

JavaScript Basics



Building Modern Web Applications - VSP2023

Karthik Pattabiraman
Kumseok Jung
Mohsen Salehi

Introduction to JavaScript

1. Introduction to JavaScript

2. Data Types

3. Statements and Expressions

4. Class Activity



Modern Web Application

- Client - a lot more interactive and actively renders content within the browser
- Application logic split between client and server – client can execute (JavaScript) code
- No need to reload the web page for updating the state of the page being displayed (DOM)
- Rich message passing interface with the server through AJAX messages



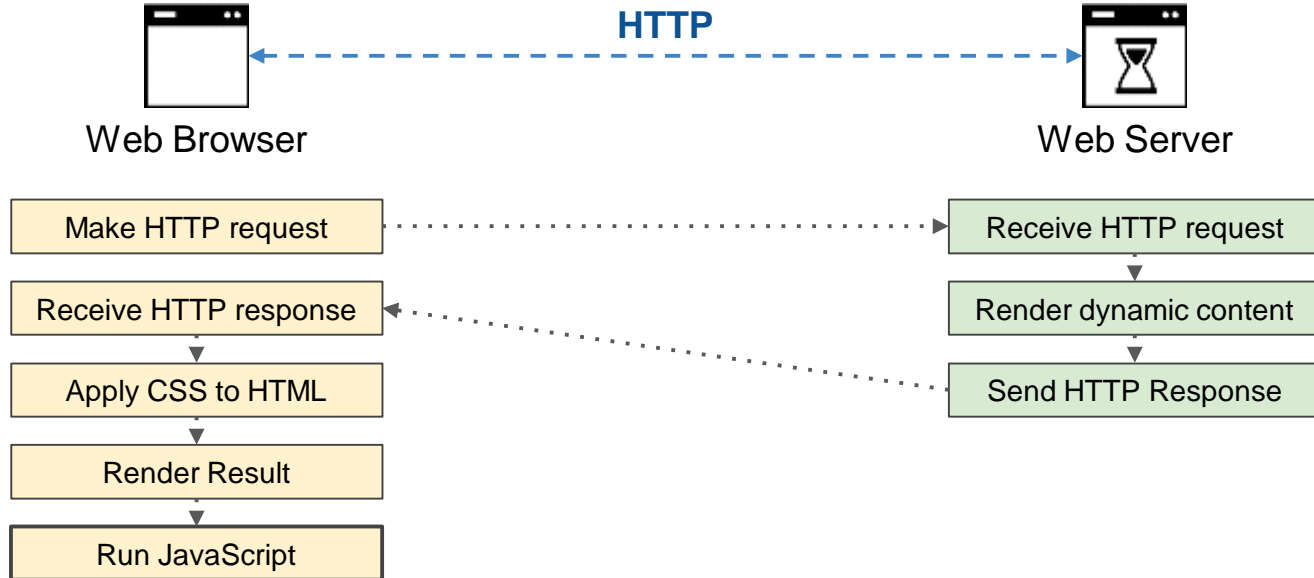
Modern Web Application: Client - JavaScript

- Unique to modern web applications and provides active functionality
- Executed when `<script>` tag is encountered or when events are triggered on DOM elements
- Is a full-fledged programming language with many advanced features (and some bad ones)



JavaScript: Browsers become Runtime Systems

Anatomy of a Web Application (2000s~)



Introduction to JavaScript: HTML `<script>` tag

1. Inline JavaScript - directly as part of the HTML document



```
1 <html>
2   <head>
3     <title>My JavaScript Page</title>
4   </head>
5   <body>
6     Hello
7     <script type="text/javascript">
8       var i = 2+2;
9       document.writeln(i);
10    </script>
11    World
12  </body>
13 </html>
```

Introduction to JavaScript: HTML `<script>` tag

2. External JavaScript - asynchronously loaded via HTTP



```
1 <html>
2   <head>
3     <title>My JavaScript Page</title>
4   </head>
5   <body>
6     Hello
7     <script type="text/javascript" src="app.js"></script>
8     World
9   </body>
10 </html>
11
12
13
```

Introduction to JavaScript: HTML <script> tag

2. External JavaScript - asynchronously loaded via HTTP



```
1 // app.js
2 alert("Hello World!");
3
4
5
6
7
8
9
10
11
12
13
```


Introduction to JavaScript: HTML `<script>` tag

2. External JavaScript - asynchronously loaded via HTTP



```
1 <html>
2   <head>
3     <title>My JavaScript Page</title>
4   </head>
5   <body>
6     Hello
7     <script type="text/javascript" src="app.js"></script>
8     World
9   </body>
10 </html>
11
12
13
```

Clear separation of document structure (HTML) from style (CSS) and from application logic (JavaScript).



Introduction to JavaScript: Comments

- Useful to document your JavaScript code!
 - Information for **other team members** and **yourself**
 - Single-line comments: `// example comment`
 - Multi-line comments: `/* example
comment */`



```
1 // This line will be ignored by the Javascript engine
2
3 greeting = "Hello"; // This is also ignored
4
5 /* <- Starting here,
6 multiple lines will be ignored
7 up to this token right here -> */
```

Introduction to JavaScript: The window Object

JavaScript VMs have many **built-in** objects

- `window` object represents the enclosing window of a browser
 - It holds the **global context** of a JavaScript program
 - In non-strict mode, **any undeclared variables** are attached to the window object



```
1 greeting = "Hello";  
2  
3 console.log(window.greeting);  
4  
5  
6  
7  
8
```

TRY IT!

Introduction to JavaScript: The window Object

JavaScript VMs have many **built-in** objects

- `window` object represents the enclosing window of a browser
 - It holds the **global context** of a JavaScript program
 - In non-strict mode, **any undeclared variables** are attached to the window object



```
1 greeting = "Hello";  
2  
3 console.log(window.greeting);  
4  
5  
6 window.greeting = "Hello";  
7  
8 console.log(greeting);
```

TRY IT!

Data Types

1. Introduction to JavaScript
- 2. Data Types**
3. Statements and Expressions
4. Class Activity



Data Types: Primitive Objects

- Boolean: `true` or `false`
- Number: `1`, `3.1412`, `1.6e3`, `01011001`
 - There is no distinction between Integers and Floating Point Numbers
- String: `"Hello"`, `'World'`



```
1 b = false;
2 n = 42;
3 s = "Hello World!";
4
5 console.log(typeof b); // prints: boolean
6 console.log(typeof n); // prints: number
7 console.log(typeof s); // prints: string
```

Data Types: Array

- Used to hold multiple objects in a sequence
- Arrays in JavaScript are **not typed** and **dynamic**
 - Can hold **any object** regardless of type
 - Can **add** or **remove** items **anytime**



```
1 my_list = [ "Hello World!", 42, true ];
2
3 console.log(my_list[0]);    // prints: Hello World!
4 console.log(my_list[1]);    // prints: 42
5 console.log(my_list[2]);    // prints: true
6
7 my_list.push("JavaScript is great!");
8
9 console.log(my_list[3]);    // prints: JavaScript is great!
```

Data Types: Array

- Used to hold multiple objects in a sequence
- Arrays in JavaScript are **not typed** and **dynamic**
 - Can hold **any object** regardless of type
 - Can **add** or **remove** items **anytime**
- Arrays can store Arrays



```
1 my_2d_list = [  
2   [ "A", "B", "C" ],  
3   [ "D", "E", "F" ],  
4   [ "G", "H", "I" ]  
5 ];  
6  
7 console.log(my_2d_list[1][2]); // prints: F
```


Data Types: Associative Array

- key-value data structure, similar to Python dictionary



```
1 dictionary = {  
2     ab: "Alberta",  
3     bc: "British Columbia",  
4     on: "Ontario",  
5     qc: "Quebec"  
6 };  
7  
8 console.log(dictionary.bc);    // prints: British Columbia  
9  
10 code = "qc";  
11 console.log(dictionary[code]); // prints: Quebec  
12  
13
```

Data Types: Associative Array

- Can be arbitrarily nested



```
1 member = {  
2   name: "Alice",  
3   age: 25,  
4   address: {  
5     province: "BC",  
6     city: "Vancouver",  
7     street: "123 Main Street"  
8   }  
9 };  
10  
11 console.log(member.address.city); // prints: Vancouver  
12 member.phone = "012-345-6789";  
13
```

Data Types: Associative Array

- Objects have **properties**

- Properties point to other Objects in the heap
- Properties can be dynamically added, removed, or re-assigned a value



```
1 member = {};  
2  
3 member.name = "Alice";  
4 member.phone = "012-345-6789";  
5  
6 console.log(member.name); // prints: Alice  
7  
8 delete member.name  
9 console.log(member.name); // prints: undefined
```

TRY IT!

Data Types: Function

- Function is also an Object

```
1  select_max = function (number_list){  
2      /* do something */  
3  };  
4  
5  console.log(select_max); // prints: [Function: select_max]  
6                          // *output may differ between browsers  
7  
8  
9  
10  
11  
12  
13
```



Data Types: Function

- Function is also an Object



```
1 select_max = function (number_list){
2     /* do something */
3 };
4
5 console.log(select_max); // prints: [Function: select_max]
6                          // *output may differ between browsers
7
8 select_max.description
9     = "Returns the maximum value from an Array of numbers";
10
11 console.log(select_max.description);
12     // prints: Returns the maximum value from an Array of numbers
13
```

Data Types: null and undefined

- `null` is actually something
 - It indicates the **absence** of a value
 - `null` itself is an object
 - Big **source of confusion**; dubbed as a major **BUG**
- `undefined` is when there is actually nothing



```
1 null_data = null;
2 undefined_data = undefined;
3
4 console.log(typeof null_data);      // prints: object
5 console.log(typeof undefined_data); // prints: undefined
6
7 console.log(window.foo); // prints: undefined
```

TRY IT!

Statements and Expressions

1. Introduction to JavaScript
2. Data Types
- 3. Statements and Expressions**
4. Class Activity



Statements and Expressions: Variable Declaration

- **var** keyword used to declare variables
- No **types** - JS is “duck-typed”



```
1  "use strict";
2
3  var width;
4  var height;
5
6  var width, height, length;
7
8  var width = 10;
9  var width = 20, height = 5, length = 10;
10 var volume = width * height * length;
11
12 console.log(volume);
```


Statements and Expressions: Assignment Statement

- `=` operator used to assign a new value to a reference
 - In strict mode, assignment is allowed only for declared variables



```
1  "use strict";
2
3  var width = 20, height = 5, length = 10;
4  var volume1 = width * height * length;
5
6  console.log(volume1);
7
8  width = 10;
9  height = 15;
10 var volume2 = width * height * length;
11
12 console.log(volume1, volume2);
```

Statements and Expressions: Binary/Unary Expression

Arithmetic

1	<code>a + b;</code>
2	<code>a - b;</code>
3	<code>a * b;</code>
4	<code>a / b;</code>
5	<code>a % b;</code>
6	
7	
8	
9	
10	
11	
12	
13	

Bitwise

1	<code>~b;</code>
2	
3	<code>a & b;</code>
4	<code>a b;</code>
5	<code>a ^ b;</code>
6	<code>a ~ b;</code>
7	<code>a << b;</code>
8	<code>a >> b;</code>
9	<code>a >>> b;</code>
10	
11	
12	
13	

Logical

1	<code>!b;</code>
2	
3	<code>a == b;</code>
4	<code>a === b;</code>
5	<code>a != b;</code>
6	<code>a !== b;</code>
7	<code>a > b;</code>
8	<code>a >= b;</code>
9	<code>a < b;</code>
10	<code>a <= b;</code>
11	
12	<code>a && b;</code>
13	<code>a b;</code>



Statements and Expressions: Binary/Unary Expression

- 2 Different notions of **equality**

- `a == b` : a and b are “equivalent”
 - Loose equality
 - Equal if the values are equivalent
 - **Type coercion** performed implicitly
- `a === b` : a and b are “identical”
 - Strict equality
 - **Type** and **value** are both equal
 - For an Object, its **value** is its location in the heap (“pointer”)



Class Activity:

What would be the output of the following code?

```
1  var x = 5;
2  console.log(x == 5);      // prints?
3  console.log(x != 5);      // prints?
4  console.log(x >= 5);      // prints?
5  console.log(x < 5);       // prints?
6
7  console.log(x == "5");     // prints?
8  console.log(x === "5");   // prints?
9  console.log(x != "5");     // prints?
10 console.log(x !== "5");    // prints?
11 console.log(x !== 5);      // prints?
12
13
```



Class Activity:

What would be the output of the following code?

```
1  var x = 5;
2  console.log(x == 5);      // prints: true
3  console.log(x != 5);      // prints: false
4  console.log(x >= 5);      // prints: true
5  console.log(x < 5);       // prints: false
6
7  console.log(x == "5");     // prints: true
8  console.log(x === "5");   // prints: false
9  console.log(x != "5");     // prints: false
10 console.log(x !== "5");    // prints: true
11 console.log(x !== 5);      // prints: false
12
13
```



Class Activity:

What would be the output of the following code?

```
1  var x = { name: "Foo", value: 5 };
2  var a = { name: "Foo", value: 5 };
3  var b = x;
4
5  console.log(a.name === x.name); // prints?
6  console.log(a.value === x.value); // prints?
7  console.log(a === x);           // prints?
8  console.log(b === x);           // prints?
9
10
11
12
13
```



Class Activity:

What would be the output of the following code?

```
1  var x = { name: "Foo", value: 5 };
2  var a = { name: "Foo", value: 5 };
3  var b = x;
4
5  console.log(a.name === x.name);    // prints: true
6  console.log(a.value === x.value);  // prints: true
7  console.log(a === x);              // prints: false
8  console.log(b === x);              // prints: true
9
10
11
12
13
```



Statements and Expressions: Call Expression

- Function calls have the form:
 - `functionName (argument1, argument2, argument3, ...)`
 - Invokes function referred by `functionName` with the given *arguments*
 - Same as many other languages



```
1 console.log("Foo");
2 alert("Foo");
3 setTimeout(alert, 1000, "Foo");
4 setInterval(alert, 1000, "Foo");
```


Statements and Expressions: Function Declaration

- Functions can be declared with the `function` keyword
 - Can accept arbitrary arguments
 - No need to specify the return type
 - Lexical scoping - functions can have local variables that inherit the local context at the time of declaration (*we will cover this in more detail later*)



```
1 function density(mass, width, height, length){
2     var volume = width * height * length;
3     return mass / volume;
4 };
5
6 density(10, 20, 5, 10);
```

Variable and Function Declaration: Hoisting

Variable and Function Declarations are **hoisted**

- **Processed before** other expressions in the program
- To avoid confusion, **best to put** Variable Declarations and Function Declarations **at the top** of the program



```
1 console.log(density); // prints: [Function: density]
2
3 function density(mass, width, height, length){
4     var volume = width * height * length;
5     return mass / volume;
6 }
```

Statements and Expressions: If, Else, Else If

- `if` statements are used to conditionally execute code
 - Has the form `if (condition) {expression} else {expression}`
 - `else` block is optional



```
1 if (temperature < 0) freezes = true;  
2 else freezes = false;
```

Statements and Expressions: If, Else, Else If

- Blocks of code can be grouped with {}



```
1  if (temperature < 0){  
2      freezes = true;  
3      boils = false;  
4  }  
5  else {  
6      freezes = false;  
7      boils = true;  
8  }  
9  
10  
11  
12
```

Statements and Expressions: If, Else, Else If

- `if` statements can be followed by multiple `else if` blocks



```
1  if (temperature < 0){
2      freezes = true;
3      boils = false;
4  }
5  else if (temperature < 100){
6      freezes = false;
7      boils = false;
8  }
9  else {
10     freezes = false;
11     boils = true;
12 }
```

Statements and Expressions: Switch

- `switch` statements can be used to handle many conditions

```
1  switch (group){  
2      case "child":  
3          fee = 0;  
4          break;  
5      case "adult":  
6          fee = 10;  
7          break;  
8      case "senior":  
9          fee = 4;  
10         break;  
11     default:  
12         break;  
13 }
```



Statements and Expressions: For

- `for` statements are used to repeat a block of code
- Similar to other languages, the `for` statement accepts 3 expressions:
 - Initial condition - run at the beginning of the loop
 - Termination condition - if this evaluates to true, the loop will exit
 - Increment expression - run after each iteration of the loop



```
1 for (      ;      ;      ){  
2     console.log(i);  
3 }
```

Statements and Expressions: For

- `for` statements are used to repeat a block of code
- Similar to other languages, the `for` statement accepts 3 expressions:
 - Initial condition - run at the beginning of the loop
 - Termination condition - if this evaluates to true, the loop will exit
 - Increment expression - run after each iteration of the loop



```
1 for (var i = 0; i < 10; i++){  
2     console.log(i);  
3 }
```


Statements and Expressions: For

- `for ... in` statements can be used to easily iterate through an object



```
1  var item = {  
2    a: 3,  
3    b: 1,  
4    c: 4  
5  }  
6  for (var key in item){  
7    console.log(key + " : " + item[key]);  
8  }  
9  
10  
11  
12  
13
```

Statements and Expressions: While, Do While

- `while` statements are similar to a `for` statement
 - Only the termination condition is specified
 - No initial condition or increment expression



```
1 while (i < 10){  
2     console.log(i);  
3     i++;  
4 }  
5  
6 do {  
7     console.log(i);  
8     i++;  
9 } while (i < 10);
```

Statements and Expressions: Try, Catch, Throw

- **try** - **catch** statements are used to handle errors
 - **try** block to enclose code that *might* throw an error
 - **catch** block to handle the error if thrown



```
1 try {  
2     myFunction(i);  
3 }  
4 catch (err){  
5     if (err.code === "FooError"){  
6         handleError(err);  
7     }  
8     else {  
9         throw err  
10    }  
11 }
```

Class Activity

1. Introduction to JavaScript
2. Data Types
3. Statements and Expressions
4. **Class Activity**



Class Activity:

Consider the function `getRandomInt(min, max)` below:

```
1 function getRandomInt(min, max){  
2   min = Math.ceil(min);  
3   max = Math.floor(max);  
4   return Math.floor(Math.random() * (max - min)) + min;  
5 };
```



Write the following functions:

1. `randomArray(n, min, max)`: returns an array of `n` random values generated between `min` and `max`
2. `sortArray(arr)`: returns an array containing all the values of `arr` sorted in ascending order