CSE1500: Web - Lecture 3 - JavaScript

Lecture Summary

The WDT Team

Ujwal Gadiraju, Sagar Chethan Kumar, Ciprian Ionescu

Contents

⚠ These summaries are not extensive overviews of lecture content and **DO NOT** cover all possible exam material. Rather, these serve as a refresher and capture **main points** for a given lecture.

Learning Goals

You should be able to do the following after the **lecture** and **assignment # 1**:

- **Create an interactive web application with client-side JavaScript. (LO4)**
- **Understand** the fundamentals of JavaScript Variables, Functions, and Scope
- **Employ** JavaScript objects.
- **Employ** JavaScript to manipulate the HTML DOM (e.g event handling, node creation...).
- **Write** interactive web applications based on click, mouse and keystroke events.
- **Develop** JavaScript code that facilitates responsive design.

1 JavaScript

JavaScript is a dynamic, interpreted, weakly-typed, and cross-platform language. It is typically used to make webpages interactive (e.g., having complex animations, clickable buttons, popup menus, etc.). *Server-side* JavaScript (ex: Node.js) allows you to add more functionality to a website on the server's end.

Dynamic Weakly-typed Example

```
a = 123;
a = a + 5;
a = "hello world";
a = a + " everyone";
console.log(a); // hello world everyone

a = "hello world";
a = a + 5;
console.log(a); // hello world5
```

Dynamic means that in JavaScript we have no fixed types associated with variables. Essentially, the interpreter determines types of values at runtime. As JavaScript is also weakly-typed it performs implicit type coercion at runtime (this is related to the notion of *strict* equality[1]).

1.1 Data Types

Like in other programming languages, JavaScript has both primitive and advanced data types:

Primitive Data Types

```
// Boolean values
var isTrue = true;
var isFalse = false;

// Numbers (only 64-bit floats)
var zero = 0;
var pi = 3.14;

// Strings (variable length, no separate character type)
var message = "6170's the best";

// No Value (null and undefined)
var nullValue = null;
var undefinedValue;
```

[1] There is actually a whole GitHub page dedicated to this.

We also have arrays (a type of advanced data type) that grow and shrink dynamically. Arrays in JavaScript are heterogenous: they can have elements of different types. Arrays have many methods for manipulation.

Arrays Experiment

```
a = ["hello", 2, null, [0, 1], "cse1500"]
a = [];
a[0] = "hello";
a[1] = "there";
a.push("everyone");
a.pop();
a.indexOf("hello");
...
a.length // What would this be?
```

Objects are another advanced data type that contain mappings between properties and values. These properties referenced with dot notation and can be added/modified on the fly. The values can be any of the types we've seen so far, including nested objects.

Objects

```
a = {hello: "there"};
a.hello // there
a.hello = "world";
a.goodbye = "for now"; // yes this works!
```

1.2 Variables and Scope

By default, variables are defined in the global scope (code examples so far). Use the `let` keyword to declare local variables, with block scope. The `const` keyword declares block scoped variables that cannot be reassigned. Note: This is **not** the same as making the variable immutable.

Variables and Scope (1)

```
let x = "John Doe";
let x = 0; // error: 'x' has already been declared
x = 0; // okay!

var x = 10;
var x = "meow"; // okay!

const PERSON = {name: "sagar"};
PERSON = {name: "ciprian"}; // error
PERSON.name = "ciprian"; // okay!
```

Pre ES6, only the `var` keyword was available to declare local variables. It has function scope, and declarations are hoisted to the top of the function.

Variable and Scope (2)

```
// variables declared with `let` and `const`
// have Block Scope in JavaScript.
// Variables declared inside a { } block cannot be
// accessed from outside the block:
if (10 < 11) {
    let x = 10;
    const y = 20;
}

console.log(x,y); // undefined undefined

// variables defined with `var` can be accessed
// outside the function.
if (10 < 11) {
    var x = 10;
    var y = 20;
}

console.log("x is "+x+ " and y is "+ y); // x is 10 and y is 20
```

1.3 Functions

Functions are declared with a name, arguments, and a body. Sometimes they can take a variable number of arguments, the special `arguments` array lists them as an array. With ES6, we can also specify default values for arguments.

Functions

```
function add(a, b) {
 return a + b;
}
function add() {
 return arguments[0] +
 arguments[1];
}
function add(a, b = 1) {
 return a + b;
}
```

Functions are like any other value: can be assigned to variables, passed as argument.

Functions Experiment

```
let students = [
 {name: "Woody", price: 10},
 {name: "Buzz", price: 8},
 {name: "Rex", price: 6},
 {name: "Slinky", price: 5.7}
];
students =
 students.filter(function(students) {
 return students.grade >= 6;
 })

students // what would this contain?
```

Arrow Functions were also introduced in ES6, which allow us to represent traditional functions in shorter syntax.

Arrow Functions

```
const sum = (a, b) => {return a + b;}
hello = () => {return "Hello World!"}
const inc = (a) => a += 1;
console.log(sum(1,2)); // 3
console.log(hello()); // Hello World!
console.log(inc(3)); // 4
```

1.4 Conditions

Comparison determine equality or difference between variable or values. The notion of *strict* equality was mentioned at the beginning of this summary. Double equals (==) in JavaScript will perform a type conversion when comparing two things. Triple equals (===) will do the same comparison as double equals but without type conversion; if the types differ, false is returned.

Strict Equality and Comparisons Experiment

```
let x = 5;
x == 5 // true
x == "5"; // true.

x === "5"; // false
x === 5; // true

// What do the following statements produce?
console.log(2 < "12"); // ?
console.log(2 == "adam"); // ?
console.log("2" > "12"); // ?
console.log("90" > ""); // ?
console.log("90" > "ujwal"); // ?
```

Similarly, we have logical, if-else, and switch conditions.

Other Conditions

```
let x = 6;
let y = 3;
console.log( (x < 10 && y > 1) );  // true
console.log( (x == 5 || y == 5) ); // false
console.log( !(x == y) ); // true

let grade = 8;
let message;
if (grade < 6) {
    message = "failed";
}
else {
    message = "passed";
}
let bsa;
switch (message) {
  case "failed":
    bsa = ":(";
    break;
  case "passed":
    bsa = ":)";
    break;
  default:
    bsa = ":0";
}

console.log(message); // Passed
console.log(bsa); // :)
```

1.5 Iterations

In JavaScript you iterate over various items in different ways:

Iterations Experiment

```
// for-loop
for(var i = 1; i <= 5; i++){
    console.log(i); // ?
}

// while-loop
let iterations = 0;
let die;
while (iterations < 10){
        die = Math.ceil(Math.random() * 10);

        if (die % 2 == 0){
                break;
        }
        iterations ++;
}
console.log("Number of iterations: ", iterations + 1); // ?
console.log("Die value: ", die); // ?

// do-while
do {
  var die1 = Math.ceil(Math.random() * 10);
  var die2 = Math.ceil(Math.random() * 10);

  console.log("Die 1:", die1, "Die 2:", die2);

  iterations++;

} while (die1 != die2);
console.log("The matched value is: ", die1); // ?
console.log("Number of iterations: ", iterations); // ?

// for-in loop
const person = {fname:"John", lname:"Doe", age:25};

let txt = "";
for (let x in person) {
  txt += person[x] + " "; // ?
}

// for-of loop
const cars = ["BMW", "Volvo", "Mini"];

let text = "";
for (let x of cars) {
  text += x;
}

console.log(text); // ?
```

1.6 DOM

As covered in `Lecture Summary 2`, the DOM enables interaction with individual HTML elements. Javascript, in specific, can use the DOM to:

1. Change all the HTML elements and their attributes

2. Change all the CSS styles
3. Remove existing HTML elements
4. React to all existing HTML events or create new ones.

Here is an overview of some important relevant methods. Can you imagine what the HTML code would need to contain for this to compile?

DOM Interactions

```
// Accessing Elements in the DOM
let userID = document.getElementById("user-id");
let divElements = document.getElementsByTagName("div");
let warningElements = document.getElementsByClassName("warning");

// changes the HTML text
document.getElementById("header").innerHTML = "Hello World!";
 // changes the attribute of an image
document.getElementById("myImage").src = "landscape.jpg";

// creates html element with <li> tag
let listElement = document.createElement("li");

let text = document.createTextNode("element 1");
listElement.appendChild(text);
```

There are a **LOT** of JavaScript methods that you can use to interact with DOM. In our course we expect you to be able to use them to complete your assignments.

1.7 Events

JavaScript operates as a single thread, capable of executing only one task at a time. Tasks that take a significant amount of time, such as animations or network requests, have the potential to obstruct the user interface.

Event Experiment (Detour)

```
function one() {
 console.log("one");
 setTimeout(two, 0);
 console.log("three");
 setTimeout(four, 0);
 console.log("five");
}
function two() {
 console.log("two");
}
function four() {
 console.log("four");
}
one() // ???
```

What gets printed to the console when we call one()? Why is this the case? This has to do with the event loop and call stack[2]. In our course, however, we focus on DOM events. An

[2]If you are interested you can read more here (not part of course content).

overview can be found here and we expect you to be able to integrate relevant ones into your assignments. Here is a basic example:

Click Event

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Add an element</title>

    <script></script>
  </head>
  <body>
    <label for="">Enter an item</label>
    <input type="text" id="menu-item" />
    <button id="btn">submit</button>

    <ul id="menu"></ul>
    <script>
      const button = document.getElementById("btn");
      button.addEventListener("click", addNewItem);

      function addNewItem() {
        let menu_item = document.getElementById("menu-item").value;
        let menu = document.getElementById("menu");

        let li = document.createElement("li");
        let text = document.createTextNode(menu_item);
        li.appendChild(text);
        menu.appendChild(li);

        document.getElementById('menu-item').value = "";
      }
    </script>
  </body>
</html>
```

2 Sample Question Types

(1) Consider the JavaScript code snippet below. We assume execution in the browser's Web Console (and thus, we know the global window object exists). What is the output on the console after the execution of this code snippet? (Answers are formatted L18_Output L19_Output)

```
1   var questions = [
2       ["Default question text", "Answer A", "Answer B", "Answer C","1"],
3       ["Default question text", "Answer A", "Answer B", "Answer C","1"],
4       ["Default question text", "Answer A", "Answer B", "Answer C","1"]
5   ];
6
7   function Quiz(title){
8       this.title = title;
9       this.questions = [];
10      this.getNumQuestions = function(){
11          return this.questions.length;
12      }
13  }
14
15  let firstQuiz = new Quiz("Rare animals");
16  let func = firstQuiz.getNumQuestions;
17
18  console.log(firstQuiz.getNumQuestions());
19  console.log(func());
```

a. 0 0

b. 0 3

c. 3 0

d. 3 3

(2) Consider the following HTML snippet:

```
1   <!DOCTYPE html>
2   <html>
3       <body>
4           <h1>Hide this button</h1>
5           <button id="b">Hide me forever</button>
6
7           <p></p>
8
9           <h2>Hide this button too</h2>
10          <button class="second">Hide me forever too</button>
11      </body>
12  </html>
```

Which of the following lines of JavaScript does **NOT** select the button defined in line 10?

a. `document.getElementsByClassName("second")[0]`

b. `document.getElementsByTagName("button")[1]`

c. `document.querySelectorAll("button")[1]`

d. `document.querySelectorLast("button")`

3 Extra Exercises

If you are feeling confident on the topic of **JavaScript** check out the following tasks[3]:

- In a web browser, go to http://example.com/ and open your JavaScript Console.
    1. Write three ways of selecting the "More information" link, one of which must include a DOM tree traversal starting from the body tag.
    2. Inside the white box, add a third paragraph of text.
    3. Add a 2nd-level heading before the paragraph you just addeds
- JavaScript W3Schools Exercises

4 References and Further Reading

Here are some useful resources for learning more about **JavaScript**:

- JavaScript Basics MIT 6.170
- W3Schools HTML DOM Overview

[3] `Note: None of these exercises are mandatory nor provide a bonus.`