COMPSCI 326 - Web Programming Intro/Refresher - JavaScript

join on the Slack #general channel as well as Zoom
please **turn on your webcam** if you can
mute at all times when you aren't asking a question

(intro first) emeryberger.com, plasma-umass.org, csrankings.org

Course overview:

This is a course in web programming, with an emphasis on both parts (especially *programming*). We'll be covering:

- JavaScript (browser-side and server-side), in some depth
- Concepts like asynchrony, authentication, caching, client-server architectures, concurrency, consistency, some data management, security, and more
- The browser and its object model (the DOM), and its interactions with HTML, CSS, and JavaScript, and AJAX

There will be in-class and out-of-class assignments and one major group project.

Course mechanics:

We will be using Slack, Piazza, and Zoom (with Google Meet as a backup). I'm going to lecture and you can follow along in my notes in Google Docs (should be readable for everyone, and won't require much bandwidth).

Background resources:

(lots of details)
let - JavaScript | MDN
JavaScript data types and data structures - JavaScript | MDN
Object - JavaScript | MDN
Functions - JavaScript | MDN

Today: JavaScript

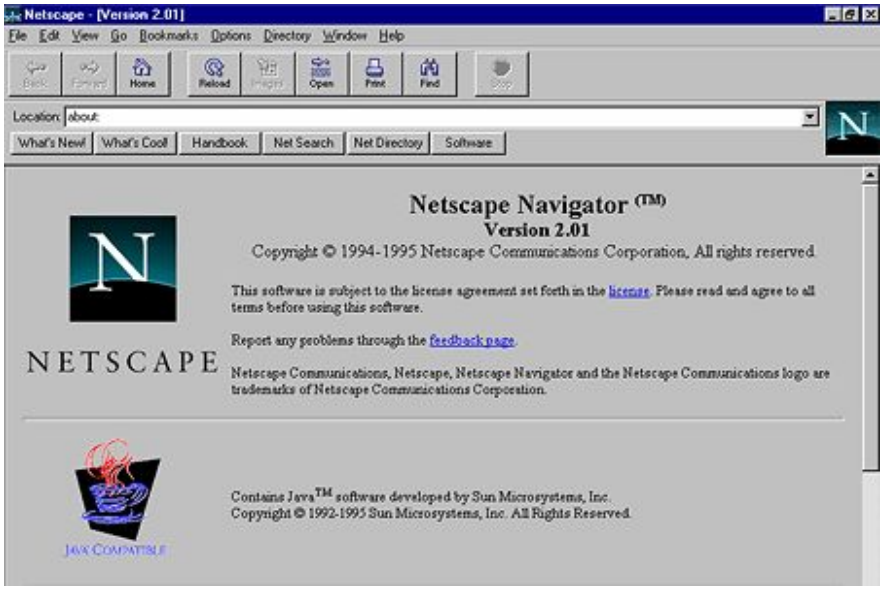
(JavaScript is mostly THE BAD PARTS; we'll try to avoid them)

50% of the class reports high familiarity with JavaScript (we'll see!), but 73% of the class reports high familiarity with Java

- introduce JavaScript by comparing to Java

In fact, JavaScript designed to be "like Java" in some way

- eventually licensed name from Sun, creator of Java
- created in 1995, for Netscape 2.0 (very early browser)



- in 10 days!
- why?
    - Java "applets"
    - various problems
- designer - Brendan Eich - originally wanted to make a language based on Scheme (a kind of LISP, a functional language)
- but had to be "like Java"

    - syntax of Java, curly braces and so on
    - result: it's kind of a mix, and is in fact based on a language called Self

JavaScript now native language of browsers

- now *also* on servers ("Node.js")

Java

- each statement ends in a semicolon
- objects, classes
- TYPES
    - "statically" typed int x; String s;
    - "boxes" have a type that never changes ("static")

[int: ____]

    - only things of that "type" can be in that box
    - int box only can hold ints, float box only holds floats, booleans…
    - variable declarations typed (or *inferred*, but that's just a convenience)
        - `String msg = "hey";`
        - `var msg = "hey";`
          // inferred type, but really it's the same as above
- SCOPE
    - "lexically" or "block" scoped
    - in braces
    - move ever outwards (which x?)

```
int x;
{
  int x;
  {
     …. x ...
  }
  … x …
}
```

JavaScript - it's improving! still lots of bad parts

- "The Good Parts" --> you should use TypeScript instead, which we will be doing later (checks lots of errors)
- looks like Java, but with less "stuff"
- can run JavaScript **directly in your browser**
    - In Chrome: **View → Developer → JavaScript Console**
- or at the command-line with **node** (from Terminal (Mac) or Command Prompt or Cmder (Windows))

```
/**
  print a string a certain number of times, prefaced by the number.
  @arg {string} str - The string to be printed.
  @arg {number} numTimes - The number of iterations.

  example:
    printN("a", 3) →
      0: a
      1: a
      2: a
**/
function printN(str, numTimes) {
  const prefaceStr = 'woo';
  for (let i = 0; i < numTimes; i++) {
    console.log(woo + ': ' + i + ': ' + str);
  }
}
printN('hello', 10);
```

- types:
  - **unlike Java**, you don't declare types!
    - just **let** — or, if the variable will never change, **const**
  - all numbers are floats (e.g., 1.2) - no distinction between those and ints
  - strings (" " **or** ' ') → most people use single quotes
    - 'mass' + 'hole' = 'masshole'
  - built-in arrays (like a list)
    - `let a = [1, 2, 3];`
  - **objects** ("dicts", "maps", "associative arrays")

```
> let person = { 'name': 'María Lopez', 'SPIRE': 12345678 };
> person
{name: "María Lopez", SPIRE: 12345678}
> person['name']
"María Lopez"
> person['SPIRE']
12345678
> person['SPIRE'] = 22345678;
> person['year'] = 2;
> person
```

```
{name: "María Lopez", SPIRE: 22345678, year: 2}
> delete person['SPIRE'];
> person
{name: "María Lopez", year: 2}
```

- Possibly bad parts:
  - **dynamic** typing
  - variables are boxes that can hold *anything*
    - `let x = 'hello';` // this is fine - notice, no types
    - `x = 12;` // this is also fine
    - can lead to nasty surprises you only detect when you run your code
      - TypeScript prevents this by letting you declare types:

```
let x : String = 'hello';
x = 'foo'; // this is fine
x = 12; // error TS2322: Type '12' is not assignable to type 'String'.
```

- it gets worse! VERY BAD PARTS
  - "**dynamic type coercion**"
    - automatically converts things *for your convenience*
  - in a way, JavaScript is "best effort" - tries to keep the program running by doing *something*, but that something can be *insane*

```
> "2" + 3

> "2" - 3

> "-1" + 0

> 0 + "1"

> [] + 1 + 2

> [] + [1]

> [] + [1,2]

> [1] + [1,2]
```

```
> [] - 1
```

Wat

- Unlike Java, JavaScript has == and === (!!!). You should, **always** use **===** for equality comparison and !== for inequalities
    - Why? because of dynamic type coercion insanity. === checks types, == doesn't.

```
> 2 == 2

> "2" == 2

> 2 === 2

> "2" === 2

> 2 == "2"

> 2 === "2"
```

- there's another thing you may see for variable declarations: **var**
    - just use **let (or const)**

---
and now, exercises!