CSE 331
Software Design & Implementation

Hal Perkins

Summer 2019

JavaScript Overview

Why?

- We've built up an application that can find walking paths on the campus
- We'd like to add a graphical user interface front-end
- Native Java Swing graphics is straightforward, but…
- These days most things are webapps
  - So we're going to build one! 😱
  - Which means learning enough JavaScript / react to draw the map and paths in a browser window and interact with the user and the back-end campus map application (your Java program!)

JavaScript – our approach

- 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 language basics, particularly key differences between JavaScript and Java
  - If you've already done JavaScript hacking this may be mostly review with some new perspective
    - But also realize our goal isn't to exhaustively cover everything – don't have time, so core ideas only
- Last two assignments this quarter:
  - HW8 draw dots and lines on an image (using JS/react)
  - HW9 – use HW8 framework as the campus path GUI

Resources

- Lectures will (try to) point out key things
- For more: start with Mozilla (MDN) JavaScript tutorial:
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript
- CodeAcademy has a good, free JavaScript basics course
- React has its own dialect of JavaScript (JSX) so we'll selectively use it's documentation
- Be real careful about web searches – the JavaScript/webapp ecosystem has way too many somewhat-to-totally incompatible or current vs. obsolete ways of doing similar things. Code snippets from the web may lead you *way* off.

Credits

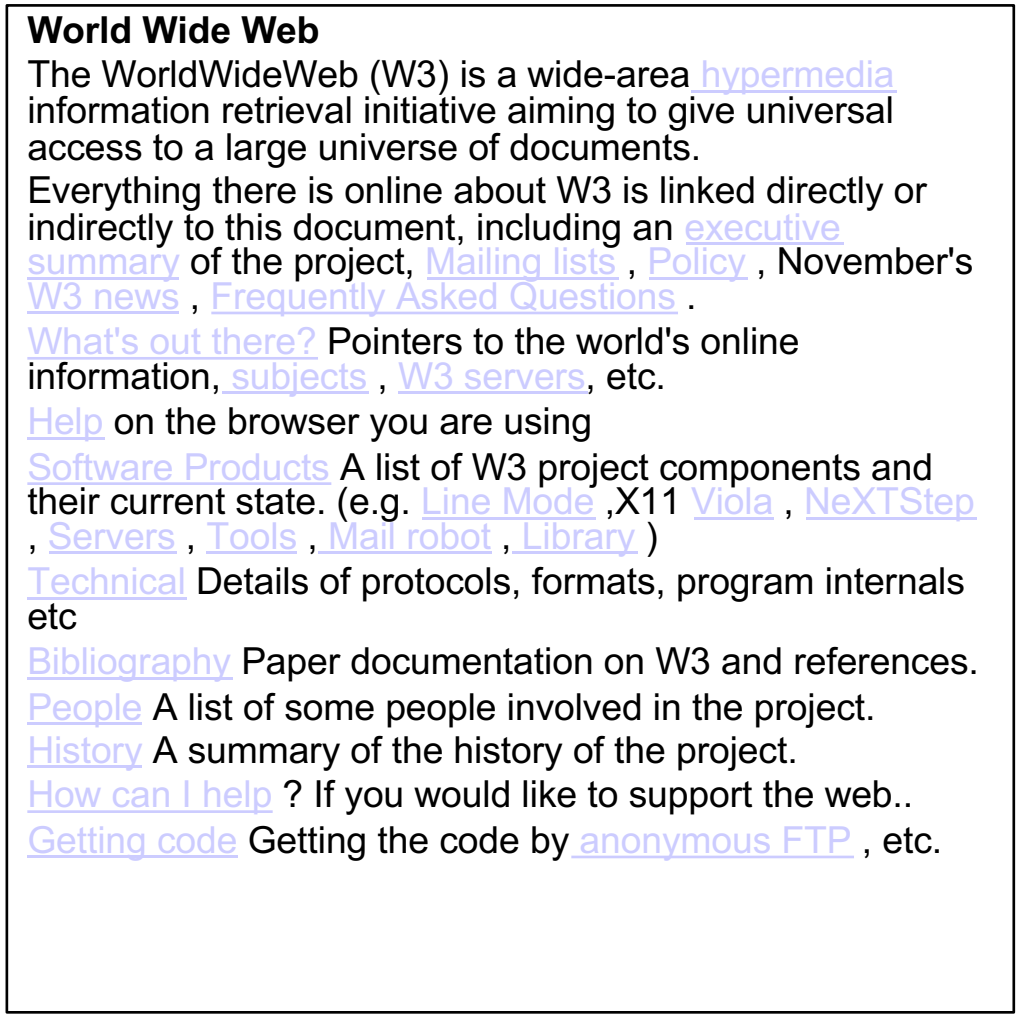
- CSE 331 project due to Andrew Gies and Avi Bhagat
- Thanks to Lauren Bricker and CSE 154 crew for their current notes (even if you took 154 recently this stuff probably will look different)
- Other material from Cay Horstmann's CSE 151 course at San Jose State
- And from wherever we can find useful things…

- Notes: JS = JavaScript. ECMAScript is the official standard version (currently v6) so you'll also see ES or ES6 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 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! (otherwise it would have been Microsoft Basic in Mozilla for scripting)
- Used to make web pages interactive:
  - Dynamic text 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
- We will try to stick to portable, generic stuff
  - Some of our libraries depend on latest version of Javascript ("ECMAScript 6"), which is supported on current versions of all major browsers
  - But for hw8/hw9 we're only supporting Chrome (at least this time around) to avoid cross-platform grief

Our plan…

- First, look at basic JavaScript language elements using Chrome as an execution engine
  - And look at how plain JS interacts with ordinary web pages
- Then…
  - The original web model was "pages linked to other pages", with some computation in individual pages
  - Modern web apps load a single page that serves as a host for running a full application program
    - Something browsers+JS were never designed for
- So…
  - There are a huge number of web/JS frameworks to make this sort of application feasible
    - We'll learn basics of React and use it for our project

A first example – embedded JS

- Originally JavaScript code was embedded in web pages. Code is executed when <script> is encountered as web page loads.
  - (all sample code is linked to the lecture on the cse331 web)

```
<html>
<head>
<title>JS Program embedded in a web page</title>
<script type="text/javascript">
    let a = 6
    let b = 7
    alert(a*b)
</script>
</head>
<body></body>
</html>
```

embedded.html

External JS files

- More commonly, JavaScript code is stored in separate files. Sample code file contents:

```
let a = 6
let b = 7
alert(a*b)
```

external.js

- The web page that uses (runs) it when the page is loaded:

```
<html>
<head>
<title>JS Program loaded
from external file</title>
<script src="external.js">
</script>
</head>
<body></body>
</html>
```

external.html

JavaScript console

Every browser has developer tools including the console, details about web pages and objects, etc.

A JS program can use `console.log("message");` to write a message to the console for debugging, recording, etc.

- "printf debugging" for JavaScript programs

In Chrome, right-click on a web page and select Inspect or pick View > Developer > Developer Tools from the menu. Click the console tab and you can see output that's been written there, plus you can enter JavaScript expressions and evaluate them. Super useful for trying things out.

console.html

Syntax and variables

- Syntax similar to Java, C, etc.
- `/*` comments `*/` or `//` comments (prefer `//`)
- Variables are "dynamically typed" (i.e., not fixed by declaration):
  - Introduced into program with **`let`**
  - Type of a var is whatever was last assigned to it

```
let x = 42;
x = "ima string now!";
```

  - Use **`const`** for constants: **`const pi = 3.14159;`**
- Semicolons are optional at ends of lines and often omitted, but also encouraged ☺

Values and types

- Primitive values do have types. 5 of them:
  - Number (floating-point only), String, Boolean, Undefined, and Null (actually an object type, but usually thought of as its own type with literal null)
- Usual numeric operations: `+ - * / ++ -- +=`, etc.
- String concatenation with `+`
- Lots of methods for strings
- Math library with the expected things

- Variables can also refer to values of object types

Control flow –just like Java

- Conditionals

```
if (condition) {
 statements
} else if (condition){
 statements
} else {
 statements
}
```

- Loops

```
while (condition) {
  statements
}

for (init; condition; update){
  statements
}
```

  - Also for-of and for-in loops
    - Be careful with these. They have "interesting" semantics and differences that you want to get right.

Boolean type

- Any value can be used as a Boolean
  - "falsey" values: **`false`**, **`0`**, **`NaN`**, "", **`null`**, **`undefined`**
  - "truthy" values: everything else (including **`true`** !)
- As expected: `>`, `<`, `<=`, `>=`, **`&&`**, **`||`**, **`!`**
- Equality `===`, `!==` (strictly equal); `==`, `!=` (loosely equal)
  - Use `===`, `!==` almost all the time. These check both types and values.
  - `==` and `!=` can surprise you with conversions
    - Try `7=="7"` vs `7==="7"`

Arrays

- Examples

```
let empty = [ ]
let names = [ "bart", "lisa" ]
let stuff = [ "wookie", 17, false ]
stuff[6] = 331   // in-between undefined
```

- Access elements with subscripts as usual
- Lots of methods – arrays can serve as lists, stacks, queues, etc.
  - length, add, concat, reverse, shift, push, pop, …

Functions (& methods!)

- JavaScript has both – like C++
  - Methods – called on objects (more later)
  - Functions – free-standing things
- Syntax is the same for both

```
function name (parameters) {
  statement
  statement
}
```

  - Use `return` *value;* statement if function should compute and yield a value on exit

Remember dynamic typing?

- Consider

```
function average(x, y) {
  return (x + y) / 2
}
```

- No surprise

```
let result = average(6,7);     // 6.5
```

- But then…

```
let answer = average('6','7'); // 33.5!
```

Functions are values

- Javascript is a functional language
  - Functions can be stored as values of variables, passed as parameters, and so on
  - Lots of powerful techniques (cf CSE 341) we won't cover for the most part
- Using the average function from the previous page:

```
let f = average

let result = f(6, 7)  // result is 6.5

f = Math.max

result = f(6,7)       // result is 7
```

Arrow functions

- A function can be defined without the function keyword. Here is a different (but works the same*) way to define average:

```
let average = (x, y) => (x + y) / 2
```

  - Idea is that the => arrow separates the parameter list from the returned value
    - **=>** can be used with multi-line functions also – see tutorials/references for details
  - Very useful when we need a "function value" to be used as an argument to some other function
    - We will see more later when we deal with user interface event handling

*There is a technical difference in how global variables (particularly **`this`**) are handled in **=>** functions compared to conventional named ones. We'll need to use **=>** in our React code because of this.

Functions as parameters

- Functions can be passed as parameters just like any other values

```
function compute(f) {
  return f(2,3);
}
function add(x,y) { return x+y; }
let sub = (x,y) => x-y;
compute(add);
compute(sub);
compute((a,b) => a*b);
```

Objects (1)

- Everything in JavaScript is a mutable object if it's not a primitive value (and some immutable values like strings also have methods)
- A JavaScript object is a set of name/value pairs (often called "properties"). Example

```
character = { name: "Lisa Simpson",
              age: 30 }
```

- Once you have an object, you reference properties with *object.property* notation:

```
character.age = 7
```

Objects (2)

- You can add properties

```
character.instrument = "horn"
```

- Bracket notation can be used to reference properties

```
character["instrument"] = "saxophone"
```

- And properties can be deleted

```
delete character.age
```

- Property names can be computed

```
what = "instrument"
character[what] = "tenor sax"
```

- See tutorials or references for more variations

Objects with methods

- Properties in a JavaScript object can include methods (functions) – just like in Java

```
let account = {
  owner: "Gandalf",
  balance: 10000,
  deposit: function(amount) {
    this.balance += amount
  }
}
```

- We call methods in the expected way:

```
account.deposit(100);
```

Objects with methods

- There is a bit of shorthand available. Instead of

```
let account = {
  …
  deposit: function(amount) {
    this.balance += amount;
  }
}
```

we can write

```
let account = {
  …
  deposit(amount) {
    this.balance += amount
  }
}
```

but the meaning is exactly the same

Creating new objects

- JavaScript has an unconventional model: it is an object-oriented language, but there are *no classes*!
  - But everything* is an object, including functions(!)

    *modulo some technical details and primitive types

- JavaScript's basic model is that objects are created by functions that return new objects…
  - All objects are related to some other object by their hidden "prototype" property
  - When we look for a property in an object, if it is not found locally, we look in its prototype object, and if not found, in that object's prototype, … until we either find it or hit the top of the chain
  - So we have something that resembles inheritance but without classes

ES6 Classes

- All of this is a bit much, so ES6 added syntax for "classes"
- ```
  class Account {
    constructor(owner, balance) {
      this.owner = owner
      this.balance = balance
    }
    deposit(amount) {
      this.balance += amount
    }
  }
  ```
- But underneath there are only objects with prototypes pointing to other objects
  - We'll ignore the details – see a good JS reference
  ```

What's next?

- How JavaScript code interacts with html elements in a web page
- Then some React basics and how to structure a JavaScript application for hw8 and hw9
- Don't miss upcoming sections and lectures!