# JavaScript Introduction

## JavaScript Can Change HTML Content

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

The example below "finds" an HTML element (with id="demo"), and changes the element content (innerHTML) to "Hello JavaScript":

### Example:

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

JavaScript accepts both double and single quotes:

### Example:

```
document.getElementById('demo').innerHTML = 'Hello JavaScript';
```

## JavaScript Can Change HTML Styles (CSS)

Changing the style of an HTML element, is a variant of changing an HTML attribute:

### Example:

```
document.getElementById("demo").style.fontSize = "35px";
```

## JavaScript Can Hide HTML Elements

Hiding HTML elements can be done by changing the `display` style:

### Example:

```
document.getElementById("demo").style.display = "none";
```

# •JavaScript Where To

## The <script> Tag

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

### Example

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

## 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.

### Example:

```
<!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 Output

## 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:

### Example:

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My First Paragraph</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>
</body>
</html>
```

## Using document.write()

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

### Example:

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
document.write(5 + 6);
</script>

</body>
</html>
```

# Using window.alert()

You can use an alert box to display data:

## Example:

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

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
window.alert(5 + 6);
</script>

</body>
</html>
```

You can skip the `window` keyword.

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

## Example:

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

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
alert(5 + 6);
</script>

</body>
</html>
```

# Using console.log()

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

## Example:

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

<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.

## Example

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

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

</body>
</html>
```

# •JavaScript Statements

## Semicolons ;

Semicolons separate JavaScript statements.

Add a semicolon at the end of each executable statement:

### Examples:

```
let a, b, c;  // Declare 3 variables
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
```

## 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 = "Hege";
let person="Hege";
```

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.

### Example:

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

# JavaScript Keywords

JavaScript statements often start with a **keyword** to identify the JavaScript action to be performed.

Here is a list of some of the keywords you will learn about in this tutorial:

| Keyword | Description |
|---|---|
| var | Declares a variable |
| let | Declares a block variable |
| const | Declares a block constant |
| if | Marks a block of statements to be executed on a condition |
| switch | Marks a block of statements to be executed in different cases |
| for | Marks a block of statements to be executed in a loop |
| function | Declares a function |
| return | Exits a function |
| try | Implements error handling to a block of statements |

# •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
```

# 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.

```
"John" + " " + "Doe"
```

# 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;
```

# 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 is Case Sensitive

All JavaScript identifiers are **case sensitive**.

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

```
let lastname, lastName;
lastName = "Doe";
lastname = "Peterson";
```

# •JavaScript Comments

## 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:

### Example:

```
// Change heading:
document.getElementById("myH").innerHTML = "My First Page";

// Change paragraph:
document.getElementById("myP").innerHTML = "My first paragraph.";
```

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

### Example:

```
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:

## Example:

```
/*
The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
in my web page:
*/
document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

# 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:

## Example:

```
//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:

## Example:

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

# •JavaScript Variables

## Variables are Containers for Storing Data

JavaScript Variables can be declared in 4 ways:

- Automatically
- Using `var`
- Using `let`
- Using `const`

- In this first example, `x`, `y`, and `z` are undeclared variables.

- They are automatically declared when first used:

## Example:

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

## Example using var:

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

# *Note*

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

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

The `var` keyword should only be used in code written for older browsers.

## Example using let:

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

## Example using const:

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

## Mixed Example:

```
const price1 = 5;
const price2 = 6;
let total = price1 + price2;
```

The two variables `price1` and `price2` are declared with the `const` keyword.

These are constant values and cannot be changed.

The variable `total` is declared with the `let` keyword.

The value `total` can be changed.

# When to Use var, let, or const?

1. Always declare variables

2. Always use `const` if the value should not be changed

3. Always use `const` if the type should not be changed (Arrays and Objects)

4. Only use `let` if you can't use `const`

5. Only use `var` if you MUST support old browsers.

# 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;
```

# JavaScript Identifiers

All JavaScript **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter.
- Names can also begin with $ and _ (but we will not use it in this tutorial).
- Names are case sensitive (y and Y are different variables).
- Reserved words (like JavaScript keywords) cannot be used as names.

# The Assignment Operator

In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator.

This is different from algebra. The following does not make sense in algebra:

```
x = x + 5
```

In JavaScript, however, it makes perfect sense: it assigns the value of x + 5 to x.

(It calculates the value of x + 5 and puts the result into x. The value of x is incremented by 5.)

# JavaScript Data Types

JavaScript variables can hold numbers like 100 and text values like "John Doe".

In programming, text values are called text strings.

JavaScript can handle many types of data, but for now, just think of numbers and strings.

Strings are written inside double or single quotes. Numbers are written without quotes.

If you put a number in quotes, it will be treated as a text string.

## Example:

```
const pi = 3.14;
let person = "John Doe";
let answer = 'Yes I am!';
```

# Declaring a JavaScript Variable

Creating a variable in JavaScript is called "declaring" a variable.

You declare a JavaScript variable with the `var` or the `let` keyword:

```
var carName;
or:
let carName;
```

After the declaration, the variable has no value (technically it is `undefined`).

To **assign** a value to the variable, use the equal sign:

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

You can also assign a value to the variable when you declare it:

```
let carName = "Volvo";
```

In the example below, we create a variable called `carName` and assign the value "Volvo" to it.

Then we "output" the value inside an HTML paragraph with id="demo":

## Example:

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

<script>
let carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
</script>
```

# One Statement, Many Variables

You can declare many variables in one statement.

Start the statement with `let` and separate the variables by **comma**:

## Example:

```
let person = "John Doe", carName = "Volvo", price = 200;
```

A declaration can span multiple lines:

## Example:

```
let person = "John Doe",
carName = "Volvo",
price = 200;
```

# Value = undefined

In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input.

A variable declared without a value will have the value `undefined`.

The variable carName will have the value `undefined` after the execution of this statement:

## Example:

```
let carName;
```

# Re-Declaring JavaScript Variables

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

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

## Example:

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

# *Note*

You cannot re-declare a variable declared with `let` or `const`.

This will not work:

```
let carName = "Volvo";
let carName;
```

# JavaScript Arithmetic

As with algebra, you can do arithmetic with JavaScript variables, using operators like `=` and `+`:

## Example:

```
let x = 5 + 2 + 3;
```

You can also add strings, but strings will be concatenated:

## Example:

```
let x = "John" + " " + "Doe";
```

# *Note*

If you put a number in quotes, the rest of the numbers will be treated as strings, and concatenated.

Now try this:

## Example:

```
let x = 2 + 3 + "5";
```

# JavaScript Dollar Sign $

Since JavaScript treats a dollar sign as a letter, identifiers containing $ are valid variable names:

## Example:

```
let $ = "Hello World";
let $$$ = 2;
let $myMoney = 5;
```

Using the dollar sign is not very common in JavaScript, but professional programmers often use it as an alias for the main function in a JavaScript library.

In the JavaScript library jQuery, for instance, the main function $ is used to select HTML elements. In jQuery `$("p");` means "select all p elements".

# JavaScript Underscore (_)

Since JavaScript treats underscore as a letter, identifiers containing _ are valid variable names:

## Example:

```
let _lastName = "Johnson";
let _x = 2;
let _100 = 5;
```

# •JavaScript 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` **can not be redeclared**.

You can not accidentally redeclare a variable declared with `let`.

With `let` you can **not** do this:

```
let x = "John Doe";

let x = 0;
```

With `var` you can:

```
var x = "John Doe";

var x = 0;
```

# Block Scope

Before ES6 (2015), JavaScript had **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:

```
var x = 10;
// Here x is 10

{
var x = 2;
// Here x is 2
}

// Here x is 2
```

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:

```
let x = 10;
// Here x is 10

{
let x = 2;
// Here x is 2
}

// Here x is 10
```

# Redeclaring

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

## Example:

```
var x = 2;
// Now x is 2

var x = 3;
// Now x is 3
```

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

## Example:

```
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:

```
let x = 2;   // Allowed

{
let x = 3;   // Allowed
}

{
let x = 4;    // Allowed
}
```