

DATA STRUCTURES & ALGORITHM IN JAVASCRIPT

Jayson N. Reales

Instructor

What is JavaScript

JavaScript was initially created to “*make web pages alive*”.

The programs in this language are called **scripts**. They can be written right in a web page's HTML and run automatically as the page loads.

Scripts are provided and executed as plain text. They don't need special preparation or compilation to run.

In this aspect, JavaScript is very different from another language called Java.

Why called JavaScript?

When JavaScript was created, it initially had another name: “**LiveScript**”. But Java was very popular at that time, so it was decided that positioning a new language as a “**younger brother**” of Java would help.

But as it evolved, JavaScript became a fully independent language with its own specification called **ECMAScript**, and now it has no relation to Java at all.

Today, JavaScript can **execute not only in the browser**, but **also on the server**, or actually on any device that has a special program called the **JavaScript engine**.

The browser has an embedded engine sometimes called a “**JavaScript virtual machine**”.

Different engines have different “codenames”. For example:

- **V8** – in Chrome, Opera and Edge.
- **SpiderMonkey** – in Firefox.
- ...There are other codenames like “Chakra” for IE, “JavaScriptCore”, “Nitro” and “SquirrelFish” for Safari, etc.

The terms above are good to remember because they are used in developer articles on the internet. We'll use them too. For instance, if “a feature X is supported by V8”, then it probably works in Chrome, Opera and Edge.

How do engines work?

Engines are complicated. But the basics are easy.

1. The engine (embedded if it's a browser) reads (“parses”) the script.
2. Then it converts (“compiles”) the script to machine code.
3. And then the machine code runs, pretty fast.

The engine applies optimizations at each step of the process. It even watches the compiled script as it runs, analyzes the data that flows through it, and further optimizes the machine code based on that knowledge.

What can in-browser JavaScript do?

Modern JavaScript is a “**safe**” programming language. It does not provide **low-level access** to memory or the CPU, because it was initially created for browsers which do not require it.

JavaScript’s capabilities greatly depend on the environment it’s running in. For instance, **Node.js** supports functions that allow JavaScript to read/write arbitrary files, perform network requests, etc.

In-browser JavaScript can do everything related to webpage manipulation, interaction with the user, and the webserver.

For instance, in-browser JavaScript is able to:

- Add new HTML to the page, change the existing content, modify styles.
- React to user actions, run on mouse clicks, pointer movements, key presses.
- Send requests over the network to remote servers, download and upload files (so-called AJAX and COMET technologies).
- Get and set cookies, ask questions to the visitor, show messages.
- Remember the data on the client-side (“local storage”).

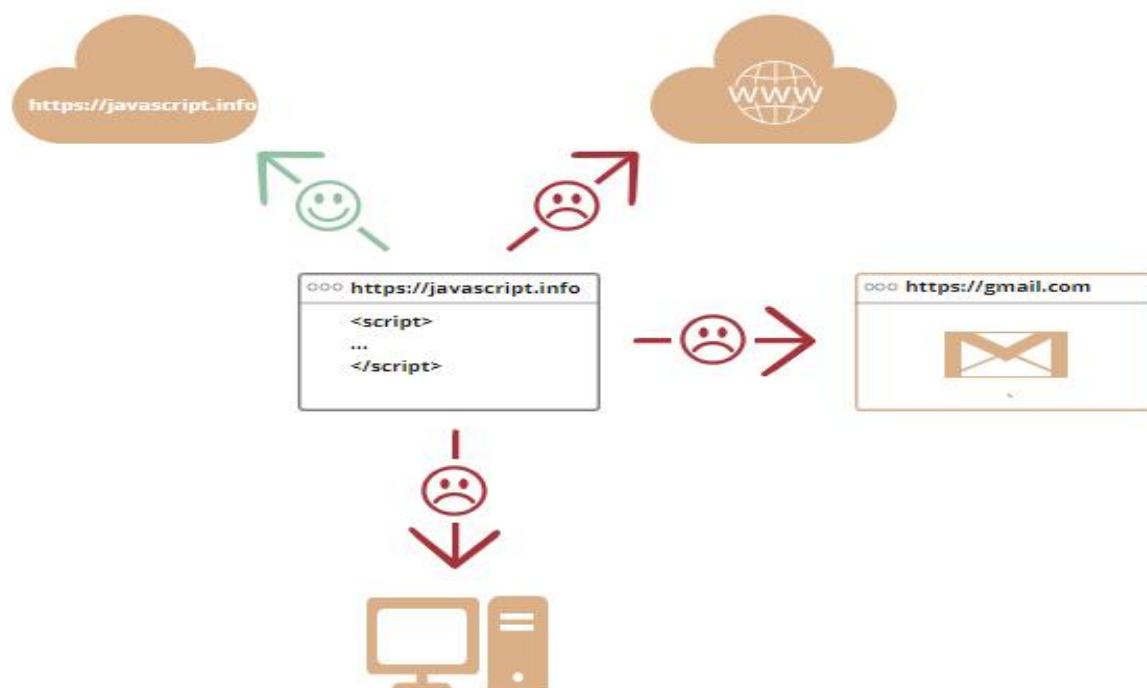
What CAN’T in-browser JavaScript do?

JavaScript’s abilities in the browser are limited to protect the user’s safety. The aim is to prevent an evil webpage from accessing private information or harming the user’s data.

Examples of such restrictions include:

- JavaScript on a webpage may not read/write arbitrary files on the hard disk, copy them or execute programs. It has no direct access to OS functions.
- Modern browsers allow it to work with files, but the access is limited and only provided if the user does certain actions, like “dropping” a file into a browser window or selecting it via an <input> tag.
- There are ways to interact with the camera/microphone and other devices, but they require a user’s explicit permission. So a JavaScript-enabled page may not sneakily enable a web-camera, observe the surroundings and send the information to the NSA.

- Different tabs/windows generally do not know about each other. Sometimes they do, for example when one window uses JavaScript to open the other one. But even in this case, JavaScript from one page may not access the other page if they come from different sites (from a different domain, protocol or port).
- This is called the “Same Origin Policy”. To work around that, both pages must agree for data exchange and must contain special JavaScript code that handles it. We’ll cover that in the tutorial.
- This limitation is, again, for the user’s safety. A page from <http://any site.com> which a user has opened must not be able to access another browser tab with the URL <http://gmail.com>, for example, and steal information from there.
- JavaScript can easily communicate over the net to the server where the current page came from. But its ability to receive data from other sites/domains is crippled. Though possible, it requires explicit agreement (expressed in HTTP headers) from the remote side. Once again, that’s a safety limitation.



Such limitations do not exist if JavaScript is used outside of the browser, for example on a server. Modern browsers also allow plugins/extensions which may ask for extended permissions.

What makes JavaScript unique?

There are at least three great things about JavaScript:

- Full integration with HTML/CSS.
- Simple things are done simply.
- Supported by all major browsers and enabled by default.

JavaScript is the only browser technology that combines these three things.

That's what makes JavaScript unique. That's why it's the most widespread tool for creating browser interfaces.

That said, JavaScript can be used to create servers, mobile applications, etc.

Languages “over” JavaScript

The syntax of JavaScript does not suit everyone's needs. Different people want different features.

That's to be expected, because projects and requirements are different for everyone.

So, recently a plethora of new languages appeared, which are transpiled (converted) to JavaScript before they run in the browser.

Modern tools make the transpilation very fast and transparent, actually allowing developers to code in another language and auto-converting it “under the hood”.

Examples of such languages:

- **CoffeeScript** is “syntactic sugar” for JavaScript. It introduces shorter syntax, allowing us to write clearer and more precise code. Usually, Ruby devs like it.
- **TypeScript** is concentrated on adding “strict data typing” to simplify the development and support of complex systems. It is developed by Microsoft.
- **Flow** also adds data typing, but in a different way. Developed by Facebook.
- **Dart** is a standalone language that has its own engine that runs in non-browser environments (like mobile apps), but also can be transpiled to JavaScript. Developed by Google.
- **Brython** is a Python transpiler to JavaScript that enables the writing of applications in pure Python without JavaScript.
- **Kotlin** is a modern, concise and safe programming language that can target the browser or Node.

There are more. Of course, even if we use one of these transpiled languages, we should also know JavaScript to really understand what we're doing.

Language Syntax

Language syntax refers to the set of rules that dictate how programs written in a particular programming language must be structured. This can include **rules** for how to **declare variables**, how to **call functions**, how to **structure control flow statements**, and so on. Syntax varies significantly between different programming languages, so it is critical to grasp the specific syntax of the language you are using. It's similar to grammar in human languages - putting words in the wrong order or including extraneous punctuation can make a sentence hard to understand, and the same applies to programming. **Incorrect syntax leads to syntax errors** which prevent your code from executing.

JavaScript Syntax

JavaScript syntax comprises a set of rules that define how to construct a JavaScript code. JavaScript can be implemented using JavaScript statements that are placed within the `<script> ... </script>` HTML tags in a web page. You can place the `<script>` tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the `<head>` tags. The `<script>` tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

```
<script>  
  // This is the part where you put your JavaScript code  
</script>
```

JavaScript Values

In JavaScript, you can have two types of values.

- Fixed values (Literals)
- Variables (Dynamic values)

JavaScript Literals

In the below code, 10 is a Number literal and 'Hello' is a string literal.

```
<script>
    document.write(10); // Number literal
    document.write("<br />"); // To add line-break
    document.write("Hello"); //String literal
</script>
```

Output ???

JavaScript Variables

In JavaScript, variables are used to store the dynamic data.

You can use the below keyword to define variables in JavaScript.

- var
- let
- const

You can use the assignment operator (equal sign) to assign values to the variable.

In the below code, variable a contains the numeric value, and variable b contains the text (string).

```
<script>
    let a = 5; // Variable Declaration
    document.write(a); // Using variable
    document.write("<br>");
```

```
    let b = "One";
```

```
    document.write(b);
```

```
</script>
```

Output???

Whitespace and Line Breaks

JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs. You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

Semicolons are Optional

Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if each of your statements are placed on a separate line. For example, the following code could be written without semicolons.

Case Sensitivity

JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

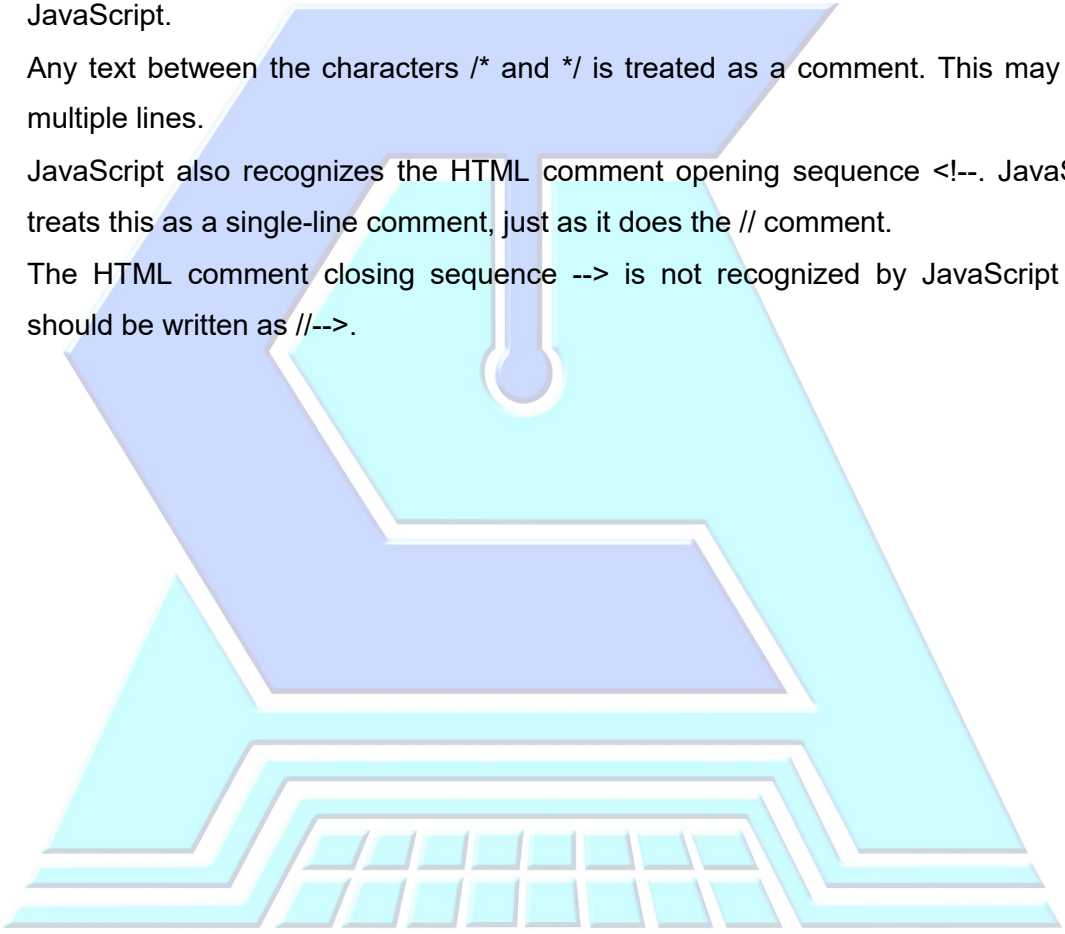
So the identifiers **Time** and **TIME** will convey different meanings in JavaScript.

JavaScript and Camel Case

- **Pascal Case** – We can create variables like SmartWatch, MobileDevice, WebDrive, etc. It represents the upper camel case string.
- **Lower Camel Case** – JavaScript allows developers to use variable names and expression names like smartwatch, mobileDevice, webDriver, etc. Here the first character is in lowercase.
- **Underscore** – We can use underscore while declaring variables like smart_watch, mobile_device, web_driver, etc.

Comments in JavaScript

JavaScript supports both C-style and C++-style comments, Thus –

- Any text between a `//` and the end of a line is treated as a comment and is ignored by JavaScript.
 - Any text between the characters `/*` and `*/` is treated as a comment. This may span multiple lines.
 - JavaScript also recognizes the HTML comment opening sequence `<!--`. JavaScript treats this as a single-line comment, just as it does the `//` comment.
 - The HTML comment closing sequence `-->` is not recognized by JavaScript so it should be written as `//-->`.
- 



JAVASCRIPT OPERATORS

Operators in JavaScript

JavaScript contains various arithmetic, logical, bitwise, etc. operators. We can use any operator in JavaScript.

1. Arithmetic Operators

1. +
2. -
3. /
4. *
5. %

2. Logical Operator

1. AND Operator or &&
2. OR Operator or ||
3. NOT Operator or !

Example

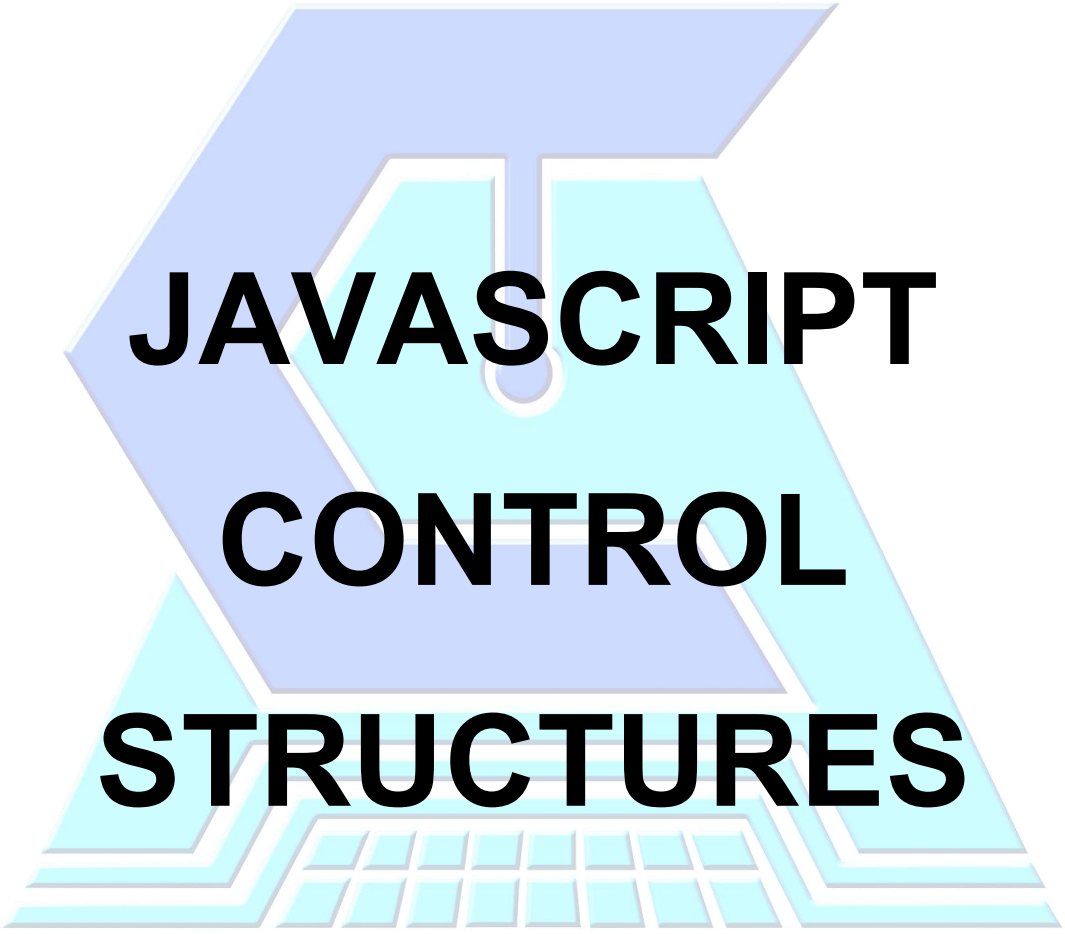
In this example, we have defined var1 and va2 and initialized them with number values. After that, we use the “*” operator to get the multiplication result of var1 and var2.

<script>

```
var1 = 10  
var2 = 20  
var3 = var1 * var2;  
var4 = 10 + 20;  
console.log(var3, " ",var4);
```

</script>

Output???



JAVASCRIPT CONTROL STRUCTURES

Control Structures

Control structures are fundamental elements in most programming languages that facilitate the flow of control through a program. There are three main types of control structures: Sequential, Selection and Iteration. Sequential control structures are the default mode where instructions happen one after another. Selection control structures (often called “conditional” or “decision” structures) allow one set of instructions to be executed if a condition is true and another if it’s false. These typically include if...else statements. Iteration control structures (also known as “loops”) allow a block of code to be repeated multiple times. Common loop structures include for, while, and do...while loops. All these control structures play a vital role in shaping the program logic.

JavaScript Control Structures

- If Statement
- Using If-Else Statement
- Using Switch Statement
- Using the Ternary Operator (Conditional Operator)
- Using For loop

Approach 1: If Statement

In this approach, we are using an if statement to check a specific condition, the code block gets executed when the given condition is satisfied.

Syntax:

```
if ( condition_is_given_here ) {  
    // If the condition is met, the code will get executed.  
}
```

Approach 2: If-Else Statement

The if-else statement will perform some action for a specific condition. If the condition met, then a particular code of action will be executed otherwise it will execute another code of action that satisfies that particular condition.

Syntax:

```
if (condition1) {  
    // Executes when condition1 is true  
}  
else {  
    // Executes when condition1 is false  
}
```

Approach 3: Switch Statement

The switch case statement in JavaScript is also used for decision-making purposes. In some cases, using the switch case statement is seen to be more convenient than if-else statements.

```
switch (expression) {
```

```
    case value1:
```

```
        statement1;
```

```
        break;
```

```
    case value2:
```

```
        statement2;
```

```
        break;
```

```
    case valueN:
```

```
        statementN;
```

```
        break;
```

```
    default:
```

```
        statementDefault;
```

```
}
```

Approach 4: Using the Ternary Operator (Conditional Operator)

The conditional operator, also referred to as the ternary operator (? :), is a shortcut for expressing conditional statements in JavaScript.

Syntax:

condition ? value if true : value if false

Approach 5: Using For Loop

In this approach, we are using for loop in which the execution of a set of instructions repeatedly until some condition evaluates and becomes false

Syntax:

```
for (statement 1; statement 2; statement 3) {  
    // Code here . . .  
}
```

Programming Fundamentals

Programming Fundamentals are the basic concepts and principles that form the foundation of any computer programming language. These include understanding variables, which store data for processing, control structures such as loops and conditional statements that direct the flow of a program, data structures which organize and store data efficiently, and algorithms which step by step instructions to solve specific problems or perform specific tasks.