JavaScript Basics

Çağatay Sönmez

01.10.2021

Arcelik Electronics Plant, R&D Team – Software Design

Agenda

- What is JavaScript
- JavaScript Data Types
- JavaScript Memory Management
- JavaScript Syntax
- JavaScript Variable Scope
- HTML5 + JavaScript APIs

What is JavaScript

- JavaScript is a scripting language used both on the client-side and server-side
- JavaScript is code statements inserted into HTML pages to be executed by the web browser
  - Since JavaScript code can run locally in a user's browser (rather than on a remote server), the browser can respond to user actions quickly, making an application more responsive
- In recent years, JavaScript has also been used in server-side applications

JavaScript and Java

- A common misconception is that JavaScript is similar or closely related to Java.
  - JavaScript copies many names and naming conventions from the programming language Java and both have a C-like syntax, the C language being their most immediate common ancestor language.
- However, the similarities end there;
  - Java has static typing; JavaScript's typing is dynamic (meaning a variable can hold an object of any type and cannot be restricted).
  - Java is loaded from compiled byte code; JavaScript is loaded as human-readable source code.
  - Java's objects are class-based; JavaScript's are prototype-based.

ECMAScript Compatibility

- ECMAScript is a JavaScript standard used to ensure the interoperability of web pages across different web browsers
- ECMAScript version is important for the web developers
  - JavaScript code should be compatible with the end users' browser
- Most of the modern browsers (except IE) support ECMAScript 2015

Browser Support for ES6 (2015)

Browser	Version	Date
Chrome	51	May 2016
Firefox	52	Mar 2017
Edge	14	Aug 2016
Safari	10	Sep 2016
Opera	38	Jun 2016

ECMAScript Versions

Ver	Official Name	Description
ES1	ECMAScript 1 (1997)	First edition
ES2	ECMAScript 2 (1998)	Editorial changes
ES3	ECMAScript 3 (1999)	Added regular expressions Added try/catch Added switch Added do-while
ES4	ECMAScript 4	Never released
ES5	ECMAScript 5 (2009) Read More	Added "strict mode" Added JSON support Added String.trim() Added Array.isArray() Added Array iteration methods Allows trailing commas for object literals
ES6	ECMAScript 2015 Read More	Added let and const Added default parameter values Added Array.find() Added Array.findIndex()
	ECMAScript 2016 Read More	Added exponential operator (**) Added Array.includes()
	ECMAScript 2017 Read More	Added string padding Added Object.entries() Added Object.values() Added async functions Added shared memory
	ECMAScript 2018 Read More	Added rest / spread properties Added asynchronous iteration Added Promise.finally() Additions to RegExp

Features of JavaScript

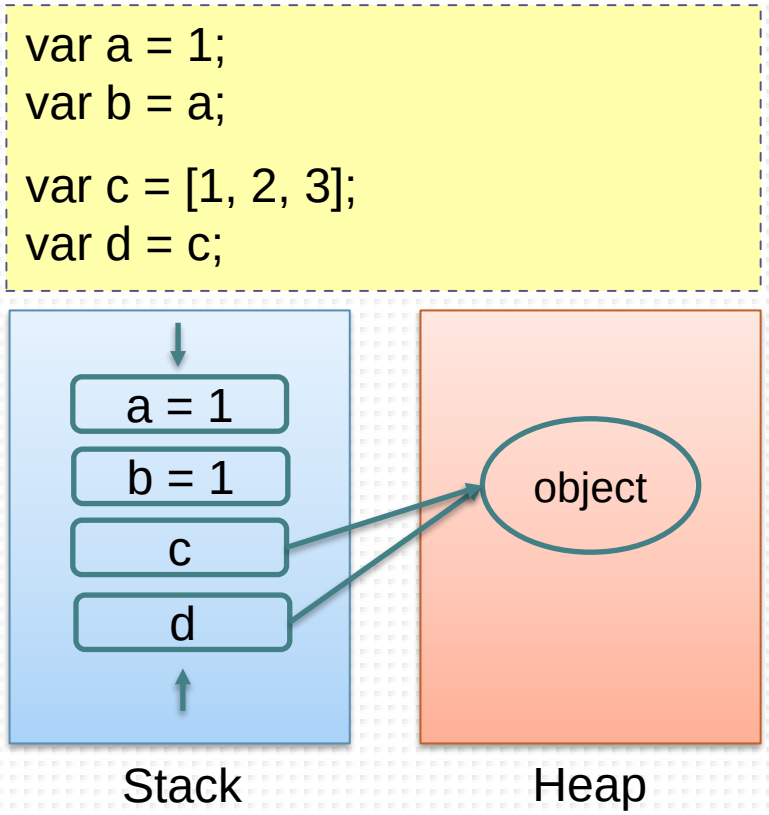
- Structured
  - JavaScript supports much of the structured programming syntax from C (e.g., if statements, while loops, switch statements, etc.).
- Dynamic typing
  - As in most scripting languages, types are associated with values, not with variables.
- Functional
  - Functions are first-class; they are objects themselves.
- Prototype-based
  - JavaScript uses prototypes instead of classes for inheritance. It is possible to simulate many class-based features with prototypes in JavaScript.

Types in JavaScript

- **Boolean**
  var myBoolean= false;
- **Number**
  var myFloat = 0.3555
- **String**
  var myString = "This is a string";
- **Object**
  var myObject = new Object();
- **Function**
  var myFunction = function(param) { return 0; };

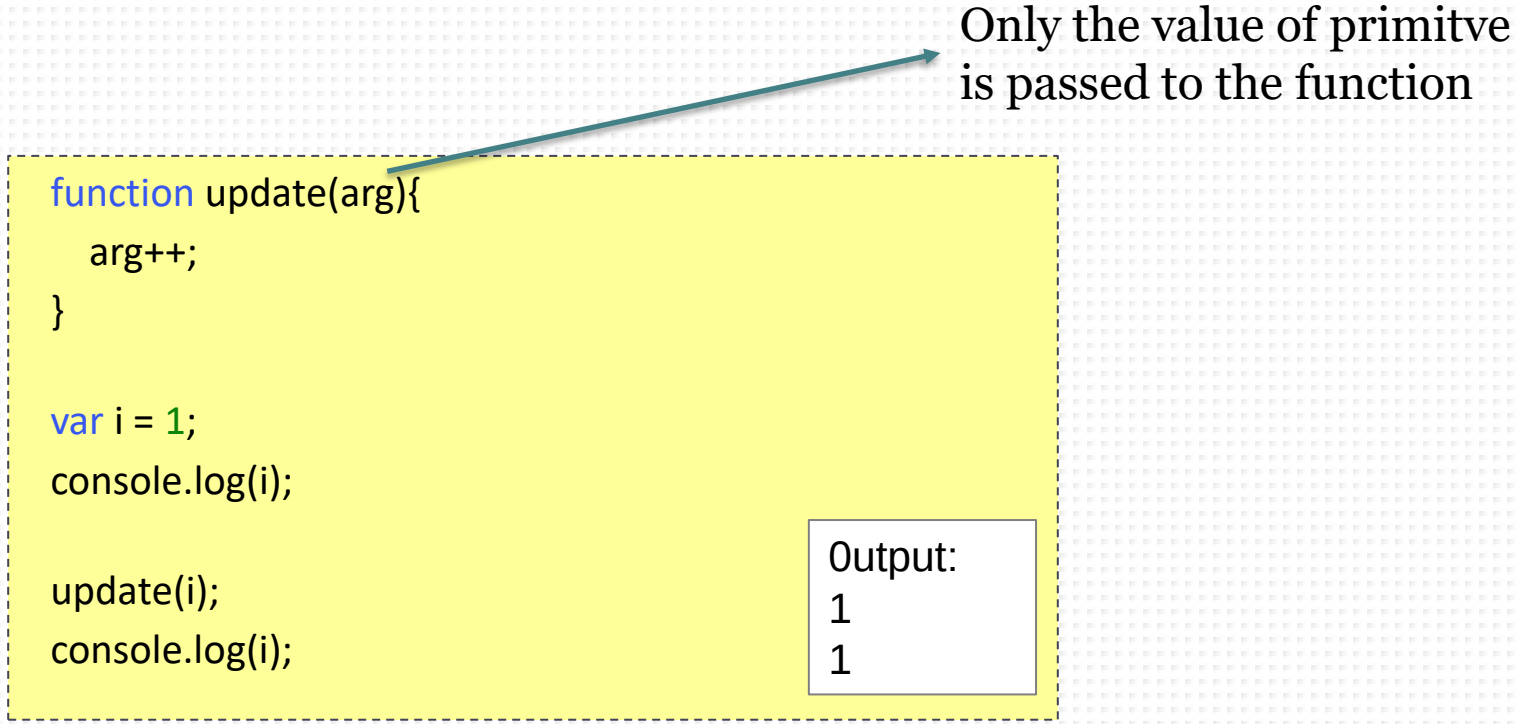
JavaScript Memory Management

- The primitives (boolean, number, string) are stored in call stack
- Non-primitives (objects) are stored at somewher in the memory (heap)



Pass by Value & Pass by Reference

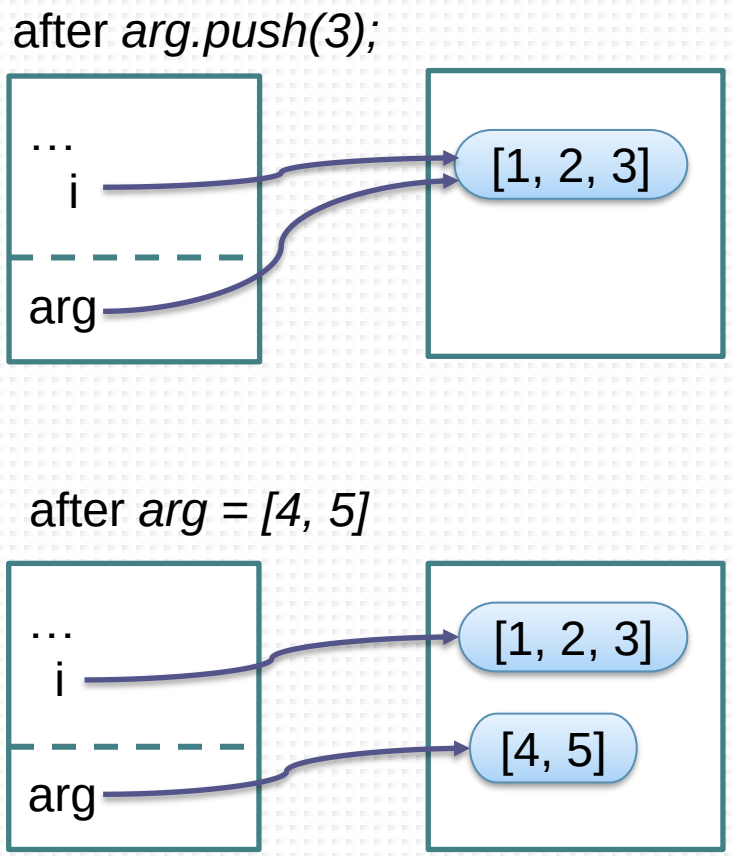
- Primitives are passed by value
- Objects are passed by *copy of a reference*



Pass by Reference Example

```
function update(arg){
   arg.push(3);
}

function reCreate(arg){
   arg = [4, 5];
}

var i = [1, 2];
console.log(i)

update(i);
console.log(i)

reCreate(i);
console.log(i)
```

Output:
[1, 2]
[1, 2, 3]
[1, 2, 3]



JavaScript Syntax

```
function myFunc(arg) {
        alert("argument is" + arg);
}
```

```
if (x == y) {
        alert("x is equal to y");
}
```

```
var flag = false;
while (flag == false) {
        alert("flag is true");
        flag = true;
}
```

```
for (i=0; i<10; i++) {
        alert("i is: " + i);
}
```

```
function Shape() { this.area=0 };
Shape.prototype.draw = function() {
        console.log("do draw");
};
```

The JavaScript typeof Operator

- The typeof operator returns the current variable type of the specified variable

```
var myVariable;
console.log ( "undefinedVar = " + typeof undefinedVariable);
console.log ( "myVariable = " + typeof myVariable );

myVariable = 10;
console.log ( "myVariable = " + typeof myVariable );

myVariable = "Hello";
console.log ( "myVariable = " + typeof myVariable );
```

```
0utput:
undefinedVar = undefined
myVariable = undefined
myVariable = number
myVariable = string
```

Variable Scope in JavaScript

- Script Level Scope
  - A variable declared with a "var" statement outside any function has a scope at the script level. A variable with a script level scope, called global variable.
- Function Level Scope
  - A variable declared with a "var" statement inside a function has a scope at the function level. A variable with a function level scope, called local variable.
- Auto-declaration
  - If a variable is used without the "var" declaration statement, it will be automatically declared with the script level scope, becoming a global variable, even it is used inside a function.

Scope Example I

```
var a=1;                                    //a globally-scoped variable

function one() {console.log(a); }           //global scope

function two(a) {console.log(a); }          //local scope

function three() {
    var a = 3;
    console.log(a);                         //local scope
}

function four() {
    if(true)
        var a=4;
 console.log(a);    //local scope, There is no block scope in JavaScript!
}

one(); two(2); three(); four();
```

Output:
1
2
3
4

Scope Example II

```
var a = 1;                          // global scope

function dangerous(){
    a = 5;                          // (automatically declared) global scope
    alert(a);
}
function myFunc() {
    alert(a);
}
function otherFunc(){
    var a = 10;                      // local scope
    alert(a);
}

myFunc();
dangerous();
myFunc();
otherFunc();
```

Output:
1
5
5
10

Scope Example III

```
// Advanced: closure
function six(){
    var foo = 6;
    return function(){
        console.log(foo);
    }
}
// Advanced: object properties
function Seven(){
  this.a = 7;
}
// Advanced: prototype-based scope resolution
Seven.prototype.a = -1;  // [object].prototype.property loses to [object].property
Seven.prototype.b = 8;

six()();
console.log(new Seven().a);
console.log(new Seven().b);
```

Output:
6
7
8

Ways of Getting Properties

- There are two ways of getting values from an object:

```
if( typeof(newObject.someKey) === "string" )
{
    //Dot syntax
    var key1 = newObject.someKey;

    //Square bracket syntax
    var key2 = newObject["someKey"];
}
```

Object Literals

- In object literals, an object is described as a set of comma-separated name/value pairs enclosed in curly braces ({}).

```
var myModule = {
    myProperty: "someValue",

    myConfig: {
        useCaching: true,
        language: "en"
    },

    myMethod: function () {
        console.log( "Caching is:" + ( this.myConfig.useCaching ) ? "enabled" : "disabled" );
    }
};

myModule.myMethod();
```

Output:
Caching is enabled

Classes in JavaScript

- JavaScript is a class-less language until ES6 version
- However, classes can be simulated using functions in earlier versions as well

```
function Car( model ) {
    this.model = model;
    this.year  = "2021";

    this.getInfo = function () {
        return this.model + " " + this.year;
    }
}

var myCar = new Car("Mercedes");
myCar.year = "2020";
console.log(myCar.getInfo());
```

0utput:
Mercedes 2020

Constructors With Prototypes

- Functions in JavaScript have a property called a prototype. When we call a JavaScript constructor, all the related prototypes are available to the new object.

```
function Car( model ) {
    this.model = model;
    this.year  = "2012";

}

Car.prototype.getInfo = function () {
        return this.model + " " + this.year;
}
```

Important Comparison Operators

- Given that x=5, the table below explains the comparison operators:

Operator	Description	Comparing	Returns
==	is equal to	x == "5"	*True*
		x == 5	*True*
===	is exactly equal to (value and type)	x === "5"	*False*
		x === 5	*True*
!=	is not equal	x != "5"	*False*
		x != 5	*False*
!==	is not equal (neither value nor type)	x !== "5"	*True*
		x !== 5	*False*

Note: Equality or strict equality operators return true only if both operands reference the **same** object.

AJAX

- Ajax stands for **A**synchronous **J**avaScript **A**nd **X**ML
- Ajax allows web pages to download data asynchronously from a web server
- It makes it possible to update parts of a web page, without reloading the whole page
- Transport data does not need to be XML, it can be in plain text or other formats such as JSON
- The **XMLHttpRequest** object is used to call ajax request
- All modern browsers support the XMLHttpRequest object

AJAX Example

```
// Create an XMLHttpRequest object
const xhttp = new XMLHttpRequest();

// Define a callback function
xhttp.onload = function() {
    document.getElementById("result").innerHTML = this.responseText;
}

// Send a request
xhttp.open("GET", "https://httpbin.org/get");
xhttp.setRequestHeader("Access-Control-Allow-Origin", "*");
xhttp.send();
```

See Example

Event Handling in JavaScript

Inline

```
<a id="myLink" onClick="doSomething()"> Click Me </a>
```

Traditional

```
document.getElementById("myLink").onclick = doSomething;
```

W3C Model

```
var linkElement = document.getElementById("myLink");
linkElement.addEventListener('click',doSomething,false);
```

Event Bubbling in JavaScript

- If you clicked the area covered by div3:

```
<html>
<script>
        function eventHandler(element) {
                alert("event is fired by " + element);
        }
</script>
<body>
<div id="div1" onclick="eventHandler(this)">
        <div id="div2" onclick="eventHandler(this)">
                <div id="div3" onclick="eventHandler(this)">
                </div>
        </div>
</div>
<body>
<html>
```

0utput:
event is fired by div3
event is fired by div2
event is fired by div1

Event Propagation in JavaScript

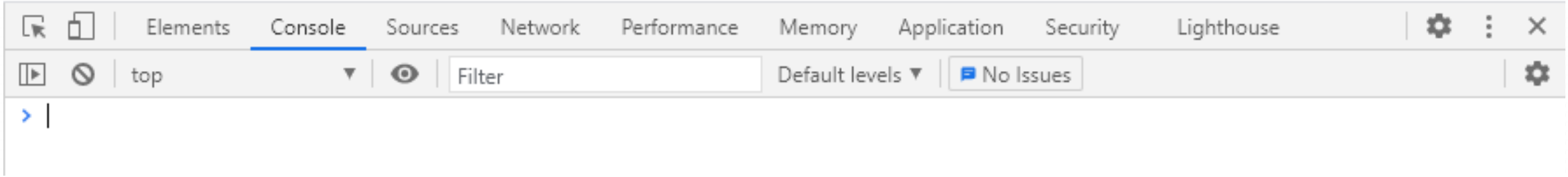
- If you clicked the area covered by div3:

```
<html>
<script>
    function eventHandler(element) {
        alert("event is fired by " + element);
        window.event.stopPropagation();
    }
</script>
<body>
<div id="div1" onclick="eventHandler(this)">
    <div id="div2" onclick="eventHandler(this)">
        <div id="div3" onclick="eventHandler(this)">
        </div>
    </div>
</div>
<body>
<html>
```

0utput:
event is fired by div3

Debugging JavaScript Code

- Chrome Dev Tools can be used for debugging web pages
- You can do many things that developers need
  - inspect or edit Elements and CSS rules
  - can see console logs
  - inspect all resources, use breakpoints on scripts etc.
  - analyse network performance, memory usage and more
  - opening web pages by imitating different devices
  - view HTTPS certificate and TLS status



HTML5 + JavaScript APIs

HTML5 Web Storage

- With HTML5 web storage, web pages can store data locally within the user's browser.
- HTML5 web storage is a better local storage than cookies.
- We can design offline applications via local storage.
- There are two new objects for storing data on the client:
  - localStorage - stores data with no expiration date
  - sessionStorage - stores data for one session

HTML5 Web Storage Example

```
// use localStorage for persistent storage
// use sessionStorage for per tab storage
saveButton.addEventListener('click', function ()
  {
    window.localStorage.setItem('value', area.value);
    window.localStorage.setItem('timestamp', (new Date()).getTime());
  }, false);

textarea.value = window.localStorage.getItem('value');
```

See Example

HTML5 Web Storage vs Cookie

- Cookies allow up to only 4 KB of data storage whereas local storage allows up to 5 MB.
- Cookies are passed with every request to the server, making the process slow and ineffective.
- Web storage data can be accessed only via client-side scripts. It is not carried over to the server via HTTP. So web storage offers a more secure method.

HTML5 Web Workers

- When executing scripts in an HTML page, the page becomes unresponsive until the script is finished.
- A web worker is a JavaScript that runs in the background, independently of other scripts, without affecting the performance of the page.
- Web workers are supported in all major browsers, except Internet Explorer.
- Chrome does not allow webworkers in file domain
  - You can start chrome as follows for development purpose
  - chrome --allow-file-access-from-files file:///<html file path>

HTML5 Web Workers Example

main.js

```
var worker = new Worker('task.js');
worker.onmessage = function(event) {
    alert(event.data);
};
worker.postMessage('data');
```

Task.js

```
self.onmessage = function(event) {
    //Do some work.
    self.postMessage("received: " + event.data);
};
```

See Example

HTML5 Web Sockets

- So HTML now does not provide only one way communication. HTML5 web socket provides full-duplex, bi-directional communication over the Web.
- Both the server and client can send data at any time, or even at the same time.
- Only the data itself is sent, without the overhead of HTTP headers, dramatically reducing bandwidth.

HTML5 Web Socket Example

```
var socket = new WebSocket('ws://html5rocks.websocket.org/echo’);

socket.onopen = function(event) {
    socket.send('Hello, WebSocket');
};

socket.onmessage = function(event) {
    alert(event.data);
};
socket.onclose = function(event) {
    alert('closed');
}
```

See Example

HTML5 Geolocation

- The HTML5 Geolocation API is used to get the geographical position of a user.
- Since this can compromise user privacy, the position is not available unless the user approves it.
- Internet Explorer 9, Firefox, Chrome, Safari and Opera support Geolocation.
- Geolocation is much more accurate for devices with GPS, like iPhone.

HTML5 Geolocation Example

```
if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(function(position) {
            var lat = position.coords.latitude ;
            var long = position.coords.longitude ;
    }, errorHandler);
}
else {
    alert("geolocation  is not supported by your browser.");
}
```

See Example

HTML5 Device Orientation

- *deviceorientation*, supplies the physical orientation of the device, expressed as a series of rotations from a local coordinate frame.
- As part of the w3c spec, you listen for a deviceOrientation event in the following manner

```
window.addEventListener('deviceorientation', function(event) {
        var a = event.alpha;
        var b = event.beta;
        var g = event.gamma;
}, false);
```

Real-Life Examples

- HTML5 Canvas
  - http://www.kevs3d.co.uk/dev/asteroids/
  - http://craftymind.com/factory/html5video/CanvasVideo.html
- HTML5 WebSockets
  - http://labs.dinahmoe.com/plink/
- CSS3 2D/3D Transitions
  - http://www.joelambert.co.uk/flux/transgallery.html
- CSS3 3D Animations
  - http://demo.marcofolio.net/3d_animation_css3/

QUESTIONS?