JavaScript

Sahalsoftware

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**

JavaScript

Sahalsoftware

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>
```

JavaScript

Sahalsoftware

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

Sahalsoftware

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>
```

JavaScript

Sahalsoftware

```
<!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:

JavaScript

Sahalsoftware

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":

JavaScript

Sahalsoftware

```
<!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>
```

JavaScript

Sahalsoftware

```
<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.

<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>

JavaScript code blocks are written between { and }

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

```
<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

JavaScript

Sahalsoftware

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.

JavaScript

Sahalsoftware

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>


<p>Redeclaring a variable with <b>let</b>, in another scope, or in another block, is allowed:</p>
```

```
<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:

JavaScript

Sahalsoftware

```
<!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`:

JavaScript

Sahalsoftware

```
<!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;
```

JavaScript

Sahalsoftware

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:

JavaScript

Sahalsoftware

```
<!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>
```

JavaScript

Sahalsoftware

```
<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

Sahalsoftware

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

Sahalsoftware

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

Sahalsoftware

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.

JavaScript

Sahalsoftware

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

Note:

- **Variables = 30%**
- **Operators + If + Loop = 30%**
- **Others = 40 %**

Types of JavaScript Operators

There are different types of JavaScript operators:

- Arithmetic Operators

JavaScript

Sahalsoftware

- 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.

Assignment

```
let x = 10;
x += 5;
```

Example 1:

<!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>

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>

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

<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:

JavaScript

Sahalsoftware

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>

Adding a number and a string, returns a string.

<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
&	AND	5 & 1	0101 & 0001	0001
|	OR	5 | 1	0101 | 0001	0101
~	NOT	~ 5	~0101	1010

^	XOR	5 ^ 1	0101 ^ 0001	0100
<<	left shift	5 << 1	0101 << 1	1010
>>	right shift	5 >> 1	0101 >> 1	0010
>>>	unsigned right shift	5 >>> 1	0101 >>> 1	0010