SESSION NINE
OCTOBER 24, 2023

AN INTRO TO JAVASCRIPT

ENDLESS APPRECIATION
TO ERIC LI +
MICHAEL FEHRENBACH
FOR THE MATERIAL

DAVIS SCHERER
SCHERERD@NEWSCHOOL.EDU

AGENDA

1. What is Javascript?
2. Javascript basics
3. Break
4. Practice

JAVASCRIPT

JavaScript is the way we add interactivity into a webpage—through actions, or user input.

JAVASCRIPT

We’re not going to learn a whole new language; let’s try and understand how to **read, use, and adapt** JS to meet our needs.

JAVASCRIPT

JavaScript was invented in 1995 by Brenden Eich. At the time, the web was entirely static—just HTML and CSS.

JAVASCRIPT

Eich was responding to a desire for dynamic webpages in developing what was then called LiveScript. He renamed it JavaScript to capitalize on the popularity of the programming language Java.

JAVASCRIPT

(It has nothing to do with Java)

JAVASCRIPT

JS runs in the browser, which means that it’s a highly accessible and modifiable. It’s the beating heart of the modern web.

PROGRAMMING (REFRESH)

MACHINE LANGUAGE

```
mov eax, num1;
imul eax, num2;
mov sum, max;
ret
```

Explicit instructions to the computer, but tedious and easy for humans to make mistakes

NATURAL LANGUAGE

“Multiply these two numbers together”

Spoken language, hard to be specific enough for a machine to interpret accurately

‘HIGH LEVEL’ LANGUAGE

```
Let p = num1*num2;
```

A compromise… harder to read than natural language but has the specificity needed for a computer to access it

LIBRARIES VS VANILLA

Developers have built innumerable JS *libraries* (aka *frameworks*) that expand the functionality of JavaScript in specific ways—around ideas, syntax, paradigms.

LIBRARIES VS VANILLA

jQuery, Node, React, Vue, Angular, D3, p5, matter—these libraries are written in and controlled by JavaScript, but allow you to do something JS doesn’t support on its own.

LIBRARIES VS VANILLA

JS written without libraries is *vanilla* or *plain* javascript. We'll start here… and talk a little about libraries as we go.

JAVASCRIPT CODING

Like CSS, JavaScript can live in 3 places.

JAVASCRIPT CODING

Lik va liv es.

INLINE as attributes

Via `<script>` elements within HTML documents

As separate `.js` files

INLINE

JS, like CSS, was first added directly into HTML as attributes in HTML elements. It was used for specific events.

INLINE

```
<button onclick="alert('The button was
  clicked!');">Click here!</button>
```

INLINE


HTML ELEMENT
JAVASCRIPT
<button onclick="alert('The button was clicked!');">Click here!</button>
HTML VALUE
(VISIBLE TEXT)
HTML ELEMENT

INLINE

This is an outdated way of coding javascript—it should still work for very simple applications, but it’s not a best practice.

<SCRIPT>

JS can also be wrapped in a special `<script>` element in an HTML document.

<SCRIPT>

* JS wrapped in <script> tags will be executed right away, and in the order they appear—much like CSS in a cascade
* The script isn’t directly applied to an element by default—so you have to target and select the element.

<SCRIPT>

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link href="/assets/styles/reset.css" rel="stylesheet">
    <link href="setup.css" rel="stylesheet">
  </head>
  <body>
    <button id="example">Click here!</button>
    <script>
      // Set up a variable for our button!
      let button = document.querySelector('#example');
    // Any CSS selector.

      button.onclick = () => { // Attach the click event.
        alert('The button was clicked!'); // Pop an alert!
      };
    </script>
  </body>
</html>
```

<SCRIPT>

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link href="/assets/styles/reset.css" rel="stylesheet">
    <link href="setup.css" rel="stylesheet">
  </head>
  <body>
    <button id="example">
    <script>
      // Set up a variable for our button.
      let button = document.querySelector('#example');
    // Any CSS selector.

      button.onclick = () => { // Attach the click event.
        alert('The button was clicked!'); // Pop an alert!
      }
    </script>
  </body>
</html>
```

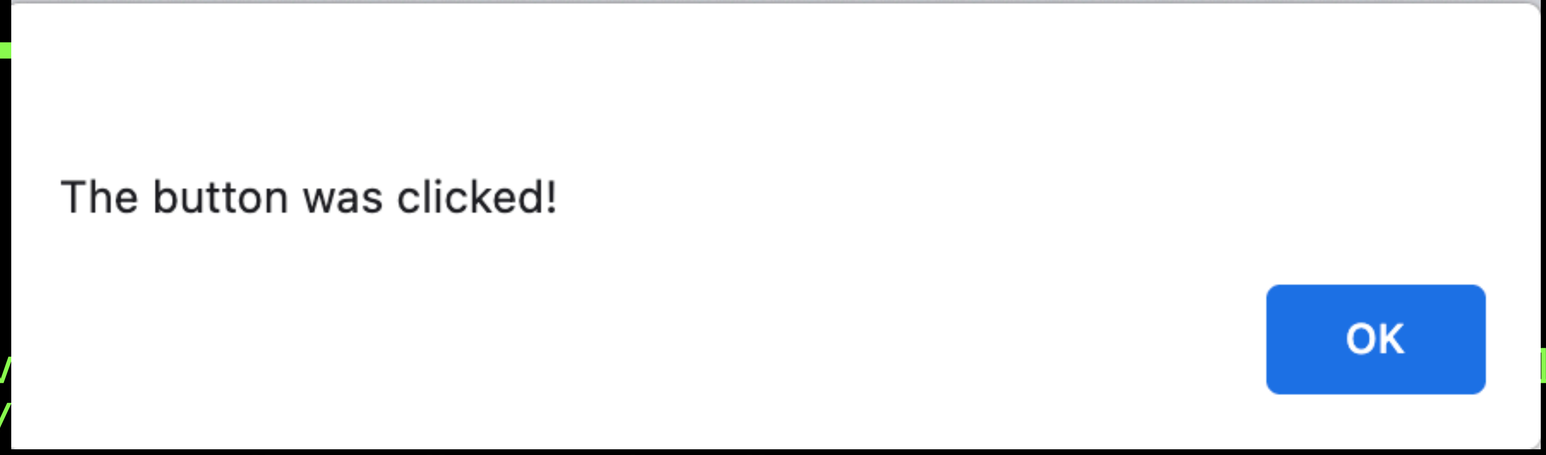
TARGETING an element

ATTACHING the event

Click here!

<SCRIPT

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="v                                                    l">
    <link href="/
    <link href="setup.css" rel="stylesheet">
  </head>
  <body>
    <button id="example">Click here!</button>
    <script>
      // Set up a variable for our button!
      let button = document.querySelector('#example');
    // Any CSS selector.

      button.onclick = () => { // Attach the click event.
        alert('The button was clicked!'); // Pop an alert!
      };
    </script>
  </body>
</html>
```



A TINY CSS NOTE

To indicate interaction…

```
button{
   cursor: pointer;
}
```

EXTERNAL .JS FILES

This is the most common, most flexible, way to include JS. Like linking CSS, it lives in the head.

EXTERNAL .JS FILES

The tag stays empty, but does need to be closed.

```
<script defer src="script.js"></script>
```

defer means that the webpage will read all of the HTML before going to implement your script

(any name)

EXTERNAL .JS FILES

You can link one or many JS files, but this keeps your various files legible and all in the same code syntax

WHAT CAN IN-BROWSER JS DO?

JavaScript doesn’t interact with a computer’s memory—so it can’t, for instance, save to disc, but it is a powerful tool forwebpage manipulation, interaction with the user, and with the webserver.

WHAT CAN IN-BROWSER JS DO?

It can:

- Add HTML to a page, changing content or styles
- React to user actions such as mouse clicks
- Download and upload files (interact w server)
- Get and set cookies or send messages to users
- Remember data in local storage

WHAT CAN'T IN-BROWSER JS DO?

It can't:

- Read/Write arbitrary files on the disk, copy them, or execute programs. It cannot interact w the Operating System
- Access other tabs that are open
- Receive information in a meaningful way from other domains

USE CASES: CLASSES

One way you may want to use javascript is to add or remove (toggle) a class from an element—triggering a transition or some sort of change in style or position.

USE CASES: CLASSES

Like with transitions, you need to define both states in your CSS.

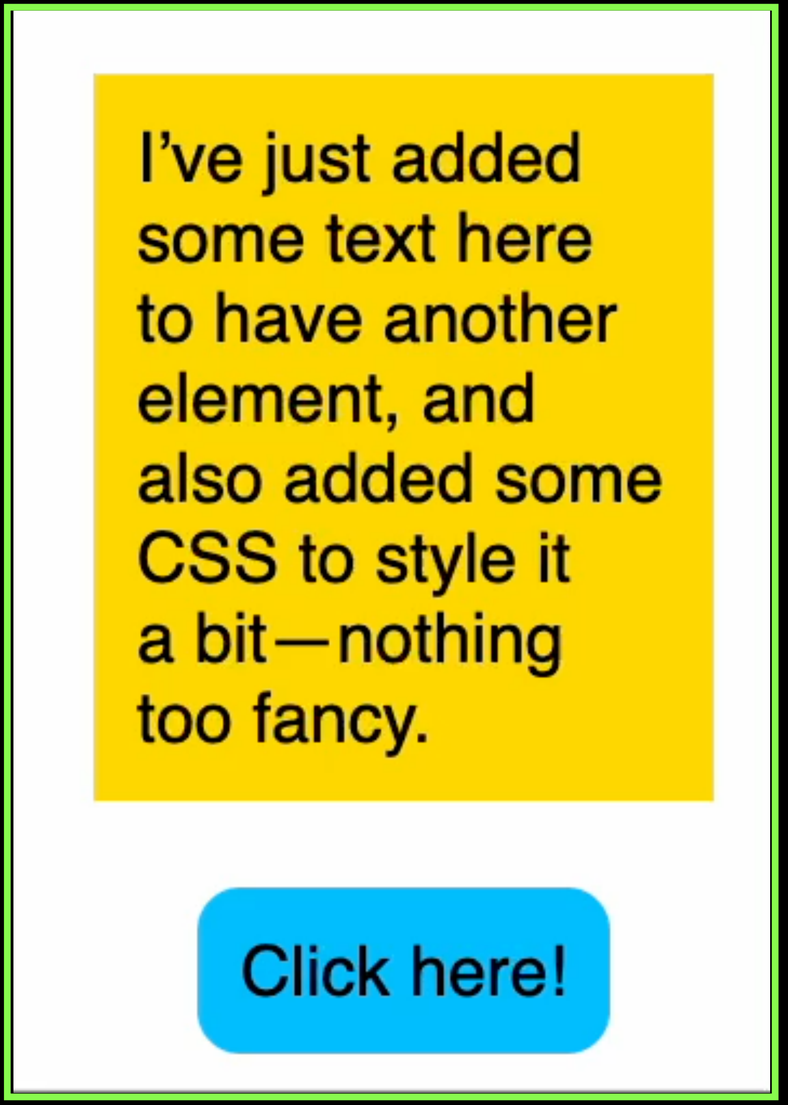
USE CASES: CLASSES

```
<!DOCTYPE html>
<html>
  <head>
    <meta name=“viewport"
    content="width=device-width,
    initial-scale=1">
    <link href="/assets/styles/
    reset.css" rel="stylesheet">
    <link href="setup.css"
    rel="stylesheet">
    <link href="style.css"
    rel="stylesheet">
    <script defer src="script.js"></
    script>
  </head>
  <body>
    <section>
      <p>I’ve just added some text
      here to have another element,
      and also added some CSS to
      style it a bit—nothing too
      fancy.</p>
    </section>
    <button id="example">Click here!
    </button>
  </body>
</html>
```

```
let highlightClass = 'highlight';
// Set up variables.

let textBlock =
document.querySelector('section');
// Any selector.

let switchButton =
document.querySelector('#example');

switchButton.onclick = () => {
// Attach the event.
  textBlock.classList.toggle(highlig
  htClass);
// Toggle the class!
};
```

```
section { /* Initial state, without
the class. */
  background-color: gold;
  padding: calc(var(--base) / 2);
  transition: all 1s ease-in-out;
/* Transition everything. */
}

section.highlight { /* When the class
is applied with JS! */
  background-color: aquamarine;
  transform: rotate(5deg);
}
```

USE CASES: CLASSES




```
let highlightClass = 'highlight';
// Set up variables.

let textBlock =
document.querySelector('section');
// Any selector.

let switchButton =
document.querySelector('#example');

switchButton.onclick = () => {
// Attach the event.
  textBlock.classList.toggle(highlig
  htClass);
// Toggle the class!
};
```

```
section { /* Initial state, without
the class. */
  background-color: gold;
  padding: calc(var(--base) / 2);
  transition: all 1s ease-in-out;
/* Transition everything. */
}

section.highlight { /* When the class
is applied with JS! */
  background-color: aquamarine;
  transform: rotate(5deg);
}
```

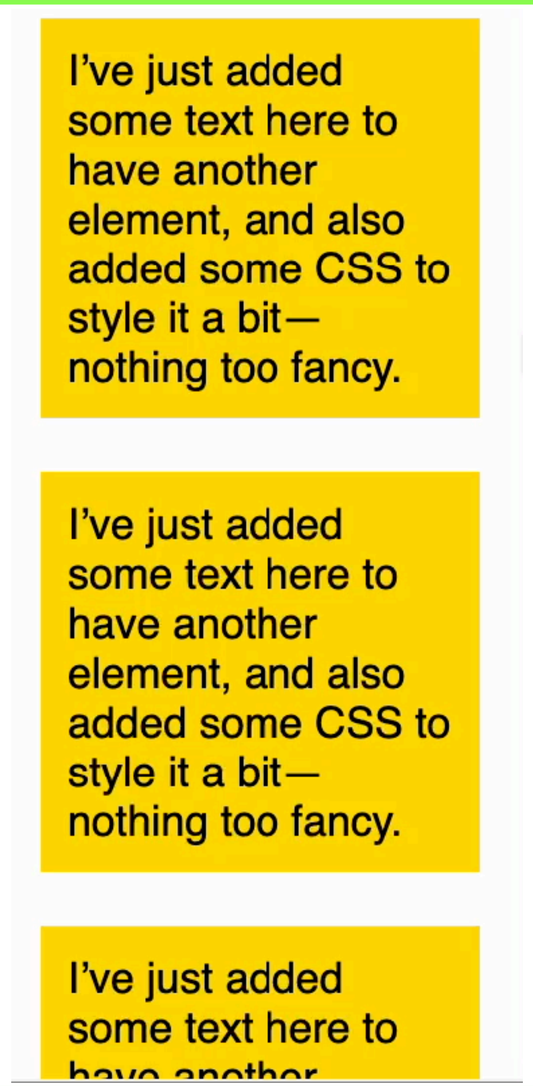
USE CASES: CLASSES

This can be used on any element in your HTML—and if you use classList.add or classList.remove, you can make it a one time toggle.

USE CASES: SCROLLING

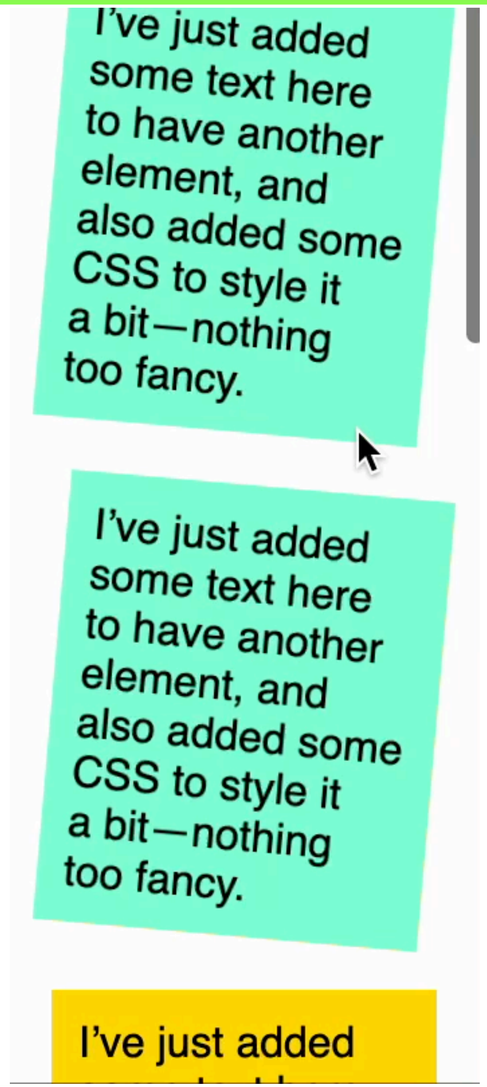
You can also use JavaScript to do something when an element enters/exits the viewport. In this case, the user's scrolling would trigger the toggle.

USE CASES: SCROLLING




```
let highlightClass = 'highlight'; // Set up variables.
let highlightBlock = document.querySelector('section:nth-child(4)');

// Set up an IntersectionObserver.
let sectionObserver = new IntersectionObserver((entries) => {
  let [entry] = entries; // Don't worry about this, for now.

  // When it is intersecting, apply the class; otherwise, remove it.
  if (entry.isIntersecting) {
    highlightBlock.classList.add(highlightClass);
  } else {
    highlightBlock.classList.remove(highlightClass);
  }
});

sectionObserver.observe(highlightBlock); // Watch for it!
```

USE CASES: SCROLLING




```
let highlightClass = 'highlight'; // Set up variables.
let highlightBlock = document.querySelectorAll('section');

// Loop through the list, doing this `forEach` one.
highlightBlocks.forEach((block) => {
  let sectionObserver = new IntersectionObserver((entries) => {
    let [entry] = entries;

    // This is a "ternary" operator——a condensed if/else.
    entry.isIntersecting ? block.classList.add(highlightClass)
    : block.classList.remove(highlightClass);
  }, {
    root: document, // This is only needed in the example iframe!
    rootMargin: '-33% 0% -33% 0%', // CSS-ish: top/right/bottom/left.
  });

  sectionObserver.observe(block); // Watch each one!
});
```

SOME VOCABULARY

Core built in types

Boolean
Number
String
Object
Array
Function

Punctuation

{
}
(
)
;
,

Arrays

a[]
a[0]
length

Objects

obj{}
obj.key
new
class
constructor()

Operations on numbers

+
-
*
/
%
++
--

Comparisons

<
<=
>
>=
===
!==

Assignment

=

Math methods

Math.min()
Math.max()
Math.round()
Math.PI

Functions

a = () => {}
function a() {}
a()
return

Operations on strings

+
""
``
`${}`
.length
.charAt()

Boolean operations

true
false
!
&&
||

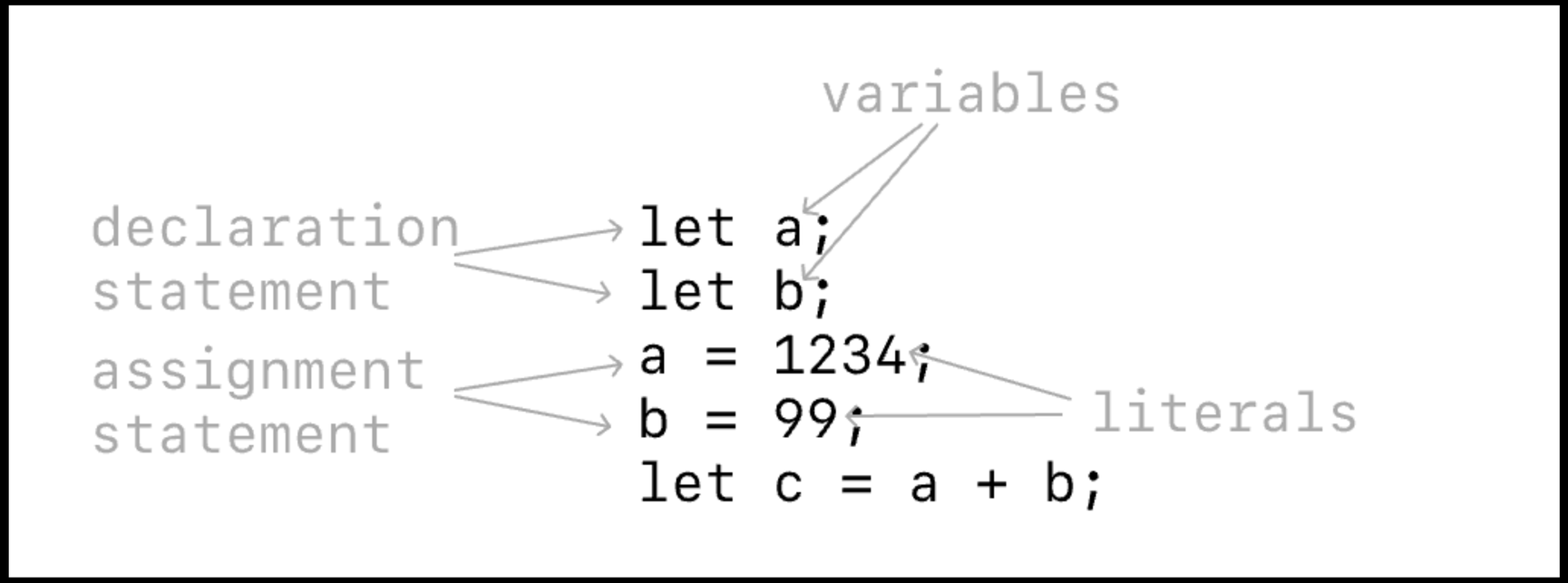
Flow control

if
else
else if
for
while

Built in methods

console.log()
querySelector()
addEventListener()

VARIABLES


variables
declaration statement
assignment statement
let a;
let b;
a = 1234;
b = 99;
let c = a + b;
literals

IF ELSE

```
let flip = Math.random();

if (flip < 0.5) {
  console.log('Heads!');
} else {
  console.log('Tails!');
}
```

Let’s code together.

HOMEWORK

- Harmonic Collection Entries 6/7 due next class.
- Find an interesting piece of javascript—that you may want to use or are intrigued by—and try and figure out how it works. You can either summarize its function, or mark up the actual code with comments. We'll be talking about these next week.
- OPTIONAL: Dig into https://javascript.info/ for in-depth JS tutorials

SESSION NINE
OCTOBER 24, 2023

THANK YOU

DAVIS SCHERER
DSCHERER@NEWSCHOOL.EDU