JavaScript

SWE 432, Fall 2017
Design and Implementation of Software for the Web
Next two lectures: JavaScript

• Today
  • Brief history of JavaScript/ECMAScript
  • Overview of core syntax and language semantics
  • Overview of key libraries
  • In class activity working with JavaScript

• Next Tuesday
  • Overview of approaches for organizing code with web apps
  • Constructs for organizing code: closures, class
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

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

```c
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){
    //...
  }
  ```
  ```javascript
  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>

```javascript
var age = 20;
```

Annoying

```javascript
var age = 20;
```
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
```
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

```
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?

```javascript
var profLaToza = {
  firstName: "Thomas",
  lastName: "LaToza",
  teaches: "SWE 432",
  office: "ENGR 44431",
  fullName: function(){
    return this.firstName + " " + this.lastName;
  }
};
```
Working with Objects

```javascript
var profJon = {
  firstName: "Thomas",
  lastName: "LaToza",
  teaches: "SWE 432",
  office: "ENGR 4431",
  fullName: function(){
    return this.firstName + " " + this.lastName;
  }
};
```

Our Object

```javascript
console.log(profLaToza.firstName); // Thomas
console.log(profLaToza["firstName"]); // Thomas
```

Accessing Fields

```javascript
console.log(profLaToza.fullName()); // Thomas LaToza
```

Calling Methods

```javascript
console.log(profLaToza.fullName); // function...
```
Destructuring

// What you write
let { firstName, lastName } = Zell

// ES6 does this automatically
let firstName = Zell.firstName
let lastName = Zell.lastName

[a, b] = [b, a]

• Convenient syntax for extracting property values into a variable
• Works with objects and arrays
JSON: JavaScript Object Notation

Open standard format for transmitting data objects.

No functions, only key / value pairs

Values may be other objects or arrays

```javascript
var profJon = {
  firstName: "Thomas",
  lastName: "LaToza",
  teaches: "SWE 432",
  office: "ENGR 4431",
  fullName: function(){
    return this.firstName + " " + this.lastName;
  }
};
```

Our Object

```javascript
var profJon = {
  firstName: "Thomas",
  lastName: "LaToza",
  teaches: "SWE 432",
  office: "ENGR 4431",
  fullName: {
    firstName: "Thomas",
    lastName: "LaToza"
  }
};
```

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

```javascript
var students = ["Alice", "Bob", "Carol"];  
var faculty = [profLaToza];  
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 profLaToza){
    console.log(prop + " : " + profLaToza[prop]);
  }
  ```

Output:
firstName: Thomas
lastName: LaToza
teaches: SWE 432
office: ENGR 4431
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 :)
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

var a = 5;
var b = 10;

console.log(`Fifteen is ${a + b} and not ${2 * a + b}.`);

// "Fifteen is 15 and not 20."

• Enable embedding expressions inside strings
• 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 } }
```

Map Collection

```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 () {};
```

Demo

- Primitives: equality
- Objects: literals, JSON, stringify / parse,
- Arrays: literals, accessing, length, sort, push, pop, map
- Strings: template literals
- Collections: Map, Set
- Functions: first class, anonymous, arrow
Exercise

https://jsfiddle.net/4sgz8dn3/
Next time

• Organizing code in web apps

• Required Readings:
  • Closures: https://medium.freecodecamp.org/lets-learn-javascript-closures-66feb44f6a44