Java Script

JavaScript is a popular web scripting language and is used for client-side and server-side development. The JavaScript code can be inserted into HTML pages that can be understood and executed by web browsers while also supporting object-oriented programming abilities.

Scripting language

Scripting language is a programming language that supports script .Scripting prefers high level language ,like python , ruby ,js. . scripting language are interpreted , donot required compilation
Used in web development

How Javascript works

JavaScript is a client-side scripting language and one of the most efficient, commonly used scripting languages. The term .**client-side scripting language means** that it runs at the client-side(or on the client machine) inside the web-browsers, but one important thing to remember is that client's web-browser also needs to support the JavaScript or it must be JavaScript enabled.

Nowadays, most of the modern web browsers support JavaScript and have their JavaScript engines. For example, Google Chrome has its own JavaScript engine called V8.

Some other web-browsers with their JavaScript engines

	Web Browser	JavaScript engines
1.	Edge	Chakra
2.	Safari	JavaScript Core
3.	Firefox	Spidermonkey

With the help of following example, we can understand how JavaScript works:

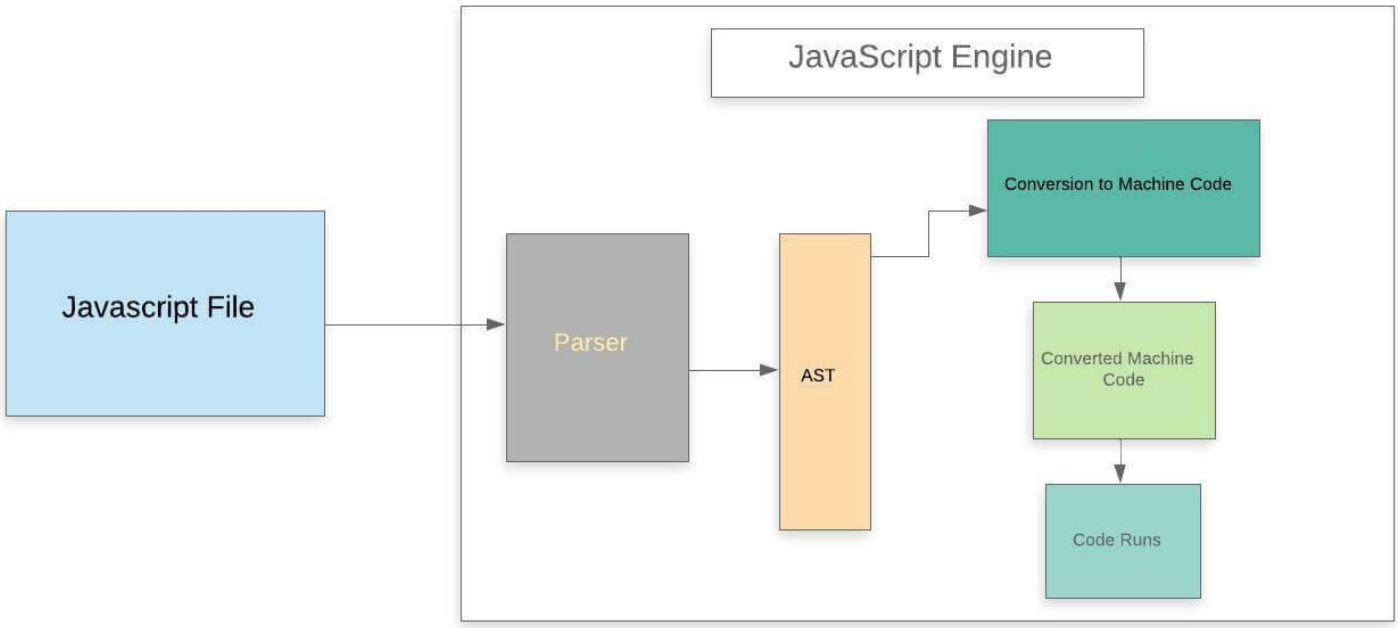
In this example, we have created a simple HTML document and added our JavaScript code in it.

```
<body>
<h1>This how javascript works</h1>
<script>
  alert("Hi,Their");
  console.log("JavaScript");
</script>
</body>
```

In the above program, we just displayed a alert message to the user by using the "alert()" method that is a pre-defined method of JavaScript. We also used the "console.log();" method and passed "JavaScript" as the String and in the inspect mode we can see "JavaScript" in the console as shown in the below output.

Output

We can understand how a typical JavaScript engine works with help of a diagram:



Whenever we run a JavaScript program inside a web browser, JavaScript code is received by the browser's engine and the engine runs the source code to obtain the output.

Step 1: Parser

This is the first stage of the engine, every time we run a JavaScript program, our code is first received by the "parser" inside the JS engine. The parser's job is to check the JavaScript code for syntactic errors in line by line manner because JavaScript is an interpretive scripting language, so whenever an error is detected by the parser, it throws a kind of error and stops execution of the code.

In short, we can say that it parses JavaScript code.

Step 2: AST

Once the parser checks all JavaScript codes and gets satisfied that there are no mistakes/errors in the code, it creates the data structure called AST (it stands for Abstract Syntax Tree).

We can easily understand what is AST with help of following example.

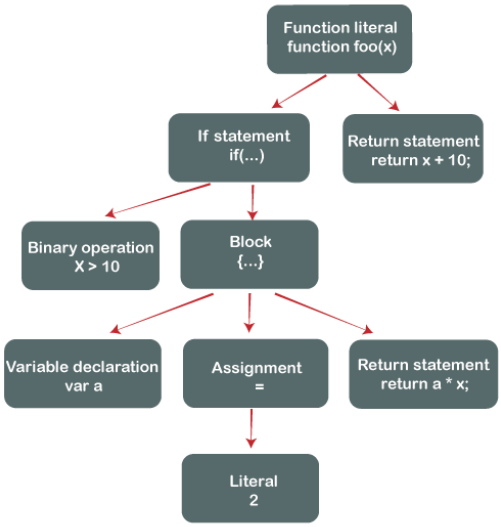
Example

Let's suppose we have a JavaScript program as given below:

Program

```
1. function fun(x) {
2. if (x > 15) {
3. var a = 4;
4. return a * x;
5.     }
6.
7. return x + 10;
8. }
```

Once the parser checks the above JavaScript code, it will create a data structure called **AST** as we have already discussed above. The created AST (Abstract Syntax Tree) looks like the given image.



Note: It is not the exact abstract syntax tree, but it is the pictorial representation of the Abstract Syntax Tree.

Step 3: Conversion to Machine code

Once the Abstract Syntax Tree is created by the parser, the JavaScript engine converts the JavaScript code into the machine code (or in the language that machine can understand).

Step 4: Machine code

When the program written in the JavaScript gets converted in the machine language (or in byte code), the converted code is sent to the system for execution, and finally, that byte code run by the system/engine just like we observe in our first example.

Where To insert Javascript

In HTML, JavaScript code is inserted between `<script>` and `</script>` tags
JavaScript in <head> or <body>
You can place any number of scripts in an HTML document.
Placing scripts at the bottom of the <body> element improves the display speed, because script interpretation slows down the display.
Scripts can also be placed in external files: `<script src="myScript.js"></script>`

JS Output

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()`.

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

Using document.write() after an HTML document is loaded, will **delete all existing HTML**
The document.write() method should only be used for testing

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

`window.print()` method in the browser to print the content of the current window.

JS Syntax

The JavaScript syntax defines two types of values:

- Fixed values
- Variable values

Fixed values are called **Literals**.= document.getElementById("demo").innerHTML = 10.50 + 10;
Variable values are called **Variables**. = <script>
let x = 6;
document.getElementById("demo").innerHTML = x;
</script>

JavaScript Comments

Not all JavaScript statements are "executed".
Code after double slashes `//` or between `/*` and `*/` is treated as a **comment**.

JavaScript Identifiers / Names

Identifiers are JavaScript names.
Identifiers are used to name variables and keywords, and functions.
The rules for legal names are the same in most programming languages.
A JavaScript name must begin with:

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

Subsequent characters may be letters, digits, underscores, or dollar signs.
Numbers are not allowed as the first character in names.
This way JavaScript can easily distinguish identifiers from numbers.
JavaScript is Case Sensitive
JavaScript uses the **Unicode** character set.

JavaScript Variables

Variables are containers for storing data
4 Ways to Declare a JavaScript Variable:

- Using `var`
- Using `let (introduce in 2015 es6)`
- Using `const (introduce in 2015 es6) (cannot change or redeclared )`
- Using nothing (called undeclared)

A variable declared without a value will have the value `undefined`.
The variable carName will have the value `undefined` after the execution of this statement:

```
let carName;
```

JavaScript Scope

Scope determines the accessibility (visibility) of variables.
JavaScript has 3 types of scope:

- Block scope
- Function scope
- Global scope

Global scope = Variables declared with `var`, `let` and `const` are quite similar when declared outside a block.

Block scope = `let` and `const`. . declaration inside block

Function scope = Variables declared with `var`, `let` and `const` are quite similar when declared inside a function.

Data Types

Primitive data type = (means predefined)	Non-Primitive data type=(defined by user)
1. String = "amarth" 2. Number =20 3. Bigint = 200000000 4. Boolean =true or false 5. Undefined = 6. Null 7. Symbol 8. object	Object Datatype 1. object ={name:"amarth",lname:"patel"} 2. Array =[1,2,3,4] 3. Date 4. function 5. map 6. set =(1,2,3,4) (unique)

String and string (method)

String is a sequence of characters.

length	`length` property returns the length of a string `(text.length)`
Slice()	slice(start, end) extracts a part of a string and returns the extracted part in a new string
Substring()	substring(*start*, *end*) Returns a specified part of the string
Replace()	`replace(replace word ,new word)` method replaces a specified value with another value in a string .
replaceALL()	`replaceALL(replace word ,new word)` text = text.`replaceAll`("cats","dogs");
touppercase()	A string is converted to upper case with `toUpperCase()`:
toLowercase()	A string is converted to lower case with `tolowerCase()`:
Concat()	`concat()` joins two or more strings =let text3 = text1.concat(" ",text2);

indexOf()	returns the **index** (position) the **first** occurrence of a string in a string `let index = text.indexOf("locate");`
lastIndexOf()	method returns the **index** of the **last** occurrence of a specified text in a string:
search()	searches a string for a string (or a regular expression) and returns the position of the match:
match()	returns an array containing the results of matching a string against a string
matchAll()	returns an iterator containing the results of matching a string against a string
includes()	returns true if a string contains a specified value
startsWith()	returns `true` if a string begins with a specified value.
endsWith()	returns `true` if a string ends with a specified value.

Regular Expression

Regular expressions can be used to perform all types of **text search** and **text replace** operations.

Syntax

```
/pattern/modifiers;
```

regular expressions are often used with the two **string methods**: `search()` and `replace()`.

Using String search() With a String

The `search()` method returns the position of the match

```
let text = "Visit W3Schools!";
let n = text.search("W3Schools"); result = 6
```

Using String search() With a Regular Expression

Use a regular expression to do a case-insensitive search for "w3schools" in a string:

```
let text = "Visit W3Schools";
let n = text.search(/w3schools/i); result= 6
```

Use String replace() With a Regular Expression

`replace(replace word ,new word)` method replaces a specified value with another value in a string .

```
let text = "Visit Microsoft!";
let result = text.replace(/microsoft/i, "W3Schools");
```

result:

```
Visit W3Schools!
```

Regular Expression Modifiers

Modifiers can be used to perform case-insensitive more global searches:

Modifier	**Description**
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than stopping after the first match)
m	Perform multiline matching

Regular Expression Patterns

Brackets are used to find a range of characters:

Expression	**Description**
[abc]	Find any of the characters between the brackets
[0-9]	Find any of the digits between the brackets
(x|y)	Find any of the alternatives separated with |

Metacharacters are characters with a special meaning:

Metacharacter	Description
\d	Find a digit
\s	Find a whitespace character
\b	Find a match at the beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b
\uxxxx	Find the Unicode character specified by the hexadecimal number xxxx

Quantifiers define quantities:

Quantifier	**Description**
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*
n?	Matches any string that contains zero or one occurrences of *n*

Number and number(method)

NaN - Not a Number = NaN is a number: typeof NaN returns number
If you use NaN in a mathematical operation, the result will also be NaN:
You can use the global JavaScript function isNaN() to find out if a value is a not a number:

```
let x = 100 / "Apple";
isNaN(x);
```

`BigInt`
Variable are used to store big integer values that are too big to be represented by a normal javascript number
In JavaScript, all numbers are stored in a 64-bit floating-point format
To create a `BigInt`, append n to the end of an integer or call `BigInt()`:
The JavaScript `typeof` a `BigInt` is "bigint":

JavaScript Number Methods

These **number methods** can be used on all JavaScript numbers:

toString()	Returns a number as a string = `x.toString();`
toExponential()	Returns a number written in exponential notation `let x = 9.656; x.toExponential(2);`
toFixed()	Returns a number written with a number of decimals `let x = 9.656; x.toFixed(0);`
toPrecision()	Returns a number written with a specified length x.toprecision(2)
ValueOf()	Returns a number as a number

Number Object Methods

These **object methods** belong to the **Number** object:

Number.isInteger()	Returns true if the argument is an integer
Number.isSafeInteger()	Returns true if the argument is a safe integer
Number.parseFloat()	Converts a string to a number
Number.parseInt()	Converts a string to a whole number

Number Properties

Property	Description
EPSILON	The difference between 1 and the smallest number > 1.
MAX_VALUE	The largest number possible in JavaScript
MIN_VALUE	The smallest number possible in JavaScript
MAX_SAFE_INTEGER	The maximum safe integer (2^{53} - 1)
MIN_SAFE_INTEGER	The minimum safe integer -(2^{53} - 1)
POSITIVE_INFINITY	Infinity (returned on overflow)
NEGATIVE_INFINITY	Negative infinity (returned on overflow)
NaN	A "Not-a-Number" value

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 (-)

Arrays

An array is a special variable, which can hold more than one value:

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

Syntax:

```
const array_name = [item1, item2, ...];
const array_name  = new Array(item1, item2, ...);
```

```
Create an array with one element: const points = [40];  = Correct
Create an array with one elements: const points = new Array(40);  error = undefined
```

Arrays are Objects

Arrays are a special type of objects. The `typeof` operator in JavaScript returns "object" for arrays.

The length Property

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

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let length = fruits.length;
```

Accessing the Last Array Element

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[fruits.length - 1];
```

Looping Array Elements

```
<script> const fruits = ["Banana", "Orange", "Apple", "Mango"];
        let fLen = fruits.length;
        let text = " ";
        for (let i = 0; i < fLen; i++) {
        text =text  + fruits[i] + "<br>"; }
        document.getElementById("demo").innerHTML = text;   </script>
```

Difference Between Arrays and Objects

In JavaScript, **arrays** use **numbered indexes**.
In JavaScript, **objects** use **named indexes**.
Arrays are a special kind of objects, with numbered index

When to Use Arrays. When to use Objects.

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**.

How to Recognize an Array

A common question is: How do I know if a variable is an array?
The problem is that the JavaScript operator `typeof` returns "`object`":

```
const fruits = ["Banana", "Orange", "Apple"];
let type = typeof fruits;
```

The typeof operator returns object because a JavaScript array is an object.
To solve this problem ECMAScript 5 (JavaScript 2009) defined a new method `Array.isArray()`:

```
Array.isArray(fruits);
```

Array Method

`toString()` : Converting Arrays to Strings

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

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

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

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

Result: Banana * Orange * Apple * Mango

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

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

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

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

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

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

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

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

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

```
const myGirls = ["Cecilie", "Lone"];
const myBoys = ["Emil", "Tobias", "Linus"];
const myChildren = myGirls.concat(myBoys);
```

`concat()` method does not change the existing arrays. It always returns a new array.

The `concat()` method can take any number of array arguments:

Example (Merging Three Arrays)

```
const arr1 = ["Cecilie", "Lone"];
const arr2 = ["Emil", "Tobias", "Linus"];
const arr3 = ["Robin", "Morgan"];
const myChildren = arr1.concat(arr2, arr3);
```

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

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi"); result = Banana,Orange,Lemon,Kiwi,Apple,Mango
```

The first parameter (2) defines the position **where** new elements should be **added** (spliced in).
The second parameter (0) defines **how many** elements should be **removed**.
With clever parameter setting, you can use `splice()` to remove elements without leaving "holes" in the array:

`slice()` : method slices out a piece of an array into a new array.
This example slices out a part of an array starting from array element 1 ("Orange"):

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(2);result : Lemon,Apple,Mango
```

Sorting Array string

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

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

Reversing an Array

The `reverse()` method reverses the elements in an array.
You can use it to sort an array in descending order:

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

Numeric Sort array

By default, the `sort()` function sorts values as **strings**.
Because of this, the `sort()` method will produce incorrect result when sorting numbers.
You can fix this by providing a **compare function (**`function(a, b){return a - b}`**)**:

Sorting in ascending order -

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

Sorting in descending order -

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

Array Spread (...)

The ... operator expands an iterable (like an array) into more elements:

```
const q1 = ["Jan", "Feb", "Mar"];
const q2 = ["Apr", "May", "Jun"];
const q3 = ["Jul", "Aug", "Sep"];
const q4 = ["Oct", "Nov", "May"];

const year = [...q1, ...q2, ...q3, ...q4];
result = Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,May
```

Arrays are Not Constants

The keyword `const` is a little misleading.
It does NOT define a constant array. It defines a constant reference to an array.
Because of this, we can still change the elements of a constant array.

Object

Objects are variables too. But objects can contain many values.
The values are written as **name:value** pairs (name and value separated by a colon).
It is a common practice to declare objects with the `const` keyword.

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

Creating a JavaScript Object

Create a single object, using an object literal.{}
Create a single object, with the keyword `new object()`.
Define an object constructor, and then create objects of the constructed type.
Create an object using `Object.create()`.

In JavaScript, almost "everything" is an object.

- Booleans can be objects (if defined with the `new` keyword)
- Numbers can be objects (if defined with the `new` keyword)
- Strings can be objects (if defined with the `new` keyword)
- Dates are always objects
- Maths are always objects
- Regular expressions are always objects
- Arrays are always objects
- Functions are always objects
- Objects are always objects

All JavaScript values, except primitives, are objects.

JavaScript Properties

Properties are the values associated with a JavaScript object.
Properties can usually be changed, added, and deleted, but some are read only.

Accessing JavaScript Properties

The syntax for accessing the property of an object is:

```
objectName.property       // person.age
```

or

```
objectName["property"]    // person["age"]
```

or

```
objectName[expression]    // x = "age"; person[x]
```

JavaScript for...in Loop

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

Syntax

```
for (let variable in object) {
  // code to be executed
}
```

The block of code inside of the `for...in` loop will be executed once for each property.

```
<script> const person = {fname:"John",lname:"Doe",age:25};
let txt = "";
for (let x in person) {
txt =txt+ person[x] + "<br>";}
document.getElementById("demo").innerHTML = txt; </script>
```

Adding New Properties

You can add new properties to an existing object by simply giving it a value.
Assume that the person object already exists - you can then give it new properties:

```
person.nationality = "English";
```

Deleting Properties

The `delete` keyword deletes a property from an object:

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

The `delete` keyword deletes both the value of the property and the property itself.
After deletion, the property cannot be used before it is added back again.

Nested Objects

Values in an object can be another object:

```
myObj = {name:"John",age:30, cars: {car1:"Ford",car2:"BMW",car3:"Fiat"} }
```

You can access nested objects using the dot notation or the bracket notation:

```
myObj.cars.car2;
```

Object Methods

JavaScript methods are actions that can be performed on objects.

A JavaScript **method** is a property containing a **function definition**.

Methods are functions stored as object properties.

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

If you access the fullName **property**, without (), it will return the **function definition**:

```
person.fullname
```

JSON.stringify

Convert object in json formate

```
let myString = JSON.stringify(person);
```

Object Accessors (Getters and Setters)

Getters and setters allow you to define Object Accessors (Computed Properties).

Getter (The get Keyword)

This example uses a `lang` property to `get` the value of the `language` property.

```
// Create an object:
const person = {
  firstName: "John",
  lastName: "Doe",
  language: "en",
  get lang() {
    return this.language;
  }
};

// Display data from the object using a getter:
document.getElementById("demo").innerHTML = person.lang;
```

JavaScript Setter (The set Keyword)

This example uses a `lang` property to `set` the value of the `language` property.

```
const person = {
  firstName: "John",
  lastName: "Doe",
  language: "",
  set lang(lang) {
    this.language = lang;
  }
};

// Set an object property using a setter:
person.lang = "en";

// Display data from the object:
document.getElementById("demo").innerHTML = person.language;
```

JavaScript Function or Getter?

What is the differences between these two examples?

```
EXAMPLE 1
const person = {
  firstName: "John",
  lastName: "Doe",
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
};

// Display data from the object using a method:
document.getElementById("demo").innerHTML = person.fullName();
```

Example 2

```
const person = {
  firstName: "John",
  lastName: "Doe",
  get fullName() {
    return this.firstName + " " + this.lastName;
  }
};
// Display data from the object using a getter:
document.getElementById("demo").innerHTML = person.fullName;
```

Example 1 access fullName as a function: person.fullName().
Example 2 access fullName as a property: person.fullName.

Object Constructors

Sometimes we need a "**blueprint**" for creating many objects of the same "type".
The way to create an "object type", is to use an **object constructor function**.
It is considered good practice to name constructor functions with an upper-case first letter.
Objects of the same type are created by calling the constructor function with the `new` keyword:

```
<!DOCTYPE html>
<html>
<body><h2>JavaScript Object Constructors</h2>

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

<script>

// Constructor function for Person objects
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
}

// Create two Person objects
const myFather = new Person("John", "Doe", 50, "blue");
const myMother = new Person("Sally", "Rally", 48, "green");

// Display age
document.getElementById("demo").innerHTML =
"My father is " + myFather.age + ". My mother is " + myMother.age + ".";

</script>

</body>
</html>
```

JavaScript Object Constructors

My father is 50. My mother is 48.

We can **not** add a new property to existing object constructor: `Person.nationality = "English";`

Prototype Inheritance

All JavaScript objects inherit properties and methods from a prototype:

- `Date` objects inherit from `Date.prototype`
- `Array` objects inherit from `Array.prototype`
- `Person` objects inherit from `Person.prototype`

The `Object.prototype` is on the top of the prototype inheritance chain:
`Date` objects, `Array` objects, and `Person` objects inherit from `Object.prototype`.

Adding Properties and Methods to Objects

Sometimes you want to add new properties (or methods) to all existing objects of a given type.
Sometimes you want to add new properties (or methods) to an object constructor.

Using the prototype Property

The JavaScript `prototype` property allows you to add new properties to object constructors:

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
}

Person.prototype.nationality = "English";
```

The JavaScript `prototype` property also allows you to add new methods to objects constructors:

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
}

Person.prototype.name = function() {
  return this.firstName + " " + this.lastName;
};
```

Sets (object)

A JavaScript Set is a collection of unique values.
Each value can only occur once in a Set.

You can create a JavaScript Set by:

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

Essential Set Methods

Method	Description
new Set()	Creates a new Set
add()	Adds a new element to the 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
Property	Description
size	Returns the number of elements in a Set

Pass an Array to the `new Set()` constructor:

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

```
<script>
// Create a Set
const letters = new Set(["a","b","c"]);
// Display set.size
document.getElementById("demo").innerHTML = letters.size;
</script>
```

Result : 3

Create a Set and add values:

```
// Create a Set
const letters = new Set();

// Add Values to the Set
letters.add("a");
letters.add("b");
letters.add("c");
```

Create a Set and add variables:

```
<script>
// Create a Set
const letters = new Set();
// Create Variables
const a = "a";
const b = "b";
const c = "c";
// Add the Variables to the Set
letters.add(a);
letters.add(b);
letters.add(c);
// Display set.size
document.getElementById("demo").innerHTML = letters.size;
</script>
```

If you want to Iterate through a set . first run value method

values() Method

The `values()` method returns a new iterator object containing all the values in a Set:
Set.values() returns a Set Iterator:

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

// Display set.size
document.getElementById("demo").innerHTML = letters.values();
</script>
```

Result : [object Set Iterator]

Now you can use the Iterator object to access the elements:

```
// List all Elements
let text = "";
for (const x of letters.values()) {
  text += x;
}
```

Maps (object)

A Map holds key-value pairs where the keys can be any datatype.
A Map remembers the original insertion order of the keys.

Map Method

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
forEach()	Calls a function for each key/value pair in a Map
entries()	Returns an iterator with the [key, value] pairs in a Map
Property	Description
size	Returns the number of elements in a Map

You can create a JavaScript Map by:

Passing an Array to `new Map()`
Create a Map and use `Map.set()`

Passing an Array to `new Map()` and get method

```
<script>
// Create a Map
const fruits = new Map([
  ["apples", 500],
  ["bananas", 300],
  ["oranges", 200]
]);
document.getElementById("demo").innerHTML = fruits.get("apples");
</script>
```

Result : 500

Create a Map and use `Map.set()` and get method

```
<script>
// Create a Map
const fruits = new Map();
// Set Map Values
fruits.set("apples", 500);
fruits.set("bananas", 300);
fruits.set("oranges", 200);
document.getElementById("demo").innerHTML = fruits.get("apples");
</script>
```

Result : 500

Delete () , size , get , set

```
<script>
// Create a Map
const fruits = new Map();

// Set Map Values
fruits.set("apples", 500);
fruits.set("bananas", 300);
fruits.set("oranges", 200);
fruits.delete("apples")
document.getElementById("demo").innerHTML =
fruits.get("apples");
document.getElementById("demoo").innerHTML = fruits.size;
</script>
```

Result:
undefined
2

The entries() Method

The `entries()` method returns an iterator object with the [key, values] in a Map

```
<script>
// Create a Map
const fruits = new Map([
  ["apples", 500],
  ["bananas", 300],
  ["oranges", 200]
]);

let text = "";
for (const x of fruits.entries()) {
  text += x + "<br>";
}

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

Result:
apples,500
bananas,300
oranges,200

Differences between Objects and Maps:

	Object	Map
Iterable	Not directly iterable	Directly iterable
Size	Do not have a size property	Have a size property
Key Types	Keys must be Strings (or Symbols)	Keys can be any datatype
Key Order	Keys are not well ordered	Keys are ordered by insertion
Defaults	Have default keys	Do not have default keys

JavaScript typeof

5 data types that can contain values:	There are 6 types of objects:	2 data types that cannot contain values:
string number boolean object function	Object Date Array String Number Boolean	null undefined

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

```
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"
```

Please observe:
The data type of NaN is number
The data type of an array is object
The data type of a date is object
The data type of null is object
The data type of an undefined variable is **undefined** *
The data type of a variable that has not been assigned a value is also **undefined** *

`undefined` and `null` are equal in value but different in type:

```
typeof undefined              // undefined
typeof null                   // object
null === undefined            // false
null == undefined             // true
```

Type Conversion

Converting Strings to Numbers = number() , parseInt() , parseFloat()
Converting Numbers to Strings = string(x) , tostring()
Converting Dates to Numbers = number()
Converting Numbers to Dates
Converting Booleans to Numbers = number()
Converting Numbers to Booleans

Date Objects

By default, JavaScript will use the browser's time zone and display a date as a full text string:
Sun May 07 2023 14:00:46 GMT+0530 (India Standard Time)

There are 9 ways to create a new 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(year,month,day,hours,minutes)
new Date(year,month,day,hours,minutes,seconds)
new Date(year,month,day,hours,minutes,seconds,ms)
new Date(milliseconds)
```

`new Date()` creates a date object with the **current date and time**:
`const d = new Date(); result =` Sun May 07 2023 14:10:44 GMT+0530 (India Standard Time)

`new Date(date string)` creates a date object from a **date string**:
`const d = new Date("October 13, 2014 11:13:00");`
`result =` Mon Oct 13 2014 11:13:00 GMT+0530 (India Standard Time)
`const d = new Date("2022-03-25");` Fri Mar 25 2022 05:30:00 GMT+0530 (India Standard Time)

JavaScript counts months from **0** to **11**:

January = 0.

December = 11.

Javascript count Sunday = 0 to Saturday =7

`const d = new Date(2018, 15, 24, 10, 33, 30);` Wed Apr 24 2019 10:33:30 GMT+0530 (India Standard Time)

If you supply only one parameter it will be treated as milliseconds.
`const d = new Date(2018);`

Date Formats

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"

The ISO format follows a strict standard in JavaScript.
The other formats are not so well defined and might be browser specific.

Parsing Dates

If you have a valid date string, you can use the `Date.parse()` method to convert it to milliseconds.

```
let msec = Date.parse("March 21, 2012");
```

Get Date Methods

new Date() Constructor

In JavaScript, date objects are created with `new Date()`.
`new Date()` returns a date object with the current date and time.
Get the Current Time

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

Date Get Methods

Method	Description
getFullYear()	Get **year** as a four digit number (yyyy)
getMonth()	Get **month** as a number (0-11)
getDate()	Get **day** as a number (1-31)
getDay()	Get weekday as a number (0-6)
getHours()	Get **hour** (0-23)
getMinutes()	Get **minute** (0-59)
getSeconds()	Get **second** (0-59)
getMilliseconds()	Get **millisecond** (0-999)
getTime()	Get time (milliseconds since January 1, 1970)

The get methods above return Local time.

The `getFullYear()` method returns the year of a date as a four digit number:

```
const d = new Date("2021-03-25");
d.getFullYear();  result = 2021

const d = new Date();
d.getFullYear(); result =2023
```

```
<script>
const months =
["January","February","March","April","May","June","July","August","September","October","November","Dece
mber"];

const d = new Date("2021-03-25");
let month = months[d.getMonth()];
document.getElementById("demo").innerHTML = month;
</script>
```
Result = March

```
<script>
const months =
["January","February","March","April","May","June","July","August","September","October","November","Dece
mber"];
const d = new Date();
let month = months[d.getMonth()];
document.getElementById("demo").innerHTML = month;
</script>
```
Result : May

```
   <script>
    let  a = new Date()
    document.getElementById("one").innerHTML= a;
    let date =a.getDate()
    let month =a.getMonth()
    let day=a.getDay()
    let hour=a.getHours()
    let minute=a.getMinutes()
    let day_name=["sun","mon","tues","wed","thus","frid","sat"]
    let month_name=["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"]
    let today_day_name=day_name[day]
    let today_month_name=month_name[month]
    document.getElementById("two").innerHTML= "today is " +  day +" " + today_day_name + " "
+today_month_name +date;

   </script>
```
Result :
Sun May 07 2023 14:38:57 GMT+0530 (India Standard Time)
today is 0 sun May7

Set Date Methods

Set Date methods let you set date values (years, months, days, hours, minutes, seconds, milliseconds) for a Date Object.

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)

The setFullYear() Method

The `setFullYear()` method sets the year of a date object. In this example to 2020:

```
const d = new Date();
d.setFullYear(2020);
```

The `setFullYear()` method can **optionally** set month and day:

```
const d = new Date();
d.setFullYear(2020, 11, 3);
```

The setMonth() Method

The `setMonth()` method sets the month of a date object (0-11):

```
const d = new Date();
d.setMonth(11);
```

`const d = new Date();`
`d.setDate(15);`
	`minutes (0-59)`
	`const d = new Date();`
	`d.setMinutes(30);`

`hour(0-23)`
`const d = new Date();`
`d.setHours(22);`
	`second(0-59)`
	`const d = new Date();`
	`d.setSeconds(30);`

Math Object

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

Math Properties (Constants)

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

JavaScript provides 8 mathematical constants that can be accessed as Math properties:

```
Math.E        // returns Euler's number
Math.PI       // returns PI
Math.SQRT2    // returns the square root of 2
Math.SQRT1_2  // returns the square root of 1/2
Math.LN2      // returns the natural logarithm of 2
Math.LN10     // returns the natural logarithm of 10
Math.LOG2E    // returns base 2 logarithm of E
Math.LOG10E   // returns base 10 logarithm of E
```

Math Methods

The syntax for Math any methods is : `Math.method(number)`

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.min()` and `Math.max()` can be used to find the lowest or highest value in a list of arguments:

```
Math.min(0, 150, 30, 20, -8, -200); result = -200
Math.max(0, 150, 30, 20, -8, -200); result = 150
```

Method	Description
abs(x)	Returns the absolute value of x
acos(x)	Returns the arccosine of x, in radians
acosh(x)	Returns the hyperbolic arccosine of x
asin(x)	Returns the arcsine of x, in radians
asinh(x)	Returns the hyperbolic arcsine of x
atan(x)	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radia
atan2(y, x)	Returns the arctangent of the quotient of its arguments
atanh(x)	Returns the hyperbolic arctangent of x
cbrt(x)	Returns the cubic root of x
ceil(x)	Returns x, rounded upwards to the nearest integer
cos(x)	Returns the cosine of x (x is in radians)
cosh(x)	Returns the hyperbolic cosine of x
exp(x)	Returns the value of E^x
floor(x)	Returns x, rounded downwards to the nearest integer

log(x)	Returns the natural logarithm (base E) of x
max(x, y, z, ..., n)	Returns the number with the highest value
min(x, y, z, ..., n)	Returns the number with the lowest value
pow(x, y)	Returns the value of x to the power of y
random()	Returns a random number between 0 and 1
round(x)	Rounds x to the nearest integer
sign(x)	Returns if x is negative, null or positive (-1, 0, 1)
sin(x)	Returns the sine of x (x is in radians)
sinh(x)	Returns the hyperbolic sine of x
sqrt(x)	Returns the square root of x
tan(x)	Returns the tangent of an angle
tanh(x)	Returns the hyperbolic tangent of a number
trunc(x)	Returns the integer part of a number (x)

Booleans

A JavaScript Boolean represents one of two values: **true** or **false**.

Boolean Values

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
For this, JavaScript has a **Boolean** data type. It can only take the values **true** or **false**.

The Boolean() Function

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

```
Boolean(10 > 9)
```

The Boolean value of **-0** (minus zero) is **false**:
`let x = -0;`
`Boolean(x);`
	The Boolean value of **null** is **false**:
	`let x = null;`
	`Boolean(x);`

The Boolean value of **""** (empty string) is **false**:
`let x = "";`
`Boolean(x);`
	The Boolean value of **false** is (you guessed it) **false**:
	`let x = false;`
	`Boolean(x);`

The Boolean value of **undefined** is **false**:
`let x;`
`Boolean(x);`
	The Boolean value of **NaN** is **false**:
	`let x = 10 / "Hallo";`
	`Boolean(x);`

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

If the variable age is a value below 18, the value of the variable voteable will be "Too young", otherwise the value of voteable will be "Old enough".

Conditional Statements if ,else , else if

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

- 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

if statement

Syntax

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

else Statement

```
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
}
```

else if Statement

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

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
}
```

This is how it works:

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

```
<script>
let 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>
```

Result: Today is Monday

Loop

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

JavaScript supports different kinds of loops:

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

For Loop

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

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

Expression 1 is executed (one time) before the execution of the code block.
Expression 2 defines the condition for executing the code block.
Expression 3 is executed (every time) after the code block has been executed.

```
<script>
    var text="";
    for(var i=0;i<=5;i++){
      text=text+"number : "+i +"<br>"
      document.getElementById("a").innerHTML=text
    }

</script>
```

```
Result:
0
1
2
3
4
5
```

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

```
Result:

4

BMW
Volvo
Saab
Ford
```

While Loop

Loops can execute a block of code as long as a specified condition is true.

Syntax

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

`while (i < 2) {`
`  text += "The number is " + i;`
`  i++;`
`}`
	Result:
	The number is 0
	The number is 1

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);
```

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

`let i = 2;`
`do {`
`  text += "The number is " + i;`
`  i++;`
`}`
`while (i < 1);`
	`"The number is " + 1;`

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
}
```

`const person = {fname:"John", lname:"Doe", age:25};`
`let text = "";`
`for (let x in person) {`
`  text += person[x];`
`}`
	Example Explained
	The 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]

For In Over Arrays

The JavaScript `for in` statement can also loop over the properties of an Array:

Syntax

```
for (variable in array) {
  code
}
```

```
const numbers = [45, 4, 9, 16, 25];
let txt = "";
for (let x in numbers) {
  txt += numbers[x];
}
```

Do not use for in over an Array if the index order is important.

The index order is implementation-dependent, and array values may not be accessed in the order you expect.

It is better to use a for loop, a for of loop, or Array.forEach() when the order is important.

Array.forEach()

The `forEach()` method calls a function (a callback function) once for each array element.

```
const numbers = [45, 4, 9, 16, 25];

let txt = "";
numbers.forEach(myFunction);

function myFunction(value, index, array)
{
  txt += value;
}
```

Note that the function takes 3 arguments:
The item value
The item index
The array itself

The example above uses only the value parameter. It can be rewritten to:

```
const numbers = [45, 4, 9, 16, 25];
let txt = "";
numbers.forEach(myFunction);

function myFunction(value) {
  txt += value;
}
```

For Of Loop

`for of` loops through the values of an iterable object.

iterable - An object that has iterable properties.

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

Syntax

```
for (variable of iterable) {
  // code block to be executed
}
```

Looping over an Array

```
const cars = ["BMW", "Volvo", "Mini"];
let text = "";
for (let x of cars) {
  text += x;
}
```

Looping over a String

```
let language = "JavaScript";

let text = "";
for (let x of language) {
text += x;
}
```

Break and Continue

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

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

Break Statement

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

In the example above, the `break` statement ends the loop ("breaks" the loop) when the loop counter (i) is 3.

The Continue Statement

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

JavaScript Labels

Label is a identifier . is used in brake and continue statement .

```
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;
```

The `continue` statement (with or without a label reference) can only be used to **skip one loop iteration**.

The `break` statement, without a label reference, can only be used to **jump out of a loop or a switch**.With a label reference, the break statement can be used to **jump out of any code block**:

```
const cars = ["BMW", "Volvo", "Saab", "Ford"];
list: {
  text += cars[0] + "<br>";
  text += cars[1] + "<br>";
  break list;
  text += cars[2] + "<br>";
  text += cars[3] + "<br>";
}
```

example :

```
 <script>
  row: for (let i = 1; i <= 5; i++){
  col : for (let j = 1; j <= i; j++) {
   document.write(j)  }
  document.write("<br>");
  }
 </script>
```

Output:

1
12
123
1234
12345

```
 <script>
  row: for (let i = 1; i <= 5; i++){
  col : for (let j = 1; j <= i; j++) {
   if (i==3) {
     break row;   }
   document.write(j)}
  document.write("<br>");
  }
 </script>
```

Output:

1
12

```
<script>
  row: for (let i = 1; i <= 5; i++){
  col : for (let j = 1; j <= i; j++) {
    if (i==3) {
      continue row;
    }
    document.write(j)
  }
  document.write("<br>");
  }
</script>
```

Output:
1
12
1234
12345

this keyword

In JavaScript, the `this` keyword refers to an object.
The `this` keyword refers to different objects depending on how it is used:

In an object method, `this` refers to the **object**.
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 received the event.
Methods like `call()`, `apply()`, and `bind()` can refer `this` to **any object**.

this in a Method

When used in an object method, `this` refers to the object.
In the example on top of this page, `this` refers to the person object.
Because the fullName method is a method of the person object.

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

this in a Function (Default)

In a function, the global object is the default binding for `this`.
In a browser window the global object is `[object Window]`:

```
function myFunction() {
  return this;
}
```

Function Definitions

JavaScript functions are **defined** with the `function` keyword.
Functions are Objects , typeof() function is object

Syntax :	Example:

`function functionName(parameters) {`
`  // code to be executed`
`}`
	`function myFunction(a, b) {`
	`  return a * b;`
	`}`

A function expression can be stored in a variable:

```
const x = function (a, b) {return a * b};
```

After a function expression has been stored in a variable, the variable can be used as a function:

```
const x = function (a, b) {return a * b};
let z = x(4, 3);
```

The function above is actually an **anonymous function** (a function without a name).
Functions stored in variables do not need function names. They are always invoked (called) using the variable name.

`arguments.length` property returns the number of arguments received when the function was invoked:

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

Arrow Functions

Arrow functions allows a short syntax for writing function expressions.
You don't need the `function` keyword, the `return` keyword, and the **curly brackets**.

```
// ES5
var x = function(x, y) {
  return x * y;
}

// ES6
const x = (x, y) => x * y;
```

Function Parameters and Arguments

Earlier in this tutorial, you learned that functions can have **parameters**:

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

Function **parameters** are the **names** listed in the function definition.

Function **arguments** are the real **values** passed to (and received by) the function.

Default Parameter Values

If y is not passed or undefined, then y = 10.

```
function myFunction(x, y = 10) {
  return x + y;
}
myFunction(5);
```

Function Rest Parameter

The rest parameter (...) allows a function to treat an indefinite number of arguments as an array:

```
function sum(...args) {
  let sum = 0;
  for (let arg of args) sum += arg;
  return sum;
}

let x = sum(4, 9, 16, 25, 29, 100, 66, 77);
```

call() Method

The `call()` method is a predefined JavaScript method.
With `call()`, an object can use a method belonging to another object.

```
<script>
const person = {
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
}
const person1 = {
  firstName:"John",
  lastName: "Doe"
}
const person2 = {
  firstName:"Mary",
  lastName: "Doe"
}
document.getElementById("demo").innerHTML = person.fullName.call(person1);
</script>
```

Result :
John Doe

call() Method with Arguments

```
<script>
const person = {
  fullName: function(city, country) {
    return this.firstName + " " + this.lastName + "," + city + "," + country;
  }
}

const person1 = {
  firstName:"John",
  lastName: "Doe"
}

const person2 = {
  firstName:"Mary",
  lastName: "Doe"
}
document.getElementById("demo").innerHTML = person.fullName.call(person1, "Oslo", "Norway");
</script>
```

Result :
John Doe,Oslo,Norway

apply() Method

The `apply()` method is similar to the `call()` method (previous chapter).
In this example the **fullName** method of **person** is applied on **person1**:

```
const person = {
  fullName: function() {
    return this.firstName + " " + this.lastName;
  } }
const person1 = {
  firstName: "Mary",
  lastName: "Doe"}
// This will return "Mary Doe":
person.fullName.apply(person1);
```

The Difference Between call() and apply()

The difference is:
The `call()` method takes arguments **separately**.
The `apply()` method takes arguments as an **array**.
The apply() method is very handy if you want to use an array instead of an argument list.

```
<script>
const person = {
  fullName: function(city, country) {
    return this.firstName + " " + this.lastName + "," + city + "," + country;
  }
}

const person1 = {
  firstName:"John",
  lastName: "Doe"
}

document.getElementById("demo").innerHTML = person.fullName.apply(person1, ["Oslo",
"Norway"]);
</script>
```

Result :
John Doe,Oslo,Norway

bind method (Function Borrowing)

With the `bind()` method, an object can borrow a method from another object.

The example below creates 2 objects (person and member).
The member object borrows the fullname method from the person object:

```
<script>
const person = {  firstName:"John", lastName: "Doe",
fullName: function() {return this.firstName + " " + this.lastName;  }}
const member = { firstName:"Hege", lastName: "Nilsen", }
let fullName = person.fullName.bind(member);
document.getElementById("demo").innerHTML = fullName();
</script>
```

Result: Hege Nilsen

Preserving this

Sometimes the `bind()` method has to be used to prevent losing this.
In the following example, the person object has a display method. In the display method, this refers to the person object:

```
<script>
const person = {
  firstName:"John",
  lastName: "Doe",
  display: function() {
    let x = document.getElementById("demo");
    x.innerHTML = this.firstName + " " + this.lastName;
  }
}

person.display();
</script>
```

Result : john doe

When a function is used as a callback, this is lost.
This example will try to display the person name after 3 seconds, but it will display undefined instead:

```
const person = {
  firstName:"John",
  lastName: "Doe",
  display: function () {
    let x = document.getElementById("demo");
    x.innerHTML = this.firstName + " " + this.lastName;
  }
}
setTimeout(person.display, 3000);
```

result : undefined undefined

The `bind()` method solves this problem.
In the following example, the `bind()` method is used to bind person.display to person.
This example will display the person name after 3 seconds:

```
const person = {
  firstName:"John",
  lastName: "Doe",
  display: function () {
    let x = document.getElementById("demo");
    x.innerHTML = this.firstName + " " + this.lastName;
  }
}

let display = person.display.bind(person);
setTimeout(display, 3000);
result : john doe
```

Closures() method

JavaScript variables can belong to the **local** or **global** scope.
Global variables can be made local (private) with **closures**.
Variables created **without** a declaration keyword (`var`, `let`, or `const`) are always global, even if they are created inside a function.

Hoisting

Hoisting in javascript is the default process behavior of moving declaration of all the variables and functions on top of the scope where scope can be either local or global.
Example 1:

```
hoistedFunction();  // " Hi There! " (is an output that is declared as function even after it is called)
function hoistedFunction(){
  console.log(" Hi There! ");
}
```

Example 2:

```
hoistedVariable = 5;
console.log(hoistedVariable); // outputs 5 though the variable is declared after it is initialized
var hoistedVariable;
```

Use Strict

- `"use strict";` Defines that JavaScript code should be executed in "strict mode".
- With strict mode, you can not, for example, use undeclared variables.

```
"use strict";
x = 3.14;
// This will cause an error because x is
not declared
```

```
"use strict";
myFunction();

function myFunction() {
  y = 3.14;   // This will also cause an
error because y is not declared
}
```

```
x = 3.14;       // This will not cause an
error.
myFunction();

function myFunction() {
  "use strict";
  y = 3.14;   // This will cause an error
}
```

Future Proof!

Keywords reserved for future JavaScript versions can NOT be used as variable names in strict mode.

Implements,interface,let,package,private,protected,public,static,yield

```
"use strict";
let public = 1500;      // This will cause an error
```

Classes

ECMAScript 2015, also known as ES6, introduced JavaScript Classes.

JavaScript Classes are templates for JavaScript Objects.

A JavaScript class is **not** an object.

It is a **template** for JavaScript objects.

Class Syntax

Use the keyword `class` to create a class.

Always add a method named `constructor()`:

```
class ClassName {
  constructor() { ... } }
```

Constructor Method

- It has to have the exact name "constructor"
- It is executed automatically when a new object is created
- It is used to initialize object properties
- If you do not define a constructor method, JavaScript will add an empty constructor method.
- The constructor method is called automatically when a new object is created.

```
<script>
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  } }
const myCar1 = new Car("Ford", 2014);
const myCar2 = new Car("Audi", 2019);
document.getElementById("demo").innerHTML =
myCar1.name + " " + myCar2.name;
</script>
```

Result:
Ford Audi

```
<script>
class Car {
  constructor(name, year) {
    this.nam = name;
    this.yea = year;  }
  age() { const date = new Date();
        return date.getFullYear() - this.yea;}
}
const myCar = new Car("Ford", 2014);
document.getElementById("demo").innerHTML =
"My car is " + myCar.age() + " years old.";
</script>
```

My car is 9 years old.

```
<script>
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
  age(x) {
    return x - this.year;
  }
}
const date = new Date();
let year = date.getFullYear();

const myCar = new Car("Ford", 2014);
document.getElementById("demo").innerHTML=
"My car is " + myCar.age(year) + " years old.";
</script>
```

Pass a parameter into the "age()" method.

My car is 9 years old.

Modules

JavaScript modules allow you to break up your code into separate files.
Modules only work with the HTTP(s) protocol.
A web-page opened via the file:// protocol cannot use import / export.

Modules are imported from external files with the `import` statement.

Modules also rely on `type="module"` in the <script> tag.

```
<script type="module">
import message from "./message.js";
</script>
```

Export

Modules with functions or variables can be stored in any external file.
There are two types of exports: Named Exports and Default Exports.

Named Exports

Let us create a file named `person.js`, and fill it with the things we want to export.
You can create named exports two ways. In-line individually, or all at once at the bottom.

In-line individually:

person.js

```
export const name = "Jesse";
export const age = 40;
```

All at once at the bottom:

person.js

```
const name = "Jesse";
const age = 40;
export {name, age};
```

Default Exports

Let us create another file, named `message.js`, and use it for demonstrating default export.
You can only have one default export in a file.

message.js

```
const message = () => {
const name = "Jesse";
const age = 40;
return name + ' is ' + age + 'years old.';
};
export default message;
```

Import

You can import modules into a file in two ways, based on if they are named exports or default exports.
Named exports are constructed using curly braces. Default exports are not.

Import from named exports

Import named exports from the file person.js:

```
import { name, age } from "./person.js";
```

Import from default exports

Import a default export from the file message.js:

```
import message from "./message.js";
```

```
<script type="module">
import { name, age } from "./person.js";

let text = "My name is " + name + ", I am
" + age + ".";
document.getElementById("demo").innerHTML
= text;
</script>
```
Result : My name is Jesse, I am 40.

```
<script type="module">
import message from "./message.js";

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

</script>
```
Result : My name is Jesse, I am 40.

Events

The change in the state of an object is known as an **Event**.

Event handling

When javascript code is included in HTML , js react over these events and allow the execution.

The process of reacting over the events is called **Event Handling**. Thus, js handles the HTML events via **Event Handlers**.

For example, when a user clicks over the browser, add js code, which will execute the task to be performed on the event.

Some of the HTML events and their event handlers are:

Mouse events:

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

Keyboard events:

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

Form events:

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

Window/Document events

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

HTML DOM

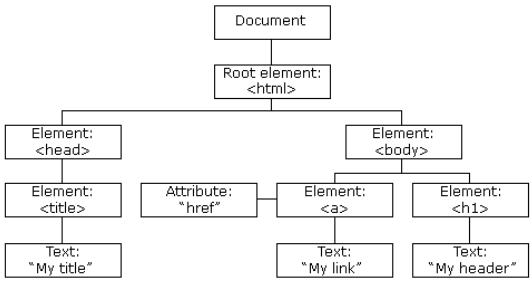
The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:

1. In the DOM, all HTML elements are defined as **objects**.
2. The **properties** of all HTML elements
3. The **methods** to access all HTML elements
4. 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.**
- With the HTML DOM, JavaScript can access and change ,remove and add the elements , attributes ,CSS Style ,events of an HTML document.

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.
The **HTML DOM** model is constructed as a tree of **Objects**:

The HTML DOM Tree of Objects



What is the DOM?

The DOM is a W3C (World Wide Web Consortium) standard.
The DOM defines a standard for accessing documents:
"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."
The W3C DOM standard is separated into 3 different parts:

1. Core DOM - standard model for all document types
2. XML DOM - standard model for XML documents
3. HTML DOM - standard model for HTML documents

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.

getElementById Method

the `getElementById` method used `id` to find the element.

innerHTML Property

`innerHTML` is a property ,is useful for getting or replacing the content of HTML elements.

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	Description
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

Adding and Deleting Elements

Method	Description
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	Description
document.getElementById(*id*).onclick = function(){*code*}	Adding event handler code to an onclick event

EXAMPLE OF FINDING HTML ELEMENT

getElementsById and element.innerHTML

```
<p id="intro">Finding HTML Elements by Id</p>
<p>This example demonstrates the <b>getElementsById</b> method.</p>
<p id="demo"></p>
<script>
        const element = document.getElementById("intro");
        document.getElementById("demo").innerHTML = "The text from the intro paragraph is: " +
element.innerHTML;
</script>
```

Result :

Finding HTML Elements by Id
This example demonstrates the **getElementsById** method.
The text from the intro paragraph is: Finding HTML Elements by Id

document.getElementsByTagName( ); AND element.innerHTML

```
<p>Finding HTML Elements by Tag Name.</p>
<p>This example demonstrates the <b>getElementsByTagName</b> method.</p>
<p id="demo"></p>
<script>
const element = document.getElementsByTagName("p");
document.getElementById("demo").innerHTML = 'The text in first paragraph (index 0) is: ' +
element[0].innerHTML;
</script>
```

RESULT :

Finding HTML Elements by Tag Name.
This example demonstrates the **getElementsByTagName** method.
The text in first paragraph (index 0) is: Finding HTML Elements by Tag Name.

This example finds the element with `id="main"`, and then finds all `<p>` elements inside `"main"`:

```
<div id="main">
  <p>Finding HTML Elements by Tag Name</p>
  <p>This example demonstrates the <b>getElementsByTagName</b> method.</p>
</div>
  <p id="demo"></p>
<script>
   const x = document.getElementById("main");
   const y = x.getElementsByTagName("p");
  document.getElementById("demo").innerHTML =
  'The first paragraph (index 0) inside "main" is: ' + y[0].innerHTML;
</script>
```

RESULT :
Finding HTML Elements by Tag Name
This example demonstrates the **getElementsByTagName** method.
The first paragraph (index 0) inside "main" is: Finding HTML Elements by Tag Name

Finding HTML Elements by Class Name = `getElementsByClassName()`

```
<p class="intro">Hello World!</p>
<p class="intro">This example demonstrates the <b>getElementsByClassName</b>
method.</p>
<p id="demo"></p>
<script>
 const x = document.getElementsByClassName("intro");
 document.getElementById("demo").innerHTML =
 'The first paragraph (index 0) with class="intro" is: ' + x[0].innerHTML;
</script>
```

RESULT:
Hello World!
This example demonstrates the getElementsByClassName method.
The first paragraph (index 0) with class="intro" is: Hello World!

Find HTML Elements by CSS Selectors `querySelectorAll()`

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.

```
<p>Finding HTML Elements by Query Selector</p>
<p class="intro">Hello World!.</p>
<p class="intro">This example demonstrates the <b>querySelectorAll</b> method.</p>
<p id="demo"></p>
<script>
  const x = document.querySelectorAll("p.intro");
  document.getElementById("demo").innerHTML =
'The first paragraph (index 0) with class="intro" is: ' + x[0].innerHTML;
</script>
Result:
Finding HTML Elements by Query Selector
Hello World!.
This example demonstrates the querySelectorAll method.
The first paragraph (index 0) with class="intro" is: Hello World!.
```

```
const x = document.querySelectorAll(".class_name");
const x = document.querySelectorAll("#id_name");
```

ALL Finding HTML Objects

Property	Description	DOM
document.anchors	Returns all <a> elements that have a name attribute	1
document.baseURI	Returns the absolute base URI of the document	3
document.body	Returns the <body> element	1
document.cookie	Returns the document's cookie	1
document.doctype	Returns the document's doctype	3
document.documentElement	Returns the <html> element	3
document.documentMode	Returns the mode used by the browser	3
document.documentURI	Returns the URI of the document	3
document.domain	Returns the domain name of the document server	1
document.embeds	Returns all <embed> elements	3
document.forms	Returns all <form> elements	1
document.head	Returns the <head> element	3
document.images	Returns all <img> elements	1
document.implementation	Returns the DOM implementation	3
document.inputEncoding	Returns the document's encoding (character set)	3
document.lastModified	Returns the date and time the document was updated	3
document.links	Returns all <area> and <a> elements that have a href attribute	1
document.readyState	Returns the (loading) status of the document	3
document.referrer	Returns the URI of the referrer (the linking document)	1
document.scripts	Returns all <script> elements	3
document.strictErrorChecking	Returns if error checking is enforced	3
document.title	Returns the <title> element	1
document.URL	Returns the complete URL of the document	1

Changing HTML Content

The easiest way to modify the content of an HTML element is by using the `innerHTML` property.
To change the content of an HTML element, use this syntax:
`document.getElementById(id).innerHTML = new HTML`

```
example= document.getElementById("p1").innerHTML = "New text!";
```

```
<script>
const element = document.getElementById("id01");
element.innerHTML = "New Heading";
</script>
```

Changing the Value of an Attribute

To change the value of an HTML attribute, use this syntax:
`document.getElementById(id).attribute = new value`

```
document.getElementById("myImage").src = "landscape.jpg";
```

Changing HTML Style

To change the style of an HTML element, use this syntax:
`document.getElementById(id).style.property = new style`

```
document.getElementById("p2").style.color = "blue";
```

```
<body> <h1 id="id1">My Heading 1</h1>
      <button type="button"
       onclick="document.getElementById('id1').style.color = 'red'">Click Me!</button>
</body>
```

Adding and Deleting Elements

`createElement()` method creates an element node.

Create a <p> element and append it to the document:

```
const para = document.createElement("p");
para.innerText = "This is a paragraph";
document.body.appendChild(para);
```

Create a <p> element and append it to an element:

```
const para = document.createElement("p");
para.innerHTML = "This is a paragraph.";
document.getElementById("myDIV").appendChild(para);
```

`removeChild()` method removes an element's child.

```
<p>Click "Remove" to remove the first item from the list:</p>
<button onclick="myFunction()">Remove</button>
<ul id="myList">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
<script>
function myFunction() {
  const list = document.getElementById("myList");
  list.removeChild(list.firstElementChild); }
</script>
```

HTML DOM EventListener

addEventListener() method

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

```
document.getElementById("myBtn").addEventListener("click", displayDate);
```

Syntax

***element*.addEventListener(*event, function, useCapture*);**

- The first parameter is the type of the event (like "`click`" or "`mousedown`")
- The second parameter is the function we want to call when the event occurs.
- The third parameter is a boolean value specifying whether to use event bubbling or event capturing. This parameter is optional.
- Note that you don't use the "on" prefix for the event; use "`click`" instead of "`onclick`".

1. You can add many event handlers to one element.
2. You can add many event handlers of the same type to one element, i.e two "click" events.
3. You can add event listeners to any DOM object not only HTML elements. i.e the window object.
4. The `addEventListener()` method makes it easier to control how the event reacts to bubbling.
5. You can easily remove an event listener by using the `removeEventListener()` method.

Add an Event Handler to an Element

Alert "Hello World!" when the user clicks on an element:

```
<button id="myBtn">Try it</button>
<script>
document.getElementById("myBtn").addEventListener("click", function() {
  alert("Hello World!");
});
</script>
```

You can also refer to an external "named" function:

```
element.addEventListener("click", myFunction);
function myFunction() { alert ("Hello World!");}
```

Add Many Event Handlers to the Same Element

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

<script>
var x = document.getElementById("myBtn");
x.addEventListener("click", myFunction);
x.addEventListener("click", someOtherFunction);

function myFunction() {
  alert ("Hello World!");
}

function someOtherFunction() {
  alert ("This function was also executed!");
}
</script>
```

You can add events of different types to the same element:

```
<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>
```

Passing Parameters

```
element.addEventListener("click", function(){ myFunction(p1, p2); });
```

Event Bubbling or Event Capturing?

- There are two ways of event propagation in the HTML DOM, bubbling and capturing.
- Event propagation is a way of defining the element order when an event occurs. If you have a <p> element inside a element, and the user clicks on the <p> element, which element's "click" event should be handled first?

In *bubbling* the inner most element's event is handled first and then the outer: the <p> element's click event is handled first, then the element's click event. Default (false)	In *capturing* the outer most element's event is handled first and then the inner: the element's click event will be handled first, then the <p> element's click event. true

With the addEventListener() method you can specify the propagation type by using the "useCapture" parameter:

```
addEventListener(event, function, useCapture);
```

The default value is false, which will use the bubbling propagation, when the value is set to true, the event uses the capturing propagation.

```
document.getElementById("myP").addEventListener("click", myFunction);
document.getElementById("myDiv").addEventListener("click", myFunction, true);
```

removeEventListener() method

The `removeEventListener()` method removes event handlers that have been attached with the addEventListener() method:

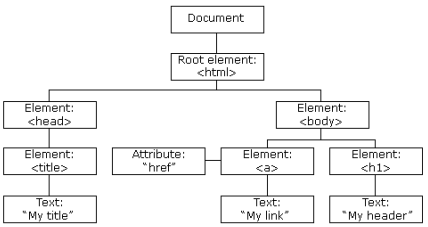
```
element.removeEventListener("mousemove", myFunction);
```

HTML DOM Navigation

With the HTML DOM, you can navigate the node tree using node relationships.

DOM Nodes

- everything in an HTML document is a node:
- The entire document is a document node
- Every HTML element is an element node
- The text inside HTML elements are text nodes
- Every HTML attribute is an attribute node (deprecated)
- All comments are comment nodes



- With the HTML DOM, all nodes in the node tree can be accessed by JavaScript.
- New nodes can be created, modified or deleted.

Node Relationships

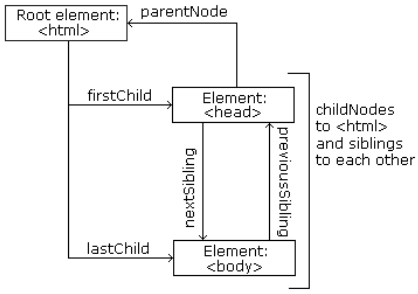
From the HTML above you can read:

`<html>` is the root node
`<html>` has no parents
`<html>` is the parent of `<head>` and `<body>`
`<head>` is the first child of `<html>`
`<body>` is the last child of `<html>`

and:

`<head>` has one child: `<title>`
`<title>` has one child (a text node): "DOM Tutorial"
`<body>` has two children: `<h1>` and `<p>`
`<h1>` has one child: "DOM Lesson one"
`<p>` has one child: "Hello world!"
`<h1>` and `<p>` are siblings

```
<html>
  <head>
    <title>DOM Tutorial</title>
  </head>
  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>
</html>
```



Navigating Between Nodes

You can use the following node properties to navigate between nodes with JavaScript:

- `parentNode`
- `childNodes[nodenumber]`
- `firstChild`
- `lastChild`
- `nextSibling`
- `previousSibling`

Child Nodes and Node Values

<body> <h1 id="id01">My First Page</h1> <script> document.getElementById("id02").innerHTML = document.getElementById("id01").innerHTML; </script> </body> Result : My First Page	<body> <h1 id="id01">My First Page</h1> <script> document.getElementById("id02").innerHTML = document.getElementById("id01").firstChild.nodeValue; </script> </body> Result : My First Page

```
<body>
<h1 id="id01">My First Page</h1>
<p id="id02"></p>
<script>
document.getElementById("id02").innerHTML =
document.getElementById("id01").childNodes[0].nodeValue;
</script>
</body>
```

Result : My First Page

nodeName Property

The `nodeName` property specifies the name of a node.

```
<h1 id="id01">My First Page</h1>
<p id="id02"></p>
<script>
document.getElementById("id02").innerHTML = document.getElementById("id01").nodeName;
</script>
```

Result : H1

Note: `nodeName` always contains the uppercase tag name of an HTML element.

nodeType Property

The `nodeType` property is read only. It returns the type of a node.

```
<h1 id="id01">My First Page</h1>
<p id="id02"></p>
<script>
document.getElementById("id02").innerHTML = document.getElementById("id01").nodeType;
</script>
RESULT : 1
```

The most important nodeType properties are:

Node	Type	Example
ELEMENT_NODE	1	<h1 class="heading">W3Schools</h1>
ATTRIBUTE_NODE	2	class = "heading" (deprecated)
TEXT_NODE	3	W3Schools
COMMENT_NODE	8	<!-- This is a comment -->
DOCUMENT_NODE	9	The HTML document itself (the parent of <html>)
DOCUMENT_TYPE_NODE	10	<!Doctype html>

Type 2 is deprecated in the HTML DOM (but works). It is not deprecated in the XML DOM.

Adding and Removing Nodes (HTML Elements)

Creating New HTML Elements (Nodes)

To add a new element to the HTML DOM, you must create the element (element node) first, and then append it to an existing element.

```
<body>

<h2>JavaScript HTML DOM</h2>
<p>Add a new HTML Element.</p>

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
const para = document.createElement("p");
const node = document.createTextNode("This is new.");
para.appendChild(node);
const element =
document.getElementById("div1");
element.appendChild(para);
</script>

</body>
```

RESULT :

JavaScript HTML DOM

Add a new HTML Element.

This is a paragraph.

This is another paragraph.

This is new.

Creating new HTML Elements - insertBefore()

The `appendChild()` method in the previous example, appended the new element as the last child of the parent.

If you don't want that you can use the `insertBefore()` method:

```
<body>

<h2>JavaScript HTML DOM</h2>
<p>Add a new HTML Element.</p>

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
const para = document.createElement("p");
const node = document.createTextNode("This is new.");
para.appendChild(node);

const element = document.getElementById("div1");
const child = document.getElementById("p1");
element.insertBefore(para,child);
</script>
```

RESULT:

JavaScript HTML DOM

Add a new HTML Element.

This is new.

This is a paragraph.

This is another paragraph.

Removing Existing HTML Elements

To remove an HTML element, use the `remove()` method:

```
<body>
<h2>JavaScript HTML DOM</h2>
<h3>Remove an HTML Element.</h3>
<div>
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>
<script>
document.getElementById("p1").remove();
</script> </body>
```

RESULT:
JavaScript HTML DOM
Remove an HTML Element.
This is another paragraph.

Replacing HTML Elements

To replace an element to the HTML DOM, use the `replaceChild()` method:

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2>This Is anotherparagraph.</p>
</div>

<script>
const para = document.createElement("p");
const node = document.createTextNode("This
is new.");
para.appendChild(node);

const parent =
document.getElementById("div1");
const child =
document.getElementById("p1");
parent.replaceChild(para, child);
</script>
```

This is new.

This is a paragraph.

HTML DOM Collections

The `getElementsByTagName()` method returns an `HTMLCollection` object.
An `HTMLCollection` object is an array-like list (collection) of HTML elements.

```
<body><h2>JavaScript HTML DOM</h2>
        <p>Hello World!</p>
        <p>Hello Norway!</p>
        <p id="demo"></p>
<script>
const myCollection =
document.getElementsByTagName("p");
document.getElementById("demo").innerHTML =
"The innerHTML of the second paragraph is: " +
myCollection[1].innerHTML ;
</script> </body>
```

Result :
JavaScript HTML DOM
Hello World!
Hello Norway!
The innerHTML of the second paragraph is:
Hello Norway!

Collection Length

The `length` property defines the number of elements in an `HTMLCollection`:

```
myCollection.length
```

The `length` property is useful when you want to loop through the elements in a collection:

Change the text color of all <p> elements:

```
const myCollection = document.getElementsByTagName("p");
for (let i = 0; i < myCollection.length; i++) {
  myCollection[i].style.color = "red";
}
```

An HTMLCollection is NOT an array!

An HTMLCollection may look like an array, but it is not.

---

jQuery?

jQuery is a lightweight, "write less, do more", JavaScript library.

The purpose of jQuery is to make it much easier to use JavaScript on your website.

The jQuery library contains the following features:

- HTML/DOM manipulation
- CSS manipulation
- HTML event methods
- Effects and animations
- AJAX
- Utilities

Why jQuery?

There are lots of other JavaScript libraries out there, but jQuery is probably the most popular, and also the most extendable.

Many of the biggest companies on the Web use jQuery, such as:

- Google
- Microsoft
- IBM
- Netflix

Adding jQuery to Your Web Pages

There are several ways to start using jQuery on your web site. You can:

- Download the jQuery library from jQuery.com
- Include jQuery from a CDN, like Google

Downloading jQuery

There are two versions of jQuery available for downloading:

- Production version - this is for your live website because it has been minified and compressed
- Development version - this is for testing and development (uncompressed and readable code)

Both versions can be downloaded from jQuery.com.

```
<head>
<script src="jquery-3.6.4.min.js"></script>
</head>
```

---

jQuery CDN

If you don't want to download and host jQuery yourself, you can include it from a CDN (Content Delivery Network).

Google is an example of someone who host jQuery:

```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></scr
ipt>
</head>
```

---

jQuery Syntax

Basic syntax is: **$(*selector*).*action*()**

- A $ sign to define/access jQuery
- A (*selector*) to "query (or find)" HTML elements
- A jQuery *action*() to be performed on the element(s)

Examples:

1. `$(this).hide()` - hides the current element.
2. `$("p").hide()` - hides all <p> elements.
3. `$(".test").hide()` - hides all elements with class="test".
4. `$("#test").hide()` - hides the element with id="test".

Document Ready Event

You might have noticed that all jQuery methods in our examples, are inside a document ready event:

```
$(document).ready(function(){
  // jQuery methods go here...
});
```

This is to prevent any jQuery code from running before the document is finished loading (is ready).
Here are some examples of actions that can fail if methods are run before the document is fully loaded:

- Trying to hide an element that is not created yet
- Trying to get the size of an image that is not loaded yet

---

jQuery Selectors

jQuery selectors allow you to select and manipulate HTML element(s).

jQuery selectors are used to "find" (or select) HTML elements based on their name, id, classes, types, attributes, values of attributes and much more.

It's based on the existing CSS Selectors, and in addition

The element Selector

You can select all <p> elements on a page like this:

```
$("p")
```

When a user clicks on a button, all <p> elements will be hidden:

```
$(document).ready(function(){
  $("button").click(function(){
    $("p").hide();
  });
})
```

The #id Selector

```
$("#test")
```

When a user clicks on a button, the element with id="test" will be hidden:

```
$(document).ready(function(){
  $("button").click(function(){
    $("#test").hide();
  }
```

The .class Selector

```
$(".test")
```

When a user clicks on a button, the elements with class="test" will be hidden:

```
$(document).ready(function(){
  $("button").click(function(){
    $(".test").hide();
  });
})
```

More Examples of jQuery Selectors

Syntax	Description
$("*")	Selects all elements
$(this)	Selects the current HTML element
$("p.intro")	Selects all <p> elements with class="intro"
$("p:first")	Selects the first <p> element
$("ul li:first")	Selects the first element of the first
$("ul li:first-child")	Selects the first element of every
$("[href]")	Selects all elements with an href attribute

$("a[target='_blank']")	Selects all <a> elements with a target attribute value equal to "_blank"
$("a[target!='_blank']")	Selects all <a> elements with a target attribute value NOT equal to "_blank"
$(":button")	Selects all <button> elements and <input> elements of type="button"
$("tr:even")	Selects all even <tr> elements
$("tr:odd")	Selects all odd <tr> elements

```
<!DOCTYPE html>
 <html><head>
<script src="jqueary.js">  </script>
<script>
$(document).ready(function (){
    $("button").click(function(){
        $("h1").hide(300)
    })
})
</script></head>
<body>
<h1>Welcome to my college</h1>
<button>click here</button>
</body> </html>
```

```
<!DOCTYPE html>
<html lang="en">
<head>   <script src="jqueary.js"></script>
  <script>
$(document).ready(function(){
  $("#b1").click(function(){
    $("div").hide()
  }) })
$(document).ready(function(){
  $("#b2").click(function(){
    $("div").show()
  }) })
  </script>
</head>
<body>
    <button id="b1">click me to hide </button>
    <button id="b2">click me to show</button>
    <div style="border: 1px solid black; width: 50%; height: 200px;">Lorem ipsum dolor sit amet consectetur adipisicing elit. Minima, facilis? Lorem ipsum dolor sit amet consectetur adipisicing elit. Atque ipsum consequatur possimus vel. Officiis accusamus, quia delectus temporibus at reiciendis harum possimus</div>
</body>
</html>
```

THIS

```
<head>
<script src="jqueary.js"></script>
<script>

$(document).ready(function(){
  $("p").click(function(){
    $(this).hide()
  })

}) </script>
</head>
<body>
    <p >helloooo 1 </p>
    <p >helloooo 2 </p>
    <p >helloooo 3 </p>
    <p >helloooo 4 </p>
</body>  </html>
```

jQuery Events

An event represents the precise moment when something happens.

Event method

Mouse Events	Keyboard Events	Form Events	Document/Window Events
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

click	**dblclick**	**mouseenter**
`$("p").click(function(){ $(this).hide(); });`	`$("p").dblclick( function(){ $(this).hide(); });`	`$("#p1").mouseenter( function(){ alert("You entered p1!"); });`
mouseleave	**mousedown**	**mouseup**
`$("#p1").mouseleave( function(){ alert("Bye! You now leave p1!"); });`	`$("#p1").mousedown( function(){ alert("Mouse down over p1!"); });`	`$("#p1").mouseup( function(){ alert("Mouse up over p1!"); });`

Hover()

hover() method takes two functions and is a combination of the mouseenter() and mouseleave() methods.

```
$("#p1").hover(function(){
  alert("You entered p1!");
},
function(){
  alert("Bye! You now leave p1!");
});
```

focus() and blur()

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("input").focus(function(){
    $(this).css("background-color", "yellow");
  });
  $("input").blur(function(){
    $(this).css("background-color", "green");
  });
});
</script>
</head>
<body>

Name: <input type="text" name="fullname"><br>
Email: <input type="text" name="email">
```

on() Method

The on() method attaches one or more event handlers for the selected elements.

```
$("p").on("click", function(){
  $(this).hide();
});
```

Attach multiple event handlers to a <p> element:

```
$("p").on({
  mouseenter: function(){
    $(this).css("background-color", "lightgray");
  },
  mouseleave: function(){
    $(this).css("background-color", "lightblue");
  },
  click: function(){
    $(this).css("background-color", "yellow");
  }});
```

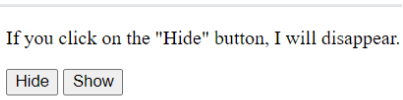
jQuery Effects
hide() and show() , toggle()

Syntax:

```
$(selector).hide(speed,callback);
$(selector).show(speed,callback);
$(selector).toggle(speed,callback);
```

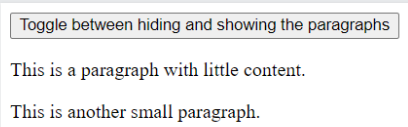
The optional speed parameter can take the following values: "slow", "fast", or milliseconds.

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("#hide").click(function(){
    $("p").hide();
  });
  $("#show").click(function(){
    $("p").show();
  });
});
</script>
</head>
<body>

<p>If you click on the "Hide" button, I will disappear.</p>

<button id="hide">Hide</button>
<button id="show">Show</button>

</body>
</html>
```



You can also toggle between hiding and showing an element with the `toggle()` method.

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("p").toggle();
  });
});
</script>
</head>
<body>

<button>Toggle between hiding and showing the paragraphs</button>

<p>This is a paragraph with little content.</p>
<p>This is another small paragraph.</p>

</body>
</html>
```

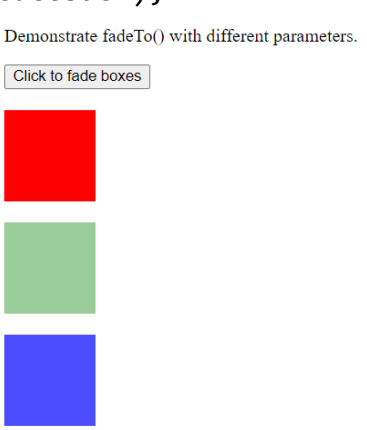


Fading Methods

With jQuery you can fade an element in and out of visibility.
jQuery has the following fade methods:

- fadeIn() | $(*selector*).fadeIn(*speed,callback*);
- fadeOut() | $(*selector*).fadeOut(*speed,callback*);
- fadeToggle() | $(*selector*).fadeToggle(*speed,callback*);
- fadeTo() | $(*selector*).fadeTo(*speed,opacity,callback*);

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("#div1").fadeTo("slow", 1);
    $("#div2").fadeTo("slow", 0.4);
    $("#div3").fadeTo("slow", 0.7);
  });
});
</script>
</head>
<body>

<p>Demonstrate fadeTo() with different parameters.</p>

<button>Click to fade boxes</button><br><br>

<div id="div1" style="width:80px;height:80px;background-color:red;"></div><br>
<div id="div2" style="width:80px;height:80px;background-color:green;"></div><br>
<div id="div3" style="width:80px;height:80px;background-color:blue;"></div>

</body>
</html>
```



Sliding Methods

With jQuery you can create a sliding effect on elements.
jQuery has the following slide methods:

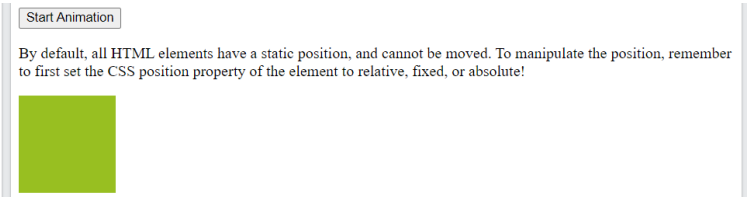
- slideDown() | $(*selector*).slideDown(*speed,callback*);
- slideUp() | $(*selector*).slideUp(*speed,callback*);
- slideToggle() | $(*selector*).slideToggle(*speed,callback*);

animate() Method

The jQuery `animate()` method is used to create custom animations.
$(*selector*).animate({*params*},*speed*,*callback*);

- The required params parameter defines the CSS properties to be animated.
- The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.
- The optional callback parameter is a function to be executed after the animation completes.
- **NOTE : By default, all HTML elements have a static position, and cannot be moved.**
  To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("div").animate({left: '250px'},3000);
  });
});
</script>
</head>
<body>

<button>Start Animation</button>

<p>By default, all HTML elements have a static position, and cannot be moved. To
manipulate the position, remember to first set the CSS position property of the element
to relative, fixed, or absolute!</p>

<div style="background:#98bf21;height:100px;width:100px;position:absolute;"></div>

</body>
</html>
```

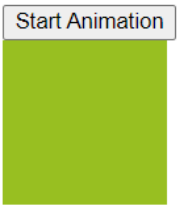



```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("div").animate({
      left: '250px',
      height: '+=150px',
      width: '+=150px'
    });
  });
});
</script>
</head>
<body>
<button>Start Animation</button>
<div style="background:#98bf21;height:100px;width:100px;position:absolute;"></div>

</body>
</html>
```

jQuery animate() - Using Pre-defined Values

You can even specify a property's animation value as "show", "hide", or "toggle":

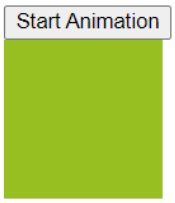
```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("div").animate({
      height: 'toggle'
    });
  });
});
</script>
</head>
<body>
<button>Start Animation</button>
<div style="background:#98bf21;height:100px;width:100px;position:absolute;"></div>

</body>
</html>
```



animate() - Uses Queue Functionality

This means that if you write multiple `animate()` calls after each other, jQuery creates an "internal" queue with these method calls. Then it runs the animate calls ONE by ONE.

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    var div = $("div");
    div.animate({height: '300px', opacity: '0.4'}, "slow");
    div.animate({width: '300px', opacity: '0.8'}, "slow");
    div.animate({height: '100px', opacity: '0.4'}, "slow");
    div.animate({width: '100px', opacity: '0.8'}, "slow");
  });
});
</script>
</head>
<body>

<button>Start Animation</button>

<div style="background:#98bf21;height:100px;width:100px;position:absolute;"></div>

</body>
</html>
```

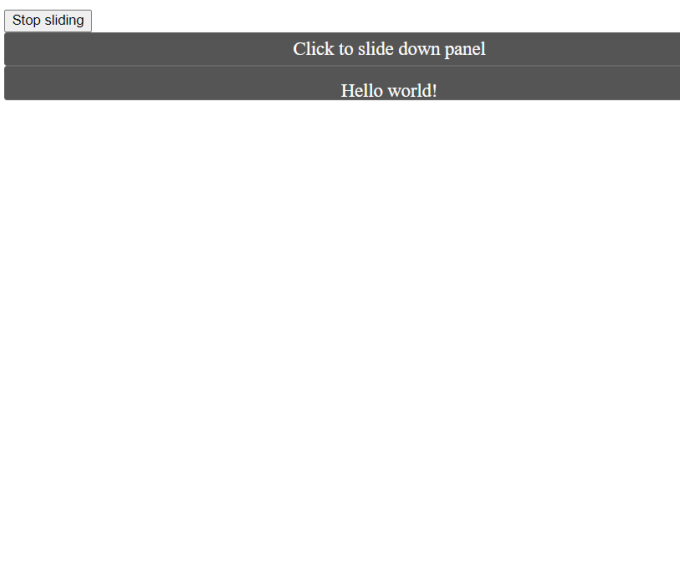


Stop() Method

The jQuery `stop()` method is used to stop an animation or effect before it is finished.
The `stop()` method works for all jQuery effect functions, including sliding, fading and custom animations.

`$(selector).stop(stopAll,goToEnd);`

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js">
</script>
<script>
$(document).ready(function(){
  $("#flip").click(function(){
    $("#panel").slideDown(5000);
  });
  $("#stop").click(function(){
    $("#panel").stop();
  });
});
</script>
<style>
#panel, #flip { padding: 5px; font-size: 18px; text-align: center;
  background-color: #555; color: white; border: solid 1px #666; border-radius: 3px;}
#panel {padding: 50px; display: none; }
</style>
</head>
<body>

<button id="stop">Stop sliding</button>

<div id="flip">Click to slide down panel</div>
<div id="panel">Hello world!</div>

</body>
</html>
```

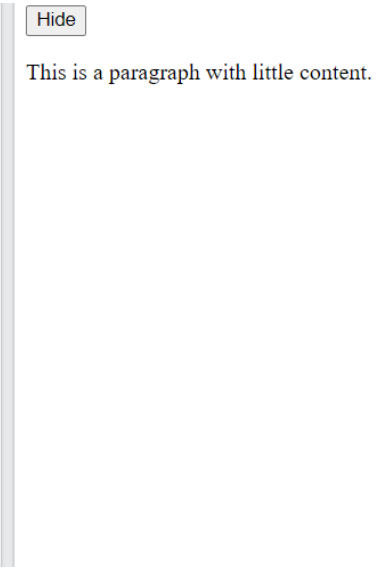


Callback Functions

A callback function is executed after the current effect is 100% finished.

Typical syntax: **$(*selector*).hide(*speed,callback*);**

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("p").hide("slow", function(){
      alert("The paragraph is now hidden");
    });
  });
});
</script>
</head>
<body>

<button>Hide</button>

<p>This is a paragraph with little content.</p>

</body>
</html>
```

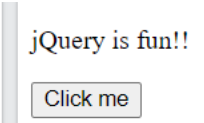


AFTER click on hide , alert function run

---

jQuery – Chaining

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("#p1").css("color", "red").slideUp(2000).slideDown(2000);
  });
});
</script>
</head>
<body>

<p id="p1">jQuery is fun!!</p>

<button>Click me</button>

</body>
</html>
```



jQuery HTML

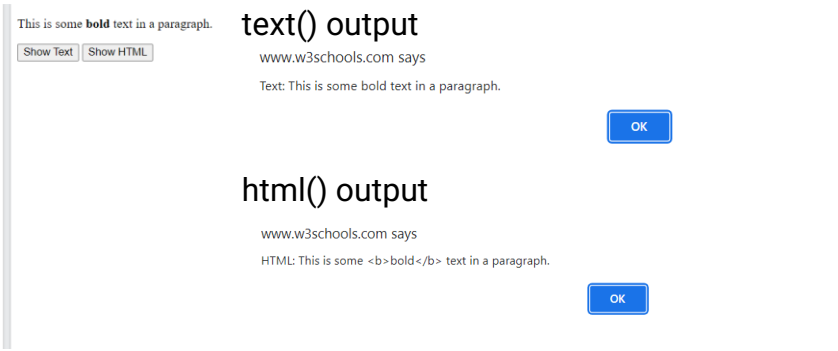
jQuery contains powerful methods for changing and manipulating HTML elements and attributes.

Get Content - text(), html(), and val()

Three simple, but useful, jQuery methods for DOM manipulation are:

- `text()` - Sets or returns the text content of selected elements
- `html()` - Sets or returns the content of selected elements (including HTML markup)
- `val()` - Sets or returns the value of form fields

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("#btn1").click(function(){
    alert("Text: " + $("#test").text());
  });
  $("#btn2").click(function(){
    alert("HTML: " + $("#test").html());
  });
});
</script>
</head>
<body>

<p id="test">This is some <b>bold</b> text in a paragraph.</p>

<button id="btn1">Show Text</button>
<button id="btn2">Show HTML</button>

</body>
</html>
```




```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.mi
n.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    alert("Value: " + $("#test").val());
  });
});
</script>
</head>
<body>
<p>Name: <input type="text" id="test" value="Mickey Mouse"></p>
<button>Show Value</button>
</body> </html>
```

Get Attributes - attr()

The jQuery `attr()` method is used to get attribute values.

The following example demonstrates how to get the value of the href attribute in a link:

Example

```
$("button").click(function(){
  alert($("#w3s").attr("href"));
});
```

Set Content - text(), html(), and val()

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js">
</script>
<script>
$(document).ready(function(){
  $("#btn1").click(function(){
    $("#test1").text("Hello world!");
  });
  $("#btn2").click(function(){
    $("#test2").html("<b>Hello world!</b>");
  });
  $("#btn3").click(function(){
    $("#test3").val("Dolly Duck");
  });
});
</script>
</head>
<body>

<p id="test1">This is a paragraph.</p>
<p id="test2">This is another paragraph.</p>

<p>Input field: <input type="text" id="test3" value="Mickey Mouse"></p>

<button id="btn1">Set Text</button>
<button id="btn2">Set HTML</button>
<button id="btn3">Set Value</button>
</body></html>
```

Add Elements in html

With jQuery, it is easy to add new elements/content.

Add New HTML Content

We will look at four jQuery methods that are used to add new content:

`append()` - Inserts content at the end of the selected elements

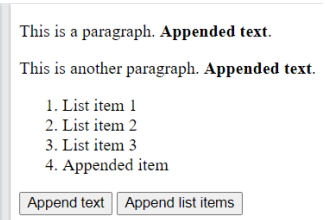
`prepend()` - Inserts content at the beginning of the selected elements

`after()` - Inserts content after the selected elements

`before()` - Inserts content before the selected elements

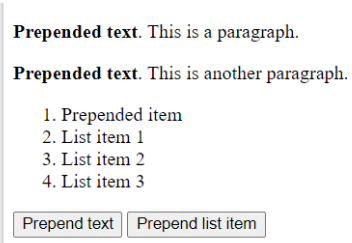
append()

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js">
</script>
<script>
$(document).ready(function(){
  $("#btn1").click(function(){
    $("p").append(" <b>Appended text</b>.");
  });

  $("#btn2").click(function(){
    $("ol").append("<li>Appended item</li>");
  });
});
</script>
</head>
<body>

<p>This is a paragraph.</p>
<p>This is another paragraph.</p>

<ol>
  <li>List item 1</li>
  <li>List item 2</li>
  <li>List item 3</li>
</ol>
<button id="btn1">Append text</button>
<button id="btn2">Append list items</button>
 </body> </html>
```



Prepend()

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js">
</script>
<script>
$(document).ready(function(){
  $("#btn1").click(function(){
    $("p").prepend("<b>Prepended text</b>. ");
  });
  $("#btn2").click(function(){
    $("ol").prepend("<li>Prepended item</li>");
  });
});
</script>
</head>
<body>

<p>This is a paragraph.</p>
<p>This is another paragraph.</p>

<ol>
  <li>List item 1</li>
  <li>List item 2</li>
  <li>List item 3</li>
</ol>

<button id="btn1">Prepend text</button>
<button id="btn2">Prepend list item</button>
 </body> </html>
```



Before() and after()

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("#btn1").click(function(){
    $("img").before("<b>Before</b>");
  });

  $("#btn2").click(function(){
    $("img").after("<i>After</i>");
  });
});
</script>
</head>
<body>

<img src="/images/w3jquery.gif" alt="jQuery" width="100" height="140"><br><br>

<button id="btn1">Insert before</button>
<button id="btn2">Insert after</button>

</body>
</html>
```

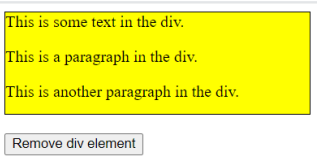
Remove() and empty() Elements

With jQuery, it is easy to remove existing HTML elements.

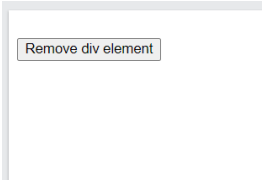
To remove elements and content, there are mainly two jQuery methods:

- `remove()` - Removes the selected element (and its child elements)
- `empty()` - Removes the child elements from the selected element

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("#div1").remove();
  });
});
</script>
</head>
<body>

<div id="div1" style="height:100px;width:300px;border:1px solid black;background-color:yellow;">

This is some text in the div.
<p>This is a paragraph in the div.</p>
<p>This is another paragraph in the div.</p>

</div>
<br>

<button>Remove div element</button>

</body>
</html>
```



Output



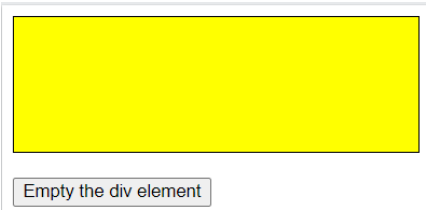
```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("#div1").empty();
  });
});
</script>
</head>
<body>

<div id="div1" style="height:100px;width:300px;border:1px solid black;background-color:yellow;">

This is some text in the div.
<p>This is a paragraph in the div.</p>
<p>This is another paragraph in the div.</p>

</div>
<br>

<button>Empty the div element</button>

</body>
</html>
```

This is some text in the div.

This is a paragraph in the div.

This is another paragraph in the div.

Empty the div element



Empty the div element

Get and Set CSS Classes

With jQuery, it is easy to manipulate the style of elements.

jQuery Manipulating CSS

jQuery has several methods for CSS manipulation. We will look at the following methods:

`addClass()` - Adds one or more classes to the selected elements

`removeClass()` - Removes one or more classes from the selected elements

`toggleClass()` - Toggles between adding/removing classes from the selected elements

`css()` - Sets or returns the style attribute

```
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js">
</script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("h1, h2, p").addClass("blue");
    $("div").addClass("important");
  });
});
</script>
<style>
.important {
  font-weight: bold;
  font-size: xx-large;}

.blue {
  color: blue;
}
</style>
</head>
<body>
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
<div>This is some important text!</div><br>
<button>Add classes to elements</button>
</body></html>
```

Heading 1

Heading 2

This is a paragraph.

This is another paragraph.

This is some important text!

Add classes to elements

JSON

- JSON stands for JavaScript Object Notation
- JSON is a format for storing and transporting data.
- JSON is often used when data is sent from a server to a web page.
- A common use of JSON is to read data from a web server, and display the data in a web page. For simplicity, this can be demonstrated using a string as input.
- The JSON syntax is derived from JavaScript object notation syntax, but the JSON format is text only.
- Because of this similarity, a JavaScript program can easily convert JSON data into native JavaScript objects.

JSON Syntax Rules

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays
- `"firstName":"John"`
- JSON names require double quotes. JavaScript names do not.
- JSON objects are written inside curly braces. `{"firstName":"John", "lastName":"Doe"}`

JSON Arrays

JSON arrays are written inside square brackets.

```
"employees":[
  {"firstName":"John", "lastName":"Doe"},
  {"firstName":"Anna", "lastName":"Smith"},
  {"firstName":"Peter", "lastName":"Jones"} ]
```

Converting a JSON Text to a JavaScript Object

First, create a JavaScript string containing JSON syntax:

```
let text = '{ "employees" : [' +
'{ "firstName":"John" , "lastName":"Doe" },' +
'{ "firstName":"Anna" , "lastName":"Smith" },' +
'{ "firstName":"Peter" , "lastName":"Jones" } ]}';
```

Then, use the JavaScript built-in function `JSON.parse()` to convert the string into a JavaScript object:

```
const obj = JSON.parse(text);
```

Finally, use the new JavaScript object in your page:

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

<script>
document.getElementById("demo").innerHTML =
obj.employees[1].firstName + " " + obj.employees[1].lastName;
</script>
```

Array.from()

The **Array.from()** static method creates a new copied Array from an iterable or array-like object.

syntax

Array.from(arrayLike)

Array.from(arrayLike, mapFn)

Array.from(arrayLike, mapFn, thisArg)

arrayLike

An iterable or array-like object to convert to an array.

mapFn Optional

A function to call on every element of the array. If provided, every value to be added to the array is first passed through this function, and mapFn's return value is added to the array instead. The function is called with the following arguments:

element

The current element being processed in the array.

index

The index of the current element being processed in the array.

thisArg Optional

Value to use as this when executing mapFn.

Return value

A new Array instance.

```
console.log(Array.from('foo'));
// Expected output: Array ["f", "o", "o"]

console.log(Array.from([1, 2, 3], x => x + x));
// Expected output: Array [2, 4, 6]
```

Target

The target property returns the element where the event occured.

The target property is read-only.

```
<body>
   <h1 id="heading">Click me!</h1>
<script>
      const heading = document.getElementById('heading');
      heading.addEventListener('click', (event) => {
         const target = event.target.innerText;
            console.log(target)
        }
        );
   </script>
</body>
</html>
Result:click me
```

```
<body onclick="myFunction(event)">
   <h1>HTML DOM Events</h1>

   <p>Click on any elements in this document to find out which element triggered the onclick event.</p>

   <button>This is a button</button>

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

   <script>
   function myFunction(event) {
     let text = event.target.innerHTML;
     document.getElementById("demo").innerHTML = text;
     console.log(event.target.innerHTML)
   }
   </script>

   </body>
```

Eval()

eval() The eval() method evaluates or executes an argument.

syntax = eval(string)

```
let x = 10;
let y = 20;
let text = "x * y";
let result = eval(text);
```

BOM Browser Object Model

The Browser Object Model (BOM) allows JavaScript to "talk to" the browser.
It represents the browser's window.

```
window.document.getElementById("header");
is the same as:
document.getElementById("header");
```

- `window.innerHeight` - the inner height of the browser window (in pixels)
- `window.innerWidth` - the inner width of the browser window (in pixels)
- `window.open()` - open a new window
- `window.close()` - close the current window
- `window.moveTo()` - move the current window
- `window.resizeTo()` - resize the current window

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

<h2>JavaScript Window</h2>

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

<script>
document.getElementById("demo").innerHTML =
"Browser inner window width: " + window.innerWidth + "px<br>" +
"Browser inner window height: " + window.innerHeight + "px";
</script>

</body>
</html>
```

JavaScript Window

Browser inner window width: 753px
Browser inner window height: 558px

Window Screen

The window.screen object contains information about the user's screen.
The `window.screen` object can be written without the window prefix.
Properties:

- `screen.width`
- `screen.height`
- `screen.availWidth`
- `screen.availHeight`
- `screen.colorDepth`
- `screen.pixelDepth`

```
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"Screen height is " + screen.height;
</script>
</body>
```

OUTPUT: Screen height is 864

Window Location

The `window.location` object can be written without the window prefix.

Some examples:

- `window.location.href` returns the href (URL) of the current page
- `window.location.hostname` returns the domain name of the web host
- `window.location.pathname` returns the path and filename of the current page
- `window.location.protocol` returns the web protocol used (http: or https:)
- `window.location.assign()` loads a new document

Window History

The `window.history` object can be written without the window prefix.

To protect the privacy of the users, there are limitations to how JavaScript can access this object.

Some methods:

- `history.back()` - same as clicking back in the browser
- `history.forward()` - same as clicking forward in the browser

History.back()

```
<html>
<head>
<script>
function goBack() {
  window.history.back()
}
</script>
</head>
<body>

<input type="button" value="Back" oncli
ck="goBack()">

</body>
</html>
```

history.forward()

```
<html>
<head>
<script>
function goForward() {
  window.history.forward()
}
</script>
</head>
<body>

<input type="button" value="Forward" onclick
="goForward()">

</body>
</html>
```

Popup Boxes = Alert (), Confirm() , and Prompt ().

JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

Alert Box

An alert box is often used if you want to make sure information comes through to the user.
When an alert box pops up, the user will have to click "OK" to proceed.

```
window.alert("sometext");
```

The `window.alert()` method can be written without the window prefix.

```
alert("I am an alert box!");
```

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

<h2>JavaScript Alert</h2>

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

<script>
function myFunction() {
  alert("I am an alert box!");
}
</script>
</body> </html>
```

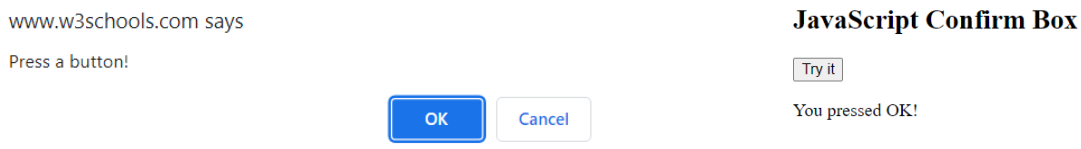
output:

Confirm Box

A confirm box is often used if you want the user to verify or accept something.
When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.
If the user clicks "OK", the box returns **true**. If the user clicks "Cancel", the box returns **false**.

```
window.confirm("sometext");
```

```
<!DOCTYPE html> <html> <body>
<h2>JavaScript Confirm Box</h2>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  var txt;
  if (confirm("Press a button!")) {
    txt = "You pressed OK!";
  } else {
    txt = "You pressed Cancel!";
  }
  document.getElementById("demo").innerHTML = txt;
}
</script> </body> </html>
```

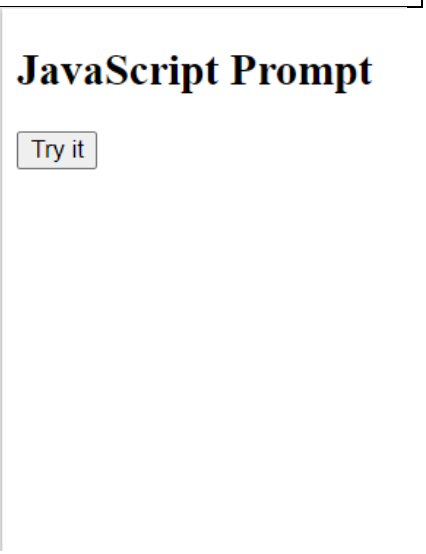


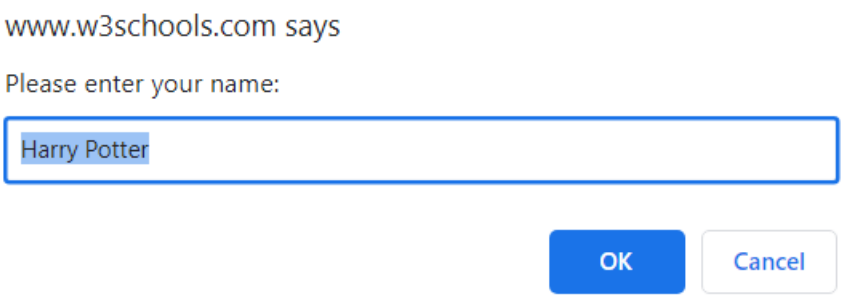
Prompt Box

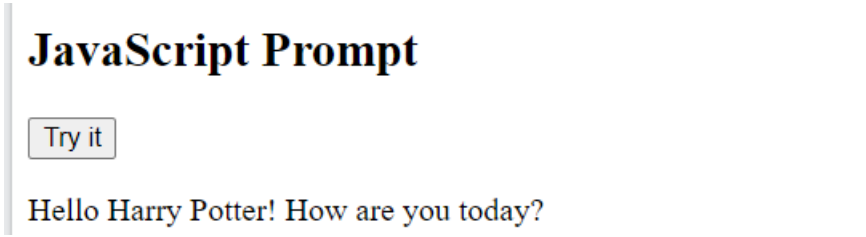
A prompt box is often used if you want the user to input a value before entering a page.
When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.
If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

```
window.prompt("sometext","defaultText");
```

```
<!DOCTYPE html> <html> <body>
<h2>JavaScript Prompt</h2>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  let text;
  let person = prompt("Please enter your name:", "Harry Potter");
  if (person == null || person == "") {
    text = "User cancelled the prompt.";
  } else {
    text = "Hello " + person + "! How are you today?";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>  </body> </html>
```







Timing Events= setTimeout() , setInterval(),clearTimeout()

The `window` object allows execution of code at specified time intervals.
These time intervals are called timing events.

- `setTimeout(function, milliseconds)`
  Executes a function, after waiting a specified number of milliseconds.

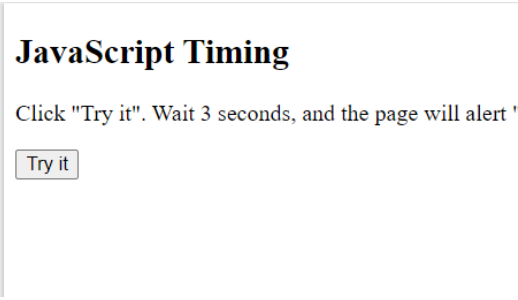
- `setInterval(function, milliseconds)`
  Same as setTimeout(), but repeats the execution of the function continuously.

- `clearTimeout(timeoutVariable)`
  stops the execution of the function specified in setTimeout().

- `clearInterval(timerVariable)`
  stops the executions of the function specified in the setInterval() method.

setTimeout(function, milliseconds)

```
<!DOCTYPE html> <html> <body>
<h2>JavaScript Timing</h2>
<p>Click "Try it". Wait 3 seconds, and the page will alert "Hello".</p>
<button onclick="setTimeout(myFunction, 3000);">Try it</button>
<script>
function myFunction() {
  alert('Hello');
}
</script>
</body>
</html>
```



clearTimeout(timeoutVariable)

```
myVar = setTimeout(function, milliseconds);
clearTimeout(myVar);
```

```
<!DOCTYPE html> <html> body>
<p>Click "Try it". Wait 3 seconds. The page will alert "Hello".</p>
<p>Click "Stop" to prevent the first function to execute.</p>
<p>(You must click "Stop" before the 3 seconds are up.)</p>

<button onclick="myVar = setTimeout(myFunction, 3000)">Try it</button>

<button onclick="clearTimeout(myVar)">Stop it</button>

<script>
function myFunction() {
  alert("Hello");
}
</script> </body> </html>
```

setInterval(function, milliseconds)

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

<p id="demo"></p>
<script>
setInterval(myTimer, 1000);

function myTimer() {
  const d = new Date();
  document.getElementById("demo").innerHTML = d.toLocaleTimeString();
}
</script>

</body>
</html>
```

A script on this page starts this clock:

9:21:56 PM

clearInterval()

```
<h2>JavaScript Timing</h2>
<p>A script on this page starts this clock:</p>

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

<button onclick="clearInterval(myVar)">Stop time</button>

<script>
let myVar = setInterval(myTimer ,1000);
function myTimer() {
  const d = new Date();
  document.getElementById("demo").innerHTML = d.toLocaleTimeString();
}
</script>
```

JavaScript Timing

A script on this page starts this clock:

9:23:16 PM

Stop time

What are Cookies?

- Cookies are data, stored in small text files, on your computer.
- When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.
- Cookies were invented to solve the problem "how to remember information about the user":
- When a user visits a web page, his/her name can be stored in a cookie.
- Next time the user visits the page, the cookie "remembers" his/her name.
- Cookies are saved in name-value pairs like:
  `username = John Doe`