Why Study JavaScript?

JavaScript is one of the **3 languages** all web developers **must** learn:

1. **HTML** to define the content of web pages
2. **CSS** to specify the layout of web pages
3. **JavaScript** to program the behavior of web pages

Web pages are not the only place where JavaScript is used. Many desktop and server programs use JavaScript. Node.js is the best known. Some databases, like MongoDB and CouchDB, also use JavaScript as their programming language.

- JavaScript Can Change HTML Content
- JavaScript Can Change HTML Attribute Values
- JavaScript Can Change HTML Styles (CSS)
- JavaScript Can Hide or Show HTML Elements

---

1. JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997.
2. ECMAScript is the official name of the language.
3. From 2015 ECMAScript is named by year (ECMAScript 2015).
4. latest version is ES10 launched in June 2019.
5. JavaScript is most widely used now a days and most preferable language for development as front-end as well back-end(Nodejs),if we want to learn JavaScript we need to understand the basics of JavaScript and ECMAScript(ES), ECMAScript is an simple standard for JavaScript and adding new features to JavaScript,
6. ECMAScript is a subset of JavaScript. JavaScript is basically ECMAScript at its core but builds upon it.

JavaScript Where To write

The <script> Tag

In HTML, JavaScript code must be inserted between `<script>` and `</script>` tags.

Scripts can be placed in the `<body>`, or in the `<head>` section of an HTML page, or in both.

```
<!DOCTYPE html>
<html>

<head>
<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>
<body>

<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```

External JavaScript

Scripts can also be placed in external files:

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>

<script src="/js/myScript1.js"></script>
```

```
<script src="https://www.w3schools.com/js/myScript1.js"></script>
```

External scripts cannot contain `<script>` tags.

External JavaScript Advantages

Placing scripts in external files has some advantages:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

Using innerHTML

To access an HTML element, JavaScript can use the `document.getElementById(id)` method.

The `id` attribute defines the HTML element. The `innerHTML` property defines the HTML content:

Using console.log()

For debugging purposes, you can use the `console.log()` method to display data.

In HTML, JavaScript programs are executed by the web browser.

Semicolons ;

Semicolons separate JavaScript statements.

```
var a, b, c;     // Declare 3 variables
var person = "John Doe", carName = "Volvo", price = 200;
```

When separated by semicolons, multiple statements on one line are allowed:

```
a = 5; b = 6; c = a + b;
```

JavaScript White Space

JavaScript ignores multiple spaces. You can add white space to your script to make it more readable.

The following lines are equivalent:

```
var person = "Hege";
var person="Hege";
```

A good practice is to put spaces around operators (= + - * /):

```
var x = y + z;
```

JavaScript Line Length and Line Breaks

For best readability, programmers often like to avoid code lines longer than 80 characters.

If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

```
document.getElementById("demo").innerHTML =
"Hello Dolly!";
```

JavaScript Keywords (always in lowercase)

A few common:

Keyword	Description
break	Terminates a switch or a loop
continue	Jumps out of a loop and starts at the top
debugger	Stops the execution of JavaScript, and calls (if available) the debugging function
do ... while	Executes a block of statements, and repeats the block, while a condition is true

for Marks a block of statements to be executed, as long as a condition is true

function Declares a function

if ... else Marks a block of statements to be executed, depending on a condition

return Exits a function

switch Marks a block of statements to be executed, depending on different cases

try ... catch Implements error handling to a block of statements

var Declares a variable

JavaScript uses the `var` keyword to **declare** variables.

Operators & variable naming conventions are almost same as in Java.

JavaScript Character Set

JavaScript uses the **Unicode** character set.

Unicode covers (almost) all the characters, punctuations, and symbols in the world.

JavaScript Comments(same like as in java)

Single Line Comments

Single line comments start with `//`.

Multi-line Comments

Multi-line comments start with `/*` and end with `*/`.

JavaScript is case-sensitive language.

Var or VAR Cannot be used for “var”

It's a good programming practice to declare all variables at the beginning of a script.

Re-Declaring JavaScript Variables

If you re-declare a JavaScript variable, it will not lose its value.

The variable `carName` will still have the value "Volvo" after the execution of these statements:

```
var carName = "Volvo";
var carName;
```

JavaScript String Operators

The `+` operator can also be used to add (concatenate) strings.

```
Expression processing happens left to right and numbers after a string
are concatenated and for numbers before string arithmetic calculation
happens.
```

```
var x = "5" + 2 + 3; //523 /concatenation

var x = 2 + 3 + "5"; //55  /arithmetic & concatenation

var x = 16 + 4 + "Volvo"; //20Volvo

var x = "Volvo" + 16 + 4; //Volvo164
```

When adding a number and a string, JavaScript will treat the number as a string.

Equal to operator:

== equal to (Relaxed equal)

=== equal value and equal type (Strict equal)

!= not equal

!== not equal value or not equal type

JavaScript Data Types

JavaScript variables can hold many **data types**: numbers, strings, boolean, objects (array, objects, null) ,undefined and more:

```
var length = 16;                                // Number
var lastName = "Johnson";                       // String

var x = (5 == 6);                                //Boolean

var cars = ["Saab", "Volvo", "BMW"];             //Array
var x = {firstName:"John", lastName:"Doe"};     // Object

var y =null // null

var z // undefined
```

JavaScript Types are Dynamic

JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

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

JavaScript Strings

A string (or a text string) is a series of characters like "John Doe".

Strings are text, written within double or single quotes:

```
"John Doe”,  'John Doe'
```

Escape Character

The solution to avoid this problem, is to use the **backslash escape character**.

The backslash (\) escape character turns special characters into string characters:

Code	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

The sequence `\"` inserts a double quote in a string:

```
var x = "We are the so-called \"Vikings\" from the north.";
```

Strings Can be Objects

Normally, JavaScript strings are primitive values, created from literals:

```
var firstName = "John";
```

But strings can also be defined as objects with the keyword `new`:

```
var firstName = new String("John");
```

```
var x = "John";
var y = new String("John");

// typeof x will return string
// typeof y will return object
```

Don't create strings as objects. It slows down execution speed.
The `new` keyword complicates the code. This can produce some unexpected results:

```
var x = "John";
var y = new String("John");

// (x == y) is true because x and y have equal values

// (x === y) is false because x and y have different types (string and object)
```

Or even worse Objects cannot be compared:

```
var x = new String("John");
var y = new String("John");

// (x == y) is false because x and y are different objects

// (x === y) is false because x and y are different objects
```

Note the difference between `(x==y)` and `(x===y)`.
Comparing two JavaScript objects will **always** return `false`.

String Methods and Properties

Primitive values, like "John Doe", cannot have properties or methods (because they are not objects).

But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

String Length

The `length` property returns the length of a string:

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

Finding a String in a String

The `indexOf()` method returns the index of (the position of) the `first` occurrence of a specified text in a string:

I string is not found returns -1.

```
var str = "Please locate where 'locate' occurs!";

var pos = str.indexOf("Please");   //0
var pos = str.indexOf("locate");   //7

var pos = str.indexOf("xyz");   //-1

var pos = str.indexOf("please");   //-1
```

JavaScript counts positions from zero.
0 is the first position in a string, 1 is the second, 2 is the third

The `lastIndexOf()` method returns the index of the **last** occurrence of a specified text in a string:

```
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate"); //21
```

Both `indexOf()`, and `lastIndexOf()` return -1 if the text is not found.

Both methods accept a second parameter as the starting position for the search:

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate", 15);
```

Searching for a String in a String

```
var str = "Please locate where 'locate' occurs!";
var pos = str.search("locate");  //7
```

Did You Notice?

The two methods, `indexOf()` and `search()`, are **equal?**

They accept the same arguments (parameters), and return the same value?

The two methods are **NOT** equal. These are the differences:

- The `search()` method cannot take a second start position argument.
- The `indexOf()` method cannot take powerful search values (regular expressions).

Numbers are written with or without decimals:

```
10.50, 1001
```

```
var y = 123e5;      // 12300000
var z = 123e-5;     // 0.00123
```

"typeof" operator:

The `typeof` operator returns the type of a variable or an expression:

```
typeof ""             // Returns "string"
typeof "John"         // Returns "string"
typeof "John Doe"     // Returns "string"

typeof 3.14           // Returns "number"
typeof (3 + 4)        // Returns "number"
```

Undefined

In JavaScript, a variable declared without a value, has the value `undefined`. The type is also `undefined`.

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

Any variable can be emptied, by setting the value to `undefined`. The type will also be `undefined`.

```
car = undefined;    // Value is undefined, type is undefined
```

Null

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`.

You can empty an object by setting it to `null or undefined`:

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
person = null;    // Now value is null, but type is still an object

or

person = undefined;   // Now both value and type is undefined
```

Difference Between Undefined and Null

`undefined` and `null` are equal in value but different in type:

```
typeof undefined           // undefined
typeof null                // object

null === undefined         // false
null == undefined          // true
```

Primitive Data

A primitive data value is a single simple data value with no additional properties and methods.

The `typeof` operator can return one of these primitive types:

- `string`
- `number`
- `boolean`
- `undefined`

```
typeof "John"                 // Returns "string"
typeof 3.14                   // Returns "number"
typeof true                   // Returns "boolean"
typeof false                  // Returns "boolean"
typeof x                      // Returns "undefined" (if x has no value)
```

Complex Data

The `typeof` operator can return one of two complex types:

- `function`
- `object`

The `typeof` operator returns "object" for objects, arrays, and null.

The `typeof` operator does not return "object" for functions.

```
typeof {name:'John', age:34} // Returns "object"
typeof [1,2,3,4]             // Returns "object" (not "array", see note
below)
typeof null                  // Returns "object"
typeof function myFunc(){}   // Returns "function"
```

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

JavaScript Functions

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is defined with the `function` keyword, followed by a **name**, followed by parentheses **()**.

Inside the function, the arguments (the parameters) behave as local variables.

```
function myFunction(p1, p2) {
  return p1 * p2;   // The function returns the product of p1 and p2
}
```

Function Invocation

The code inside the function will execute when "something" **invokes** (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

Why Functions?

You can reuse code: Define the code once, and use it many times.

You can use the same code many times with different arguments, to produce different results.

Functions Used as Variable Values

Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations.

```
var x = toCelsius(77);
var text = "The temperature is " + x + " Celsius";
```

Local Variables

Variables declared within a JavaScript function, become **LOCAL** to the function.

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

Local variables are created when a function starts, and deleted when the function is completed.

```
// code here can NOT use carName

function myFunction() {
  var carName = "Volvo";
  // code here CAN use carName
}
```

```
// code here can NOT use carName
```

JavaScript Objects: stores key-value pair

JavaScript objects are containers for **named values** called properties or methods.

```
var person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

Object Properties

The **name:values** pairs in JavaScript objects are called **properties**:

Property	Property Value
firstName	John
lastName	Doe

Accessing Object Properties

You can access object properties in two ways:

objectName.propertyName *or*

objectName["propertyName"]

The **this** Keyword

In a function definition, `this` refers to the "owner" of the function.

In the example above, `this` is the **person object** that "owns" the `fullName` function.

In other words, `this.firstName` means the `firstName` property of **this object**.

Accessing Object Methods

```
name = person.fullName();
```

If you access a method **without** the () parentheses, it will return the **function definition**:

```
name = person.fullName;
```

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.

JavaScript Events

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

In the example above, the JavaScript code changes the content of the element with id="demo".

In the next example, the code changes the content of its own element (using `this.innerHTML`):

```
<button onclick="this.innerHTML = Date()">The time is?</button>
```

Common HTML Events:

Here is a list of some 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

What can JavaScript Do?

Event handlers can be used to handle, and verify, user input, user actions, and browser actions:

- Things that should be done every time a page loads
- Things that should be done when the page is closed
- Action that should be performed when a user clicks a button
- Content that should be verified when a user inputs data