basic JS), call a “register” function and pass to it the function to call when the event happens (react), similar things in other async systems
  - When an event happens (button press, text added to dialog, timer expires, data read, etc. etc.) the code that is registered ahead of time will be called

# Before next class...

---

1. Watch the [TS Introduction](#) video posted on Panopto before next lecture
2. Start on the [Prep. Quiz: HW7](#) to get practice with generics
  - Will need to apply generics and implement Dijkstra's algorithm
3. If you are uncomfortable with generics, start [HW7](#) early
  - Will need to apply generics
  - Useful for implementing Dijkstra's algorithm on a **Graph<Double>**