JavaScript
SWE 432, Fall 2018
Web Application Development
Review: Course Topics

• How do we organize, structure and share information?

• How to make applications that are delivered through browsers
  • JavaScript, front-end and back-end development, programming models, testing, performance, privacy, security, scalability, deployment, etc.

• How to design user interactions, focusing on browsers
  • User-centered design, user studies, information visualization, visual design, etc.
Show and Tell

Editors' Notes

Low-lit grooves meet high drama in the swooshing, swirling ambient-ish music of Tycho. The Bay Area producer's fourth album *Awake* mixes layered guitar strum, electric bass, and electronic wash with beats that are chilled-out but emphatic too. The panoramic title track starts the show in a melancholy mood that soon progresses toward brightness ("Montana") and, in time, drifting serenity ("Dye"). Gripping melodies and loud-quiet dynamics cribbed from post-rock set up a sense of suspense teased out in deft and diverging instrumentals, each of which seems to tell a story of its own.
Today

• Brief history of JavaScript/ECMAScript

• Overview of core syntax and language semantics

• Overview of key libraries

• HW1 Discussion

• In class activity working with JavaScript

• Next:
  • Testing and tooling; NodeJS (video, no class next week)

• Upcoming: Lunch with Professor?
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


ES1 ES2 ES3 “AJAX” jQuery ES5 ES6

Mocha/LiveScript/JavaScript 1.0
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

### Pastebins

<table>
<thead>
<tr>
<th>JS</th>
<th>HTML</th>
<th>CSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>var a = 5;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>var b = 10;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>console.log('Fifteen is ${a + b} and not ${2 * a + b}.');</td>
<td></td>
<td></td>
</tr>
<tr>
<td>// &quot;Fifteen is 15 and not 20.&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Examples: [JSFiddle](https://jsfiddle.net), [JSBin](https://jsbin.com), [seeCode.run](https://seeCode.run)

- We’ll often use seeCode.run to try out examples
Variables

• Variables are *loosely* typed
  • String:
    
    ```javascript
    var strVar = 'Hello';
    ```
  • Number:
    
    ```javascript
    var num = 10;
    ```
  • Boolean:
    
    ```javascript
    var bool = true;
    ```
  • Undefined:
    
    ```javascript
    var undefined;
    ```
  • Null:
    
    ```javascript
    var nulled = null;
    ```
  • Objects (includes arrays):
    
    ```javascript
    var intArray = [1, 2, 3];
    ```
  • Symbols (named magic strings):
    
    ```javascript
    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
  ```javascript
  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:
    ```javascript
    {
      let a = 3;
    }
    //a is undefined
    ```
Loops and Control Structures

- **if** - pretty standard
  ```javascript
  if (myVar >= 35) {
      //...
  } else if (myVar >= 25) {
      //...
  } else {
      //...
  }
  ```

- Also get **while**, **for**, and **break** as you might expect
  ```javascript
  while (myVar > 30) {
      //...
  }

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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td>Equality</td>
<td><code>age == 20</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>age == '20'</code></td>
</tr>
<tr>
<td><code>!=</code></td>
<td>Inequality</td>
<td><code>age != 21</code></td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>Greater than</td>
<td><code>age &gt; 19</code></td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>Greater or Equal</td>
<td><code>age &gt;= 20</code></td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>Less than</td>
<td><code>age &lt; 21</code></td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>Less or equal</td>
<td><code>age &lt;= 20</code></td>
</tr>
<tr>
<td><code>===</code></td>
<td>Strict equal</td>
<td><code>age === 20</code></td>
</tr>
<tr>
<td><code>!==</code></td>
<td>Strict Inequality</td>
<td><code>age !== '20'</code></td>
</tr>
</tbody>
</table>
Functions

- At a high level, syntax should be familiar:
  
  ```javascript
  function add(num1, num2) {
    return num1 + num2;
  }
  ```

- Calling syntax should be familiar too:
  
  ```javascript
  var num = add(4, 6);
  ```

- Can also assign functions to variables!
  
  ```javascript
  var magic = function(num1, num2){
    return num1+num2;
  }
  var myNum = magic(4, 6);
  ```

- Why is this cool?
Default Values

```javascript
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

```javascript
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

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

```javascript
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?

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

```javascript
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?)
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;
  }
};
```

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

```javascript
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**
  
  ```javascript
  var numberOfStudents = students.length;
  ```

- **Join**
  
  ```javascript
  var classMembers = students.concat(faculty);
  ```

- **Sort**
  
  ```javascript
  var sortedStudents = students.sort();
  ```

- **Reverse**
  
  ```javascript
  var backwardsStudents = sortedStudents.reverse();
  ```

- **Map**
  
  ```javascript
  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):
  ```javascript
  for(var student of students)
  {
      console.log(student);
  } //Prints out all student names
  ```

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

27
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

Template Literals

• Enable embedding expressions **inside** strings
  ```javascript
  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

```javascript
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}}
```

```javascript
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 () {}
```

HW1 Discussion


http://autolab.cs.gmu.edu/
Exercise

https://jsfiddle.net/4sgz8dn3/