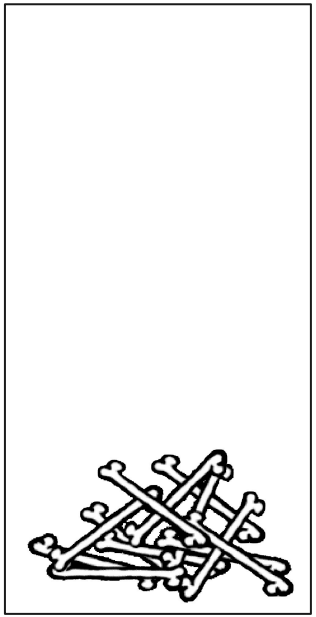
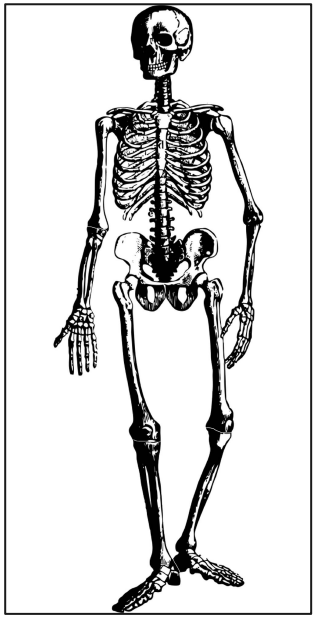
JavaScript

Lecture 5

So What is a Web Page Really?



CONTENT



STRUCTURE

STYLE

BEHAVIOR

Why do we talk about these things separately?

Separation of Concerns

- This is a concept from Software Engineering that every part of a program should address a separate "concern".
- In web programming, we define those concerns as:
  - Content (words, images)
  - Structure/Meaning (HTML)
  - Style/Appearance (CSS)
  - Behavior
- What happens if we don't separate them?
  - https://repl.it/@afitzg/UnseparateConcerns#index.html

What we've learned so far

- How to write content for a webpage using `HTML5`
- How to add styles to a webpage using `CSS` and linking a `CSS` file to an `HTML` file

CSC 196W Modules

1. Web page structure and appearance with HTML5 and CSS.
2. **Client-side interactivity with JS DOM and events.**
3. Using web services (API's) as a client with JS.
4. Writing JSON-based web services with a server-side language.
5. Storing and retrieving information in a database with SQL and server-side programs.

JavaScript is to Java as ...

Grapefruit → Grape
Carpet → Car
Hamster → Ham
Catfish → Cat
Horse → Horseradish

What is JavaScript?

- A lightweight "scripting" programming language
- Created in 1995 by Brendan Eich (original prototype created in 10 days and called LiveScript)
- NOT related to Java other than name and some syntactic similarities...
- Used to define interactivity for web pages.

Why JavaScript and not another language?

Popularity.

The early web browsers supported it as a lightweight and flexible way to add interactivity to pages.

Microsoft created their own version, called JScript, but the open source browsers (notably Firefox and Chrome) put all their effort into JavaScript.

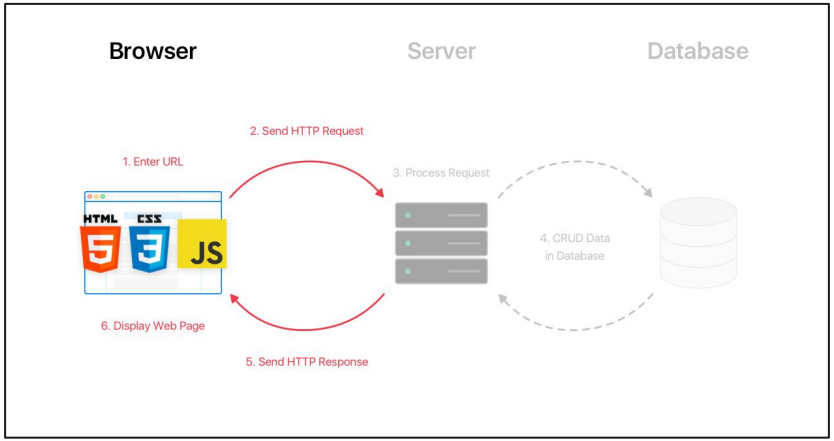
Now: If you want to run anything other than JavaScript in the browser… it's Very Hard™ (often impossible)

Web page Behavior with JavaScript

Now that we know how to add content and styles to a web page, let's explore how to add responsive behavior

We'll use these building blocks to dynamically update what you see on a web page in response to clicks, text input, timers, etc.

Terminology: Client-Side Scripting



Client-side script: Code that runs on the user's computer and does not need a server to run (just a browser!).
Client-side JavaScript is usually run after HTML and CSS have been loaded on the browser (e.g. from a server response).

Often, this JavaScript manipulates the page or responds to user actions through "event handlers".

JS: Adding Behavior to HTML/CSS

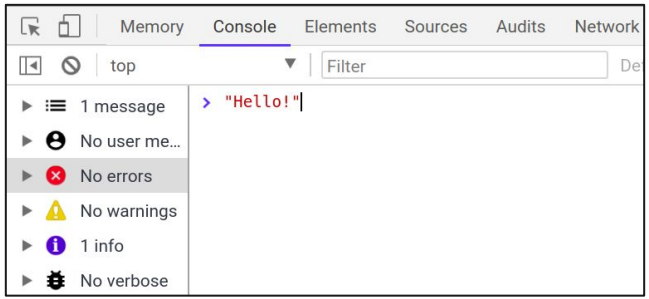
We can use write JavaScript functions to...

- Insert dynamic text into HTML (ex: username)
- React to events (ex: page load, user's mouse click)
- Get information about a user's computer (ex: what browser they are using)

Today: Following Along

As an interpreted programming language, JS is great to interact with a line at a time (similar to Python, but very different than Java). Where do you start?

The easiest way to dive in is with the Chrome browser's Console tab in the same inspector tool you've used to inspect your HTML/CSS.



Until we learn how to interact with the HTML DOM with JS, we recommend experimenting with the following code examples using this console to get comfortable with the basic syntax and behavior.

Our First JavaScript Statement: `console.log`

Used to output values to the browser console, most often used to debug JS programs. You can think of this as `System.out.println` in Java or `print` in Python.

```
console.log(“message”);
```

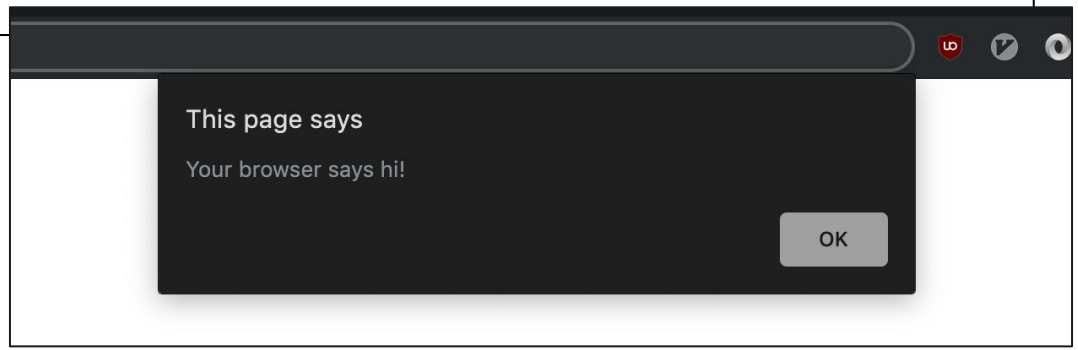
```
console.log(“The answer is: ” + 42);
```

The `alert` function

A JS function that pops up a dialog box with a message - not ideal in practice, but sometimes a recommended debugging tool when first learning JS. **Don't include alert statements in any of your assignments.**

```
alert(“message”);
```

```
alert("Your browser says hi!");
```



Comments (similar to Java)

```
// single-line comment

/**
* multi-line
* comment
*/
```

Identical to Java's comment syntax

Recall: 3 comment syntaxes

- `HTML: <!-- comment -->`
- `CSS/Java/JS: /* comment */`
- `Java/JS: // comment`

Variables

```
// template
let name = expression;

// examples
let level = 23;
let accuracyRate = 0.99;
let name = "Pikachu";
```

Variables are declared with the let keyword (case-sensitive). You may also see `var` used instead of `let` - this is an older convention with weaker scope - **DO NOT USE** `var` anywhere

Question: What does weaker scope mean? Why is this a "bad" thing?

CQG: Use camelCasing for variable (and function) names

“Types” in JavaScript

```
let level = 23; // Number
let accuracyRate = 0.99; // Number
let name = "Pikachu"; // String
let temps = [55, 60, 57.5]; // Array
```

Types are not specified, but JS does have types ("loosely-typed")

- `Number, Boolean, String, Array, Object, Function, Null, Undefined`
- Can find out a variable's type by calling typeof, but usually this is poor practice (why?)
- Note: Type conversion isn't always what you expect...

A Note about Declaring Types in JavaScript

If you've programmed in a statically-typed language like Java, you will recall that when declaring variables, you must specify their type which **must** always stay the same.

```
boolean isValid = "hello!"; // error
```

In a dynamically-typed language like JavaScript, you don't need to specify the type (just use `let` or `const`) and you may change the type the variable refers to later in execution.

```
let isValid = true; // no error
isValid = "hello!";
isValid = 1;
```

In a dynamically-typed language like JavaScript, you don't need to specify the type (just use let or const) and you may change the type the variable refers to later in execution.

Number Type

```
let enrollment = 99;
let medianGrade = 2.8;
let credits = 5 + 4 + (2 * 3);
```

- Integers and real numbers are the same type (no `int` vs. `double`). All numbers in JS are floating point numbers.
- Same operators: `+ - * / % ++ -- = += -= *= /= %=` and similar precedence to Java.
- Many operators auto-convert types: `"2" * 3` is `6`
- `NaN` ("Not a Number") is a return value from operations that have an undefined numerical result (e.g. dividing a String by a Number).

Practice!

String type

```
let nickName = "Sparky O'Sparkz";                    // "Sparky O'Sparks"
let fName = nickName.substring(0, s.indexOf(" ")); // "Sparky"
let len = nickName.length;                           // 15
let name = 'Pikachu';                                // can use "" or ''
```

Methods: charAt, charCodeAt, fromCharCode, indexOf, lastIndexOf, replace, split, substring, toLowerCase, toUpperCase

More about Strings

Escape sequences behave as in Java: `\' \" \& \n \t \\`
To convert between Numbers and Strings:

```
let count = 10;                                 // 10
let stringedCount = "" + count;                 // "10"
let puppyCount = count + " puppies, yay!";      // "10 puppies, yay!"
let magicNum = parseInt("42 is the answer");    // 42
let mystery = parseFloat("Am I a number?");     // NaN
```

To access characters of a String s, use s[index] or s.charAt(index):

```
let firstLetter  = puppyCount[0];                                  // "1"
let fourthLetter = puppyCount.charAt(3);                           // "p"
let lastLetter   = puppyCount.charAt(puppyCount.length - 1); // "!"
```

Common Bugs when Using Strings

While Strings in JS are fairly similar to those you'd use in Java, there are a few special cases that you should be aware of.

- Remember that `length` is a property (not a method, as it is in Java)
- Concatenation with +: `1 + 1` is `2`, but `"1" + 1` and `1 + "1"` are both `"11"`!

Practice: `repeat`

Special Values: `null` and `undefined`.

```
let foo = null;
let bar = 9;
let baz;

/* At this point in the code,
* foo is null
* bar is 9
* baz is undefined
*/
```

`undefined`: declared but has not yet been assigned a value
`null`: exists, but was specifically assigned an empty value or `null`. Expresses intentional a lack of identification.
A good motivating overview of `null` vs. `undefined`

Note: This takes some time to get used to, and remember this slide if you get confused later.

Arrays

```
let name = [];                          // empty array
let names = [value, value, ..., value]; // pre-filled
names[index] = value;                   // store element
```

```
let types = ["Electric", "Water", "Fire"];
let pokemon = [];         // []
pokemon[0] = "Pikachu";   // ["Pikachu"]
pokemon[1] = "Squirtle"; // ["Pikachu", "Sqiurtle"]
pokemon[3] = "Magikarp"; // ["Pikachu", "Sqiurtle", undefined, "Magikarp"]
pokemon[3] = "Gyarados"; // ["Pikachu", "Sqiurtle", undefined, "Gyarados"]
```

Two ways to initialize an array
`length` property (grows as needed when elements are added)

Some Notes on Typing

As you write JS programs, you may run into some silent bugs resulting from odd typing behavior in JS. Automatic type conversion, or coercion, is a common, often perplexing, source of JS bugs (even for experienced JS programmers).

Why is this important to be aware of? You'll be writing programs which use variables and conditional logic. Knowing what is considered truthy/false and how types are evaluated (at a high level) can make you a much happier JS developer (some practice)

Examples of some "not-so-obvious" evaluations:

```
2 < 1 < 2;// true
0 + "1" + 2;// "012"
[] + [];// ""
"1" / null;// Infinity
[+!![]]+[[![]+!![]]];// "11"
```

This is worth 3 minutes of your viewing pleasure. (starting at 1:20)

Defining Functions

```
// template
function name(params) {
  statement;
  statement;
  ...
  statement;
}

// example
function myFunction() {
  console.log("Hello!");
  alert("Your browser says hi!");
}
```

The above could be the contents of basics.js linked to our HTML page

Statements placed into functions can be evaluated in response to user events

JS Function vs. Java Method

```
function repeat(str, n) {
  let result = str;
  for (let i = 1; i < n; i++) {
    result += str;
  }
  return result;
  }
  let repeatedStr = repeat("echo...", 3); // "echo...echo...echo..."
}
```

```
public static String repeat(String str, int n) {
  String result = str;
  for (int i = 1; i < n; i++) {
    result += str;
  }
  return result;
  }
  String repeatedStr = repeat("echo...", 3); // "echo...echo...echo..."
}
```

JS Function vs. Python Function

```
function repeat(str, n) {
  let result = str;
  for (let i = 1; i < n; i++) {
    result += str;
  }
  return result;
  }
  let repeatedStr = repeat("echo...", 3); // "echo...echo...echo..."
}
```

```
def repeat(str, n):
  result = str;
  for i in range(1, n):
    result = result + str;
  return result;

  repeatedStr = repeat("echo...", 3) // "echo...echo...echo..."
```