CS 146: Intro to Web Programming and Project Development

Instructor: Iraklis Tsekourakis

Lieb 213

Email: itsekour@stevens.edu

JavaScript

JavaScript Data Types

- JavaScript variables can hold many **data types**: numbers, strings, arrays, objects and more:

```
var length = 16;                                    // Number
var lastName = "Johnson";                           // String
var cars = ["Saab", "Volvo", "BMW"];                // Array
var x = {firstName:"John", lastName:"Doe"};         // Object
```

The Concept of Data Types

```
var x = 16 + "Volvo";
var x = "Volvo" + 16;
var x = 16 + 4 + "Volvo";
var x = "Volvo" + 16 + 4;
```

JavaScript Dynamic Data Types

```
var x;                  // Now x is undefined
var x = 5;              // Now x is a Number
var x = "John";         // Now x is a String
```

```
var carName = "Volvo XC60";   // Using double quotes
var carName = 'Volvo XC60';   // Using single quotes
```

```
var x1 = 34.00;       // Written with decimals
var x2 = 34;          // Written without decimals
```

```
var x = true;
var y = false;
```

JS: The typeof Operator

- You can use the JavaScript **typeof** operator to find the type of a JavaScript variable:

```
typeof "John"                 // Returns "string"
typeof 3.14                   // Returns "number"
typeof false                  // Returns "boolean"
typeof [1,2,3,4]              // Returns "object" (not "array", see note below)
typeof {name:'John', age:34}  // Returns "object"

var person;                   // Value is undefined, type is undefined
```

- The typeof operator returns "object" for arrays because in JavaScript arrays are objects

NULL value

- In JavaScript null is "nothing"; it is supposed to be something that doesn't exist
- Unfortunately, in JavaScript, the data type of null is an object
- *You can consider it a bug in JavaScript that typeof null is an object; it should be null*

```
typeof undefined             // undefined
typeof null                  // object
null === undefined           // false
null == undefined            // true
```

JavaScript Functions

- A JavaScript function is a block of code designed to perform a particular task
- Function Syntax:

```
function name(parameter1, parameter2, parameter3) {
    code to be executed
}
```

- Function **parameters** are the **names** listed in the function definition
- Function **arguments** are the real **values** received by the function when it is invoked
- Inside the function, the arguments (the parameters) behave as local variables

More on Functions

- Functions can be invoked by
  - Events
  - JS code
  - Auto/self-invoked
- Return statement

```
var x = myFunction(4, 3);          // Function is called, return value will end up in x

function myFunction(a, b) {
    return a * b;                  // Function returns the product of a and b
}
```

- You can reuse code
- You need () to invoke a function!

JavaScript Objects

```
<body>

 <p>Creating a JavaScript Object.</p>

 <p id="demo"></p>

 <script>
  var person = {
   firstName : "John",
   lastName  : "Doe",
   age       : 50,
   eyeColor  : "blue",
   getName   : function(){return person.firstName + " is " + person.age + "
years old.";}
  };

  document.getElementById("demo").innerHTML =
  person.getName();
 </script>

</body>
```

Do Not Declare Strings, Numbers, and Booleans as Objects!

- When a JavaScript variable is declared with the keyword "new", the variable is created as an object:

```
var x = new String();        // Declares x as a String object
var y = new Number();        // Declares y as a Number object
var z = new Boolean();       // Declares z as a Boolean object
```

- Avoid String, Number, and Boolean objects; they complicate your code and slow down execution speed

Scope!

- Variables declared within a JavaScript function, become **LOCAL** to the function
- Local variables have **local scope**: They can only be accessed within the function

```
// code here can not use carName

function myFunction() {
    var carName = "Volvo";

    // code here can use carName

}
```

- Pros?
- Cons?

Global JavaScript Variables

- A variable declared outside a function, becomes **GLOBAL**
- A global variable has **global scope**: All scripts and functions on a web page can access it

```
var carName = " Volvo";

// code here can use carName

function myFunction() {

    // code here can use carName

}
```

Automatic Global Variables

- If you assign a value to a variable that has not been declared, it will automatically become a **GLOBAL** variable
- This code example will declare a global variable **carName**, even if the value is assigned inside a function

```
myFunction();

// code here can use carName

function myFunction() {
    carName = "Volvo";
}
```

- Do NOT create global variables unless you intend to
- In "Strict Mode" automatically global variables will fail; why should it?

JavaScript Events

- With JS, we can ”react” to HTML events
- An HTML event can be something the browser does, or something a user does
  - An HTML web page has finished loading
  - An HTML input field was changed
  - An HTML button was clicked
- HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements

```
<button onclick="document.getElementById('demo').innerHTML=Date()">The time is?
</button>
```

Common HTML Events

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

String Methods

- String Length:

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

- Finding a String in a String:

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate");
```

```
var str = "Please locate where 'locate' occurs!";
var pos = str.search("locate");
```

```
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate");
```

- Both the indexOf(), and the lastIndexOf() methods return -1 if the text is not found
- JavaScript counts positions from zero
  0 is the first position in a string, 1 is the second, 2 is the third..

Number Methods

- toString()

```
var x = 123;
x.toString();            // returns 123 from variable x
(123).toString();        // returns 123 from literal 123
(100 + 23).toString();   // returns 123 from expression 100 + 23
```

- toExponential()

```
var x = 9.656;
x.toExponential(2);      // returns 9.66e+0
x.toExponential(4);      // returns 9.6560e+0
x.toExponential(6);      // returns 9.656000e+0
```

Number Properties

Property	Description
MAX_VALUE	Returns the largest number possible in JavaScript
MIN_VALUE	Returns the smallest number possible in JavaScript
NEGATIVE_INFINITY	Represents negative infinity (returned on overflow)
NaN	Represents a "Not-a-Number" value
POSITIVE_INFINITY	Represents infinity (returned on overflow)

```
var x = Number.MAX_VALUE;
```

JS Arrays

- What if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

```
var car1 = "Saab";
var car2 = "Volvo";
var car3 = "BMW";
```

- The solution is an array!

```
var cars = ["Saab", "Volvo", "BMW"];
```

- ***Never put a comma after the last element (like "BMW",).***
  The effect is inconsistent across browsers.

```
var cars = new Array("Saab", "Volvo", "BMW");
```

JS Arrays

- Access Array elements by index

```
var name = cars[0];
```

- Access the full Array

```
document.getElementById("demo").innerHTML = cars;
```

- The length property

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.length;                          // the length of fruits is 4
```

- Arrays are Objects (for JS)

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

typeof fruits;                  // returns object
```

  - How do we recognize arrays?

```
Array.isArray(fruits);      // returns true
```

JS Array Methods

- toString()

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

- pop()

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();              // Removes the last element ("Mango") from fruits
```

- push()

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");        //  Adds a new element ("Kiwi") to fruits
```

- shift()

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift();             // Returns "Banana"
```

- unshift()

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");    // Adds a new element "Lemon" to fruits
```

JS Array Sort

- The **sort()** method sorts an array alphabetically
- The **reverse()** method reverses the elements in an array

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();              // Sorts the elements of fruits
fruits.reverse();           // Reverses the order of the elements
```

- Numeric Sort & the Compare Function

```
var points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a - b});

var points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return b - a});
```

Conditions & Switch

```
if (time < 10) {
    greeting = "Good morning";
} else if (time < 20) {
    greeting = "Good day";
} else {
    greeting = "Good evening";
}
```

```
switch (new Date().getDay()) {
    case 6:
        text = "Today is Saturday";
        break;
    case 0:
        text = "Today is Sunday";
        break;
    default:
        text = "Looking forward to the Weekend";
}
```

Loops

```
for (i = 0; i < 5; i++) {
    text += "The number is " + i + "<br>";
}
```

```
for (x in person) {
    text += person[x];
}
```

```
while (i < 10) {
    text += "The number is " + i;
    i++;
}
```

```
do {
    text += "The number is " + i;
    i++;
}
while (i < 10);
```