Sayed Jamaludin Afghani university
Computer Science Faculty
Software Engineering department

Script programming (JavaScript)

Ketabton.com

Lecture by: Tawhidullah babarzai

Phone #: +93778716554

Email: tawhidullah1111@gmail.com

Contents

JavaScript introduction

JavaScript is the programming language of HTML and web page

That can behavior the web page.

That can find the HTML element and change the element content .

- The below example of change the element content.

```
<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>
```

JavaScript can change the style of an HTML element, is a variant of changing an HTML attribute.

- Example of changing the style of HTML element

```
<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>
```

JavaScript can hide HTML element

- Example:

```
<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 code must be inserted between <script> and </script>

<script> opening tag

</script> closing tag

Such as :

```
<html>

<body>

<h2>JavaScript in Body</h2>

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

<script>

document.getElementById("demo").innerHTML = "My First JavaScript";

</script>

</body>

</html>
```

JavaScript Functions

A JavaScript `function` is a block of JavaScript code, that can be executed when "called" and it is used to perform the specific task.

a function can be called when an **event** occurs, like when the user clicks a button.

- three place we can use the script

The script opening and closing tag we can use

1: In head place .

2: Use before the body tag.

3: External place.

The External place page extension should .**js** And the internal place like this <script src =”the name of External page ”></script>

Example of head place:

```
<html>

<head>
</head>
<body>

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

</body>
</html>
```

Example: of before of body tag:

```
<html>

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

```
</script>
</head>
<body>

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

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

The Example of External place:

```
<html>

<body>

<h1 id=”do”></h1>

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

</body>

</html>
```

External:

The name of the page **myScript.js**

```
Document.getElemnetbyId(‘do’).innerHTMl = “the External page”;
```

- Some advantage of External file.
  - It separates HTML and code
  - It makes HTML and JavaScript easier to read and maintain
  - Cached JavaScript files can speed up page loads

Display the javaScript out put

JavaScript can display Data in deferent way

Using innerHTMl

Using document.write();

Using window.alert();

Using console.log();

Example Of innerHTML.

```
<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>
```

Example Of document.write().

```
<html>
<body>

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

<script>
document.write(5 + 6);
</script>

</body>
</html>
```

Example of window.alert().

```
<html>
<body>

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

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

```
</script>

</body>
</html>
```

Example of console.log().

```
<html>
<body>

<script>
console.log(5 + 6);
</script>

</body>
</html>
```

JavaScript Statements

JavaScript statements are composed of:

Values, Operators, Expressions, Keywords, and Comments.

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

Example:

<html>

<body>

<h2>JavaScript Statements</h2>

In HTML, JavaScript statements are executed by the browser.

<script>

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

</script>

</body>

```
</html>
```

The second Example:

```
<html>

<body>



<h2>JavaScript Statements</h2>

<p>A <b>JavaScript program</b> is a list of <b>statements</b> to be
executed by a computer.</p>



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

<script>

var x, y, z;  // Statement 1

x = 5;    // Statement 2

y = 6;    // Statement 3

z = x + y;  // Statement 4

document.getElementById("demo").innerHTML =

"The value of z is " + z + ".";

</script>

</body></html>
```

JavaScript Code Blocks

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

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

Example:

```
<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 Dolly!";
  document.getElementById("demo2").innerHTML = "How are you?";
}
</script>
</body></html>
```

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
break	Terminates a switch or a loop

continue	Jumps out of a loop and starts at the top
debugger	Stops the execution of JavaScript, and calls (if available) the debugging function
do ... while	Executes a block of statements, and repeats the block, while a condition is true
For	Marks a block of statements to be executed, as long as a condition is true
function	Declares a function
if ... else	Marks a block of statements to be executed, depending on a condition
return	Exits a function
switch	Marks a block of statements to be executed, depending on different cases
try ... catch	Implements error handling to a block of statements
Var	Declares a variable

JavaScript keywords are reserved words. Reserved words cannot be used as names for variables.

JavaScript Syntax

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

```
var x, y, z;           // How to declare variables
x = 5; y = 6;       // How to assign values
z = x + y;          // How to compute values
```

JavaScript Values

The JavaScript syntax defines two types of values: Fixed values and variable values.

Fixed values are called **literals**. Variable values are called **variables**.

JavaScript Variables

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

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

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

<html>

<body>

<h2>JavaScript Variables</h2>

In this example, x is defined as a variable.

Then, x is assigned the value of 6:

<script>

var x;

```
x = 6;

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

</script>

</body></html>
```

JavaScript Operators

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

```
<html>

<body>

<h2>JavaScript Operators</h2>

<p>JavaScript uses arithmetic operators to compute values (just like algebra).</p>

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

<script>

document.getElementById("demo").innerHTML = (5 + 6) * 10;

</script>

</body></html>
```

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

```
<html>

<body>

<h2>Assigning JavaScript Values</h2>

<p>In JavaScript the = operator is used to assign values to variables.</p>

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

<script>
```

```
  var x, y;

    x = 5;

    y = 6;

document.getElementById("demo").innerHTML = x +y;
</script><body></html>
```

JavaScript Expressions

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

```
<html>

<body>

<h2>JavaScript Expressions</h2>

<p>Expressions compute to values.</p>

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

<script>

document.getElementById("demo").innerHTML = 5 * 10;

</script>

</body></html>

<html>

<body>

<h2>JavaScript Expressions</h2>

<p>Expressions compute to values.</p>

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

<script>

document.getElementById("demo").innerHTML = "John" + " "  + "Doe";
```

```
</script>

</body>
```

JavaScript Keywords

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

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

```
<html>

<body>

<h2>The var Keyword Creates Variables</h2>

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

<script>

var x, y;

x = 5 + 6;

y = x * 10;

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

</script>

</body></html>
```

JavaScript Comments

JavaScript has two type of comment

- Single line comment
- Multi line comment

  The single line comment **like** this double back slash

  String name = “hellow bro”;//

The multi line comment **like** this single back slash and staric /* end staric and back slash*/

/*

Var a = 10;

Var b =20;

Var sum = a+b; */

JavaScript Variables

JavaScript variables are containers for storing data values

In this example, x, y, and z, are variables:

```
<html>
<body>
<h2>JavaScript Variables</h2>
<p>In this example, x, y, and z are variables.</p>
<p id="demo"></p>
<script>
var x = 5;
var y = 6;
var z = x + y;
document.getElementById("demo").innerHTML = "The value of z is: " + z;
</script>
</body> </html>
```

JavaScript Identifiers

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

These unique names are called **identifiers**.

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

JavaScript identifiers are case-sensitive.

The Assignment Operator

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

This is different from algebra.

The "equal to" operator is written like == in JavaScript.

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.

```
<html><body>

<h2>JavaScript Variables</h2>

<p>Strings are written with quotes.</p>

<p>Numbers are written without quotes.</p>

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

```
<script>

var pi = 3.14;

var person = "John Doe";

var answer = 'Yes I am!';

document.getElementById("demo").innerHTML =

pi + "<br>" + person + "<br>" + answer;

</script>

</body></html>
```

Declaring (Creating) JavaScript Variables

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

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

Example:

Var carName;

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

One Statement, Many Variables

You can declare many variables in one statement.

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

Example:

```
<html>

<body>

<h2>JavaScript Variables</h2>

<p>You can declare many variables in one statement.</p>
```

```
<p id="demo"></p>
<script>
var person = "John Doe", carName = "Volvo", price = 200;
document.getElementById("demo").innerHTML = carName;
</script></body>
```

What is the deferent between Undefined and Null;

Undefined: the variable declare with out the value .

Null: the variable declare with empty value.

Example of undefined:

```
Var a;
Console.log(a);
```

Example of Null:

```
Var a = " ";
Console.log(a);
```

JavaScript Operators

The **assignment** operator (=) assigns a value to a variable.

```
var x = 10;
```

The **addition** operator (+) adds numbers:

```
<html><body>
<h2>The + Operator</h2>
<p id="demo"></p>
<script>
var x = 5;
var y = 2;
var z = x + y;
document.getElementById("demo").innerHTML = z;
```

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

The **multiplication** operator (*) multiplies numbers.

```
<html>
<body>
<h2>The * Operator</h2>
<p id="demo"></p>
<script>
var x = 5;
var y = 2;
var z = x * y;
document.getElementById("demo").innerHTML = z;
</script>
</body></html>
```

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.

Example:

```
<html>
<body>
<h2>The += Operator</h2>
<p id="demo"></p>
<script>
var x = 10;
x += 5;
document.getElementById("demo").innerHTML = x;
</script>
</body></html>
```

JavaScript String Operators

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

Example:

```
<html>
<body>
<h2>JavaScript Operators</h2>
<p>The + operator concatenates (adds) strings.</p>
<p id="demo"></p>
<script>
var txt1 = "John";
var txt2 = "Doe";
document.getElementById("demo").innerHTML = txt1 + " " + txt2;
</script>
</body></html>
```

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

Example:

```
<html>
<body>
<h2>JavaScript Operators</h2>
<p>The assignment operator += can concatenate strings.</p>
<p id="demo"></p>
<script>
txt1 = "What a very ";
txt1 += "nice day";
document.getElementById("demo").innerHTML = txt1;
</script>
```

```
</body>
```

Adding Strings and Numbers

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

Example:

```
<html>

<body>

<h2>JavaScript Operators</h2>

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

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

<script>

var x = 5 + 5;

var y = "5" + 5;

var z = "Hello" + 5;

document.getElementById("demo").innerHTML =

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

</script>

</body></html>
```

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 Bitwise Operators

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
<<	Zero fill left shift	5 << 1	0101 << 1	1010	10
>>	Signed right shift	5 >> 1	0101 >> 1	0010	2

>>>	Zero fill right shift	5 >>> 1	0101 >>> 1	0010	2

JavaScript Data Types

JavaScript variables can hold many **data types**: numbers, strings, objects and more:

```
var length = 16;                               // Number
var lastName = "Johnson";                      // String
var x = {firstName:"John", lastName:"Doe"};    // Object
```

JavaScript Strings

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

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

<html>

<body>

<h2>JavaScript Strings</h2>

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

<script>

var carName1 = "Volvo XC60";

var carName2 = 'Volvo XC60';

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

carName1 + "<br>" +

carName2;

</script>

</body></html>
```

JavaScript Numbers

JavaScript has only one type of numbers.

Numbers can be written with, or without decimals:

```
<html>

<body>

<h2>JavaScript Numbers</h2>

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

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

<script>

var x1 = 34.00;

var x2 = 34;

var x3 = 3.14;

document.getElementById("demo").innerHTML =

x1 + "<br>" + x2 + "<br>" + x3;

</script>

</body></html>
```

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

```
<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>

var y = 123e5;

var z = 123e-5;document.getElementById("demo").innerHTML =

y + "<br>" + z;

</script>

</body></html>
```

JavaScript Arrays

JavaScript arrays are written with square brackets.

Array items are separated by commas.

```
<html>

<body>

<h2>JavaScript Arrays</h2>

<p>Array indexes are zero-based, which means the first item is [0].</p>

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

<script>

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

document.getElementById("demo").innerHTML = cars[0];
```

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

JavaScript Objects

JavaScript objects are written with curly braces `{}`.

Object properties are written as name:value pairs, separated by commas.

```
<html>
<body>
<h2>JavaScript Objects</h2>
<p id="demo"></p>
<script>
var person = {
  firstName : "John",
  lastName  : "Doe",
  age     : 50,
  eyeColor  : "blue"
};
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>
</body></html>
```

The 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:

```
<html>

<body>


<h2>JavaScript typeof</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 "John" + "<br>" +

typeof "John Doe";

</script>

</body></html>
```

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).

```
<html>

<body>

<h2>JavaScript Functions</h2>
```

This example calls a function which performs a calculation, and returns the result:

<script>

function myFunction(p1, p2) {

  return p1 * p2;

}

document.getElementById("demo").innerHTML = myFunction(4, 3);

</script>

</body></html>

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
}
```

why function we use?

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

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

Example:

```
<html>

<body>

<h2>JavaScript Functions</h2>

<p>This example calls a function to convert from Fahrenheit to Celsius:</p>

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

<script>

function toCelsius(f) {

  return (5/9) * (f-32);

}

document.getElementById("demo").innerHTML = toCelsius(77);

</script>

</body></html>
```

Local Variables

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

Local variables can only be accessed from within the function.

Example:

```
<html>

<body>

<h2>JavaScript Functions</h2>
```

```
<p>Outside myFunction() carName is undefined.</p>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
myFunction();
function myFunction() {
  var carName = "Volvo";
  document.getElementById("demo1").innerHTML =
  typeof carName + " " + carName;
}
document.getElementById("demo2").innerHTML =
typeof carName;
</script>

</body></html>
```

JavaScript Objects

Every present thing is called object.

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.model = 500

car.weight = 850kg

car.color = white

car.start()

car.drive()

car.brake()

car.stop()

Methods

car.start()

car.drive()

car.brake()

car.stop()

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

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

Example:

```
<html>
<body>
<h2>JavaScript Objects</h2>
<p id="demo"></p>
<script>
// Create an object:
var car = {type:"Fiat", model:"500", color:"white"};
// Display some data from the object:
```

document.getElementById("demo").innerHTML = "The car type is " + car.type;

</script>

</body></html>

Object Properties

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

Property	Property Value
firstName	John
lastName	Doe
Age	50
eyeColor	blue

Accessing Object Properties

You can access object properties in two ways:

objectName.propertyName

or

objectName["propertyName"]

Example:

<html>

```
<body>


<h2>JavaScript Objects</h2>
<p>There are two different ways to access an object property.</p>
<p>You can use person.property or person["property"].</p>
<p id="demo"></p>
<script>
// Create an object:
var person = {
  firstName: "John",
  lastName : "Doe",
  id     :  5566
};
// Display some data from the object:
document.getElementById("demo").innerHTML =
person.firstName + " " + person.lastName;
</script>
</body></html>
```

The **this** Keyword

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

Instead of the object we can use **this** keyword

Example:

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

JavaScript Events

HTML events are **"things"** that happen to HTML elements.

When JavaScript is used in HTML pages, JavaScript can **"react"** on these events.

Common HTML Events

Here is a list of some common HTML events:

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key

onload	The browser has finished loading the page

What can JavaScript Do?

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

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

Many different methods can be used to let JavaScript work with events:

- HTML event attributes can execute JavaScript code directly
- HTML event attributes can call JavaScript functions

JavaScript Strings

JavaScript strings are used for storing and manipulating text.

JavaScript Strings

A JavaScript string is zero or more characters written inside quotes.

Example:

```
<html>

<body>

<h2>JavaScript Strings</h2>

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

<script>
```

```
var x = "John Doe";  // String written inside
quotesdocument.getElementById("demo").innerHTML = x;

</script>

</body>

</html>
```

You can use single or double quotes:

Example:

```
<html>

<body>

<h2>JavaScript Strings</h2>

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

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

<script>

var carName1 = "Volvo XC60"; // Double quotes

var carName2 = 'Volvo XC60'; // Single quotes

document.getElementById("demo").innerHTML =

carName1 + " " + carName2;

</script>

</body></html>
```

String Length

The length of a string is found with the built-in `length` property :

Example:

```
<html>
```

```
<body>

<h2>JavaScript String Properties</h2>

<p>The length property returns the length of a string:</p>

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

<script>

var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

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

</script>

</body></html>
```

Special Characters

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

Example:

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

the solution of the problem

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

Example:

```
<html>

<body>

<h2>JavaScript Strings</h2>

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

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

<script>
```

```
var x = "We are the so-called \"Vikings\" from the north.";
document.getElementById("demo").innerHTML = x;
</script>
</body></html>
```

Strings Can be Objects

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

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

String Length

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

Example:

```
<html>
<body>

<h2>JavaScript String Properties</h2>
<p>The length property returns the length of a string:</p>
<p id="demo"></p>
<script>
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
document.getElementById("demo").innerHTML = txt.length;
</script>
</body></html>
```

Finding a String in a String

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

Example:

```
<html>

<body>

<h2>JavaScript String Methods</h2>

<p>The indexOf() method returns the position of the first occurrence of a specified text:</p>

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

<script>

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

var pos = str.indexOf("locate");

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

</script>

</body></html>
```

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

Example:

```
<html>

<body>

<h2>JavaScript String Methods</h2>

<p>The lastIndexOf() method returns the position of the last occurrence of a specified text:</p>

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

```
<script>
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate");
document.getElementById("demo").innerHTML = pos;
</script>
</body></html>
```

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

Example:

```
<html>
<body>
<h2>JavaScript String Methods</h2>
<p>Both indexOf(), and lastIndexOf() return -1 if the text is not found:</p>
<p id="demo"></p>
<script>
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("John");
document.getElementById("demo").innerHTML = pos;
</script>
</body></html>
```

Extracting String Parts

There are 3 methods for extracting a part of a string:

- `slice(start, end)`
- `substring(start, end)`

- `substr(start, length)`

The slice() Method

`slice()` extracts a part of a string and returns the extracted part in a new string.

The method takes 2 parameters: the start position, and the end position (end not included).

This example slices out a portion of a string from position 7 to position 12 (13-1):

Example:

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(7, 13);
```

```
<html>

<body>



<h2>JavaScript String Methods</h2>

<p>The slice() method extract a part of a string

and returns the extracted parts in a new string:</p>

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

<script>

var str = "Apple, Banana, Kiwi";

var res = str.slice(7,13);

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

</script>

</body></html>
```

If a parameter is negative, the position is counted from the end of the string.

This example slices out a portion of a string from position -12 to position -6:

Example:

```
<html>

<body>

<h2>JavaScript String Methods</h2>

<p>The slice() method extract a part of a string

and returns the extracted parts in a new string:</p>

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

<script>

var str = "Apple, Banana, Kiwi";

var res = str.slice(-12,-6);

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

</script>

</body></html>
```

If you omit the second parameter, the method will slice out the rest of the string:

```
<html>

<body>

<h2>JavaScript String Methods</h2>

<p>The slice() method extract a part of a string

and returns the extracted parts in a new string:</p>

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

<script>

var str = "Apple, Banana, Kiwi";
```

```
var res = str.slice(7);
document.getElementById("demo").innerHTML = res;
</script>
</body></html>
```

or, counting from the end:

```
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The slice() method extract a part of a string
and returns the extracted parts in a new string:</p>
<p id="demo"></p>
<script>
var str = "Apple, Banana, Kiwi";
var res = str.slice(-12)
document.getElementById("demo").innerHTML = res;
</script>
</body></html>
```

he substring() Method

`substring()` is similar to `slice()`.

The difference is that `substring()` cannot accept negative indexes.

Example:

```
<html>
<body>
<h2>JavaScript String Methods</h2>
<p>The substr() method extract a part of a string
and returns the extracted parts in a new string:</p>
<p id="demo"></p>
<script>
var str = "Apple, Banana, Kiwi";
var res = str.substring(7,13);
document.getElementById("demo").innerHTML = res;
</script>
</body></html>
```

If you omit the second parameter, `substr()` will slice out the rest of the string.

Example:

```
<html>
<body>
<h2>JavaScript String Methods</h2>
<p>The substr() method extract a part of a string
and returns the extracted parts in a new string:</p>
<p id="demo"></p>
<script>
var str = "Apple, Banana, Kiwi";
var res = str.substr(7);
```

```
document.getElementById("demo").innerHTML = res;
</script>
</body></html>
```

Replacing String Content

The `replace()` method replaces a specified value with another value in a string:

Exampl:

```
<html>
<body>
<h2>JavaScript String Methods</h2>
<p>Replace "Microsoft" with "W3Schools" in the paragraph below:</p>
<button onclick="myFunction()">Try it</button>
<p id="demo">Please visit Microsoft!</p>
<script>
function myFunction() {
  var str = document.getElementById("demo").innerHTML;
  var txt = str.replace("Microsoft","W3Schools");
  document.getElementById("demo").innerHTML = txt;
}
</script>
</body></html>
```

Converting to Upper and Lower Case

A string is converted to upper case with `toUpperCase()`:

Example:

```
<html>
<body>
<p>Convert string to upper case:</p>
<button onclick="myFunction()">Try it</button>
<p id="demo">Hello World!</p>
<script>
function myFunction() {
  var text = document.getElementById("demo").innerHTML;
  document.getElementById("demo").innerHTML = text.toUpperCase();
}
</script>
</body></html>
```

The concat() Method

`concat()` joins two or more strings:

Example:

```
<html>
<body>
<h2>JavaScript String Methods</h2>
<p>The concat() method joins two or more strings:</p>
<p id="demo"></p>
<script>
```

```
var text1 = "Hello";

var text2 = "World!";

var text3 = text1.concat(" ",text2);

document.getElementById("demo").innerHTML = text3;

</script>

</body></html>
```

String.trim()

The `trim()` method removes whitespace from both sides of a string:

Example:

```
<html>

<body>

<h2>JavaScript String.trim()</h2>

<p>Click the button to alert the string with removed whitespace.</p>

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

<p><strong>Note:</strong> The trim() method is not supported in Internet Explorer 8 and earlier versions.</p>

<script>

function myFunction() {

  var str = "     Hello World!     ";

  alert(str.trim());

}

</script>

</body></html>
```

Converting a String to an Array

A string can be converted to an array with the `split()` method:

Example:

```
<html>

<body>

<p>Click "Try it" to display the first array element, after a string split.</p>

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

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

<script>

function myFunction() {

  var str = "a,b,c,d,e,f";

  var arr = str.split(",");

  document.getElementById("demo").innerHTML = arr[0];

}

</script>

</body></html>
```

JavaScript Numbers

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

Example:

```
<html>

<body>

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

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

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

<script>

var x = 3.14;

var y = 3;

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

</script>

</body></html>
```

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

```
<html>

<body>

<h2>JavaScript Numbers</h2>

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

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

<script>

var x = 123e5;

var y = 123e-5;

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

</script>

</body></html>
```

If you add a number and a string, the result will be a string concatenation:

```
<html>
```

```
<body>

<h2>JavaScript Numbers</h2>

<p>If you add a number and a numeric string, the result will be a concatenated string:</p>

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

<script>

var x = 10;

var y = "20";

var z = x + y;

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

</script>

</body></html>
```

Numeric Strings

JavaScript strings can have numeric content:

```
var x = 100;          // x is a number

var y = "100";        // y is a string
```

The toFixed() Method

`toFixed()` returns a string, with the number written with a specified number of decimals:

```
<html>
```

```
<body>

<h2>JavaScript Number Methods</h2>

<p>The toFixed() method rounds a number to a given number of digits.</p>

<p>For working with money, toFixed(2) is perfect.</p>

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

<script>

var x = 9.656;

document.getElementById("demo").innerHTML =

  x.toFixed(0) + "<br>" +

  x.toFixed(2) + "<br>" +

 x.toFixed(4) + "<br>" +

  x.toFixed(6);

</script>

</body></html>
```

The toPrecision() Method

`toPrecision()` returns a string, with a number written with a specified length:

```
<html>

<body>

<h2>JavaScript Number Methods</h2>

<p>The toPrecision() method returns a string, with a number written with a specified length:</p>

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

<script>
```

```
var x = 9.656;

document.getElementById("demo").innerHTML =

  x.toPrecision() + "<br>" +

  x.toPrecision(2) + "<br>" +

  x.toPrecision(4) + "<br>" +

  x.toPrecision(6);

</script>

</body></html>
```

The parseInt() Method

`parseInt()` parses a string and returns a whole number. Spaces are allowed. Only the first number is returned:

```
<html>

<body>

<h2>JavaScript Global Functions</h2>

<p>The global JavaScript function parseInt() converts strings to numbers:</p>

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

<script>

document.getElementById("demo").innerHTML =

  parseInt("10") + "<br>" +

  parseInt("10.33") + "<br>" +

  parseInt("10 6") + "<br>" +

  parseInt("10 years") + "<br>" +
```

```
 parseInt("years 10");
</script>
</body></html>
```

Number Properties

Property	Description
MAX_VALUE	Returns the largest number possible in JavaScript
MIN_VALUE	Returns the smallest number possible in JavaScript
POSITIVE_INFINITY	Represents infinity (returned on overflow)
NEGATIVE_INFINITY	Represents negative infinity (returned on overflow)
NaN	Represents a "Not-a-Number" value

JavaScript MIN_VALUE and MAX_VALUE

`MAX_VALUE` returns the largest possible number in JavaScript.

```
<html>
<body>
<h2>JavaScript Number Object Properties</h2>
<p>MAX_VALUE returns the largest possible number in JavaScript.</p>
```

```
<p id="demo"></p>
<script>
var x = Number.MAX_VALUE;
document.getElementById("demo").innerHTML = x;
</script>
</body></html>
```

JavaScript Arrays

JavaScript arrays are used to store multiple values in a single variable.

Example:

```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p id="demo"></p>
<script>
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>
</body></html>
```

Creating an Array

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

Syntax:

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

Example:

```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p id="demo"></p>
<script>
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>
</body></html>
```

Using the JavaScript Keyword new

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

```
<html><body>
<h2>JavaScript Arrays</h2>
<p id="demo"></p>
<script>
var cars = new Array("Saab", "Volvo", "BMW");
document.getElementById("demo").innerHTML = cars;
</script>
</body></html>
```

Access the Elements of an Array

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

Example:

```
<html>

<body>

<h2>JavaScript Arrays</h2>

<p>JavaScript array elements are accessed using numeric indexes (starting from 0).</p>

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

<script>

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

document.getElementById("demo").innerHTML = cars[0];

</script>

</body></html>
```

Changing an Array Element

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

Example:

```
<html>

<body>

<h2>JavaScript Arrays</h2>

<p>JavaScript array elements are accessed using numeric indexes (starting from 0).</p>

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

```
<script>
var cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
document.getElementById("demo").innerHTML = cars;
</script>
</body></html>
```

Access the Full Array

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

Example:

```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p id="demo"></p>
<script>
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>
</body></html>
```

Arrays are Objects

Arrays are a special type of objects.

Arrays use **numbers** to access its "elements".

But Objects use **names** to access its "members". In this example, `person.firstName` returns John:

Example:

```
<html>

<body>

<h2>JavaScript Objects</h2>

<p>JavaScript uses names to access object properties.</p>

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

<script>

var person = {firstName:"John", lastName:"Doe", age:46};

document.getElementById("demo").innerHTML = person["firstName"];

</script>

</body></html>
```

The length Property

The `length` property of an array returns the length of an array (the number of array elements).

Example:

```
<html>

<body>

<h2>JavaScript Arrays</h2>

<p>The length property returns the length of an array.</p>

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

```
<script>

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

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

</script>

</body></html>
```

Accessing the First Array Element

Example:

```
<html>

<body>

<h2>JavaScript Arrays</h2>

<p>JavaScript array elements are accesses using numeric indexes (starting from 0).</p>

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

<script>

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

first = fruits[0];

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

</script>

</body>

</html></html>
```

Accessing the Last Array Element

Example:

```
<html>

<body>

<h2>JavaScript Arrays</h2>

<p>JavaScript array elements are accesses using numeric indexes (starting from 0).</p>

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

<script>

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

var last = fruits[fruits.length-1];

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

</script>

</body></html>
```

Looping Array Elements

The safest way to loop through an array, is using a `for` loop:

Example:

```
<html>

<body>

<h2>JavaScript Arrays</h2>

<p>The best way to loop through an array is using a standard for loop:</p>

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

```
<script>
var fruits, text, fLen, i;
fruits = ["Banana", "Orange", "Apple", "Mango"];
fLen = fruits.length;

text = "<ul>";
for (i = 0; i < fLen; i++) {
  text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";
document.getElementById("demo").innerHTML = text;
</script>
</body></html>
```

Adding Array Elements

The easiest way to add a new element to an array is using the `push()` method:

Example:

```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p>The push method appends a new element to an array.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
```

```
<script>

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

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


function myFunction() {

  fruits.push("Lemon");

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

}

</script>

</body></html>
```

The Difference Between Arrays and Objects

In JavaScript, **arrays** use **numbered indexes**.

In JavaScript, **objects** use **named indexes**.

When to Use Arrays. When to use Objects.

- JavaScript does not support associative arrays.
- You should use **objects** when you want the element names to be **strings (text)**.
- You should use **arrays** when you want the element names to be **numbers**.

Converting Arrays to Strings

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

Example:

```
<html>
```

```
<body>

<h2>JavaScript Array Methods</h2>

<h2>toString()</h2>

<p>The toString() method returns an array as a comma separated string:</p>

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

<script>

var 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.

```
<html>

<body>

<h2>JavaScript Array Methods</h2>

<h2>join()</h2>

<p>The join() method joins array elements into a string.</p>

<p>It this example we have used " * " as a separator between the
elements:</p>

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

<script>

var 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.

pop is used to remove the last element of array

```
<html>

<body>

<h2>JavaScript Array Methods</h2>

<h2>pop()</h2>

<p>The pop() method removes the last element from an array.</p>

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

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

<script>

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

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

fruits.pop();

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

</script>

</body></html>
```

Push is used to add element at the end of the array

Example:

```
<html>

<body>

<h2>JavaScript Array Methods</h2>
```

```
<h2>push()</h2>

<p>The push() method appends a new element to an array.</p>

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

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

<script>

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

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

function myFunction() {

  fruits.push("Kiwi");

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

}

</script>

</body></html>
```

Shifting Elements

The `shift()` method removes the first array element

```
<html>

<body>

<h2>JavaScript Array Methods</h2>

<h2>shift()</h2>

<p>The shift() method removes the first element of an array (and "shifts" all other elements to the left):</p>

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

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

<script>

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

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

fruits.shift();

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

</script>

</body></html>
```

The `unshift()` method adds a new element to an array (at the beginning),

```
<html>

<body>

<h2>JavaScript Array Methods</h2>

<h2>unshift()</h2>

<p>The unshift() method adds new elements to the beginning of an
array.</p>

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

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

<script>

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

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

function myFunction() {

  fruits.unshift("Lemon");

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

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

Changing Elements

Array elements are accessed using their **index number**:

```
<html>
<body>
<h2>JavaScript Array Methods</h2>
<p>Array elements are accessed using their index number:</p>
<p id="demo1"></p><p id="demo2"></p>
<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
fruits[0] = "Kiwi";
document.getElementById("demo2").innerHTML = fruits;
</script>
</body></html>
```

Deleting Elements

can be deleted by using the JavaScript operator `delete`:

```
<html>
<body>
<h2>JavaScript Array Methods</h2>
```

```
<p>Deleting elements leaves undefined holes in an array.</p>

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

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

<script>

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

document.getElementById("demo1").innerHTML =

"The first fruit is: " + fruits[0];

delete fruits[0];

document.getElementById("demo2").innerHTML ="The first fruit is: " +
fruits[0];

</script>

</body></html>
```

Splicing an Array

The `splice()` method can be used to add new items to an array

Where you want and also and more then one through the index[];

```
<html>

<body>

<h2>JavaScript Array Methods</h2>

<h2>splice()</h2>

<p>The splice() method adds new elements to an array.</p>

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

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

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

<script>

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

document.getElementById("demo1").innerHTML = "Original Array:<br>" +
fruits;

function myFunction() {

  fruits.splice(2, 0, "Lemon", "Kiwi");
document.getElementById("demo2").innerHTML = "New Array:<br>" + fruits;

}

</script>

</body></html>
```

Merging (Concatenating) Arrays

The `concat()` method creates a new array by merging (concatenating) existing arrays:

Example:

```
<html>

<body>

<h2>JavaScript Array Methods</h2>

<h2>concat()</h2>

<p>The concat() method is used to merge (concatenate) arrays:</p>
```

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

<script>

var myGirls = ["Cecilie", "Lone"];

var myBoys = ["Emil", "Tobias", "Linus"];

var myChildren = myGirls.concat(myBoys);

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

</script>

</body></html>
```

Creating Date Objects

Date objects are created with the `new Date()` constructor.

There are **4 ways** to create a new date object:

```
new Date()
new Date(year, month, day, hours, minutes, seconds, milliseconds)
new Date(milliseconds)
new Date(date string)
```

```
Example:

<html>

<body>

<h2>JavaScript new Date()</h2>

<p>Using new Date(), creates a new date object with the current date and time:</p>

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

<script>

var d = new Date();
```

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

</script>

</body></html>
```

Second Example:

```
<html>

<body>

<h2>JavaScript Date()</h2>

<p>The toUTCString() method converts a date to a UTC string (a date display standard):</p>

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

<script>

var d = new Date();

document.getElementById("demo").innerHTML = d.toUTCString();

</script>

</body></html>
```

Third Example:

```
<html>

<body>

<h2>JavaScript toDateString()</h2>

<p>The toDateString() method converts a date to a date string:</p>

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

<script>

var d = new Date();
```

```
document.getElementById("demo").innerHTML = d.toDateString();
</script>
</body></html>
```

JavaScript Get Date Methods

These methods can be used for getting information from a date object:

Method	Description
getFullYear()	**Get the year as a four digit number (yyyy)**
getMonth()	**Get the month as a number (0-11)**
getDate()	**Get the day as a number (1-31)**
getHours()	**Get the hour (0-23)**
getMinutes()	**Get the minute (0-59)**
getSeconds()	**Get the second (0-59)**
getMilliseconds()	**Get the millisecond (0-999)**

getTime()	**Get the time (milliseconds since January 1, 1970)**
getDay()	**Get the weekday as a number (0-6)**
Date.now()	**Get the time. ECMAScript 5.**

Exmaple one of that :

```
<html>
<body>
<h2>JavaScript getFullYear()</h2>
<p>The getFullYear() method returns the full year of a date:</p>
<p id="demo"></p>
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d.getFullYear();
// document.getElementById("demo").innerHTML = d.getMonth();
// document.getElementById("demo").innerHTML = d.getDate();
// etc……
</script>
</body></html>
```

JavaScript Math Object

The JavaScript Math object allows you to perform mathematical tasks on numbers.

Example:

```
<html>
<body>
<h2>JavaScript Math.PI</h2>
<p>Math.PI returns the ratio of a circle's circumference to its diameter:</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = Math.PI;
</script>
</body></html>
```

Math.pow()

`Math.pow(x, y)` returns the value of x to the power of y:

```
<html>
<body>
<h2>JavaScript Math.pow()</h2>
<p>Math.pow(x,y) returns the value of x to the power of y:</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = Math.pow(8,2);
</script>
</body></html>
```

Math.sqrt()

`Math.sqrt(x)` returns the square root of x:

```
<html>
<body>
<h2>JavaScript Math.sqrt()</h2>
<p>Math.sqrt(x) returns the square root of x:</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = Math.sqrt(64);
</script>
</body></html>
```

Math.abs()

`Math.abs(x)` returns the absolute (positive) value of x:

```
<html>
<body>
<h2>JavaScript Math.abs()</h2>
<p>Math.abs(x) returns the absolute (positive) value of x:</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = Math.abs(-4.4);
</script>
</body></html>
```

Math.random()

`Math.random()` returns a random number between 0 (inclusive), and 1 (exclusive):

```
<html>

<body>

<h2>JavaScript Math.random()</h2>

<p>Math.random() returns a random number between 0 and 1:</p>

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

<script>

document.getElementById("demo").innerHTML = Math.random();

</script>

</body></html>
```

```
<html>

<body>

<p>Display the value of Boolean(10 > 9):</p>

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

The Boolean() Function

You can use the `Boolean()` function to find out if an expression (or a variable) is true:

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

<script>

function myFunction() {

  document.getElementById("demo").innerHTML = Boolean(10 > 9);
```

```
}

</script>

</body></html>
```

Comparison and Logical Operators

Comparison and Logical operators are used to test for `true` or `false`.

Comparison operators are used in logical statements to determine equality or difference between variables or values.

the table below explains the comparison operators:

Operator	Description	Comparing	Returns	Try it
==	equal to	x == 8	false	Try it »
		x == 5	true	Try it »
		x == "5"	true	Try it »
===	equal value and equal type	x === 5	true	Try it »
		x === "5"	false	Try it »
!=	not equal	x != 8	true	Try it »

!==	not equal value or not equal type	x !== 5	false	Try it »
		x !== "5"	true	Try it »
		x !== 8	true	Try it »
>	greater than	x > 8	false	Try it »
<	less than	x < 8	true	Try it »
>=	greater than or equal to	x >= 8	false	Try it »
<=	less than or equal to	x <= 8	true	Try it »

How Can it be Used

Comparison operators can be used in conditional statements to compare values and take action depending on the result:

```
if (age < 18) text = "Too young";
```

Logical Operators

Logical operators are used to determine the logic between variables or values.

Given that `x = 6` and `y = 3`, the table below explains the logical operators:

Operator	Description	Example	Try it
&&	and	(x < 10 && y > 1) is true	Try it »
||	or	(x == 5 || y == 5) is false	Try it »
!	not	!(x == y) is true	

Conditional (Ternary) Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Syntax

```
variablename = (condition) ? value1:value2
```

```
<html>

<body>

<p>Input your age and click the button:</p>

<input id="age" value="18" />

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

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

<script>

function myFunction() {

  var age, voteable;

  age = document.getElementById("age").value;

  voteable = (age < 18) ? "Too young":"Old enough";

  document.getElementById("demo").innerHTML = voteable + " to vote.";

}

</script>

</body></html>
```

JavaScript if else and else if

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

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

Example:

```
<html>

<body>

<p>Display "Good day!" if the hour is less than 18:00:</p>

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

<script>
```

```
if (new Date().getHours() < 18) {
  document.getElementById("demo").innerHTML = "Good day!";
}
</script>
</body></html>
```

The Second Example:

```
<html>
<body>
<p>Click the button to display a time-based greeting:</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  var hour = new Date().getHours();
  var greeting;
  if (hour < 18) {
    greeting = "Good day";
  } else {
    greeting = "Good evening";
  }
  document.getElementById("demo").innerHTML = greeting;
}
</script>
</body></html>
```

The JavaScript Switch Statement

Use the `switch` statement to select one of many code blocks to be executed.

Syntax

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

Example:

<html>

<body>

<script>

var day;

switch (new Date().getDay()) {

  case 0:

    day = "Sunday";

    break;

  case 1:

    day = "Monday";

    break;

  case 2:

```
    day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case  6:
    day = "Saturday";
}
document.getElementById("demo").innerHTML = "Today is " + day;
</script>
</body></html>
```

The break Keyword

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

This will stop the execution of inside the block.

The default Keyword

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

JavaScript Loops

Instead of writing:

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
text += cars[4] + "<br>";
text += cars[5] + "<br>";
```

You can write:

```
<html>

<body>

<h2>JavaScript Loops</h2>

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

<script>

var cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];

var text = "";

var i;

for (i = 0; i < cars.length; i++) {

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

}

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

</script>
```

</body></html>

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
- `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 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:

<html>

<body>

<h2>JavaScript while</h2>

<script>

var text = "";

var i = 0;

while (i < 10) {

  text += "
The number is " + i;

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

JavaScript Break and Continue

The `break` statement "jumps out" of a loop.

The `continue` statement "jumps over" one iteration in the loop.

Exmaple:

```
<html>
<body>
<p>A loop with a break.</p>
<p id="demo"></p>
<script>
var text = "";
var i;
for (i = 0; i < 10; i++) {
  if (i === 3) { break; }
  text += "The number is " + i + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body></html>
```

Example of continue:

```
<html>

<body>

<p>A loop which will skip the step where i = 3.</p>

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

<script>

var text = "";

var i;

for (i = 0; i < 10; i++) {

  if (i === 3) { continue; }

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

}

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

</script>

</body></html>
```

JavaScript Errors - Throw and Try to Catch

The `try` statement lets you test a block of code for errors.

The `catch` statement lets you handle the error.

The `throw` statement lets you create custom errors.

The `finally` statement lets you execute code, after try and catch, regardless of the result.

Exmaple:

```
<html>

<body>

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

<script>

try {

  adddlert("Welcome guest!");

}

catch(err) {

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

}

</script>

</body></html>
```

JavaScript try and catch

The `try` statement allows you to define a block of code to be tested for errors while it is being executed.

The `catch` statement allows you to define a block of code to be executed, if an error occurs in the try block.

The JavaScript statements `try` and `catch` come in pairs:

```
try {
  Block of code to try
}
catch(err) {
  Block of code to handle errors
}
```

The throw Statement

The `throw` statement allows you to create a custom error.

Technically you can **throw an exception (throw an error)**.

The exception can be a JavaScript `String`, a `Number`, a `Boolean` or an `Object`:

Example:

```
<html>
<body>
<p>Please input a number between 5 and 10:</p>
<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="p01"></p>
<script>
function myFunction() {
  var message, x;
  message = document.getElementById("p01");
  message.innerHTML = "";
  x = document.getElementById("demo").value;
  try {
    if(x == "")  throw "empty";
    if(isNaN(x)) throw "not a number";
    x = Number(x);
    if(x < 5)  throw "too low";
    if(x > 10)   throw "too high";
  }
```

```
  catch(err) {
    message.innerHTML = "Input is " + err;
  }
}
</script>
</body></html>
```

The finally Statement

The `finally` statement lets you execute code, after try and catch, regardless of the result:

Syntax

```
try {
  Block of code to try
}
catch(err) {
  Block of code to handle errors
}
finally {
  Block of code to be executed regardless of the try / catch result
}
```

Example:

```
<html>
<body>

<p>Please input a number between 5 and 10:</p>
<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="p01"></p>
```

```
<script>
function myFunction() {
  var message, x;
  message = document.getElementById("p01");
  message.innerHTML = "";
  x = document.getElementById("demo").value;
  try {
    if(x == "")  throw "is empty";
    if(isNaN(x)) throw "is not a number";
    x = Number(x);
    if(x > 10)   throw "is too high";
    if(x < 5)  throw "is too low";
  }
  catch(err) {
    message.innerHTML = "Input " + err;
  }
  finally {
    document.getElementById("demo").value = "";
  }
}
</script>
</body></html>
<html>
```

```
<body>

<h2>The JavaScript <b>this</b> Keyword</h2>

<p>In this example, <b>this</b> represents the <b>person</b>
object.</p>

<p>Because the person object "owns" the fullName method.</p>

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

<script>

// Create an object:

var 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>
```

JavaScript Let

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

These two keywords provide **Block Scope** variables (and constants) in JavaScript.

```
var carName = "Volvo";

// code here can use carName

function myFunction() {
  // code here can also use carName
}
```

Exmaple:

```
<html>

<body>

<h2>JavaScript Scope</h2>

<p>A GLOBAL variable can be accessed from any script or function.</p>

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

<script>

var carName = "Volvo";

myFunction();

function myFunction() {

 document.getElementById("demo").innerHTML = "I can display " + carName;

}

</script>

</body></html>
```

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

Exmaple:

<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>
```

JavaScript - HTML DOM Methods

HTML DOM methods are **actions** you can perform (on HTML Elements).

HTML DOM properties are **values** (of HTML Elements) that you can set or change.

What is the HTML DOM?

The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:

- The HTML elements as **objects**
- The **properties** of all HTML elements
- The **methods** to access all HTML elements
- The **events** for all HTML elements

In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

JavaScript - HTML DOM Methods

HTML DOM methods are **actions** you can perform (on HTML Elements).

HTML DOM properties are **values** (of HTML Elements) that you can set or change

Example:
<html>

<body>

<h2>My First Page</h2>

<script>

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

</script>

</body></html>

In the example above, `getElementById` is a **method**, while `innerHTML` is a **property**.

The HTML DOM Document Object

The document object represents your web page.

If you want to access any element in an HTML page, you always start with accessing the document object.

Finding HTML Elements

Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

Changing HTML Elements

Property	Discription
element.innerHTML = *new html content*	Change the inner HTML of an element
element.*attribute* = *new value*	Change the attribute value of an HTML element
element.style.*property* = *new style*	Change the style of an HTML element

Method	Discription
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element

document.appendChild(*element*)	Add an HTML element
document.replaceChild(*new, old*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

Adding Events Handlers

Method

document.getElementById(*id*).onclick = function(){*code*}

Description

Adding event handler code to an onclick event

Finding HTML Objects

Property	Description
Document.anchors	Returns all <a> elements that have a name attribute
document.applets	Returns all <applet> elements (Deprecated in HTML5)
document.baseURI	Returns the absolute base URI of the document
document.body	Returns the <body> element
document.cookie	Returns the document's cookie
document.doctype	Returns the document's doctype
document.documentElement	Returns the <html> element
document.documentMode	Returns the mode used by the browser
document.documentURI	Returns the URI of the document

document.domain Returns the domain name of the document server

JavaScript HTML DOM Elements

Finding HTML Elements

- Finding HTML elements by id
- Finding HTML elements by tag name
- Finding HTML elements by class name
- Finding HTML elements by CSS selectors
- Finding HTML elements by HTML object collections

Finding HTML Element by Id

The easiest way to find an HTML element in the DOM, is by using the element id.

This example finds the element with `id="intro"`:

```
<html>

<body>

<h2>Finding HTML Elements by Id</h2>

<p id="intro">Hello World!</p>

<p>This example demonstrates the <b>getElementsById</b>
method.</p>

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

<script>

var myElement = document.getElementById("intro");

document.getElementById("demo").innerHTML =

"The text from the intro paragraph is " + myElement.innerHTML;

</script>
```

```
</body></html>

<html>

<body>

<h2>Finding HTML Elements by Tag Name</h2>

<p>Hello World!</p>

<p>This example demonstrates the <b>getElementsByTagName</b>
method.</p>

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

<script>

var x = document.getElementsByTagName("p");

document.getElementById("demo").innerHTML =

'The text in first paragraph (index 0) is: ' + x[0].innerHTML;

</script>

</body></html>
```

Finding HTML Elements by CSS Selectors

If you want to find all HTML elements that match a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the `querySelectorAll()` method.

This example returns a list of all `<p>` elements with `class="intro"`.

Example:

```
<html>

<body>
```

```
<h2>Finding HTML Elements by Query Selector</h2>

<p>Hello World!</p>

<p class="intro">The DOM is very useful.</p>

<p class="intro">This example demonstrates the <b>querySelectorAll</b>
method.</p>

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

<script>

var x = document.querySelectorAll("p.intro");

document.getElementById("demo").innerHTML =

'The first paragraph (index 0) with class="intro": ' + x[0].innerHTML;

</script>

</body></html>
```

Finding HTML Elements by HTML Object Collections

Exmaple:

```
<html>

<body>

<h2>Finding HTML Elements Using document.forms</h2>

<form id="frm1" action="https://www.w3schools.com/action_page.php">

  First name: <input type="text" name="fname" value="Donald"><br>

  Last name: <input type="text" name="lname" value="Duck"><br><br>
```

```
  <input type="submit" value="Submit">
</form>
<p>Click "Try it" to display the value of each element in the form.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  var x = document.forms["frm1"];
  var text = "";
  var i;
  for (i = 0; i < x.length ;i++) {
    text += x.elements[i].value + "<br>";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>
</body></html>
```

JavaScript HTML DOM - Changing CSS

Changing HTML Style

To change the style of an HTML element, use this syntax:

```
document.getElementById(id).style.property = new style
```

The following example changes the style of a `<p>` element:

Example:

```
<html>
<body>
<p id="p1">Hello World!</p>
<p id="p2">Hello World!</p>
<script>
document.getElementById("p2").style.color = "blue";
document.getElementById("p2").style.fontFamily = "Arial";
document.getElementById("p2").style.fontSize = "larger";
</script>
<p>The paragraph above was changed by a script.</p>
</body></html>
```

JavaScript HTML DOM Animation

Style the Elements

The container element should be created with style = "`position: relative`".

The animation element should be created with style = "`position: absolute`".

```
<html>
```

```
<style>
#container {
  width: 400px;
  height: 400px;
  position: relative;
  background: yellow;
}
#animate {
  width: 50px;
  height: 50px;
  position: absolute;
  background-color: red;
}
</style>
<body>
<p><button onclick="myMove()">Click Me</button></p>
<div id ="container">
  <div id ="animate"></div>
</div>
<script>
function myMove() {
```

```
  var elem = document.getElementById("animate");
  var pos = 0;
  var id = setInterval(frame, 5);
  function frame() {
    if (pos == 350) {
      clearInterval(id);
    } else {
      pos++;
      elem.style.top = pos + "px";
      elem.style.left = pos + "px";
    }
  }
}
</script>
</body></html>
```

JavaScript HTML DOM Events

Reacting to Events

A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.

To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

Examples of HTML events:

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

Exmaple:

```
<html>

<body>

<p>Click the button to display the date.</p>

<button onclick="displayDate()">The time is?</button>

<script>

function displayDate() {

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

}

</script>

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

</body></html>
```

Second Example:

```
<html>

<body>

<p>Click the button to display the date.</p>

<button onclick="displayDate()">The time is?</button>

<script>

function displayDate() {

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

```
}
</script>
<p id="demo"></p>
</body></html>
```

JavaScript HTML DOM EventListener

The addEventListener() method

Add an event listener that fires when a user clicks a button:

Example:

```
<html>
<body>

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method to add many events on the same button.</p>

<button id="myBtn">Try it</button>

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

```
<script>
var x = document.getElementById("myBtn");
x.addEventListener("mouseover", myFunction);
x.addEventListener("click", mySecondFunction);
x.addEventListener("mouseout", myThirdFunction);

function myFunction() {
  document.getElementById("demo").innerHTML += "Moused
over!<br>";
}

function mySecondFunction() {
  document.getElementById("demo").innerHTML += "Clicked!<br>";
}function myThirdFunction() {
  document.getElementById("demo").innerHTML += "Moused
out!<br>";
}
</script>
</body></html>
```