JavaScript – Part 1

Lesson 01: **Introduction**

- ✓ JavaScript is the world's most popular programming language.
- ✓ JavaScript is the programming language of the Web.
- ✓ JavaScript is easy to learn.
- ✓ This tutorial will teach you JavaScript from basic to advance.

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

This tutorial covers every version of JavaScript:

- The Original JavaScript ES1 ES2 ES3 (1997-1999)
- The First Main Revision ES5 (2009)
- The Second Revision ES6 (2015)
- The Yearly Additions (2016, 2017, 2018)

Commonly Asked Questions

- How do I get JavaScript?
- Where can I download JavaScript?
- How much do full stack JavaScript developers make?
- Are JavaScript coders in demand?
- Who is father of JavaScript? **Brendan Eich**

Lesson 02: JavaScript Can Change HTML Content (Tags)

Text Editor:

- Download Visual Studio.

One of many JavaScript HTML methods is `getElementById()`

`Example:`

```
<!DOCTYPE html>

<html>

<body>


<h2>What Can JavaScript Do?</h2>


<p id="demo">JavaScript can change HTML content.</p>


<button type="button" onclick='document.getElementById("demo").innerHTML = "Hello JavaScript!"'>Click Me!</button>


</body>

</html>
```

Note:

JavaScript accepts both double and single quotes:

Lesson 03. JavaScript Can Change HTML Attribute Values

In this example JavaScript changes the value of the `src` (source) attribute of an `<img>` tag:

Example:

```
<!DOCTYPE html>

<html>

<body>


<h2>What Can JavaScript Do?</h2>

<p>JavaScript can change HTML attribute values.</p>


<p>In this case JavaScript changes the value of the src (source) attribute of an image.</p>

<img id="myImage" src="off.png" style="width:100px" >

<br>

<button onclick="document.getElementById('myImage').src='on.png'">Turn on the light</button>

<br>

<button onclick="document.getElementById('myImage').src='off.png'">Turn off the light</button>

</body>

</html>
```

Lesson 04. JavaScript Can Change HTML Styles (CSS)

Example:

```
<!DOCTYPE html>

<html>

<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change the style of an HTML element.</p>

<button type="button" onclick="document.getElementById('demo').style.fontSize='35px'">Click Me!</button>


</body>

</html>
```

Lesson 05: JavaScript Can Hide HTML Elements (Tags)

```
<!DOCTYPE html>

<html>

<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can hide HTML elements.</p>

<button type="button" onclick="document.getElementById('demo').style.display='none'">Click Me!</button>

</body>

</html>
```

JavaScript Can Show HTML Elements

```
<!DOCTYPE html>

<html>

<body>


<h2>What Can JavaScript Do?</h2>


<p>JavaScript can show hidden HTML elements.</p>


<p id="demo" style="display:none">Hello JavaScript!</p>


<button type="button" onclick="document.getElementById('demo').style.display='block'">Click
Me!</button>


</body>

</html>
```

Lesson 06: Where to (Head or Body)

The <script> Tag

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

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript in Body</h2>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
</body>
</html>
```

Note:

Old JavaScript examples may use a type attribute: <script type="text/javascript">.

The type attribute is not required. JavaScript is the default scripting language in HTML.

JavaScript in <head> or <body>

You can place any number of scripts in an HTML document.

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

JavaScript in <head>

In this example, a JavaScript `function` is placed in the `<head>` section of an HTML page.

The function is invoked (called) when a button is clicked:

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}</script></head>
<body>
<h2>Demo JavaScript in Head</h2>
<p id="demo">A Paragraph.</p>
<button type="button" onclick="myFunction()">Try it</button>
</body>
```

```
</html>
```

JavaScript in <body>

In this example, a JavaScript `function` is placed in the `<body>` section of an HTML page.

The function is invoked (called) when a button is clicked:

```
<!DOCTYPE html>

<html>

<body>

<h2>Demo JavaScript in Body</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

<script>

function myFunction() {

  document.getElementById("demo").innerHTML = "Paragraph changed.";

}

</script>

</body>

</html>
```

Lesson 07: Where to (External JavaScript)

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

Scripts can also be placed in external files:

External file: myScript.js

External scripts are practical when the same code is used in many different web pages.

JavaScript files have the file extension **.js**.

To use an external script, put the name of the script file in the `src` (source) attribute of a `<script>` tag:

Example

```
<script src=”myScript.js”> </script>
```

```
<!DOCTYPE html>
<html>
    <head>

    </head>
<body>

<h2>Demo External JavaScript</h2>

<script>

    document.write("I am a testing External JavaScript");

</script>

</body>
</html>
```

Example2:

```
<html>

<body>

    <h2>Demo External JavaScript</h2>


    <script src="05.1.myScript.js"> </script>
</body>
</html>
```

Note:

Create External file .js

Type this code and save .js

```
Document.write("I am a testing External JavaScript");
```

Note:

To add several script files to one page - use several script tags:

Example

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

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

External References

An external script can be referenced in 3 different ways:

- With a full URL (a full web address)
- With a file path (like /js/)
- Without any path

This example uses a **full URL** to link to myScript.js:

```
<script src="https://www.sahalsoftware.com/js/myScript.js"></script>
```

This example uses a **file path** to link to myScript.js:

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

This example uses no path to link to myScript.js:

Example

```
<script src=”myScript.js”> </script>
```

Lesson 08: JavaScript Display Possibilities

JavaScript can "display" data in different ways:

- Writing into an HTML element, using `innerHTML`.
- Writing into the HTML output using `document.write()`.
- Writing into an alert box, using `window.alert()`.
- Writing into the browser console, using `console.log()`.

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:

```
<!DOCTYPE html>

<html>

<body>

<h2>My First Web Page</h2>

<p>My First Paragraph.</p>
```

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

<script>

document.getElementById("demo").innerHTML = 5 + 6;

</script>

</body>

</html>
```

Note:

Changing the innerHTML property of an HTML element is a common way to display data in HTML.

Using document.write()

For testing purposes, it is convenient to use `document.write()`:

```
<!DOCTYPE html>

<html>

<body>


<h2>My First Web Page</h2>

<p>My first paragraph.</p>

<p>Never call document.write after the document has finished loading.

It will overwrite the whole document.</p>

<script>
```

```
document.write(5 + 6);

</script>

</body>

</html>
```

Note:

Using document.write() after an HTML document is loaded, will **delete all existing HTML**:

Example:

```
<!DOCTYPE html>

<html>

<body>

<h2>My First Web Page</h2>

<p>My first paragraph.</p>

<button type="button" onclick="document.write(5 + 6)">Try it</button>

</body>

</html>
```

Note:

The document.write() method should only be used for testing.

Using window.alert()

You can use an alert box to display data:

```
<!DOCTYPE html>
<html>
<body>

<h2>My First Web Page</h2>
<p>My first paragraph.</p>
<script>
window.alert(5 + 6);
</script>
</body>
</html>
```

Note:

You can skip the `window` keyword.

In JavaScript, the window object is the global scope object, which means that variables, properties, and methods by default belong to the window object. This also means that specifying the `window` keyword is optional:

Using console.log()

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

```
<!DOCTYPE html>

<html>

<body>

<h2>Activate Debugging</h2>

<p>F12 on your keyboard will activate debugging.</p>

<p>Then select "Console" in the debugger menu.</p>

<p>Then click Run again.</p>

<script>

console.log(5 + 6);

</script>

</body>

</html>
```

JavaScript Print

JavaScript does not have any print object or print methods.

You cannot access output devices from JavaScript.

The only exception is that you can call the `window.print()` method in the browser to print the content of the current window.

```
<!DOCTYPE html>

<html>

<body>

<h2>The window.print() Method</h2>

<p>Click the button to print the current page.</p>

<button onclick="window.print()">Print this page</button>

</body>

</html>
```

Lesson 09: JavaScript Statements

JavaScript Programs

A **computer program** is a list of "instructions" to be "executed" by a computer.

In a programming language, these programming instructions are called **statements**.

A **JavaScript program** is a list of programming **statements**.

JavaScript Statements

JavaScript statements are composed of:

Values, Operators, Expressions, Keywords, and Comments.

This statement tells the browser to write "Hello Sahalsoftware." inside an HTML element with id="demo":

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Statements</h2>

<p>In HTML, JavaScript statements are executed by the browser.</p>

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

<script>

document.getElementById("demo").innerHTML = "Hello Sahalsoftware.";

</script>

</body>

</html>
```

Note:

Most JavaScript programs contain many JavaScript statements.

The statements are executed, one by one, in the same order as they are written.

JavaScript programs (and JavaScript statements) are often called JavaScript code.

Semicolons ;

Semicolons separate JavaScript statements.

Add a semicolon at the end of each executable statement:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Statements</h2>
<p>JavaScript statements are separated by semicolons.</p>
<p id="demo1"></p>
<script>
let a, b, c;
a = 5;
b = 6;
c = a + b;
document.getElementById("demo1").innerHTML = c;
</script>
</body>
</html>
```

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

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

Same as:

```
a = 5;          // Assign the value 5 to a
b = 6;          // Assign the value 6 to b
c = a + b;      // Assign the sum of a and b to c
```

Note:

Sometimes, you might see examples without semicolons. Ending statements with semicolon is not required, but highly recommended.

Lesson 10. 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:

Let person = “Ahmed”

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

Let 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:

```
<!DOCTYPE html>

<html>

<body>



<h2>JavaScript Statements</h2>

<p>

The best place to break a code line is after an operator or a comma.

</p>

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

<script>

document.getElementById("demo").innerHTML =

"Hello Sahalsoftware!";

</script>

</body>

</html>
```

Lesson 11. JavaScript Code Blocks

JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.

The purpose of code blocks is to define statements to be executed together.

One place you will find statements grouped together in blocks, is in JavaScript functions:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Statements</h2>

<p>JavaScript code blocks are written between { and }</p>

<button type="button" onclick="myFunction()">Click Me!</button>

<p id="demo1"></p>

<p id="demo2"></p>

<script>

function myFunction() {

  document.getElementById("demo1").innerHTML = "Hello Mohamed!";

  document.getElementById("demo2").innerHTML = "How are you?";

}

</script>

</body>

</html>
```

Lesson 12: JavaScript Comments

JavaScript comments can be used to explain JavaScript code, and to make it more readable.

JavaScript comments can also be used to prevent execution, when testing alternative code.

Single Line Comments

Single line comments start with `//`.

Any text between `//` and the end of the line will be ignored by JavaScript (will not be executed).

This example uses a single-line comment before each code line:

```
<!DOCTYPE html>

<html>

<body>

<h1 id="myH"></h1>

<p id="myP"></p>

<script>

// Change heading:

document.getElementById("myH").innerHTML = "JavaScript Comments";

// Change paragraph:

document.getElementById("myP").innerHTML = "My first paragraph.";

</script>

</body>

</html>
```

This example uses a single line comment at the end of each line to explain the code:

```
let x = 5;       // Declare x, give it the value of 5
let y = x + 2;   // Declare y, give it the value of x + 2
```

Multi-line Comments

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

Any text between `/*` and `*/` will be ignored by JavaScript.

This example uses a multi-line comment (a comment block) to explain the code:

```
<!DOCTYPE html>

<html>

<body>

<h1 id="myH"></h1>

<p id="myP"></p>

<script>

/*

The code below will change

the heading with id = "myH"

and the paragraph with id = "myP"

*/

document.getElementById("myH").innerHTML = "JavaScript Comments";

document.getElementById("myP").innerHTML = "My first paragraph.";

</script>

</body>

</html>
```

Using Comments to Prevent Execution

Using comments to prevent execution of code is suitable for code testing.

Adding `//` in front of a code line changes the code lines from an executable line to a comment.

This example uses // to prevent execution of one of the code lines:

```
//document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

This example uses a comment block to prevent execution of multiple lines:

```
/*
document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
*/
```

Lesson 13: JavaScript Variables

What are Variables?

Variables are containers for storing data (storing data values).

3 Ways to Declare a JavaScript Variable:

- Using `var`
- Using `let`

- Using `const`

In this example, `x`, `y`, and `z`, are variables, declared with the `var` keyword:

```
var x = 5;
var y = 6;
var z = x + y;
```

In this example, `x`, `y`, and `z`, are variables, declared with the `let` keyword:

```
let x = 5;
let y = 6;
let z = x + y;
```

In this example, `x`, `y`, and `z`, are variables, declared with the `const` keyword:

```
const x = 5;
const y = 6;
const z = x + y;
```

From all the examples above, you can guess:

- x stores the value 5
- y stores the value 6
- z stores the value 11

When to Use JavaScript var?

Always declare JavaScript variables with `var`, `let`, or `const`.

The `var` keyword is used in all JavaScript code from 1995 to 2015.

The `let` and `const` keywords were added to JavaScript in 2015.

If you want your code to run in older browser, you must use `var`.

When to Use JavaScript const?

If you want a general rule: always declare variables with `const`.

If you think the value of the variable can change, use `let`.

Just Like Algebra

Just like in algebra, variables hold values:

Let x = 5;

Let y = 6;

Just like in algebra, variables are used in expressions:

Let z = x + y;

From the example above, you can guess that the total is calculated to be 11.

Note

Variables are containers for storing values.

LET

The `let` keyword was introduced in ES6 (2015).

Variables defined with `let` cannot be Redeclared.

Variables defined with `let` must be Declared before use.

Variables defined with `let` have Block Scope.

Cannot be Redeclared

Variables defined with `let` cannot be **redeclared**.

You cannot accidentally redeclare a variable.

Example:

```
let x = "John Doe";

let x = 0;

// SyntaxError: 'x' has already been declared
```

With `var` you can:

```
var x = "John Doe";

var x = 0;
```

Block Scope

Before ES6 (2015), JavaScript had only **Global Scope** and **Function Scope**.

ES6 introduced two important new JavaScript keywords: `let` and `const`.

These two keywords provide **Block Scope** in JavaScript.

Variables declared inside a **{ }** block cannot be accessed from outside the block:

Example

```
{
    Let x = 2;
}
// x can NOT be used here
```

Variables declared with the var keyword can NOT have block scope.

Variables declared inside a { } block can be accessed from outside the block.

Example

```
{
    var x = 2;
}
// x can be used here
```

Redeclaring Variables

Redeclaring a variable using the var keyword can impose problems.

Redeclaring a variable inside a block will also redeclare the variable outside the block:

Example

```
<!DOCTYPE html>

<html>

<body>

<h2>Redeclaring a Variable Using var</h2>

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

<script>

var  x = 10;

// Here x is 10

{

var x = 2;

// Here x is 2

}



// Here x is 2

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>
```

Redeclaring a variable using the `let` keyword can solve this problem.

Redeclaring a variable inside a block will not redeclare the variable outside the block:

Example

```
<!DOCTYPE html>

<html>

<body>

<h2>Redeclaring a Variable Using let</h2>

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

<script>

let  x = 10;

// Here x is 10

{

  let x = 2;

  // Here x is 2

}

// Here x is 10

document.getElementById("demo").innerHTML = x;

</script>


</body>

</html>
```

Browser Support

The `let` keyword is not fully supported in Internet Explorer 11 or earlier.

Re-declaring - var

Redeclaring a JavaScript variable with `var` is allowed anywhere in a program:

Example:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript let</h2>

<p>Redeclaring a JavaScript variable with <b>var</b> is allowed anywhere in a program:</p>

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

<script>

var x = 2;

// Now x is 2



var x = 3;

// Now x is 3

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

</script>

</body>

</html>

With `let`, redeclaring a variable in the same block is NOT allowed:

```
var x = 2;    // Allowed
let x = 3;    // Not allowed

{
let x = 2;    // Allowed
let x = 3;    // Not allowed
}

{
let x = 2;    // Allowed
var x = 3;    // Not allowed
}
```

Redeclaring a variable with `let`, in another block, is allowed:

Example

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript let</h2>

Redeclaring a variable with let, in another scope, or in another block, is allowed:

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


<script>

let x = 2;   // Allowed


{

  let x = 3;   // Allowed

}


{

  let x = 4;   // Allowed

}

document.getElementById("demo").innerHTML = x;

</script>


</body>

</html>
```

Let Hoisting

Variables defined with `var` are **hoisted** to the top and can be initialized at any time.

Meaning: You can use the variable before it is declared:

```
<!DOCTYPE html>

<html>

<body>


<h2>JavaScript Hoisting</h2>


<p>With <b>var</b>, you can use a variable before it is declared:</p>


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


<script>

carName = "Volvo";

document.getElementById("demo").innerHTML = carName;

var carName;

</script>


</body>

</html>
```

Variables defined with `let` are also hoisted to the top of the block, but not initialized.

Meaning: Using a `let` variable before it is declared will result in a `ReferenceError`:

```
<!DOCTYPE html>

<html>

<body>


<h2>JavaScript Hoisting</h2>

<p>With <b>let</b>, you cannot use a variable before it is declared.</p>

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


<script>

try {

  carName = "Saab";

  let carName = "Volvo";

}

catch(err) {

  document.getElementById("demo").innerHTML = err;

}

</script>


</body>

</html>
```

Const

The `const` keyword was introduced in ES6 (2015).

Variables defined with `const` cannot be Redeclared.

Variables defined with `const` cannot be Reassigned.

Variables defined with `const` have Block Scope.

Cannot be Reassigned

A `const` variable cannot be reassigned:

```
<!DOCTYPE html>

<html>

<body>
```

```
<h2>JavaScript const</h2>


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


<script>
try {
  const PI = 3.141592653589793;
  PI = 3.14;
}
catch (err) {
  document.getElementById("demo").innerHTML = err;
}
</script>


</body>
</html>
```

Must be Assigned

JavaScript `const` variables must be assigned a value when they are declared:

Correct

const PI = 3.14159265359

```
const PI;
PI = 3.14159265359;
```

When to use JavaScript const?

As a general rule, always declare a variable with `const` unless you know that the value will change.

Use `const` when you declare:

- A new Array
- A new Object
- A new Function
- A new RegExp

Constant Objects and Arrays

The keyword `const` is a little misleading.

It does not define a constant value. It defines a constant reference to a value.

Because of this you can NOT:

- Reassign a constant value
- Reassign a constant array
- Reassign a constant object

But you CAN:

- Change the elements of constant array
- Change the properties of constant object

Constant Arrays

You can change the elements of a constant array:

Example:

```
<!DOCTYPE html>

<html>

<body>



<h2>JavaScript const</h2>



<p>Declaring a constant array does NOT make the elements
unchangeable:</p>



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



<script>

// Create an Array:

const cars = ["Saab", "Volvo", "BMW"];



// Change an element:

cars[0] = "Toyota";
```

```
// Add an element:
cars.push("Audi");

// Display the Array:
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

But you can NOT reassign the array:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript const</h2>

<p>You can NOT reassign a constant array:</p>

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

<script>
```

```
try {
  const cars = ["Saab", "Volvo", "BMW"];
  cars = ["Toyota", "Volvo", "Audi"];
}
catch (err) {
  document.getElementById("demo").innerHTML = err;
}
</script>


</body>
</html>
```

Constant Objects

You can change the properties of a constant object:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript const</h2>

<p>Declaring a constant object does NOT make the objects properties unchangeable:</p>
```

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


<script>

// Create an object:

const car = {type:"Fiat", model:"500", color:"white"};


// Change a property:

car.color = "red";


// Add a property:

car.owner = "Johnson";


// Display the property:

document.getElementById("demo").innerHTML = "Car owner is " + car.owner;

</script>


</body>

</html>
```

But you can NOT reassign the object:

```
<!DOCTYPE html>

<html>
```

```
<body>

<h2>JavaScript const</h2>

<p>You can NOT reassign a constant object:</p>

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

<script>
try {
  const car = {type:"Fiat", model:"500", color:"white"};
  car = {type:"Volvo", model:"EX60", color:"red"};
}
catch (err) {
  document.getElementById("demo").innerHTML = err;
}
</script>

</body>
</html>
```

Browser Support

The `const` keyword is not supported in Internet Explorer 10 or earlier.

Block Scope - Const

Declaring a variable with `const` is similar to `let` when it comes to **Block Scope**.

The x declared in the block, in this example, is not the same as the x declared outside the block:

```
<!DOCTYPE html>

<html>

<body>


<h2>JavaScropt const variables has block scope</h2>


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


<script>

const  x = 10;

// Here x is 10


{
```

```
const x = 2;

// Here x is 2

}


// Here x is 10

document.getElementById("demo").innerHTML = "x is " + x;

</script>


</body>

</html>
```

Redeclaring

Redeclaring a JavaScript `var` variable is allowed anywhere in a program:

Example

```
var x = 2;       // Allowed
var x = 3;       // Allowed
x = 4;           // Allowed
```

Redeclaring an existing `var` or `let` variable to `const`, in the same scope, is not allowed:

```
var x = 2;       // Allowed
const x = 2;     // Not allowed

{
let x = 2;       // Allowed
```

```
const x = 2;    // Not allowed
}

{
const x = 2;    // Allowed
const x = 2;    // Not allowed
}
```

Reassigning an existing `const` variable, in the same scope, is not allowed:

Example

```
const x = 2;       // Allowed
x = 2;             // Not allowed
var x = 2;         // Not allowed
let x = 2;         // Not allowed
const x = 2;       // Not allowed

{
  const x = 2;    // Allowed
  x = 2;          // Not allowed
  var x = 2;      // Not allowed
  let x = 2;      // Not allowed
  const x = 2;    // Not allowed
}
```

Redeclaring a variable with `const`, in another scope, or in another block, is allowed:

Example

```
const x = 2;        // Allowed

{
  const x = 3;    // Allowed
}

{
```

```
  const x = 4;    // Allowed
}
```

Const Hoisting

Variables defined with `var` are **hoisted** to the top and can be initialized at any time.

Meaning: You can use the variable before it is declared:

Example

This is OK:

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

Variables defined with `const` are also hoisted to the top, but not initialized.

Meaning: Using a `const` variable before it is declared will result in a `ReferenceError`:

```
alert (carName);
const carName = "Volvo";
```

Lesson 14. JavaScript Syntax

JavaScript syntax is the set of rules, how JavaScript programs are constructed:

```
// How to create variables:
var x;
let y;

// How to use variables:
x = 5;
y = 6;
let z = x + y;
```

JavaScript Values

The JavaScript syntax defines two types of values:

- Fixed values
- Variable values

Fixed values are called **Literals**.

Variable values are called **Variables**.

JavaScript Literals

The two most important syntax rules for fixed values are:

1. **Numbers** are written with or without decimals:

```
10.50

1001
```

2. **Strings** are text, written within double or single quotes:

```
"Mohamed Jama"

'Mohamed Jama'
```

JavaScript Variables

In a programming language, **variables** are used to **store** data values.

JavaScript uses the keywords `var`, `let` and `const` to **declare** variables.

An **equal sign** is used to **assign values** to variables.

In this example, x is defined as a variable. Then, x is assigned (given) the value 6:

```
let x;
x = 6;
```

JavaScript Operators

JavaScript uses **arithmetic operators** (`+ - * /`) to **compute** values:

```
(5 + 6) * 10
```

JavaScript uses an **assignment operator** (`=`) to **assign** values to variables:

```
let x, y;
x = 5;
y = 6;
```

JavaScript Expressions

An expression is a combination of values, variables, and operators, which computes to a value.

The computation is called an evaluation.

For example, 5 * 10 evaluates to 50:

```
5 * 10
```

Expressions can also contain variable values:

```
x * 10
```

The values can be of various types, such as numbers and strings.

For example, "Mohamed" + " " + "Jama", evaluates to "Mohamed Jama":

```
"Mohamed" + " " + "Jama"
```

JavaScript Keywords

JavaScript **keywords** are used to identify actions to be performed.

The `let` keyword tells the browser to create variables:

```
let x, y;
x = 5 + 6;
y = x * 10;
```

The `var` keyword also tells the browser to create variables:

```
var x, y;
x = 5 + 6;
y = x * 10;
```

In these examples, using `var` or `let` will produce the same result.

JavaScript Comments

Not all JavaScript statements are "executed".

Code after double slashes `//` or between `/*` and `*/` is treated as a **comment**.

Comments are ignored, and will not be executed:

```
let x = 5;   // I will be executed

// x = 6;   I will NOT be executed
```

JavaScript Identifiers / Names

Identifiers are JavaScript names.

Identifiers are used to name **variables** and **keywords**, and **functions**.

The rules for legal names are the same in most programming languages.

A JavaScript name must begin with:

- A letter (A-Z or a-z)
- A dollar sign ($)
- Or an underscore (_)

Subsequent characters may be letters, digits, underscores, or dollar signs.

Note

Numbers are not allowed as the first character in names.

This way JavaScript can easily distinguish identifiers from numbers.

JavaScript is Case Sensitive

All JavaScript identifiers are **case sensitive**.

The variables `lastName` and `lastname`, are two different variables:

```
let lastname, lastName;
lastName = "Mohamed";
lastname = "Jama";
```

JavaScript does not interpret **LET** or **Let** as the keyword **let**.

JavaScript and Camel Case

Historically, programmers have used different ways of joining multiple words into one variable name:

Hyphens:

first-name, last-name, master-card, inter-city.

Hyphens are not allowed in JavaScript. They are reserved for subtractions.

Underscore:

first_name, last_name, master_card, inter_city.

Upper Camel Case (Pascal Case):

FirstName, LastName, MasterCard, InterCity.

Lower Camel Case:

JavaScript programmers tend to use camel case that starts with a lowercase letter:

firstName, lastName, masterCard, interCity.

Lesson 15. JavaScript Operators

Types of JavaScript Operators

There are different types of JavaScript operators:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Conditional Operators
- Type Operators
- Bitwise Operators

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic on numbers:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

The **addition assignment** operator (+=) adds a value to a variable.

Adding JavaScript Strings

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

Example

```
let text1 = "Mohamed";
let text2 = "Jama";
let text3 = text1 + " " + text2;
```

The result of text3 will be:

```
Mohamed Jama
```

Example2:

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Arithmetic</h1>

<h2>The += Operator</h2>

<script>

var x = 10;

x += 5;

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

</script>

</body>

</html>
```

The `+=` assignment operator can also be used to add (concatenate) strings:

Example

```
let text1 = "What a very ";
text1 += "nice day";
```

The result of text1 will be:

```
What a very nice day
```

When used on strings, the + operator is called the concatenation operator.

Adding Strings and Numbers

Adding two numbers, will return the sum, but adding a number and a string will return a string:

Example

```
let x = 5 + 5;
let y = "5" + 5;
let z = "Hello" + 5;
```

The result of *x*, *y*, and *z* will be:

```
10
55
Hello5
```

Example 3:

```
<!DOCTYPE html>

<html>

<body>


<h1>JavaScript Operators</h1>


<p>Adding a number and a string, returns a string.</p>


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


<script>

let x = 5 + 5;

let y = "5" + 5;

let z = "Hello" + 5;

document.getElementById("demo").innerHTML =

x + "<br>" + y + "<br>" + z;

</script>


</body>
```

```
</html>


If you add a number and a string, the result will be a string!
```

Lesson 16 JavaScript Comparison

Operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than

<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

JavaScript Logical Operators

Operator	Description
&&	logical and
||	logical or
!	logical not

JavaScript Type Operators

Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

JavaScript Bitwise Operators

Bit operators work on 32 bits numbers.

Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.

Operator	Description	Example	Same as	Result	Decimal
&	AND	5 & 1	0101 & 0001	0001	1
|	OR	5 | 1	0101 | 0001	0101	5

~	NOT	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	0100	4
<<	left shift	5 << 1	0101 << 1	1010	10
>>	right shift	5 >> 1	0101 >> 1	0010	2
>>>	unsigned right shift	5 >>> 1	0101 >>> 1	0010	2

Lesson 17: JavaScript Data Types

JavaScript has 8 Datatypes

1. String
2. Number
3. Bigint
4. Boolean
5. Undefined
6. Null
7. Symbol
8. Object

The Object Datatype

The object data type can contain:

1. An object
2. An array
3. A date

Note

A JavaScript variable can hold any type of data.

The Concept of Data Types

In programming, data types is an important concept.

To be able to operate on variables, it is important to know something about the type.

Without data types, a computer cannot safely solve this:

```
let x = 16 + "Volvo";
```

Does it make any sense to add "Volvo" to sixteen? Will it produce an error or will it produce a result?

JavaScript will treat the example above as:

```
let x = "16" + "Volvo";
```

Note

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

Example 1

```
<!DOCTYPE html>

<html>
```

```
<body>


<h2>JavaScript</h2>


<p>When adding a string and a number, JavaScript will treat the number as a string.</p>


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


<script>

let x = "Volvo" + 16;

document.getElementById("demo").innerHTML = x;

</script>


</body>

</html>
```

JavaScript Types are Dynamic

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

Example 2

```
<!DOCTYPE html>

<html>
```

```
<body>

<h2>JavaScript Data Types</h2>

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

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

<script>
let x;        // Now x is undefined
x = 5;        // Now x is a Number
x = "Mohamed";    // Now x is a String

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

</body>
</html>
```

Lesson 18: JavaScript Strings

A string (or a text string) is a series of characters like "Mohamed Jama".

Strings are written with quotes. You can use **single** or **double** quotes:

Example 1

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Strings</h2>

<p>Strings are written with quotes. You can use single or double quotes:</p>

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

<script>
let firstName = "Mohamed Jama";
let lastName = 'Sahal';

document.getElementById("demo").innerHTML =
firstName + "<br>" +
lastName;
```

```
</script>


</body>

</html>
```

Note:

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

Example 2

```
<!DOCTYPE html>

<html>

<body>


<h2>JavaScript Strings</h2>


<p>You can use quotes inside a string, as long as they don't match the quotes surrounding the string:</p>


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


<script>

let answer1 = "It's alright";

let answer2 = "He is called 'Mohamed'";
```

```
let answer3 = 'He is called "Sahal"';


document.getElementById("demo").innerHTML =
answer1 + "<br>" +
answer2 + "<br>" +
answer3;
</script>


</body>
</html>
```

String Length

To find the length of a string, use the built-in `length` property:

Example 3

```
<!DOCTYPE html>
<html>
<body>


<h1>JavaScript Strings</h1>
<h2>The length Property</h2>
```

The length of the string is:

<script>

let text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

document.getElementById("demo").innerHTML = text.length;

</script>

</body>

</html>

Escape Character

Because strings must be written within quotes, JavaScript will misunderstand this string:

```
let text = "We are the so-called "Vikings" from the north.";
```

The string will be chopped to "We are the so-called ".

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:

Example 4

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Strings</h1>

<p>The escape sequence \" inserts a double quote in a string.</p>

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

<script>
```

```
let text = "Waxaan tijaabinayaa sida ay \"ESCAPE Character\"u shaqeynayso.";

document.getElementById("demo").innerHTML = text;

</script>


</body>

</html>
```

Six other escape sequences are valid in JavaScript:

Code	**Result**
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Horizontal Tabulator

\v	Vertical Tabulator

The 6 escape characters above were originally designed to control typewriters, teletypes, and fax machines. They do not make any sense in HTML.

Breaking Long Code Lines

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:

Example 5

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Statements</h2>

<p>

The best place to break a code line is after an operator or a comma.

</p>
```

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

<script>

document.getElementById("demo").innerHTML =

"Hello Sahalsoftware!";

</script>


</body>

</html>
```

Note:

You can also break up a code line **within a text string** with a single backslash:

Example 6

```
<!DOCTYPE html>

<html>

<body>


<h1>JavaScript Strings</h1>


<p>

You can break a code line within a text string with a backslash.
```

```
</p>

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

<script>

document.getElementById("demo").innerHTML = "Hello \

Sahalsoftware!";

</script>


</body>

</html>
```

The \ method is not the preferred method. It might not have universal support. Some browsers do not allow spaces behind the \ character.

Note:

A safer way to break up a string, is to use string addition:

Example 7

```
<!DOCTYPE html>

<html>

<body>


<h1>JavaScript Strings</h1>
```

```
<p>The safest way to break a code line in a string is using string addition.</p>

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

<script>
document.getElementById("demo").innerHTML = "Hello " +
"Sahalsoftware!";
</script>
</body>
</html>
```

Note:

You cannot break up a code line with a backslash:

Example 8

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Statements</h2>
```

```
<p id="demo">You cannot break a code line with a \ backslash.</p>


<script>

document.getElementById("demo").innerHTML = \

"Hello Sahalsoftware.";

</script>


</body>

</html>
```

Lesson 19. JavaScript Numbers

All JavaScript numbers are stored as decimal numbers (floating point).

JavaScript has only one type of number. Numbers can be written with or without decimals.

Example 1

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Numbers</h2>

<p>Numbers can be written with, or without decimals:</p>
```

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

<script>

let price1 = 34.00;

let price2 = 34;

let price3 = 3.14;


document.getElementById("demo").innerHTML =

price1 + "<br>" + price2 + "<br>" + price3;

</script>


</body>

</html>
```

Exponential Notation

Extra large or extra small numbers can be written with scientific (exponential) notation:

Example 2

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Numbers</h2>
```

```
<p>Extra large or extra small numbers can be written with scientific (exponential) notation:</p>

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

<script>

let y = 123e5;

let z = 123e-5;

document.getElementById("demo").innerHTML =

y + "<br>" + z;

</script>


</body>

</html>
```

Note

Most programming languages have many number types:

Whole numbers (integers):
byte (8-bit), short (16-bit), int (32-bit), long (64-bit)

Real numbers (floating-point):
float (32-bit), double (64-bit).

Javascript numbers are always one type:
double (64-bit floating point).

Lesson 20. JavaScript BigInt (3)

- All JavaScript numbers are stored in a a 64-bit floating-point format.
- JavaScript BigInt is a new datatype (ES2020) that can be used to store integer values that are too big to be represented by a normal JavaScript Number.
- In JavaScript, all numbers are stored in a 64-bit floating-point format (IEEE 754 standard).
- `BigInt` is the second numeric data type in JavaScript (after `Number`).
- With `BigInt` the total number of supported data types in JavaScript is 8
- Arithmetic between a `BigInt` and a `Number` is not allowed (type conversion lose information).
- A `BigInt` can not have decimals.
- `BigInt` is supported in all browsers since September 2020

Example 1

```
<!DOCTYPE html>
<html>
<body>
<h1>JavScript Bigint</h1>
<p>A BigInt can not have decimals.</p>
```

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

<p>You cannot perform math between a BigInt type and a Number type.</p>

<script>
let x = BigInt("12345678901234567890123456789O");
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Lesson 21: Booleans, Undefined & Null (4,5, & 6)

- Booleans can only have two values: `true` or `false`.
- in programming, you will need a data type that can only have one of two values, like (YES / NO) (ON / OFF) (TRUE / FALSE).
- For this, JavaScript has a **Boolean** data type. It can only take the values **true** or **false**.

Example 1

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Booleans</h2>

<p>Booleans can have two values: true or false:</p>

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

<script>
let x = 5;
let y = 5;
let z = 6;

document.getElementById("demo").innerHTML =
(x == y) + "<br>" + (x == z);
</script>

</body>
```

```
</html>
```

Booleans are often used in conditional testing.

and so on.

Undefined

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

Example 2

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Operators</h1>
<h2>The typeof Operator</h2>
<p>The value (and the data type) of a variable with no value is
<b>undefined</b>.</p>

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

```
<script>

let car;

document.getElementById("demo").innerHTML =

car

</script>


</body>

</html>
```

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

Empty Values (Null)

An empty value has nothing to do with `undefined`.

An empty string has both a legal value and a type.

Example 3

```
<!DOCTYPE html>

<html>

<body>


<h2>JavaScript</h2>

<p>An empty string has both a legal value and a type:</p>
```

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

<script>

let car = "";

document.getElementById("demo").innerHTML =

"The value is: " +

car + "<br>" +

"The type is: " + typeof car;

</script>


</body>

</html>
```

Lesson 22: Objects & Symbol (7 & 8)

Symbol

The JavaScript Symbol is a primitive data type, just like Number, String, Boolean, etc. It represents a unique identifier and can be used in various ways. Symbols are used to create object properties, for example, when you want to assign a unique identifier to an object.

- const mySymbol = Symbol();

JavaScript Objects

Real Life Objects, Properties, and Methods

In real life, a car is an **object**.

A car has **properties** like weight and color, and **methods** like start and stop:

Object	Properties	Methods
	car.name = Fiat	car.start()
	car.model = 500	car.drive()
	car.weight = 850kg	car.brake()
	car.color = white	car.stop()

- All cars have the same **properties**, but the property **values** differ from car to car.

- All cars have the same **methods**, but the methods are performed **at different times**.
- JavaScript objects are written with **curly** braces { }.
- Object properties are written as **name:value** pairs, separated by commas.
- You define (and create) a JavaScript object

JavaScript Objects

You have already learned that JavaScript variables are containers for data values.

This code assigns a **simple value** (Fiat) to a **variable** named car:

```
let car = "NOHA";
```

Objects are variables too. But objects can contain many values.

This code assigns **many values** (Fiat, 500, white) to a **variable** named car:

Example 1

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Objects</h2>
<p id="demo"></p>
<script>
// Create an object:
const car = {
type:"Fiat",
model:"500",
color:"white"
};
// Display some data from the object:
document.getElementById("demo").innerHTML =
car.type + "<br>" +
car.model + "<br>" +
car.color ;


</script>
</body>
</html>
```

The values are written as **name:value** pairs (name and value separated by a colon).

It is a common practice to declare objects with the `const` keyword.

Example 2

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Objects</h2>
<p id="demo"></p>

<script>
const person = {
  firstName : "Abdirahman",
  age     : 25
 };
document.getElementById("demo").innerHTML =
person.firstName + " waxa uu jiraa " + person.age + " sano.";
</script>
</body>
</html>
```

The object (person) in the example above has 2 properties: firstName, age.

Lesson 23: typeof Operator

You can use the JavaScript `typeof` operator to find the type of a JavaScript variable.

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

Example 1 (String & Number)

```
<!DOCTYPE html>

<html>

<body>


<h1>JavaScript Operators</h1>

<h2>The typeof Operator</h2>

<p>The typeof operator returns the type of a variable or an expression.</p>


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


<script>

document.getElementById("demo").innerHTML =

typeof "" + "<br>" +
```

```
typeof "Mohamed" + "<br>" +
typeof "Abdirisak Mohamed" + "<br>" +
typeof 0 + "<br>" +
typeof 314 + "<br>" +
typeof 3.14 + "<br>" +
typeof (3) + "<br>" +
typeof (3 + 4);

</script>
</body>
</html>
```

Example 2 (BigInt)

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Numbers</h1>
<h2>BigInt typeof</h2>

<p>The typeof a BigInt is:</p>
<p id="demo"></p>
```

```
<script>
let x = BigInt("9999999999999999");
document.getElementById("demo").innerHTML = typeof x;
</script>


</body>
</html>
```

Example 3 (undefined)

```
<!DOCTYPE html>
<html>
<body>


<h1>JavaScript Operators</h1>
<h2>The typeof Operator</h2>
<p>The value (and the data type) of a variable with no value is
<b>undefined</b>.</p>
<p id="demo"></p>


<script>
let car;
document.getElementById("demo").innerHTML =
```

typeof car;

</script>

</body>

</html>

JavaScript – Part 2

Lesson 24. JavaScript Functions

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

A JavaScript function is executed when "something" invokes it (calls it).

JavaScript Function Syntax

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

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:
(*parameter1, parameter2, ...*)

The code to be executed, by the function, is placed inside curly brackets: **{}**

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

Function **parameters** are listed inside the parentheses () in the function definition.

Function **arguments** are the **values** received by the function when it is invoked.

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

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)

Function Return

When JavaScript reaches a `return` statement, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a **return value**. The return value is "returned" back to the "caller":

Example 1

Calculate the product of two numbers, and return the result:

Why Functions?

With functions you can reuse code

You can write code that can be used many times.

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

Example 1

Use External JavaScript

```
Function Salaan() {

        console.log("Salaan sare iga gudoon”)

}

Salaan()
```

Example 2

Use External JavaScript

```
function Salaan(name) {

    console.log('Salaan sare iga gudoon ' + name);

}

Salaan('Mohamed')
```

Example 3

```
<!DOCTYPE html>
<html>
<body>



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


<script>
let x = myFunction(4, 3);
document.getElementById("demo").innerHTML = x;
function myFunction(a, b) {
  return a * b;
}
</script>


</body>
```

```
</html>
```

Lesson 25. Accessing Object Properties

You can access object properties in two ways:

- *objectName.propertyName*
- *objectName["propertyName"]*

Example A

```
person.lastName;
```

Example B

```
person["lastName"];
```

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

Example 1

```
<!DOCTYPE html>
<html>
<body>
```

```
<h2>JavaScript Objects</h2>
<p id="demo"></p>

<script>
const person = {
  firstName : "Abdirahman",
  age     : 25
 };
document.getElementById("demo").innerHTML =
person.firstName + " waxa uu jiraa " + person.age + " sano.";
</script>
</body>
</html>
```

Example 2

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Objects</h2>
<p id="demo"></p>
```

```
<script>
const person = {
  firstName : "Abdirahman",
  age     : 25
 };
document.getElementById("demo").innerHTML =
person["firstName"] + " waxa uu jiraa " + person.age + " sano.";
</script>
</body>
</html>
```

Object Methods

Objects can also have **methods**.

Methods are **actions** that can be performed on objects.

Methods are stored in properties as **function definitions**.

Property	**Property Value**
firstName	Abdirahman
lastName	Jama

age	25
eyeColor	white
fullName	function() {return this.firstName + " " + this.lastName;}

A method is a function stored as a property.

Example 3: Method

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>
<p>An object method is a function definition, stored as a property value.</p>

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

<script>
// Create an object:
```

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

// Display data from the object:
document.getElementById("demo").innerHTML = person.fullName();
</script>

</body>
</html>
```

Lesson 26 JavaScript Arrays

JavaScript arrays are written with square brackets.

Array items are separated by commas.

The following code declares (creates) an array called cars, containing three items (car names):

Example 1

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
const cars = ["Saab","Volvo","BMW"];

document.getElementById("demo").innerHTML = cars[0];
</script>
```

```
</body>

</html>
```

Array indexes are zero-based, which means the first item is [0], second is [1],

Why Use Arrays?

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

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

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

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Creating an Array (1)

Using an array literal is the easiest way to create a JavaScript Array.

Syntax:

```
const array_name = [item1, item2, ...];
```

It is a common practice to declare arrays with the `const` keyword.

Example

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

Spaces and line breaks are not important. A declaration can span multiple lines:

Example

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

You can also create an array, and then provide the elements:

Example

```
const cars = [];
cars[0]= "Saab";
cars[1]= "Volvo";
cars[2]= "BMW";
```

Using the JavaScript Keyword new

The following example also creates an Array, and assigns values to it:

Example

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

Lesson 27 Accessing Array Elements (2)

You access an array element by referring to the **index number**:

Example 1

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>
<script>
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
</script>
```

```
</body>

</html>
```

Note: Array indexes start with 0.

[0] is the first element. [1] is the second element.

Changing an Array Element (3)

This statement changes the value of the first element in `cars`:

Example 2

```
<!DOCTYPE html>

<html>

<body>


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


<script>

const cars = ["Saab", "Volvo", "BMW"];

cars[0] = "Noha";

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

</script>


</body>
```

```
</html>
```

Access the Full Array (4)

With JavaScript, the full array can be accessed by referring to the array name:

Example 3

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Arrays</h1>

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

<script>
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>
```

```
</body>

</html>
```

Accessing the Last Array Element (5)

Example 4

```
<!DOCTYPE html>

<html>

<body>

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

<script>

const fruits = ["Banana", "Orange", "Apple", "Mango"];

document.getElementById("demo").innerHTML = fruits[fruits.length-1];

</script>

</body>

</html>
```

Lesson 28. JavaScript Array Methods

Array length	Array join()
Array toString()	Array delete()

Array pop()	Array concat()
Array push()	Array flat()
Array shift()	Array splice()
Array unshift()	Array slice()

JavaScript Array length (1)

The `length` property returns the length (size) of an array:

Example 1

```
<!DOCTYPE html>

<html>

<body>

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


<script>

const fruits = ["Banana", "Orange", "Apple", "Mango"];

let size = fruits.length;

document.getElementById("demo").innerHTML = size;

</script>


</body>

</html>
```

JavaScript Array toString() (2)

The JavaScript method `toString()` converts an array to a string of (comma separated) array values.

Example 2

```
<!DOCTYPE html>

<html>

<body>

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

<script>

const fruits = ["Banana", "Orange", "Apple", "Mango"];

document.getElementById("demo").innerHTML = fruits.toString();

</script>


</body>

</html>
```

The `join()` method also joins all array elements into a string. (3)

It behaves just like `toString()`, but in addition you can specify the separator:

Example 3

```
<!DOCTYPE html>
<html>
<body>

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

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
</script>

</body>
</html>
```

Popping and Pushing

When you work with arrays, it is easy to remove elements and add new elements.

This is what popping and pushing is:

Popping items **out** of an array, or pushing items **into** an array.

JavaScript Array pop() (4)

The `pop()` method removes the last element from an array:

Example 4

```
<!DOCTYPE html>

<html>

<body>

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

<script>

const fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.pop();

document.getElementById("demo").innerHTML = fruits;

</script>

</body>
```

```
</html>
```

JavaScript Array push() (5)

The `push()` method adds a new element to an array (at the end):

Example 5

```
<!DOCTYPE html>
<html>
<body>

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

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Avocado");
document.getElementById("demo2").innerHTML = fruits;
</script>

</body>
</html>
```

Lesson 29. JavaScript Sorting Arrays

Sorting an Array

The `sort()` method sorts an array alphabetically:

Example 1

```
<!DOCTYPE html>
<html>
<body>
<p id="demo1"></p>
<p id="demo2"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;

fruits.sort();
document.getElementById("demo2").innerHTML = fruits;
</script>
```

```
</body>

</html>
```

Reversing an Array

The `reverse()` method reverses the elements in an array:

Example 2

```
<!DOCTYPE html>

<html>

<body>


<p id="demo1"></p>

<p id="demo2"></p>

<script>

const fruits = ["Banana", "Orange", "Apple", "Mango"];

document.getElementById("demo1").innerHTML = fruits;


fruits.reverse();

document.getElementById("demo2").innerHTML = fruits;

</script>


</body>
```

```
</html>
```

Numeric Sort

By default, the `sort()` function sorts values as **strings**.

This works well for strings ("Apple" comes before "Banana").

If numbers are sorted as strings, "25" is bigger than "100", because "2" is bigger than "1".

Because of this, the `sort()` method will produce incorrect result when sorting numbers.

You can fix this by providing a **compare function**:

Example 3

```
<!DOCTYPE html>

<html>

<body>

<p id="demo1"></p>

<p id="demo2"></p>

<script>

const points = [40, 100, 1, 5, 25, 10];

document.getElementById("demo1").innerHTML = points;

points.sort(function(a, b){return a - b});

document.getElementById("demo2").innerHTML = points;
```

```
</script>

</body>

</html>
```

Use the same trick to sort an array descending:

Example 4

```
<!DOCTYPE html>

<html>

<body>

<p>Sort the array in descending order:</p>


<p id="demo1"></p>

<p id="demo2"></p>


<script>

const points = [40, 100, 1, 5, 25, 10];

document.getElementById("demo1").innerHTML = points;


points.sort(function(a, b){return b - a});

document.getElementById("demo2").innerHTML = points;
```

```
</script>


</body>

</html>
```

Lesson 30. if, else, and else if

Conditional statements are used to perform different actions based on different conditions.

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

- Use `if` to specify a block of code to be executed, if a specified condition is true
- Use `else` to specify a block of code to be executed, if the same condition is false
- Use `else if` to specify a new condition to test, if the first condition is false
- Use `switch` to specify many alternative blocks of code to be executed

The if Statement

Use the `if` statement to specify a block of JavaScript code to be executed if a condition is true.

Syntax

```
if (condition) {
  //  block of code to be executed if the condition is true
}
```

Note that `if` is in lowercase letters. Uppercase letters (If or IF) will generate a JavaScript error.

Example 1

Make a "Good day" greeting if the hour is less than 18:00:

The result of greeting will be:

```
Good day
```

```
<!DOCTYPE html>

<html>

<body>

<p id="demo">Good Evening!</p>

<script>

if (new Date().getHours() < 18) {

 document.getElementById("demo").innerHTML = "Good day!";

}
```

```
</script>

</body>

</html>
```

The else Statement

Use the `else` statement to specify a block of code to be executed if the condition is false.

```
if (condition) {
  //  block of code to be executed if the condition is true
} else {
  //  block of code to be executed if the condition is false
}
```

Example 2

If the hour is less than 18, create a "Good day" greeting, otherwise "Good evening":

```
<!DOCTYPE html>

<html>

<body>

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

<script>

const hour = new Date().getHours();

let greeting;

if (hour < 18) {

  greeting = "Good day";

} else {
```

```
 greeting = "Good evening";

}

document.getElementById("demo").innerHTML = greeting;

</script>

</body>

</html>
```

The else if Statement

Use the `else if` statement to specify a new condition if the first condition is false.

Syntax

```
if (condition1) {
  //  block of code to be executed if condition1 is true
} else if (condition2) {
  //  block of code to be executed if the condition1 is false and
condition2 is true
} else {
  //  block of code to be executed if the condition1 is false and
condition2 is false
}
```

Example 3

If time is less than 10:00, create a "Good morning" greeting, if not, but time is less than 20:00, create a "Good day" greeting, otherwise a "Good evening":

```
<!DOCTYPE html>

<html>
```

```
<body>

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

<script>

const time = new Date().getHours();

let greeting;

if (time < 10) {

  greeting = "Good morning";

} else if (time < 20) {

  greeting = "Good day";

} else {

  greeting = "Good evening";

}

document.getElementById("demo").innerHTML = greeting;

</script>

</body>

</html>
```

Lesson 31. Five conditions

if, else, and else if

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
const time = new Date().getHours();
let greeting;
if (time < 10) {
  greeting = "Subax wanaagsan";
} else if (time < 15) {
  greeting = "Duhur wanaagsan";
} else if (time < 18) {
  greeting = "Casar Wanaagsan";
} else if (time < 22) {
```

```
 greeting = "Fiid wanaagsan";

} else {

 greeting = "Habeen wanaagsan";

}

document.getElementById("demo").innerHTML = greeting;

</script>

</body> </html>
```

Lesson 32. JavaScript Switch Statement

The switch statement is used to perform different actions based on different conditions.

Syntax

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

This is how it works:

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- If there is no match, the default code block is executed.

The break Keyword

When JavaScript reaches a `break` keyword, it breaks out of the switch block.

This will stop the execution inside the switch block.

It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway.

Note: If you omit the break statement, the next case will be executed even if the evaluation does not match the case.

The default Keyword

The `default` keyword specifies the code to run if there is no case match:

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
let text;
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";
}
document.getElementById("demo").innerHTML = text;
</script>


</body>
</html>
```

Lesson 33. JavaScript Loops (1)

Loops can execute a block of code a number of times. If you want to run the same code over and over again, each time with a different value.

Different Kinds of Loops

JavaScript supports different kinds of loops:

- `for` - loops through a block of code a number of times
- `for/in` - loops through the properties of an object
- `for/of` - loops through the values of an iterable object
- `while` - loops through a block of code while a specified condition is true
- `do/while` - also loops through a block of code while a specified condition is true

The For Loop

The `for` statement creates a loop with 3 optional expressions:

```
for (expression 1; expression 2; expression 3) {
  // code block to be executed
}
```

Expression 1 is executed (one time) before the execution of the code block.

Expression 2 defines the condition for executing the code block.

Expression 3 is executed (every time) after the code block has been executed.

Example 1

```
<!DOCTYPE html>

<html>

<body>


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


<script>

let text = "";


for (let i = 0; i < 5; i++) {
```

```
  text += "The number is " + i + "<br>";
}


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


</body>
</html>
```

From the example above, you can read:

Expression 1 sets a variable before the loop starts (let i = 0).

Expression 2 defines the condition for the loop to run (i must be less than 5).

Expression 3 increases a value (i++) each time the code block in the loop has been executed.

Expression 1

Normally you will use expression 1 to initialize the variable used in the loop (let i = 0).

This is not always the case. JavaScript doesn't care. Expression 1 is optional.

You can initiate many values in expression 1 (separated by comma):

Example 2

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript For Loop</h2>

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

<script>
const cars = ["BMW", "Volvo", "Saab", "Ford"];
let i, len, text;
for (i = 0, len = cars.length, text = ""; i < len; i++) {
  text += cars[i] + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

And you can omit expression 1 (like when your values are set before the loop starts):

Example 3

```
<!DOCTYPE html>

<html>

<body>


<h2>JavaScript For Loop</h2>


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


<script>

const cars = ["BMW", "Volvo", "Saab", "Ford"];

let i = 2;

let len = cars.length;

let text = "";


for (; i < len; i++) {

  text += cars[i] + "<br>";

}
```

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

</script>


</body>

</html>
```

Expression 2

Often expression 2 is used to evaluate the condition of the initial variable.

This is not always the case. JavaScript doesn't care. Expression 2 is also optional.

If expression 2 returns true, the loop will start over again. If it returns false, the loop will end.

If you omit expression 2, you must provide a **break** inside the loop. Otherwise the loop will never end. This will crash your browser.

Expression 3

Often expression 3 increments the value of the initial variable.

This is not always the case. JavaScript doesn't care. Expression 3 is optional.

Expression 3 can do anything like negative increment (i--), positive increment (i = i + 15), or anything else.

Expression 3 can also be omitted (like when you increment your values inside the loop):

Example 4

```
<!DOCTYPE html>

<html>

<body>


<h2>JavaScript For Loop</h2>


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


<script>

const cars = ["BMW", "Volvo", "Saab", "Ford"];


let i = 0;

let len = cars.length;

let text = "";


for (; i < len; ) {

  text += cars[i] + "<br>";
```

```
 i++;
}
document.getElementById("demo").innerHTML = text;
</script>


</body>
</html>
```

Lesson 34. JavaScript While Loop

The While Loop

The `while` loop loops through a block of code as long as a specified condition is true.

Syntax

```
while (condition) {
  // code block to be executed
}
```

Example 1

In the following example, the code in the loop will run, over and over again, as long as a variable (i) is less than 10:

```
<!DOCTYPE html>

<html>

<body>



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



<script>

let text = "";

let i = 1;

while (i < 10) {

  text += "<br>The number is " + i;

  i++;

}

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

</script>

</body>

</html>

If you forget to increase the variable used in the condition, the loop will never end. This will crash your browser.

The Do While Loop

The `do while` loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

```
do {
  // code block to be executed
}
while (condition);
```

Example 2

The example below uses a `do while` loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

<!DOCTYPE html>

<html>

<body>

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

<script>

let text = ""

let i = 1;


do {

  text += "<br>The number is " + i;

  i++;

}

while (i < 10);


document.getElementById("demo").innerHTML = text;

</script>


</body>

</html>
```