JavaScript

SWE 432, Fall 2019

Web Application Development

Review: Course Topics

You are here. 

- JavaScript and Backend development (first half of semester)
    - JavaScript, back-end development, programming models, testing, performance, privacy, security, scalability, deployment, etc.
- Frontend development and user experience design (second half of semester)
    - Templates and data binding, React, user-centered design, user studies, information visualization, visual design, etc.

Today

- Brief history of JavaScript/ECMAScript
- Overview of core syntax and language semantics
- Overview of key libraries
- In class activity working with JavaScript
- Next:
  - Testing and tooling

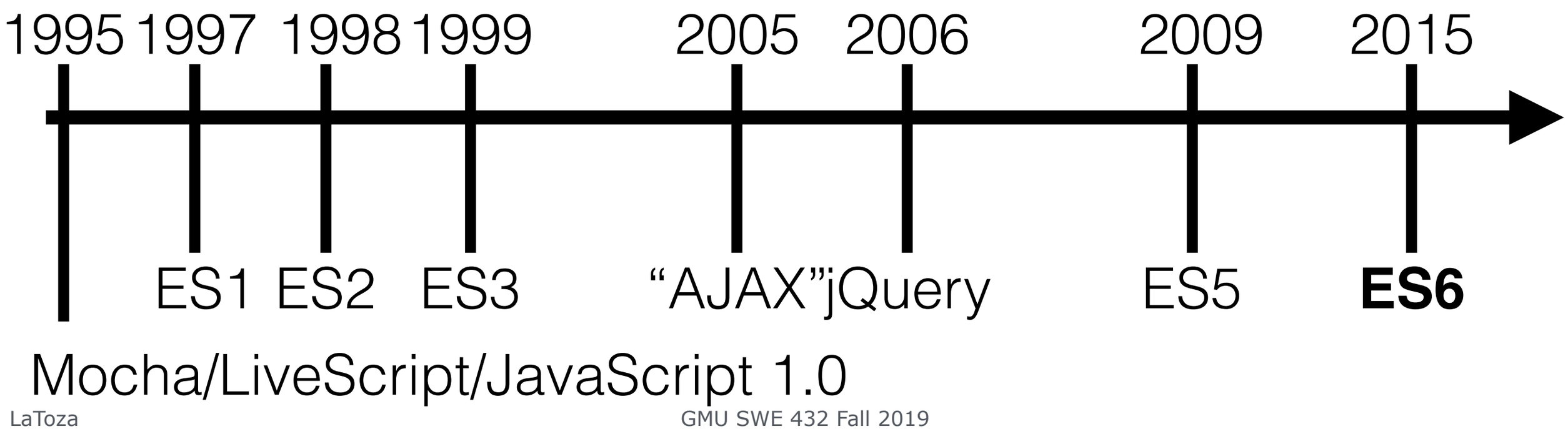
Survey

Go to:
b.socrative.com, Click student login
Room name: SWE432
Student ID: Your G-number (Including the G)

Reminder: Survey can only be completed if you are in class. If you are not in class and do it you will be referred directly to the honor code board, no questions asked, no warning.

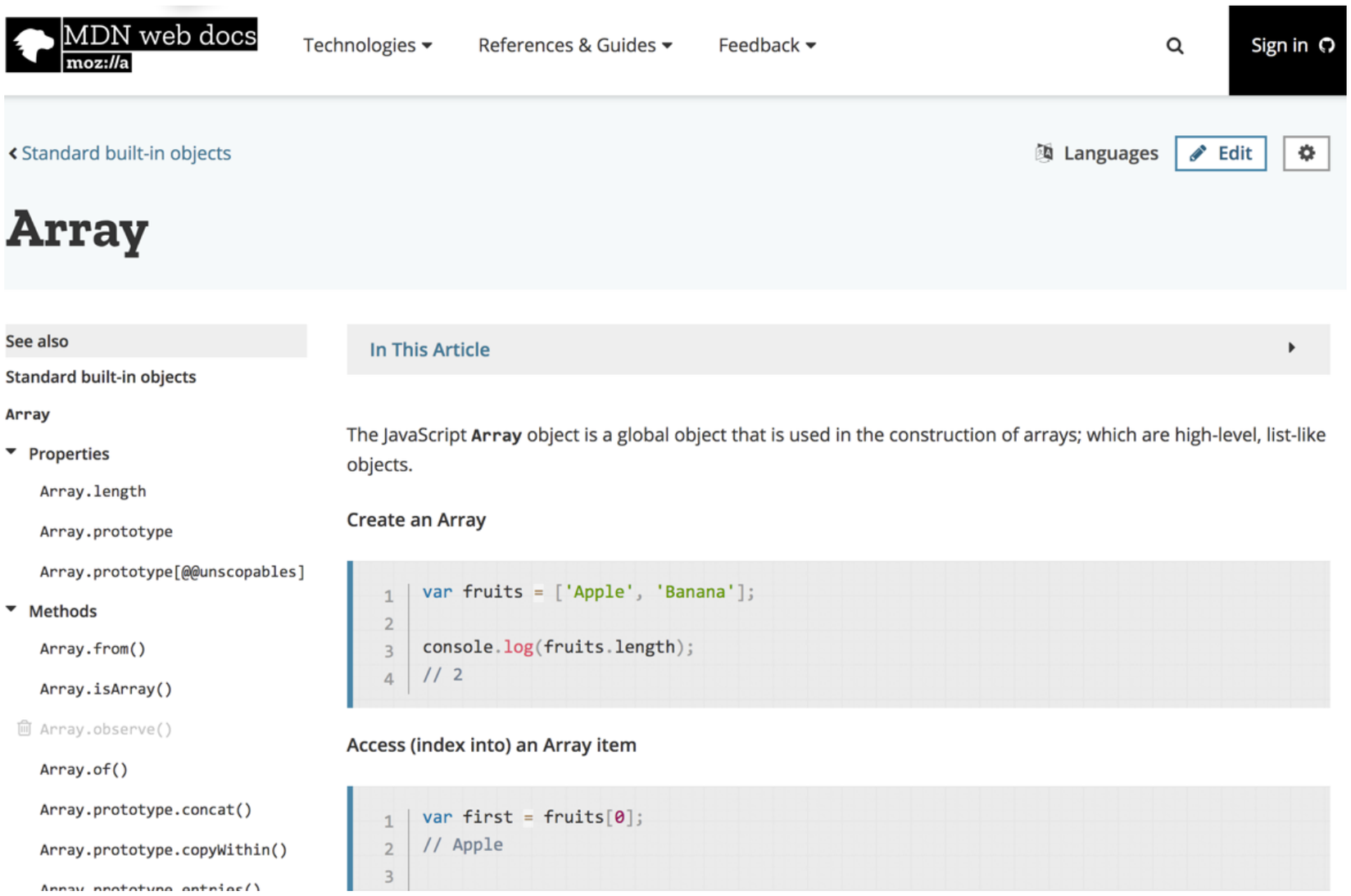
JavaScript: Some History

- JavaScript: 1995 at Netscape (supposedly in only 10 days)
  - No relation to Java (maybe a little syntax, that's all)
  - Naming was marketing ploy
  - ECMAScript -> International standard for the language



Reference materials

- Not any “official” documentation
- Most definitive source for JavaScript, DOM, HTML, CSS: Mozilla Development Network (MDN)
- StackOverflow posts, blogs often have good examples



https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Pastebins

- Code snippet hosted on the web with an in-browser editor
- Used to share code and experiment with small code snippets
- Examples: JSFiddle, JSBin

Variables

- Variables are *loosely* typed
  - String:
    ```
    var strVar = 'Hello';
    ```
  - Number:
    ```
    var num = 10;
    ```
  - Boolean:
    ```
    var bool = true;
    ```
  - Undefined:
    ```
    var undefined;
    ```
  - Null:
    ```
    var nulled = null;
    ```
  - Objects (includes arrays):
    ```
    var intArray = [1,2,3];
    ```
  - Symbols (named magic strings):
    ```
    var sym = Symbol('Description of the symbol');
    ```
  - Functions (We'll get back to this)
- Names start with letters, $ or _
- Case sensitive

Const

- Can define a variable that cannot be assigned again using const

```
const numConst = 10; //numConst can’t be changed
```

- For objects, properties may change, but object identify may not.

More Variables

- Loose typing means that JS figures out the type based on the value

```
let x; //Type: Undefined
x = 2; //Type: Number
x = 'Hi'; //Type: String
```

- Variables defined with let (but not var) have block scope
  - If defined in a function, can only be seen in that function
  - If defined outside of a function, then global. Can also make arbitrary blocks:

```
{
    let a = 3;
}
//a is undefined
```

Loops and Control Structures

- `if` - pretty standard

```
if (myVar >= 35) {
    //...
} else if(myVar >= 25){
    //...
} else {
    //...
}
```

- Also get `while`, `for`, and `break` as you might expect

```
while(myVar > 30){
    //...
}

for(var i = 0; i < myVar; i++){
      //...
      if(someOtherVar == 0)
        break;
}
```

Operators

```
var age = 20;
```

Operator	Meaning	Examples
==	Equality	`age == 20` `age == '20'`
!=	Inequality	`age != 21`
>	Greater than	`age > 19`
>=	Greater or Equal	`age >= 20`
<	Less than	`age < 21`
<=	Less or equal	`age <= 20`
===	Strict equal	`age === 20`
!==	Strict Inequality	`age !== '20'`

Annoying

Functions

- At a high level, syntax should be familiar:

```
function add(num1, num2) {
    return num1 + num2;
}
```

- Calling syntax should be familiar too:

```
var num = add(4,6);
```

- Can also assign functions to variables!

```
var magic = function(num1, num2){
    return num1+num2;
}
var myNum = magic(4,6);
```

- Why might you want to do this?

Default Values

```
function add(num1=10, num2=45) {
    return num1 + num2;
}

var r = add(); // 55
var r = add(40); //85
var r = add(2,4); //6
```

Rest Parameters

```
function add(num1, ... morenums) {
    var ret = num1;
    for(var i = 0; i < morenums.length; i++)
        ret += morenums[i];
    return ret;
}
```

```
add(40,10,20); //70
```

=> Arrow Functions

- Simple syntax to define short functions *inline*
- Several ways to use

Parameters

```
var add = (a,b) => {
            return a+b;
          }
```

```
var add = (a,b) => a+b;
```

If your arrow function only has one expression, JavaScript will automatically add the word "return"

Objects

- What are objects like in other languages? How are they written and organized?
- Traditionally in JS, no *classes*
- Remember - JS is not really typed… if it doesn't care between a number and a string, why care between two kinds of objects?

```
var profHacker = {
    firstName: "Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: "ENGR 6409”,
    fullName: function(){
        return this.firstName + " " + this.lastName;
    }
};
```

Working with Objects

```
var profLaToza = {
    firstName: "Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: "ENGR 6409”,
    fullName: function(){
        return this.firstName + " " + this.lastName;
    }
};
```

Our Object

```
console.log(profHacker.firstName); //Alyssa
console.log(profHacker[“firstName”]); //Alyssa
```

Accessing Fields

```
console.log(profHacker.fullName()); //Alyssa P Hacker
```

Calling Methods

```
console.log(profHacker.fullName);//function...
```

Bind and This

```
var profHacker = {
  firstName: "Alyssa",
  lastName: "P Hacker",
  teaches: "SWE 432",
  office: "ENGR 6409",
  fullName: function(){
     return this.firstName + " " + this.lastName;
  }
};
```

```
var func = profHacker.fullName;
console.log(func())//undefined undefined
```

This occurs because when the function is called, ‘this’ refers to the ‘this’ that calls it (who knows what that is… the file itself?)

Binding This

```
var func = profHacker.fullName.bind(profHacker);
console.log(func()); //Alyssa P Hacker


var ben = {
  firstName: "Ben",
  lastName: "Bitdiddle"
};
var func = profHacker.fullName.bind(ben);
console.log(func()); //Ben Bitdiddle
```

The bind() function lets you pre-set the arguments for a function (starting with what ‘this’ is)

JSON: JavaScript Object Notation

Open standard format for transmitting *data* objects.

- No functions, only key / value pairs
- Values may be other objects or arrays

```
var profHacker = {
    firstName: "Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: “ENGR 6409”,
    fullName: function(){
        return this.firstName + " " + this.lastName;
    }
};
```

Our Object

```
var profHacker = {
    firstName: "Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: "ENGR 6409",
    fullName: {
        firstName: “Alyssa”,
        lastName: “P Hacker”}
};
```

JSON Object

Interacting w/ JSON

- Important functions
- JSON.parse(jsonString)
  - Takes a *String* in JSON format, creates an *Object*
- JSON.stringify(obj)
  - Takes a Javascript *object*, creates a JSON *String*
- Useful for persistence, interacting with files, debugging, etc.
  - e.g., console.log(JSON.stringify(obj));

Arrays

- Syntax similar to C/Java/Ruby/Python etc.
- Because JS is loosely typed, can mix types of elements in an array
- Arrays automatically grow/shrink in size to fit the contents

```
var students = ["Alice", "Bob", "Carol"];
var faculty = [profHacker];
var classMembers = students.concat(faculty);
```

Arrays are actually objects… and come with a bunch of "free" functions

Some Array Functions

- Length
```
var numberOfStudents = students.length;
```
- Join
```
var classMembers = students.concat(faculty);
```
- Sort
```
var sortedStudents = students.sort();
```
- Reverse
```
var backwardsStudents = sortedStudents.reverse();
```
- Map
```
var capitalizedStudents = students.map(x =>
                                  x.toUpperCase());
 //["ALICE","BOB","CAROL"]
```

For Each

- JavaScript offers two constructs for looping over arrays and objects

- For **of** (iterates over values):

```
for(var student of students)
{
    console.log(student);
} //Prints out all student names
```

- For **in** (iterates over keys):

```
for(var prop in profHacker){
    console.log(prop + ": " + profHacker[prop]);
}
```

Output:
firstName: Alyssa
lastName: P Hacker
teaches: SWE 432
office: ENGR 6409

Arrays vs Objects

- Arrays are Objects
- Can access elements of both using syntax

```
var val = array[idx];
```

- Indexes of arrays must be integers
- Don’t find out what happens when you make an array and add an element with a non-integer key :)

String Functions

- Includes many of the same String processing functions as Java
- Some examples
  - var stringVal = ‘George Mason University’;
  - stringVal.endsWith(‘University’) // returns true
  - stringVal.match(….) // matches a regular expression
  - stringVal.split(‘ ‘) // returns three separate words

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

Template Literals

- Enable embedding expressions **inside** strings

```
var a = 5;
var b = 10;
console.log(`Fifteen is ${a + b} and
not ${2 * a + b}.`);
// "Fifteen is 15 and not 20."
```

- Denoted by a back tick grave accent `, **not** a single quote

Set Collection

```
var mySet = new Set();

mySet.add(1); // Set { 1 }
mySet.add(5); // Set { 1, 5 }
mySet.add(5); // Set { 1, 5 }
mySet.add('some text'); // Set { 1, 5, 'some text' }
var o = {a: 1, b: 2};
mySet.add(o);

mySet.add({a: 1, b: 2}); // o is referencing a different object so this is okay

mySet.has(1); // true
mySet.has(3); // false, 3 has not been added to the set
mySet.has(5);              // true
mySet.has(Math.sqrt(25));  // true
mySet.has('Some Text'.toLowerCase()); // true
mySet.has(o); // true

mySet.size; // 5

mySet.delete(5); // removes 5 from the set
mySet.has(5);    // false, 5 has been removed

mySet.size; // 4, we just removed one value
console.log(mySet);// Set {1, "some text", Object {a: 1, b: 2}, Object {a: 1, b: 2}}
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set

Map Collection

```
var myMap = new Map();

var keyString = 'a string',
    keyObj = {},
    keyFunc = function() {};

// setting the values
myMap.set(keyString, "value associated with 'a string'");
myMap.set(keyObj, 'value associated with keyObj');
myMap.set(keyFunc, 'value associated with keyFunc');

myMap.size; // 3

// getting the values
myMap.get(keyString);    // "value associated with 'a string'"
myMap.get(keyObj);       // "value associated with keyObj"
myMap.get(keyFunc);      // "value associated with keyFunc"

myMap.get('a string');   // "value associated with 'a string'"
                         // because keyString === 'a string'
myMap.get({});           // undefined, because keyObj !== {}
myMap.get(function() {}) // undefined, because keyFunc !== function () {}
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map

Exercise

https://jsfiddle.net/4sgz8dn3/