JAVASCRIPT

INDEX

Javascript Tutorial :

1) Javascript Introduction

2) Javascript Where to

3) Javascript Output

4) Javascript Statements

5) Javascript Comments

6) Javascript Variables

7) Javascript Const

8) Javascript Operators

@coders.world

9) Javascript Arithmetic

10) Javascript Data Types

11) Javascript Functions

12) Javascript Objects

13) Javascript Events

14) Javascript String Methods

15) Javascript Numbers

16) Javascript Arrays

17) Javascript Array Methods

18) Javascript Array Search

19) Javascript Date Formats

20) Javascript Math object

21) Javascript Math Methods

@coders_world

22) Javascript Comparison

23) Javascript Logical Operators

24) Javascript If, else, and else if

25) Javascript Switch Statement

26) Javascript Javascript Break

27) Javascript For Loop

28) Javascript While Loop

29) Javascript Break and Continue

30) Javascript Sets

31) Javascript Maps

32) Javascript Type Conversion

33) Javascript Bitwise Operations

34) Javascript Operator precedence

@coders.world

35) Javascript Errors

36) Javascript this keywords

37) Javascript Arrow function

38) Javascript JSON

39) Javascript Best practices

40) Javascript Initialize Variables.

@coders_world

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").inne
rHTML = "Hello JavaScript";
```

JavaScript can change HTML Attribute Values

In this example Javascript changes the value of the src (source) attribute of an <img> tag.

Javascript functions and events

A Javasript function is a block of Javascript code, that can be executed When "called" For.

@coders.world

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

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.

Javasript 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.

Example ↓

```
<Script>
 function myfunction () {
    document.getElementById("demo").in
 nerHTML = "paragraph changed.";
}
</Script>
```

@coders.world

JavaScript in <body>

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

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

Example ↓

```
<!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>
```

External Javascript

Scripts can also be placed in exeternal files:

External file: myScript.js

```
function myFunction () {
document.getElementById ("demo"). inne
rHTML = "paragraph changed.";
}
```

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:

JavaScript Output

Javascript Display posibilities

Javascript can "display" data in different ways:

@coders_world

- 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().

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>
```

@coders.world

Example:

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

<script>
function myFunction () {

document.getElementById ("demo"). inne
rHTML = "paragraph changed.";
}

</script >
```

External JavaScript

Scripts can also be placed in external files.

External file: myScript.js

```
function myfunction() {

document.getElementById ("demo"). inne
rHTML = "paragraph Changed.";
}
```

@coders.world

Javascript Output *

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 document.write()

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

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

@coders_world

Using Window.alert()

You can use an alert box to display data.

Example ↓

```
<Script>
Window.alert ( 5 + 6 );
</Script>
```

Using console.log()

For debugging purposes, you can call the consol.log() Method in the browser to display data.

Example ↓

```
<Script>
console.log ( 5 + 6 );
<Script>
```

Javascript Statements :

Javascript statements are composed of:

Values, Operators, Expressions, keywords, and comments.

@coders_world

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

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

Semicolons

Semicolons separate Javascript statements.

Add a semicolon at the end of each executable statement.

Example ↓

```
let a, b, c;        // Declare 3 variables
a = 5 ;             // Assign the value 5
to a
b = 6 ;             // Assign the value 6
to a
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.

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 Code Blocks

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

Example →

```
function myFunction () {
    document.getElementById ("demo1").innerHTML = "Hello Dolly!";
```

@codersworld

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

Javascript Keywords

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

Our Reserved Words References lists all Javascript keywords.

JavaScript Keywords

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

Keyword	Description
var	Declares a variable
let	Declares a block variable
const	Declares a block constant.

@coders.world

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 - Comments :

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

Single Line Comments

Single line comments start with //.

@coders.world

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

Example ↓

```
// change heading:
document.getElementById ("myp") . inner
HTML = " My first paragraph.";
```

Multi-line Comments

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

Example ↓

```
/*
The code below will change
the heading with id = "MyH"
and the paragraph with id = "myp"
in my web page:
*/
document.getElementById ("myH"). inner
HTML = "My first page";
```

@coders-world

JAVAScript Variables :

Variables are Containers for Storing Data

Javascript variables can be declared in 4 Ways:

- Automatically
- Using var
- Using let
- Using const

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 ↓

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

@coders.world

Example ↓

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

Mixed Example ↓

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

JavaScript Let ★

The let keyword was introduced in ES6 (2015)

Variables declared with let have Block Scope.

variables declared with let must be Declared before use

variables declared with let cannot be Redeclared in the Same Scope.

@coders.world

Example ↓

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

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

Javasript 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

Javascript Operators :

Types of Javascript Operators - There are different types of Javascript Operators.

@coders.world

- Arithmetic Operators
- Assignment Operators
- Comparison operators
- String Operators
- Logical Operators
- Bitwise Operators
- Ternary operators
- Type operators.

Javascript Arithmetic Operators

Arithmetic Operators are used perform arithmetic on Numbers.

Example ↓

```
let a = 3;
let x = (100 + 50) * a;
```

Operator	Description
+	Addition
-	Subtraction

@coders.world

*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement.

Javascript Assignment Operators

Assignment Operators assign values to Javascript variables.

The Addition Assignment Operator (+=) Adds a value to a variable.

Assignment

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

@coders.world

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

Javascript Comparision Operators

Operator	Description
==	equal to
===	equal value and equal type
!=	Not equal

@coders.world

!==	Not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary Operator.

Javascript String Comparison

All the comparision operators above can also be used on strings.

Example:

```
let text1 = "A";
let text2 = "B";
let result = text1 < text2;
```

@coders.world

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.

@codders.world

Operator	Description	Example	same as
&	AND	5 & 1	0101 & 0001
|	OR	5 | 1	0101 | 0001
~	NOT	~5	~0101
^	XOR	5 ^ 1	0101 ^ 0001
<<	left-shift	5 << 1	0101 << 1
>>	right shift	5 >> 1	0101 >> 1
>>>	Unsigned Right shift	5 >>> 1	0101 >>> 1

Result →	0001	0101	1010	0100	1010	0010	0010
Decimal →	1	5	10	4	10	2	2

@coders_world

JAVAScript Arithmetic

Javascript Arithmetic Operators

Arithmetic Operators perform arithmetic on numbers (literals or variables).

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation
/	Division
%	Modulus
++	Increment
--	Decrement

@coders.world

Javascript Data Types

Javascript has 8 Datatypes

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

The Object Datatype

The object data type can contain.

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

Example →

```
// Numbers:
let length = 16;
let weight = 7.5;

// Strings:
```

@coders.world

```
let color = "yellow";
let lastname = "Johnson";

// Booleans :
let x = true ;
let y = false ;

// Object :
const person = {firstname : "John",
lastname : "Doe" };

// Array Object :
const cars = [ "saab", "volvo",
"BMW" ];

// Date object :
const date = new Date ("2022-03-25");
```

The Concept of Data Types

In programming, data types is an important concept.

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

@coders.world

Without data types, a computer cannot safely solve this.

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

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

Javascript will treat the example above as.

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

Javascript Objects

Javascript objects are written with curly braces { }.

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

Example:

```
const person = {firstname : "John",
lastname : "Doe", age:50,
eyecolor : "blue"};
```

@Coders.world

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.

Examples↓

typeof "" // Returns "String"
typeof "John" // Returns "String"
typeof "John Doe" // Returns "String"

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).

@coders.world

Javascript Function Syntax

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

The parentheses may include parameter names separated by commas:
(parameter1, paramete2, ...)

```
function name (parameter 1,
parameter2, Parameter 3) {
  // code to be executed
}
```

Function Invocation

The code inside the function will execute when "Something" invokes (calls) the function.

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

@coders.world

Function Return

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

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

Example ↓

```
// function is called, the return
value will end up in x
let x = myfunction( 4, 3 );

function myfunction (a, b) {
// Function returns the product of a
and b
return a * b;
}
```

Local variables

Example ↓

```
function myfunction () {
let carName = "volvo";
// code here CAN use carName
}
```

@coders.world

JavaScript Objects :

Real Life Objects, Properties, and Methods

In real life, a car is an object.

A car has properties like weight and color, and Methods like Start and Stop:

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

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

Example:

```
const person = {
  firstName : " John",
  lastName : " Doe",
  age : 50,
  eyecolor : " blue",
};
```

@coders_world

JavaScript Events

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

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

Here are some examples of HTML events:

- An HTML Web page has finished loading
- An HTML input field was changed
- An HTML button was clicked.

Often, when events happen, you may want to do something.

HTML allows event handler attributes, with Javascript code, to be added to HTML elements.

With single quotes.

<element event = 'some Javascript'>

With double quotes. → <element event = "Some Javascript">

@Coders.World

Example:

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

JavaScript String Methods

Basic String Methods

Javascript Strings are primitive and immutable. All String methods produces a new String without altering the original string.

String length
String charAt()
String charCodeAt()
String at()
String [ ]
String slice()
String substring()
String substr()

String toUpperCase()
String toLowerCase()
String concat()
String trim()
String trimStart()
String trimEnd()
String padStart()
String padEnd()
String repeat()
String replace()
String replaceAll()
String split()

JavaScript String Length

The length property returns the length of a String.

Example 1,

```
let text =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ";
let length = text.length;
```

@coders.world

Extracting String characters

There are 4 Methods for extracting string characters.

- The at (position) Method
- The charAt (position) Method
- The charCodeAt (position) Method
- Using property access [] like in arrays

Example ↓

```
let text = "HELLO WORLD";
let char = text.charAt(0);
```

Extracting String parts

There are 3 Methods for extracting a part of a String.

- slice (start, end)
- substring (start, end)
- substr (start, length)

Example →
```
let text = "Apple","Banana", "kiwi";
let part = text.slice(7, 13);
```

@coders.world

Converting to Upper and Lower Case

A String is converted to upper case with toUpperCase():

A String is converted to lower case with toLowerCase():

Example↓

```
let text1 = "Hello World";
let text2 = text1.toUpperCase();
```

Javascript String toLowerCase()

Example↓

```
Let text1 = "Hello World";
String
Let text2 = text1.toLowerCase();
```

text2 is text1 converted to lower

Javascript String Concat()

Concat() Joins two or More Strings.

Example ↓

```
let text1 = " Hello";
let text2 = " World";
let text3 = text1.concat( " ", text2);
```

Javascript String trim()

The trim() Method removes Whitespace from both sides of a string.

Example ↓

```
let text1 = " Hello World!
    ";
let text2 = text1.trim();
```

Javascript String ReplaceAll()

In 2021, Javascript introduced the String Method replaceAll():

Example →

```
text =
text.replaceAll(" cats", " Dogs");
text =
text.replaceAll(" cats", " dogs");
```

@coders.world

Javascript String Split()

A String can be converted to an array with the Split() Method.

Example↓

```
text.split(",")          // Split on commas
text.split(" ")          // Split on spaces
text.split("|")          // Split on pipe
```

JavaScript String indexOf()

Example↓

```
let text = "please locate where 'locate' occurs!";
let index = text.indexOf("Locate");
```

Javascript String lastIndexOf()

The lastIndexOf() Method returns the index of the last occurrence of a specified text in a String.

@coders.world

Example:

```
let text= "please locate where
'locate occurs!";
let index =
text.lastIndexOf("locate");
```

Javascript String Search()

The search() Method searches a string for a string (or a regular expression) and returns the position of the match:

Example:

```
let text = "please locate where
'locate' occurs!";
text.search("locate");
```

Javascript String Starts With()

The StartWith() Method returns true if a string begins with a specified value.

Otherwise it returns False.

@coders.world

Example ↓

```
let text = "Hello World, Welcome to
the universe.";
text.startsWith("Hello");
```

Javascript Numbers :

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

Example ↓

```
Let x = 3.14;          // A number with
decimals
Let y = 3;             // A Number without
decimals
```

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

Example ↓

```
let x = 123e5;         // 12300000
let y = 123e-5;        // 0.000123
```

@codersworld

Javascript Number Methods

The Number Methods can be used on all Javascript Numbers.

Method	Description
toString()	Returns a number as a string
toExponential()	Returns a number written in exponential notation.
toFixed()	Returns a number with a number of decimals.
toprecision()	Returns a number written with a specified length.
valueOf()	Returns a number as a number.

Converting Variables to Numbers

There are 3 Javascript Methods that can be used to convert a variable to a number.

@coders.world

Method	Description
Number()	Returns a Number converted from its argument.
parseFloat()	parses its argument and returns a Floating point number.
parseInt()	parses its arguement and returns a Whole Number.

Number Object Methods

These Object Methods belong to a Number Object.

Method	Description
Number.isInteger()	Returns true if the arguement is an integer
Number.isSafeInteger()	Returns true if arguement is a safe integer.
Number.parseFloat()	Converts a String to a Number

@coders.world

Number.parseInt() Converts a string to a whole number.

JavaScript Arrays :

Why Use Arrays ?

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

Creating An Array

Syntax

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

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

Example → `const cars = [ "Saab", "volvo", "BMW" ];`

@coders.world

Accessing Array Elements

Access an array element By referring to the index Number.

```
Const cars = ["Saab", "Volvo", "BMW"];
Let car = cars[0];
```

Changing An Array Element

This Statement changes the value of the First element in cars.

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

Converting an Array to a String

The Javascript Method toString() converts an array to a string of (comma separated) array values.

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

@coders.world

```
document.getElementById("demo").inne
rHTML = fruits.toString();
```

Result :

Banana, Orange, Apple, Mango

Javascript Array Methods :

Basic Array Methods

Array length()
Array toString()
Array at()
Array join()
Array pop()
Array push()
Array shift()
Array unshift()
Array delete()
Array concat()
Array copyWithin()
Array Flat()
Array splice()
Array toSpliced()
Array slice()

@coders.world

Javascript Array length

The length property returns the length (size) of an Array.

Example ↓

```
let size = Fruits.length;
```

Javascript Array toString()

The Javascript Method toString() converts an array to a String (comma separated) array values.

Example ↓

```
document.getElementById ("demo").inne
rHTML = Fruits.toString ();
```

Result

Banana, Orange, Apple, Mango

Javascript Array at()

ES2022 introduce the array Method at():

@codeers.world

Example ↓

```
const fruits = [ "Banana", "Orange",
"Apple", "Mango" ];
let fruit = fruits.at (2);
```

Javascript Array Join()

The Join() Method also joins all array elements into a string.

Example ↓

```
const fruits = [ "Banana", "Orange",
"Apple", "Mango" ];
document.getElementById ("demo").inne
rHTML = fruits.join (" * ");
```

Javascript Array pop()

The pop() Method removes the last element from an array.

Example →
```
const fruits = ["Banana", "Orange", "Apple"];
fruits.pop();
```

Javascript Array push()

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

Example ↓

```
Const fruits = ["Banana", "Orange",
"Apple", "Mango"];
fruits.push("Kiwi");
```

Javascript Array length

The length property provides an easy way to append new element to an array.

Example ↓

```
fruits [fruits.length] = "Kiwi";
```

Javascript Array delete()

Example ↓

```
Const fruits = ["Banana", "Orange",
"Apple", "Mango"];
delete fruits [0];
```

@codeis.world

JavaScript Array concat()

The concat() Method creates a new array by merging (concatenating) existing arrays.

Example ↓

```
const mychildren =
myGirls.concat(myBoys);
```

JavaScript Array Search:

Array Find and Search Methods

Array indexOf()
Array lastIndexOf()
Array Includes()

Array find()
Array findIndex()
Array findLast().
Array findLastIndex()

Javascript Array includes()

ECMAScript 2016 introduced Array.includes() to arrays. This allows as to check if an element is present in an array.

@coders.world

Example ↓

```
fruits.includes("Mango");    //  is true
```

Javascript Array IndexOf()

The indexOf() Method searches an array for an element value and returns its position.

Example ↓

```
const fruits = ["Apple", "Orange", "Apple",
  "Mango"];
Let position =
fruits.indexof("Apple") + 1;
```

Javascript Array Sort

The Sort() Method sorts an array alphabetically.

Example ↓

```
const fruits = ["Banana", "Orange",
"Apple", "Mango"];
fruits.sort();
```

@coders.world

Reversing an Array

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

Example ↓

```
Const fruits = [ "Banana", "Orange",
"Apple", "Mango" ];
fruits.reverse();
```

Numeric Sort

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

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

Fix this by providing a compare function.

Example ↓

```
Const points = [ 40, 100, 1, 5, 25,
10 ]
points.sort (function (a,b) { return a - b });
```

@coders.world

Javascript Date Objects

Javascript Date objects let us work with dates:

Sat Apr 13 2024 21:18:42 GMT + 0530
(India Standard Time)

Year: 2024	Month: 4	Day: 13

Hours: 21	Minutes: 18

Seconds: 42

Examples ↓

```
const d = new Date();
```

Note

Date objects are static. The "clock" is not "running".

The computer clock is ticking, date objects are not.

@coders.world

Javascript Date Output

By default, Javascript will use the browser's time zone and display a date as a full text string.

Sat Apr 13 2024 21:18:42 GMT+0530
(India Standard Time)

Creating Date Objects

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

There are 9 ways to create a new code date object.

```
new Date()
new Date (date string)
new Date (year, Month)
new Date (year, Month, day)
new Date (year, Month, day, hours)
new Date (years, Month, day, hours, minutes)
new Date (year, Month, day, hours, minutes, second)
new Date (year, Month, day, hours, minutes, sec. ms)
      Date (milliseconds).
```

@coders.world

Note

Javascript counts months from 0 to 11.

January = 0
December = 11

Javascripts Date Formats :

JavaScript Date Input

There are generally 3 types of Javascript date input formats.

Type	Example
ISO Date	"2015-03-25" (The international Standard)
Short Date	"03-25-2015"
Long Date	"Mar 25 2015" or "25 Mar 2015"

@coders.world

Set Date Methods

Set date methods are used for setting a part of a date.

Method	Description
setDate()	Set the day as a number (1-31)
setfullyear()	Set the year (optionally Month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the Month (0-11)
setseconds()	Set the seconds (0-59)
SetTime()	Set the time (milliseconds since January 1, 1970)

@codersworld

Javascript Math Object :

The Javascript Math object allows you to perform Mathematical tasks on Numbers.

Example ↓

```
Math.PI;
```

Unlike other objects, the Math object has no Constructor.

The Math object is static.

All Methods and properties can be used without creating a Math Object first.

Math properties (constants)

The syntax for any Math property is :
Math.property.

Math Properties (Constants)

The syntax for any Math property is :
Math.property.

@Coders.World

Example ↓

Math.E // returns Euler's number

Math.PI // returns the PI

Math.SQRT2 // returns the square root of 1/2

Math.LN2 // returns the natural logirithm of 2

Math.LN10 // returns the natural logirithm of 10

Math.LOG2E // returns base 2 logarithm of E

Math.LOG10E // returns base 10 logarithm of E

Math.SQRT2 // returns the square root of 2.

Math Methods :

Syntax → Math.method(number)

@coders.world

Number to Integer

There are 4 common Methods to round a Number to an integer.

Math.round (x)	Returns x rounded to its nearest Integer.
Math.ceil (x)	Returns x rounded up to its nearest integer
Math.floor (x)	Returns x rounded down to its nearest integer
Math.trunc(x)	Returns the integer part of x (new in ES6)

Math.round ()

Math.round (x) returns the nearest integer.

Example]

```
Math.round ( 4.6 );
```

@codersworld

Math.random()

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

Example ↓

```
// Returns a random number.
Math.random();
```

Boolean Values Javascript Booleans

Very often, in programming, you will need a data type that can only have one of two values like.

- YES / NO
- ON / OFF
- TRUE / FALSE

The Boolean Function

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

Example → Boolean(10 > 9)

@CodersWorld

Javascript Comparison

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

Operator	Description	Comparing	Returns
==	equal to	x == 8	False
		x == 5	True
		x == "5"	True
===	equal value and equal Type	x === 5	True
		x === "5"	False
!=	Not equal	x != 8	True
!==	Not equal value or not equal type	x !== 5	False
		x !== "5"	True
		x !== 8	True
>	greater than	x > 8	False
<	less than	x < 8	True
>=	greater than or equal to	x >= 8	False

@coders.world

<=	less than or equal to	x<=8	True.

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 to
    buy alcohol";
```

Logical operators :

Logical operators are used to determine the logic between the logical values.

Operator	Description	Example
&&	and	(x<10 && y>1) is true
||	or	(x==5 || y==5) is false

!	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
```

Example↓

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

Javascript If, else, and else if :

Conditional Statements

- Use if to specify a block of code to be executed, if a specified condition is true.

- Use else to specify a block of code to be executed, if the same condition is false

- Use else if to specify a new condition to test, if the first condition is false

- Use switch to specify many alternative blocks of code to be executed.

The if statement

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

Syntax

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

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

@coders.world

The else Statement

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

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

Example ↓

```
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

Result

Good evening

The else if statement

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

Syntax

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

Javascript Switch statement :

Switch statement is used to perform different actions based on different conditions.

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

@coders.world

Syntax

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

This is how it Works :

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

example ↓

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

@coders.world

```
    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 " ;
}
```

Result

Sunday

@coders.world

Javascript Break :

The Break Keyword

This will stop the execution inside the switch block.

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

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

The default keyword

Default keyword specifies the code to run if there is no case match.

Example ↓

```
switch (new Date().getDay()) {
        case 6 :
            text = " Today is saturday " ;
        break ;
      case 0 :
```

```
        text = " Today is sunday ";
        break;
    default:
        text = "Looking forward to the
    Weekend";
}
```

Result

Today is sunday

Javascript For Loop:

Loops can execute a block of code a number of times.

Javascript Loops

Loops are handy, if you want to run the same code over and over again, each time with a different level. value.

you can write →

```
for (let i = 0; i < cars.length;
i++) {
    text += cars[i] + "<br>";
}
```

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>";
```

Different Kinds of Loops

- For - loops through a block of code a number of times.
- for/in - loops through the properties of an Object
- For/of - loops through the values of an iterable Object.
- While - loops through a block of code While a specified condition is true
- do/While - also loops through a block of code While a specified condition is true.

@coders.world

The For loop

The For statement creates a loop with 3 optional expressions.

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

Example:

```
for (let i = 0; i < 5; i++) {
  text += " The number is" + i + "<br>";
}
```

The For In Loop

The Javascript for in statement loops through the properties of an object.

Syntax

```
for (key in object) {
    // code block to be executed
}
```

@coders.world

Example:

```
const person = {fname : "John",
lname :"Doe", age : 25 };

Let text = " ";
for ( let x in person) {
    text += person [x];
}
```

Example Explained

- For in loop iterates over a person object
- Each iteration returns a key (x)
- The key is used to access the value of the key.
- The value of the key is person [x]

The For Of Loop

The Javascript for of statement loops through the values of an iterable object.

It lets you loop over iterable data structures such as Arrays, Strings, Maps, NodeLists, and More:

@coders.world

Syntax

```
For (Variable of iterable) {
    // code block to be executed.
}
```

Javascript While Loop :

The While loop

While loop loops through a block of code as long as a specified condition is true.

Syntax

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

Example 1

```
While (i < 10) {
    text += " The number is " + i;
    i++;
}
```

@coders.world

The Do While Loop

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

Syntax

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

Example 1

```
do {
   text += " The number is " + i;
   i++ ;
}
        (i < 10);
```

Javascript Break and continue

you have already seen the break statement

@coders.world

used in an earlier chapter of this tutorial. It was used to "jump out" of a switch() statement.

The break statement can also be used to jump out of a loop.

The Continue Statement

The Continue Statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 3:

Example ↓

```
for (let i = 0; i < 10; i++) {
    (i===3) { continue; }
  text t = " The number is " + i + "<br>";
}
```

Javascript Labels

To label Javascript statements you precede the

Statements With a label name and a colon.

label : Statements

The break and the continue statements are the only Javascript statements that can "jump out of" a code block.

Syntax

```
break labelname;
continue labelname;
```

JavaScript Sets :

A Javascript set is a collection of unique values. Each value can only occurs once in set.

How to create A Set

- passing an Array to new Set()
- Create a new set and use add() to add values
- Create a New set and use add() to add variables.

@coders.world

Example ↓

```
// Create a set
const letters = new
Set(["a","b","c"]);
```

Essential Set Methods

Method	Description
new Set()	Creates a new set
add()	Adds a new element to set
delete()	Removes an element from a set
has()	Returns true if a value exists in the set
forEach()	Invokes a callback for each element in the set.
values()	Returns an iterator with all the values in a set.

@coders.world

JavaScript Maps :

A Map holds key-value pairs where the keys can be any datatype.

A Map remembers the original insertion order of the keys.

Essential Map Methods

Method	Description
new Map()	Creates a new Map.
set()	Sets the value for a key in a Map
get()	Gets the value for a key in a Map
delete()	Removes a Map element specified by the key.
has()	Returns true if a key exists in a Map.

@coders_world

forEach () — Calls a function for each key/value pair in a Map

entries () — Returns an iterator with the [key, value] pairs in a Map

How to Create a Map

- passing an Array to new Map ()
- Create a Map use Map.set ()

The New Map () Method

Example ↓

```
// Create a Map
const Fruits = new Map ([
  ["apples", 500],
  ["bananas, 300],
  ["Oranges, 200]

]);
```

The get () Method

Example ↓

```
Fruits.get("apples");      // Returns 500
```

Javascript Type Conversion

Javascript variables can converted to a new variable and another data type.

- By the use of Javascript Function
- Automatically by Javascript itself.

Converting Strings to Numbers

Examples ↓

```
Number("3.14")
Number(Math.PI)
Number(" ")
Number("")
```

The typeof Operator

Use the typeof operator to find the data type of a Javascript variable.

@codersworld

Example:

typeof "John"
Returns "String"
typeof 3.14
Returns "Number"
typeof NAN
Returns "Number"
typeof False
Returns "boolean"
typeof [1, 2, 3, 4]
Returns "Object"
typeof { name: 'John', age: 34 }
Returns "Object"
typeof new Date()
Returns "Object"
typeof function () { }
Returns "Function"
typeof myCar
Returns "undefined" *
typeof Null
Returns "Object"

Number Methods

In the chapter Number Methods, you will

@coders.world

Find More methods that can be used to convert Strings to numbers.

Method	Description
Number()	Returns a number, converted from its arguement
parseFloat ()	parses a string and returns a floating point number
parseInt()	parses a string and returns an integer.

Javascript Bitwise Operations :

Operator	Name	Description
&	AND	sets each bit to 1 if both bits are 1
|	OR	sets each bit to 1 if one of two bits is 1
^	XOR	sets each bit to 1 if only one of two bits is 1

@codera.world

~	NOT	Inverts all the bits
<<	zero fill left shift	shifts left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	shifts right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off.
>>>	Zero fill right shift	shifts right by pushing zeros in from the left, and let the rightmost bits fall off.

Javascript Bitwise AND

Operation	Result
0 & 0	0
0 & 1	0

@coders.world

1 & 0	0
1 & 1	1

Javascript Bitwise OR

Operation	Result
0/0	0
0/1	1
1/0	1
1/1	1

Javascript Bitwise XOR

Operation	Result
0 ^ 0	0
0 ^ 1	1
1 ^ 0	1
1 ^ 1	0

@coders_world

Javascript Bitwise AND (&)

Decimal	Binary
5	00000000000000000000000000000101
1	00000000000000000000000000000001
5 & 1	00000000000000000000000000000001
	(1)

Javascript Bitwise OR (|)

Decimal	Binary
5	00000000000000000000000000000101
1	00000000000000000000000000000001
5 | 1	00000000000000000000000000000101
	(5)

Javascript Bitwise XOR (^)

Decimal	Binary
5	00000000000000000000000000000101
1	00000000000000000000000000000001
5 ^ 1	00000000000000000000000000000100
	(4)

@coders.world

Javascript Bitwise NOT (~)

Decimal	Binary
5	00000000000000000000000000000101
~5	11111111111111111111111111111010
	(-6)

Converting Decimal to Binary

Example 1

```
function dec2bin(dec) {
    return (dec >>> 0).toString(2);
}
```

Javascript Operator precedence :

Operator precedence describes the order in which operations are performed in an arithmetic expression.

Multiplication (*) and division (/) have higher precedence than addition (+) and subtraction (-).

@coders.world

As in traditional Mathematics, Multiplication is done first.

```
let x = 100 + 50 * 3;
```

When using parentheses, operations inside the parentheses are computed first.

```
let x = (100 + 50) * 3;
```

Operations with the same precedence (like * and /) are computed from left to right.

```
Let x = 100 / 50 * 3;
```

Javascript Errors

Throw, and Try.... Catch.... Finally

The try statement defines a code block to run (to try).

The catch statement defines a code block to handle any error.

The Finally Statement defines a code block

@coders.world

run regardless to the result.

The throw statement defines a custom error.

The throw Statement

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

The exception can be a Javascript String, a Number, a Boolean or an object.

```
throw "Too big";        // throw a text
throw 500;              // throw a number
```

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
}

The Javascript this keyword

Example:

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

What is this ?

In Javascript, the this keyword refers to an Object.

@coders.world

Which object depends on how this is being invoked (used or called).

Alone, this refers to the global object

In a function, this refers to the global object.

In a function, in strict mode, this is undefined.

In an event, this refers to the element that recieved the event.

Methods like call(), apply(), and bind() can refer this to any object.

this in a Method

```
fullName : function () {
    return this.firstName + " " +
this.LastName;
}
```

this Alone

In a browser window the global object is

[object Window]:

Example ↓

```
Let X = this;
```

Javascript Arrow Function:

Arrow functions were introduced in ES6

Arrow functions use to write shorter function syntax.

Before Arrow:

```
hello = function () {
    return "Hello World!";
}
```

With Arrow Function

```
hello =() => {
    return "Hello World!";
}
```

It gets shorter! If the function has only one

@coders.world

statement, and the statement returns a value, you can remove the brackets and the return keyword.

Javascript JSON

JSON is a format for storing and transporting data.

JSON is often used when data is sent from a server to a web page.

What is JSON?

→ JSON stands for Javascript object notation

→ JSON is a lightweight data inchange format

→ JSON is language independent *

→ JSON is "self-describing" and easy to understand.

Example↓

This JSON syntax defines an employees object an Array of 3 employee records (objects):

@coders.world

JSON Example ↓

```
{
  "employee":[
    "firstName": "John",
  "LastName": "Doe" };
   {"firstName": "Anna",
  "LastName":"Smith"},
   {"firstName : "Peter",
  "LastName": "Jones"}
  ]
}
```

JSON Syntax Rules

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

JavaScript Best Practices :

Avoid Global Variables

Minimize the use of global variables.

@coders.world

This includes all data types, Objects, and functions.

Global variables and functions can be overwritten by other scripts.

Use local variables instead, and learn how to use closures.

Declarations on Top

It is a good coding practice to put all declarations at the top of each script or function.

This will:

- Give cleaner code
- provide a single place to look for local variables
- Make it easier to avoid unwanted (implied) global variables.
- Reduce the possibility of unwanted re-declarations.

@codeers.world

Initialize Variables

It is good coding practice to initialize variable when you declare them.

This will:

- Give cleaner code
- Provide a single place to initialize variables.
- Avoid undefined values.

Don't Use New Object()

- Use " " instead of new String()
- Use 0 instead of new number()
- Use false instead of new Boolean()
- Use { } instead of new Object()
- Use [ ] instead of new Array()
- Use /()/ instead of new RegExp()
- Use function () { } instead of new function()

~ o ~

@Coders.World