JavaScript is the programming language of HTML and the Web.

JavaScript Can Change HTML Content

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

This example uses the method to "find" an HTML element (with id="demo") and changes the element content (**innerHTML**) to "Hello JavaScript":

Example

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

Try it Yourself »

JavaScript accepts both double and single quotes:

Example

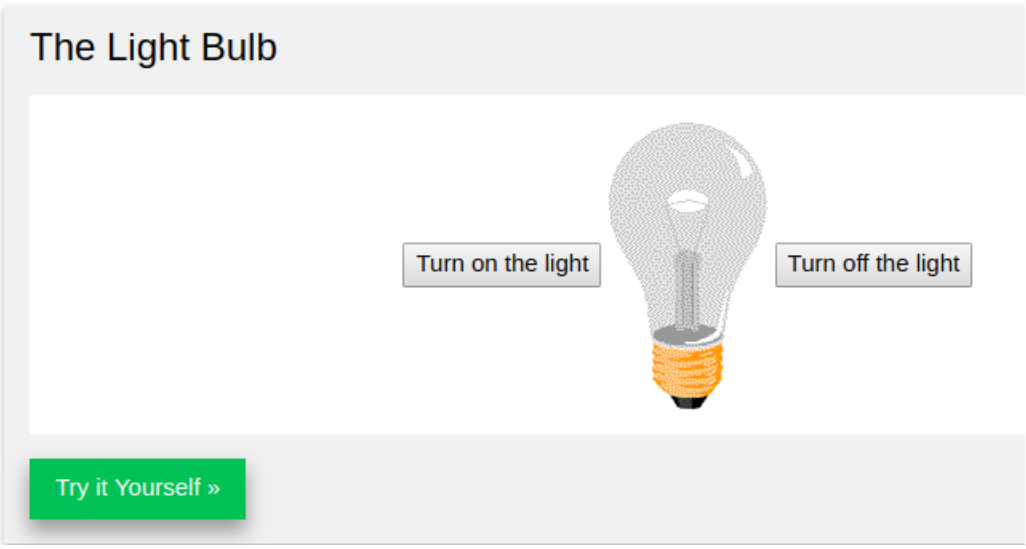
```
document.getElementById('demo').innerHTML = 'Hello JavaScript';
```

Try it Yourself »

Nice Example

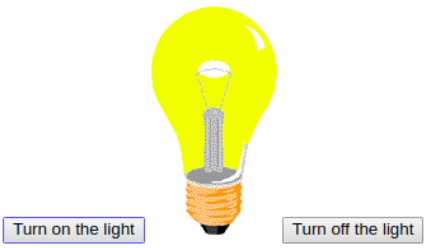
JavaScript Can Change HTML Attribute Values

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



https://www.w3schools.com/js/tryit.asp?filename=tryjs_intro_lightbulb

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

<button
onclick="document.getElementById('myImage').src='pic_bulbon.gif'">Tu
rn on the light</button>

<img id="myImage" src="pic_bulboff.gif" style="width:100px">

<button
onclick="document.getElementById('myImage').src='pic_bulboff.gif'">T
urn off the light</button>

</body>
</html>
```



JS css Example

JavaScript Can Hide HTML Elements

https://www.w3schools.com/js/tryit.asp?filen

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

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can hide HTML elements.</p>

<button type="button"
onclick="document.getElementById('demo').style.display='none'">Click
Me!</button>


</body>
</html>
```

JavaScript in <head> or <body>

JavaScript in <head> or <body>

You can place any number of scripts in an HTML document.

Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

> Placing scripts at the bottom of the <body> element improves the display speed, because script compilation slows down the display.

JavaScript in <head>

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

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

Example

```
<!DOCTYPE html>
<html>

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

<body>

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

</body>
</html>
```

```
<html>
<body>

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

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

</body>
</html>
```

External JavaScript

External file: myScript.js

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

External scripts are practical when the same code is used in many different web pages.

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

<h2>External JavaScript</h2>

<p id="demo">A Paragraph.</p>

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

<p>(myFunction is stored in an external file called "myScript.js")
</p>

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

</body>
</html>
```

External JavaScript Advantages

Placing scripts in external files has some advantages:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

To add several script files to one page - use several script tags:

Example

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

External References

External scripts can be referenced with a full URL or with a path relative to the current web page.

This example uses a full URL to link to a script:

Example

```
<script src="https://www.w3schools.com/js/myScript1.js"></script>
```

What is Hoisting in JavaScript ? Most important question in JS developer interview

In JavaScript, a variable can be used before it has been declared يعرف – يعلن .

Example 1 gives the same result as **Example 2**:

Example 1

```
x = 5; // Assign 5 to x

elem = document.getElementById("demo"); // Find an element
elem.innerHTML = x;                     // Display x in the element

var x; // Declare x
```

Try it Yourself »

Example 1 gives the same result as **Example 2**:

Example 1

```
x = 5; // Assign 5 to x

elem = document.getElementById("demo"); // Find an element
elem.innerHTML = x;                     // Display x in the element

var x; // Declare x
```

Try it Yourself »

Hoisting is JavaScript's default behavior of moving all declarations to the top of the current scope (to the top of the current script or the current function).

The math operations with Converting the ‘string’

the + operation converts (the number when adding to string ‘’) to string.

```
var x = '8 '+ 2;
console.log(x);
console.log(typeof x);
```

```
8 2
string
```

Important Note 1:

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

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

The result of adding "5" + 2 + 3:

523

Important Note 2:

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

The result of adding 2 + 3 + "5":

55

---

the - * / % operations convert (the string ‘’ when adding to number) to number.

```
var x = '8 '- 2;
console.log(x);
console.log(typeof x);

var y = '8 '* 2;
console.log(y);
console.log(typeof y);

var z = '8 '/ 2;
console.log(z);
console.log(typeof z);

var w = '8 ' % 2;
console.log(w);
console.log(typeof w);
```

```
6
number
16
number
4
number
0
number
```

JS HTML DOM addEventListener()

-The addEventListener() method attaches an event handler to the specified element. تقوم بإرفاق معالج أحداث بالعنصر المحدد.

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

- event : Required. A String that specifies the name of the event.

events: https://www.w3schools.com/jsref/dom_obj_event.asp

- function: Required. Specifies the function to run when the event occurs.

- `document.getElementById("id").addEventListener("click", function(){});`

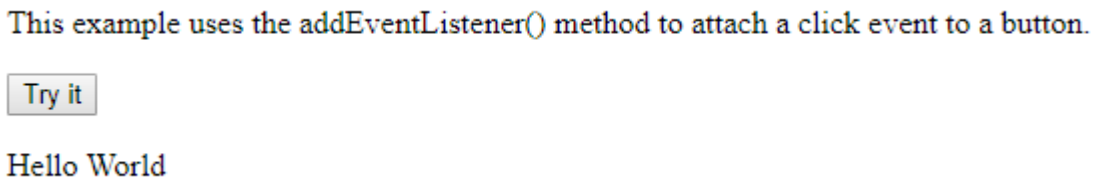
- **Tip:** Use the removeEventListener() method to remove an event handler that has been attached with the addEventListener() method.

-Tip: Use the document.addEventListener() method to attach an event handler to the document.

Ex:

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

<p>This example uses the addEventListener() method to attach a click event to a
button.</p>

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

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

<script>
document.getElementById("myBtn").addEventListener("click", function(){
    document.getElementById("demo").innerHTML = "Hello World";
});
</script>

</body>
</html>
```

In Browser



Logical operation ! && ||

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

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

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

who execute first :

1. **not !**
2. **and &&**
3. **or ||**

Conditional (Ternary ثلاثي) Operator

```
var isRich = true;

isRich ? console.log('True, he is Rich') : console.log('False, he is not Rich');
```

That means :

```
var isRich = true;

switch (isRich) {
    case true:
        console.log('True, he is Rich');
        break;
    case false:
        console.log('False, he is not Rich');
        break;
}
```

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.

Example

The getDay() method returns the weekday as a number between 0 and 6.

If today is neither Saturday (6) nor Sunday (0), write a default message:

```
switch (new Date().getDay()) {
    case 6:
        text = "Today is Saturday";
        break;
    case 0:
        text = "Today is Sunday";
        break;
    default:
        text = "Looking forward to the Weekend";
}
```

Operator precedence - order

The following table is ordered from higheWst (20) to lowest (1) precedence.

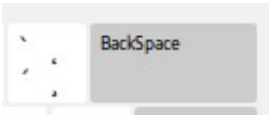
https://developer.mozilla.org/en-US/docs/eb/JavaScript/Reference/Operators/Operator_Precedence

logical for numbers and string

0 zero	as logic is false
all numbers else zero	as logic is true
‘ ‘	as logic is false
‘ncjdncjdnccdkl’	as logic is true
empty array	as logic is false

To insert variables inside string

use ` `



not ‘ ‘

ex:

```
console.log(`you have orderd ${numBalls}, will cost ${price}$`);
document.getElementById('result').innerHTML = `you have orderd ${numBalls}, will cost ${price}$` ;
```

The else if Statement (note : *and the another down if will not be executed) that means only one if will be executed*

```
if (condition1) {
  block of code to be executed if condition1 is true and the another down will not be executed
}

else if (condition2) {
  block of code to be executed if the condition1 is false and condition2 is true and the another down will not be executed
}

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

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

- Writing into an HTML element, using **innerHTML**.
- Writing into the HTML output using **document.write()**.
- Writing into an alert box, using **window.alert()**.
- Writing into the browser console, using **console.log()**.

Using innerHTML

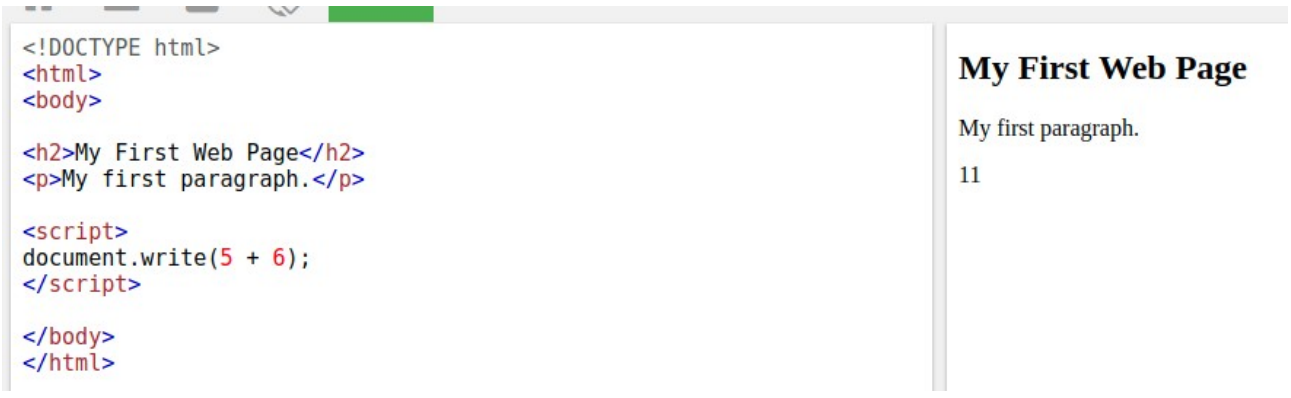
Using innerHTML

To access an HTML element, JavaScript can use the **document.getElementById(id)** method.

The **id** attribute defines the HTML element. The **innerHTML** property defines the HTML content:

Using document.write()

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



Using document.write() after an HTML document is fully loaded, will **delete all existing HTML**:

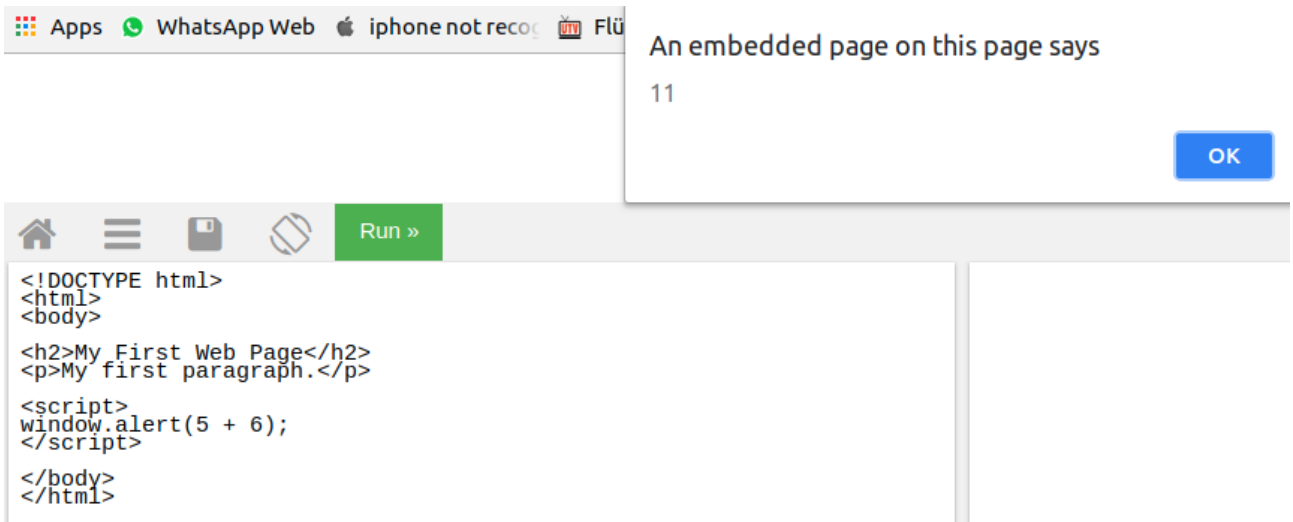
```
<!DOCTYPE html>
<html>
<body>

<h2>My First Web Page</h2>
<p>My first paragraph.</p>

<button type="button" onclick="document.write(5 + 6)">Try
it</button>

</body>
</html>
```

11

> The document.write() method should only be used for testing.

Using window.alert()

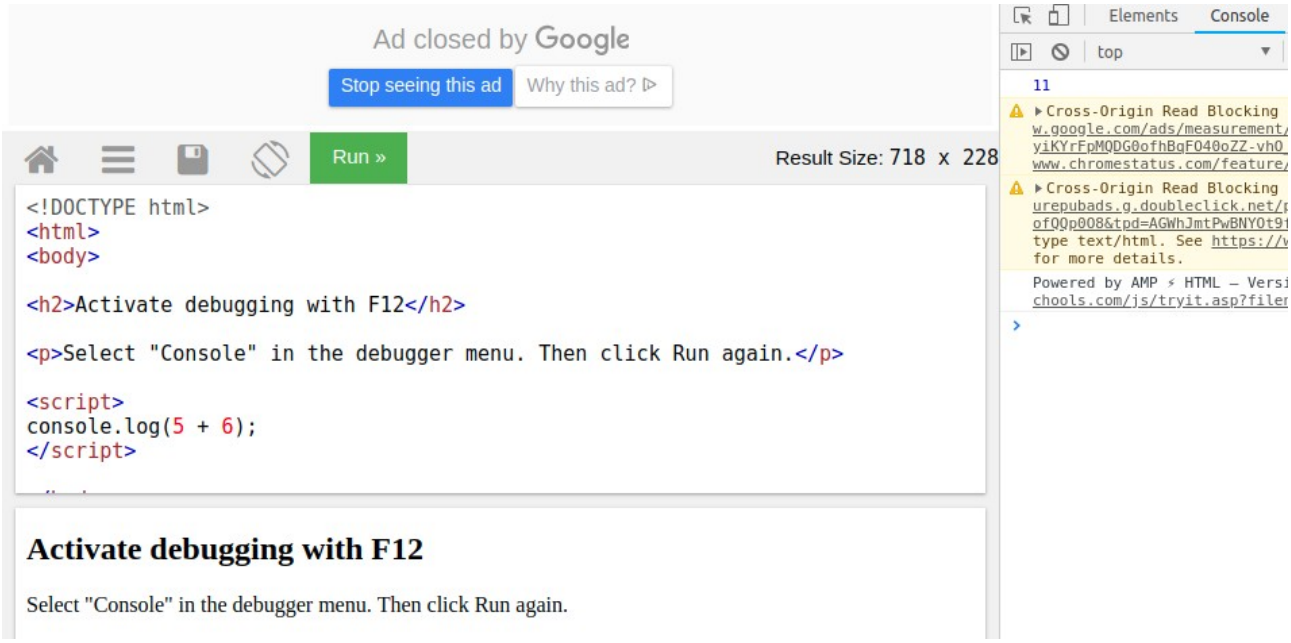
You can use an alert box to display data:



Using console.log()

For debugging purposes, you can use the **console.log()** method to display data.

> You will learn more about debugging in a later chapter.



JavaScript Statements

```
var x, y, z;     // Statement 1
x = 5;           // Statement 2
y = 6;           // Statement 3
z = x + y;       // Statement 4
```

```
a = 5; b = 6; c = a + b;
```

JavaScript White Space

JavaScript ignores multiple spaces. You can add white space to your script to make it more readable.

The following lines are equivalent:

```
var person = "Hege";
var person="Hege";
```

A good practice is to put spaces around operators (= + - * /):

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

JavaScript Line Length and Line Breaks

For best readability, programmers often like to avoid code lines longer than 80 characters.

If a JavaScript statement does not fit on one line, the best place to break it, is after an operator:

Example

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

Try it Yourself »

Variable = Variable + ‘ Hallo ‘

Variable += ‘ Hallo ‘ ;

Classy if – one line

```
var x=prompt();
var givenNumber=parseInt(x);
givenNumber!==0?console.log(-givenNumber):console.log(0);
```

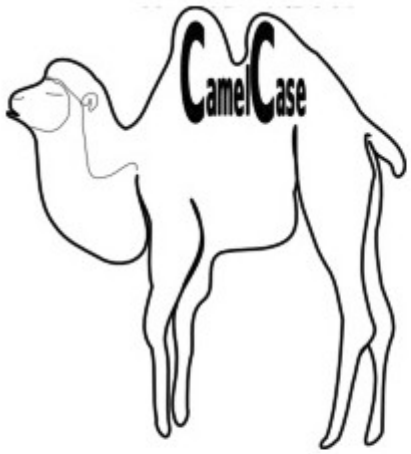
var x=prompt();
var givenNumber=parseInt(x);
givenNumber!==0?console.log(-givenNumber):console.log(0);

JavaScript Identifiers

In JavaScript, the first character must be a letter, or an underscore (_), or a dollar sign ($).

Subsequent characters may be letters, digits, underscores, or dollar signs.

JavaScript and Camel Case



Lower Camel Case:

JavaScript programmers tend to use camel case that starts with a lowercase letter:

firstName, lastName, masterCard, interCity.

Hyphens:

first-name, last-name, master-card, inter-city.

> Hyphens are not allowed in JavaScript. They are reserved for subtractions.

Underscore:

first_name, last_name, master_card, inter_city.

Upper Camel Case (Pascal Case):

FirstName, LastName, MasterCard, InterCity.

The For Loop

```
for (var i = 0; i < 5; i++) {
    // code block to be executed
}
```

The For/In Loop

The JavaScript for/in statement loops through the properties of an object:

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

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

Ex:

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

</body>
</html>
```

John Doe 25

The While Loop

```
var i = 0;
while (i < 10) {
    // code block to be executed
    i++;
}
```

The Do/While Loop

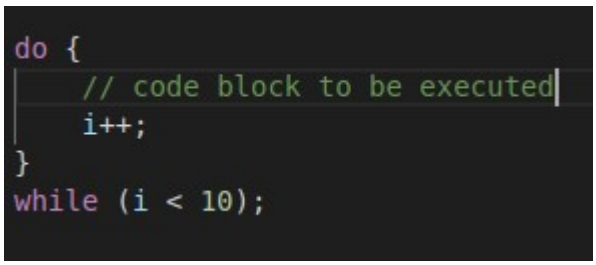
The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

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

Example

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:



Comparing For and While

The loop in this example uses a **for loop** to collect the car names from the cars array:

Example

```
var cars = ["BMW", "Volvo", "Saab", "Ford"];
var i = 0;
var text = "";

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

The loop in this example uses a **while loop** to collect the car names from the cars array:

Example

```
var cars = ["BMW", "Volvo", "Saab", "Ford"];
var i = 0;
var text = "";

while (cars[i]) {
    text += cars[i] + "<br>";
    i++;
}
```

JavaScript Functions

Ex:

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

<script>
function myFunction(p1, p2) {
    return p1 * p2;
}
document.getElementById("demo").innerHTML = myFunction(4, 3);
</script>
```

12

---

important

The () Operator Invokes the Function

Using the example above, toCelsius refers to the function object, and toCelsius() refers to the function result.

Accessing a function without () will return the function definition instead of the function result:

Example

```
function toCelsius(fahrenheit) {
    return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius;
```

function toCelsius(f) { return (5/9) * (f-32); }

---

important

Local Variables

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

Local variables can only be accessed from within the function.

Example

```
// code here can NOT use carName

function myFunction() {
    var carName = "Volvo";
    // code here CAN use carName
}

// code here can NOT use carName
```

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

Local variables are created when a function starts, and deleted when the function is completed.

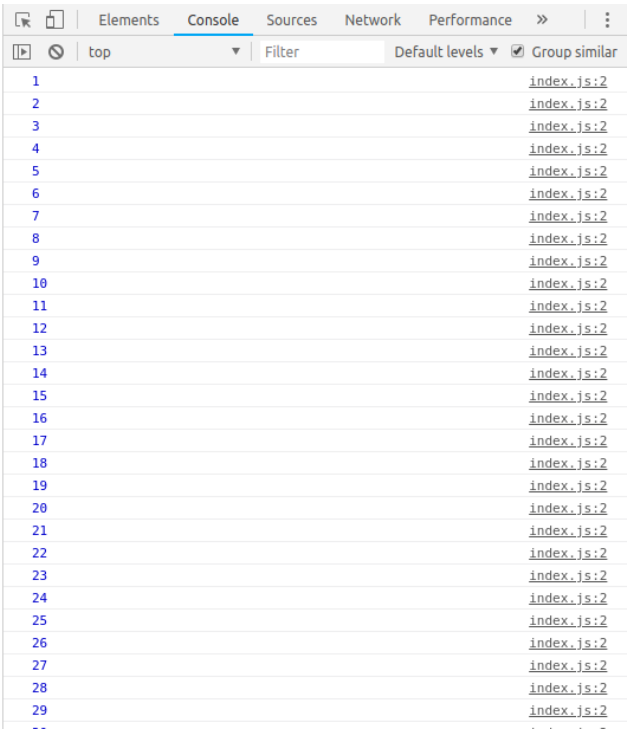
Note: js function is hoisted

(Important) Function Return

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

Ex: recursive function will stop will reaches a return

```
function printNums(intialPoint) {
    console.log(intialPoint);
    if (intialPoint===100) {
        return;             // the function will stop here
    }
    intialPoint++;
    printNums(intialPoint);
}


printNums(1);
```



Variables

It's a good programming practice to declare all variables at the beginning of a script.

You can declare many variables in one statement.

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

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

Try it Yourself »

A declaration can span multiple lines:

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

Try it Yourself »

Re-Declaring JavaScript Variables

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

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

Example

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

JavaScript Let and const

ECMAScript 2015

ES2015 introduced two important new JavaScript keywords: **let** and **const**.

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

Before ES2015, JavaScript had only two types of scope: **Global Scope** and **Function Scope**.

Global Scope

Variables declared **Globally** (outside any function) have **Global Scope**.

Example

```
var carName = "Volvo";

// code here can use carName

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

Try it Yourself »

Global variables can be accessed from anywhere in a JavaScript program.

Function Scope

Variables declared **Locally** (inside a function) have **Function Scope**.

Example

```
// code here can NOT use carName

function myFunction() {
    var carName = "Volvo";
    // code here CAN use carName
}

// code here can NOT use carName
```

Try it Yourself »

Local variables can only be accessed from inside the function where they are declared.

.....

JavaScript Block Scope

JavaScript Block Scope

Variables declared with the **var keyword** can not have **Block Scope**.

Variables declared inside a block **{}** can be accessed from outside the block.

Example

```
{
    var x = 2;
}
// x CAN be used here
```

Before ES2015 JavaScript did not have **Block Scope**.

Variables declared with the **let keyword** can have Block Scope.

Variables declared inside a block **{}** can not be accessed from outside the block:

Example

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

Redeclaring Variables

Redeclaring Variables

Redeclaring a variable using the **var keyword** can impose problems.

Redeclaring a variable inside a block will also redeclare the variable outside the block:

Example

```
var x = 10;
// Here x is 10
{
    var x = 2;
    // Here x is 2
}
// Here x is 2
```

Redeclaring a variable using the **let keyword** can solve this problem.

Redeclaring a variable inside a block will not redeclare the variable outside the block:

Example

```
var x = 10;
// Here x is 10
{
    let x = 2;
    // Here x is 2
}
// Here x is 10
```

Loop Scope

Using **var** in a loop:

Example

```
var i = 5;
for (var i = 0; i < 10; i++) {
    // some statements
}
// Here i is 10
```

Using **let** in a loop:

Example

```
let i = 5;
for (let i = 0; i < 10; i++) {
    // some statements
}
// Here i is 5
```

Function Scope

Variables declared with **var** and **let** are quite similar when declared inside a function.

They will both have **Function Scope**:

```
function myFunction() {
    var carName = "Volvo";   // Function Scope
}
```

```
function myFunction() {
    let carName = "Volvo";   // Function Scope
}
```

Global Scope

Variables declared with **var** and **let** are quite similar when declared outside a block.

They will both have **Global Scope**:

```
var x = 2;       // Global scope
```

```
let x = 2;       // Global scope
```

Global Variables in HTML

With JavaScript, the global scope is the JavaScript environment.

In HTML, the global scope is the window object.

Global variables defined with the **var** keyword belong to the window object:

Example

```
var carName = "Volvo";
// code here can use window.carName
```

Try it Yourself »

Global variables defined with the **let** keyword do not belong to the window object:

Example

```
let carName = "Volvo";
// code here can not use window.carName
```

Try it Yourself »

Redeclaring

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

Example

```
var x = 2;

// Now x is 2

var x = 3;

// Now x is 3
```

Redeclaring a **var** variable with **let**, in the same scope, or in the same block, is not allowed:

Example

```
var x = 2;       // Allowed
let x = 3;       // Not allowed

{
    var x = 4;   // Allowed
    let x = 5   // Not allowed
}
```

Redeclaring a **let** variable with **let**, in the same scope, or in the same block, is not allowed:

Example

```
let x = 2;       // Allowed
let x = 3;       // Not allowed

{
    let x = 4;   // Allowed
    let x = 5;   // Not allowed
}
```

Redeclaring a **let** variable with **var**, in the same scope, or in the same block, is not allowed:

Example

```
let x = 2;       // Allowed
var x = 3;       // Not allowed

{
    let x = 4;   // Allowed
    var x = 5;   // Not allowed
}
```

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

Example

```
let x = 2;       // Allowed

{
    let x = 3;   // Allowed
}

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

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

Example

```
let x = 2;        // Allowed

{
    let x = 3;    // Allowed
}

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

Hoisting

Variables defined with **var** are hoisted to the top. (Js Hoisting)

You can use a variable before it is declared:

Example

```
// you CAN use carName here
var carName;
```

Variables defined with **let** are not hoisted to the top.

Using a **let** variable before it is declared will result in a ReferenceError.

The variable is in a "temporal dead zone" from the start of the block until it is declared:

Example

```
// you can NOT use carName here
let carName;
```

JavaScript Const

Variables defined with **const** behave like **let** variables, except they cannot be reassigned:

Example

```
const PI = 3.141592653589793;
PI = 3.14;      // This will give an error
PI = PI + 10;   // This will also give an error
```

Assigned when Declared

JavaScript const variables must be assigned a value when they are declared:

Incorrect

```
const PI;
PI = 3.14159265359;
```

Correct

```
const PI = 3.14159265359;
```

IIFE's Function

An IIFE (Immediately Invoked Function Expression تعبير دالة مستحث فورا) ,
is a JavaScript function that runs as soon as it is defined يتم تشغيلها بمجرد تحديدها.

```
let add=function (a,b) {
    return a+b;
}

console.log(add(5,2));
```

Note: The IIFE's Function is not Hoisting

Functional programming (interview question)
= function Argument as function(x, y, function)

the function accepts an argument(variable) as function.

```
let add=function (a,b) {       //normal function 1
    return a+b;
}

let sub=function (a,b) {       //mormal function 2
    return a-b;
}

let multi_Func=function (x,y,fn) {     //function with function Argument
    return console.log(fn(x,y));
}

multi_Func(4,2,add);
multi_Func(8,4,sub);
```

```
6
4
```

Objected-oriented programming

JavaScript objects are containers for **named values** called properties or methods.

The values are written as **name:value** pairs (name and value separated by a colon).

Ex1:

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

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

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

<h2>JavaScript Objects</h2>

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

<script>
// Create an object:
var person = {firstName:"John", lastName:"Doe", age:50,
eyeColor:"blue"};

// Display some data from the object:
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>

</body>
</html>
```

JavaScript Objects

John is 50 years old.

Ex2:

```javascript
person = {
firstName:"John",
lastName:"Doe",
age:50,
eyeColor:"blue",
info: function () {
    return this.firstName + " " + this.lastName + this.age;
},
changeAge: function (x) {
    this.age=x;
    return this.age;
}


ole.log(typeof (person));
ole.log(person.eyeColor);
ole.log(person.info());
ole.log(person.changeAge(76));
ole.log(person.info());
```

```
object
blue
John Doe50
76
John Doe76
```

Note: it can to create new properties later without declaring in the object.

```
let PayperMonth = {

    July:1000,
    August:2000,
    Septemper:5000,

    }

let  calculateAvberage = function(obj) {

    let  sum = 0;
    //create new properties count later without declaring in the object
    obj.count=0;
    for(let props in obj ) {

        obj.count++;
        }
        return obj.count;
    }

console.log(PayperMonth);

console.log(calculateAvberage(PayperMonth));

console.log(PayperMonth);
```

```
▶ {July: 1000, August: 2000, Septemper: 5000}
4
▶ {July: 1000, August: 2000, Septemper: 5000, count: 4}
```

Object Inheritance

```
let man={
    bankAcount_$:1000,
    residenceCountry:'Germany ',
}

let jake=Object.create(man);
let daniel=Object.create(man);

jake.firstName='Ali';
jake.lastName='Alsaher';

jake.showAcount=function () {
    return `${this.firstName}`+"has"+`${this.bankAcount_$}`;
}

daniel.firstName='mhd';
daniel.lastName='Amin';
```

```
console.log(jake.showAcount());
```

```
Ali has 1000
```

or

```
let man={
    bankAcount_$:1000,
    residenceCountry:'Germany ',
}

let jake=Object.create(man);
let daniel=Object.create(man);

let mhd={
    inherite:Object.create(man),    // the mhd object has inherite object
    firstName:'Ali',
    lastName:'Alsaher',

    showAcount:function () {
    return `${this.firstName}`+" has "+`${this.inherite.bankAcount_$}`;
    }
}

console.log(mhd.showAcount());
```

this

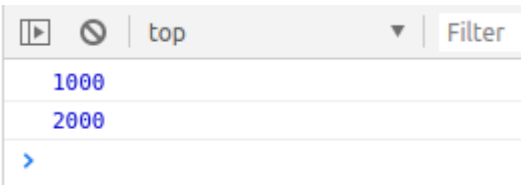
In a function definition, **this** refers to the "**caller**" of the function.

1 - This.numOfsales +=; ===> this=**owner** the function

2- if an object(ALI) calls the above via inherits ===> this=ALI
that means This.numOfsales=ALI. NumOfsales
so the compiler will replace “this” with “ALI”.

Ex:

```
let man={
    bankAcount_$:1000,
    residenceCountry:'Germany ',
    print: function(){
        return this.bankAcount_$;
    },
}

console.log(man.print()); // the owner call the function     this =man

let jake=Object.create(man);
jake.bankAcount_$=2000;
console.log(jake.print());  // the object call the function     this=jake
```



JavaScript Arrays

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

What is an Array?

An array is a special variable, which can hold more than one value at a time.

Creating an Array

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

Syntax:

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

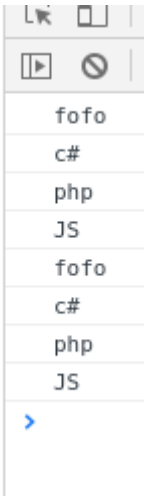
Example

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

the data type of array is : object

ex:

```
let test=['fofo',"c#","php","JS"];

// it is used to handel with HTML elements (forntend)
for (let i = 0; i < test.length; i++) {
  console.log(test[i]);
}

// it is used to handel with (backend)
for (const x of test) {
  console.log(x);
}
```

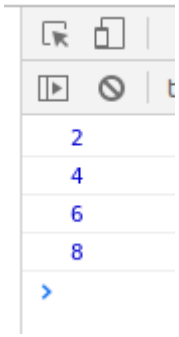


the first item is ; [0]
the last item is ; [length - 1]

```
console.log("the last element "+testResult[testResult.length-1]);
```

Array.includes() to compare

```
let all=[1,2,3,4,5,6,7,8,9];
let notLike=[1,3,5,7,9];

for (const x of all) {
  if (notLike.includes(x)) {
    continue;
  }
  console.log(x);
}
```



Array's methods

Pushing

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

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");       //  Adds a new element ("Kiwi") to fruits
```

Syntax

```
array.splice(index, howmany, item1, ....., itemX)
```

Parameter Values

Parameter	Description
index	Required. An integer that specifies at what position to add/remove items, Use negative values to specify the position from the end of the array
howmany	Optional. The number of items to be removed. If set to 0, no items will be removed
item1, ..., *itemX*	Optional. The new item(s) to be added to the array

Example

At position 2, add the new items, and remove 1 item:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 1, "Lemon", "Kiwi");
```

Try it Yourself »

Example

At position 2, remove 2 items:

```
var fruits = ["Banana", "Orange", "Apple", "Mango", "Kiwi"];
fruits.splice(2, 2);
```

Merging (Concatenating) Arrays

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

Example (Merging Two Arrays)

```
var myGirls = ["Cecilie", "Lone"];
var myBoys = ["Emil", "Tobias", "Linus"];
var myChildren = myGirls.concat(myBoys);     // Concatenates (joins) myGirls and
myBoys
```

Example (Merging Three Arrays)

```
var arr1 = ["Cecilie", "Lone"];
var arr2 = ["Emil", "Tobias", "Linus"];
var arr3 = ["Robin", "Morgan"];
var myChildren = arr1.concat(arr2, arr3);     // Concatenates arr1 with arr2 and
arr3
```

The concat() method can also take values as arguments:

Example (Merging an Array with Values)

```
var arr1 = ["Cecilie", "Lone"];
var myChildren = arr1.concat(["Emil", "Tobias", "Linus"]);
```

---

to reverse array **array.reverse()**
to sort array **array.sort()**

```
let amount=[5,5,2,6,1];
amount.reverse();
amount.sort();
```

or

```
let amount=[5,5,2,6,1];
amount.reverse().sort();
```

Converting an Array to String

array.join()

The join() method joins the elements of an array into a string, and returns the string.

The elements will be separated by a specified separator. The default separator is comma (,).

Syntax

```
array.join(separator)
```

Parameter Values

Parameter	Description
separator	Optional. The separator to be used. If omitted, the elements are separated with a comma

ex:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var energy = fruits.join();
```

Banana,Orange,Apple,Mango

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var energy = fruits.join(" and ");
```

Banana and Orange and Apple and Mango

Converting a String to an Array

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

Example

```
var txt = "a,b,c,d,e";   // String
txt.split(",");          // Split on commas
txt.split(" ");          // Split on spaces
txt.split("|");          // Split on pipe
```

```
function myFunction() {
    var str = "a,b,c,d,e,f";
    var arr = str.split(",");
    document.getElementById("demo").innerHTML = arr[0];
}
```

Try it

a

If the separator is omitted, the returned array will contain the whole string in index [0].

If the separator is "", the returned array will be an array of single characters:

Example

```
var txt = "Hello";       // String
txt.split("");           // Split in characters
```

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

<script>
var str = "Hello";
var arr = str.split("");
var text = "";
var i;
for (i = 0; i < arr.length; i++) {
    text += arr[i] + "<br>"
}
document.getElementById("demo").innerHTML = text;
</script>
```

H
e
l
l
o

String Variable is like array

```
// string interpolation by using the left-side quotes
let putVariables = `Strings ${can} also contain expression within the right quotes`;

// The string concatenation (+) operator
let anotherWayToConstruct =  'Strings ' + can + ' also be concatenated!';
```

```
anotherWayToConstruct.length // returns every characters string
anotherWayToConstruct[5] // returns the sixth character of an element like an array
anotherWayToConstruct.toUpperCase() // Converts all characters to upper case
anotherWayToConstruct.toLowerCase() // converts all characters to lower case.
anotherWayToConstruct.search('concat') // returns if the 'concat' is inside the anotherWayToConstruct
```

Nested Arrays

```
// An array can hold also arrays as elements.

let namesAndResults = [
  ['Nour', 92],
  ['Jake', 67]
]

// Now if you want to access Nour's array in general you know what you have to do.
console.Log(namesAndResults[0]);

// What if i want to access nour's test result and compare it with jake's test result?
// These are inside the nested arrays right? Check this out.

console.Log(namesAndResults[0][1] > namesAndResults[1][1]) // returns true. The element that is
// in outer array in position 0 returns an array. Of this inner array give me the element
// in index 1.
```

Ex:

```
let amount=[5,5,2,6,1];
let groceries=['chocolate','bananas','rice','beers','deodorant'];
let shoppingCart=[];

for (let i = 0; i < amount.length; i++) {
    shoppingCart.push([amount[i],groceries[i]]);

    console.log(`Please buy ${shoppingCart[i][0]}x ${shoppingCart[i][1]}`);
}
 //or to print

 for (const item of shoppingCart) {
    console.log(`or Please buy ${item[0]}x ${item[1]}`);
 }
```

```
Please buy 5x chocolate
Please buy 5x bananas
Please buy 2x rice
Please buy 6x beers
Please buy 1x deodorant
or Please buy 5x chocolate
or Please buy 5x bananas
or Please buy 2x rice
or Please buy 6x beers
or Please buy 1x deodorant
>
```

inheritance (Constructor+Prototype)

1-New Constructor منشئ جديد

Another way to create objects is by creating a constructor pattern and then fill the properties with the values as shown

```
let Person = function(firstName, lastName, age) {
  this.firstName = firstName;
  this.lastName = lastName;
  this.age = age;
}

let jake = new Person('Kostas', 'Diakogiannis', 30);
let jake = new Person('Mauro', 'Cifuentes', 46);
```

2-New Constructor+ Prototype inheritance

A better way to set an object to inherit the properties of another object is by the Object.setPrototypeOf.

Syntax

```
Object.setPrototypeOf(obj, prototype)
```

Parameters

`obj`

The object which is to have its prototype set.

`prototype`

The object's new prototype (an object or `null`).

Return value

The specified object.

```
let Student = function(firstName, lastName, age, email, nationality) {
  this.firstName = firstName;
  this.lastName = lastName;
  this.age = age;
  this.email = email;
  this.nationality = nationality;
}

let mauro = new Student('Mauro', 'Cifuentes', 45, 'some@example.com', 'Chilean');

let Latinos = {Language: 'spanish'};

Object.setPrototypeOf(mauro, Latinos);

console.log(mauro);
```

Date Objects

Creating Date Objects

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

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

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

new Date()

new Date() creates a new date object with the **current date and time**:

Example

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

new Date(*year, month, ...*)

new Date(*year, month, ...*) creates a new date object with a **specified date and time**.

7 numbers specify year, month, day, hour, minute, second, and millisecond (in that order):

Example

```
var d = new Date(2018, 11, 24, 10, 33, 30, 0);
```

new Date(*dateString*)

new Date(dateString) creates a new date object from a **date string**:

Example

```
var d = new Date("October 13, 2014 11:13:00");
```

```
new Date('10/13/2018')
new Date('Oct 12 2018')
new Date(2018,10,11)
```

JavaScript Stores Dates as Milliseconds

JavaScript stores dates as number of milliseconds since January 01, 1970, 00:00:00 UTC (Universal Time Coordinated).

> Zero time is January 01, 1970 00:00:00 UTC.

Now the time is: **1539245250282** milliseconds past January 01, 1970

01 January 1970 **plus** 100 000 000 000 milliseconds is approximately 03 March 1973:

Example

```
var d = new Date(100000000000);
```

Try it Yourself »

January 01 1970 **minus** 100 000 000 000 milliseconds is approximately October 31 1966:

Example

```
var d = new Date(-100000000000);
```

Try it Yourself »

Example

```
var d = new Date(86400000);
```

Try it Yourself »

One day (24 hours) is 86 400 000 milliseconds.

JavaScript Date Input

There are generally 3 types of JavaScript date input formats:

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

The ISO format follows a strict standard in JavaScript.

The other formats are not so well defined and might be browser specific.

Date Getters

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

Method	Description
getFullYear()	Get the **year** as a four digit number (yyyy)
getMonth()	Get the **month** as a number (0-11)
getDate()	Get the **day** as a number (1-31)
getHours()	Get the **hour** (0-23)
getMinutes()	Get the **minute** (0-59)
getSeconds()	Get the **second** (0-59)
getMilliseconds()	Get the **millisecond** (0-999)
getTime()	Get the time (milliseconds since January 1, 1970)
getDay()	Get the weekday as a number (0-6)
Date.now()	Get the time. ECMAScript 5.

Set Date Methods

Set Date methods are used for setting a part of a date:

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

Return
Return will break everything (function, while, for, if, ...) in any code.
That mean the last line will execute, is “Return”.

JavaScript Array find() Method

Or high order function for array

Syntax

```
array.find(function(currentValue, index, arr),thisValue)
```

Parameter Values

Parameter	Description
function(currentValue, index, arr)	Required. A function to be run for each element in the array. Function arguments: **Argument** / **Description** *currentValue* — Required. The value of the current element *index* — Optional. The array index of the current element *arr* — Optional. The array object the current element belongs to
thisValue	Optional. A value to be passed to the function to be used as its "this" value. If this parameter is empty, the value "undefined" will be passed as its "this" value

Ex:

Example

Get the value of the first element in the array that has a value of 18 or more:

```
var ages = [3, 10, 18, 20];

function checkAdult(age) {
    return age >= 18;
}

function myFunction() {
    document.getElementById("demo").innerHTML = ages.find(checkAdult);
}
```

18

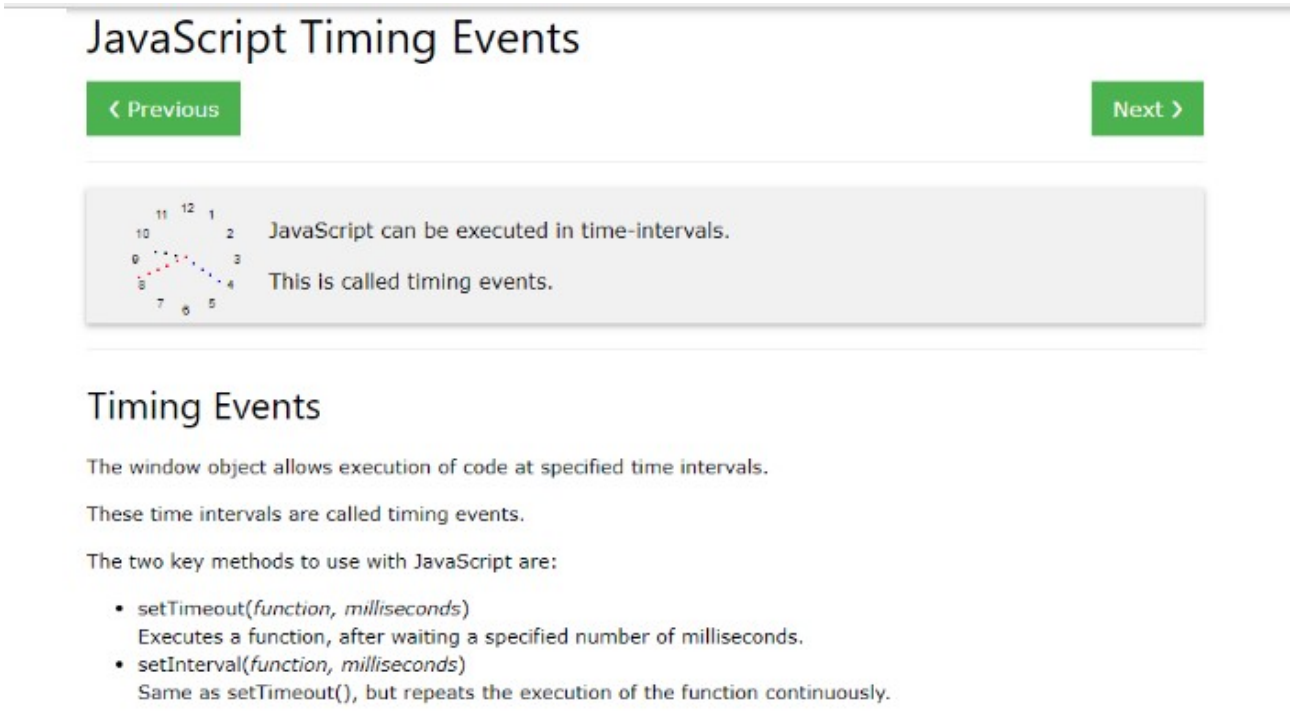
JavaScript Assignment Operators

JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Timing Events



Attention: Both timing events (setInterval and setTimeout) run asynchronously.

Setting an interval function

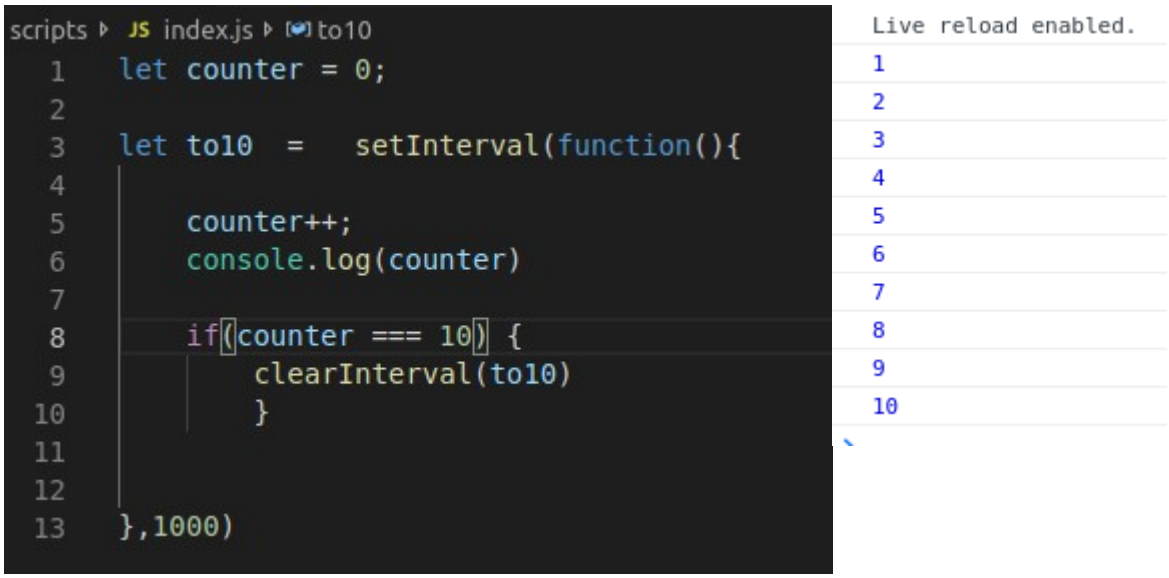
Setting an interval function

Set an interval function executes a specific piece of code periodically every when we define so. (every 5 sec, 5 mins etc).

The function accepts 2 arguments, the first argument is a function that is going to be executed, and the second argument is a number in milliseconds.

For example if we put as a first argument a function that prints 'Hello World' to the console, and 1000 milliseconds as the second argument, we will eventually print a 'hello word' message to the console every second.

In order to stop executing we call the clearInterval function that accepts as an argument the name of the interval that we want to stop.



Setting a timeout function

Setting a timeout function

Setting a timeout function follows exactly the same principles with the interval, but this time the body function is not executed periodically until is cleared, but waits instead, for a given amount of time and executes itself only once.

In this case if we put the same function as before that prints 'hello world' message to the console, and 3000 milliseconds, a 'hello world' message will appear to the console after 3 seconds but only once!

It is mainly used for popping up windows after 5 seconds (giving first some time to the new user to get himself/herself used to the content).

```
10
11  let printHelloAfterTime = setTimeout(function() {
12    console.log('Hello Timeout!!!');
13  }, 5000)
14
```

Selecting DOM elements

Generally speaking there are many different ways to select elements from an HTML document. We use these functions:

document.body // returns the whole body of the html document

document.getElementById('unique') // returns the element with the id set to 'unique'

document.getElementsByTagName('H3') // returns an array with all h3 elements

document.getElementsByClassName('bigSize') // returns an array with with this class elems

document.querySelector('p.intro') // CSS selector type. Returns the first element match

document.querySelectorAll('div img') // returns an array with images inside a div

HTML DOM querySelector() Method

Definition and Usage

The querySelector() method returns the first element that matches a specified *CSS selector(s)* in the document.

Note: The querySelector() method only returns the first element that matches the specified selectors. To return all the matches, use the querySelectorAll() method instead.

If the selector matches an ID in document that is used several times (Note that an "id" should be unique within a page and should not be used more than once), it returns the first matching element.

Syntax

```
document.querySelector(CSS selectors)
```

Parameter Values

Parameter	Type	Description
CSS selectors	String	Required. Specifies one or more CSS selectors to match the element. These are used to select HTML elements based on their id, classes, types, attributes, values of attributes, etc. For multiple selectors, separate each selector with a comma. The returned element depends on which element that is first found in the document (See "More Examples"). **Tip:** For a list of all CSS Selectors, look at our CSS Selectors Reference.

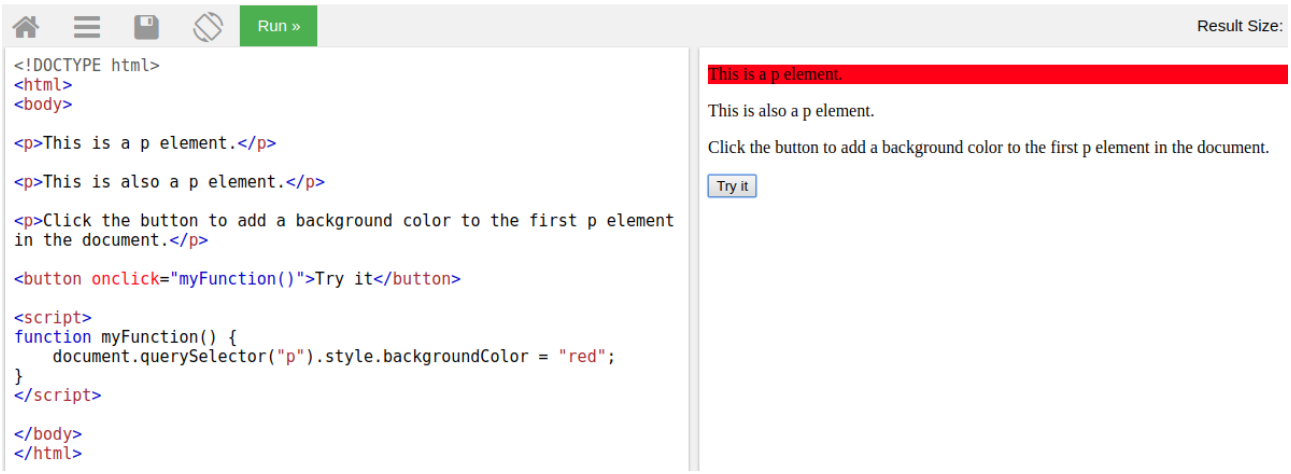
More Examples

Example

Get the first <p> element in the document:

```
document.querySelector("p");
```

Try it Yourself »

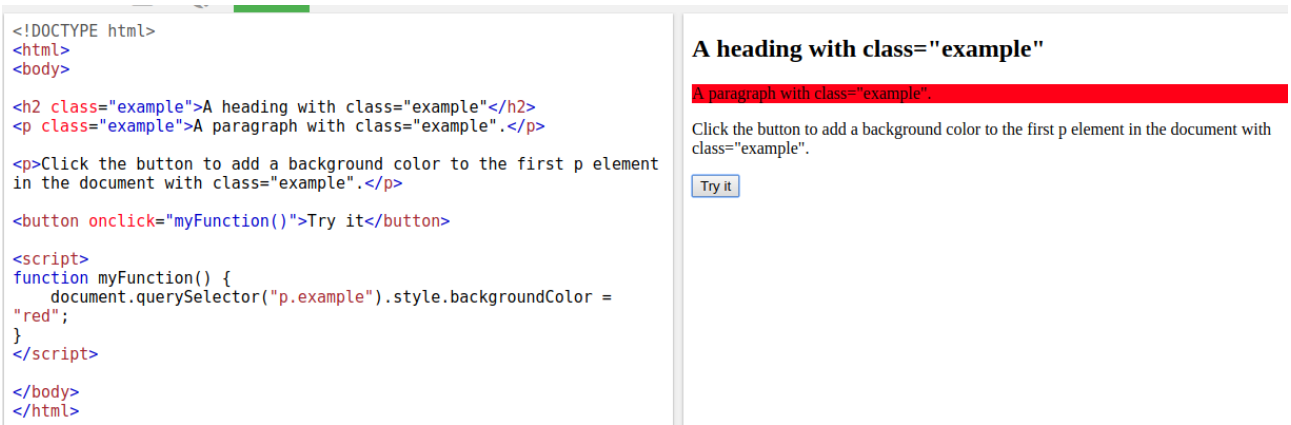


Example

Get the first <p> element in the document with class="example":

```
document.querySelector("p.example");
```

Try it Yourself »



Example

Change the text of an element with id="demo":

```
document.querySelector("#demo").innerHTML = "Hello World!";
```

Try it Yourself »

Example

Get the first <p> element in the document where the parent is a element.

```
document.querySelector("div > p");
```

Try it Yourself »

Run » Result Size: 635 x 46

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
    border: 1px solid black;
    margin-bottom: 10px;
}
</style>
</head>
<body>

<div>
  <h3>H3 element</h3>
  <p>I am a p element, my parent is a div element.</p>
</div>

<div>
  <h3>H3 element</h3>
  <p>I am a p element, my parent is also a div element.</p>
</div>

<p>Click the button to change the background color of the first p
element in the document where the parent is a div element.</p>
```

H3 element

I am a p element, my parent is a div element.

H3 element

I am a p element, my parent is also a div element.

Click the button to change the background color of the first p element in the document where the parent is a div element.

Try it

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

<script>
function myFunction() {
    var x = document.querySelector("div > p");
    x.style.backgroundColor = "red";
}
</script>

</body>
</html>
```

Example

Get the first <a> element in the document that has a "target" attribute:

```
document.querySelector("a[target]");
```

Try it Yourself »

Run » Result Size: 635

```
<head>
<style>
a[target] {
    background-color: yellow;
}
</style>
</head>
<body>

<p>The CSS selector a[target] makes sure that all links with a
target attribute gets a yellow background:</p>

<a href="https://www.w3schools.com">w3schools.com</a>
<a href="http://www.disney.com" target="_blank">disney.com</a>
<a href="http://www.wikipedia.org" target="_top">wikipedia.org</a>

<p>Click the button to add a red border to the first link that has
a target attribute.</p>

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

<script>
function myFunction() {
    document.querySelector("a[target]").style.border = "10px solid
red";
function myFunction() {
    document.querySelector("a[alt]").style.border = "10px solid
red";
}
</script>

</body>
```

The CSS selector a[target] makes sure that all links with a target attribute gets a yellow background:

w3schools.com disney.com wikipedia.org

Click the button to add a red border to the first link that has a target attribute.

Try it

Example

This example demonstrates how multiple selectors work.

Assume that you have two elements: a <h2> and a <h3> element.

The following code will add a background color to the first <h2> element in the document:

```
<h2>A h2 element</h2>
<h3>A h3 element</h3>

document.querySelector("h2, h3").style.backgroundColor = "red";
```

Try it Yourself »

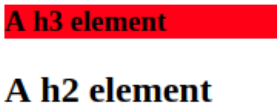
However, if the <h3> element was placed before the <h2> element in the document. The <h3> element is the one that will get the red background color.

```
<h3>A h3 element</h3>
<h2>A h2 element</h2>

document.querySelector("h2, h3").style.backgroundColor = "red";
```

Try it Yourself »

Run »

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

<h3>A h3 element</h3>
<h2>A h2 element</h2>

<script>
document.querySelector("h2, h3").style.backgroundColor = "red";
</script>

</body>
</html>
```



A h2 element

HTML DOM classList Property

Definition and Usage

The classList property returns the class name(s) of an element, as a DOMTokenList object.

This property is useful to add, remove and toggle CSS classes on an element.

The classList property is read-only, however, you can modify it by using the add() and remove() methods.

Cross-browser solution: The classList property is not supported in IE9 and earlier. However, you can use the className property or regular expressions for a cross-browser solution (see "More Examples" on the bottom of this page).

Example

Add the "mystyle" class to a element:

Methods

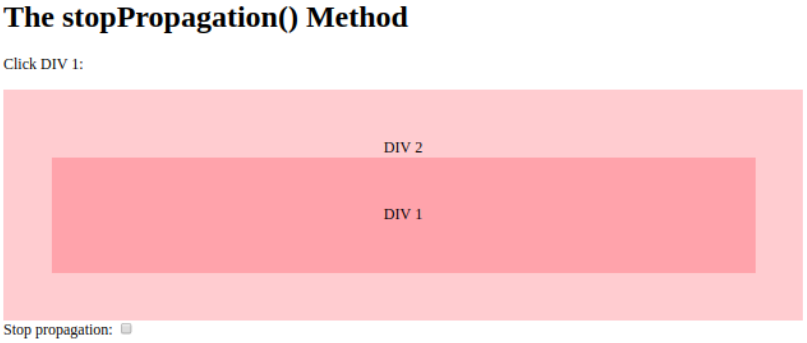
Method	Description
add(*class1*, *class2*, ...)	Adds one or more class names to an element. If the specified class already exist, the class will not be added
contains(*class*)	Returns a Boolean value, indicating whether an element has the specified class name. Possible values: • true - the element contains the specified class name • false - the element does not contain the specified class name
item(*index*)	Returns the class name with a specified index number from an element. Index starts at 0. Returns *null* if the index is out of range
remove(*class1*, *class2*, ...)	Removes one or more class names from an element. **Note:** Removing a class that does not exist, does NOT throw an error
toggle(*class*, true|false)	Toggles between a class name for an element. The first parameter removes the specified class from an element, and returns false. If the class does not exist, it is added to the element, and the return value is true. The optional second parameter is a Boolean value that forces the class to be added or removed, regardless of whether or not it already existed. For example: Remove a class: *element*.classList.toggle("classToRemove", false); Add a class: *element*.classList.toggle("classToAdd", true); **Note:** The second parameter is not supported in Internet Explorer or Opera 12 and earlier.

stopPropagation() نشر Event Method

Propagation means bubbling up to parent elements *or* capturing down to child elements. يعني الانتشار فقاعات تصل إلى العناصر الرئيسية أو الالتقاط إلى العناصر الفرعية.
The `stopPropagation()` method prevents propagation of the same event from being called.

Or
the element that has `stopPropagation()` , it will not respond to any `addEventListener()` from the parent.

Ex:



https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_event_stoppropagation

HTML DOM classList Property

Definition and Usage

The classList property returns the class name(s) of an element, as a DOMTokenList object.

This property is useful to add, remove and toggle CSS classes on an element.

The classList property is read-only, however, you can modify it by using the add() and remove() methods.

Syntax

```
element.classList
```

Example

Add the "mystyle" class to a element:

```
document.getElementById("myDIV").classList.add("mystyle");
```

Methods

Method	Description
add(*class1, class2, ...*)	Adds one or more class names to an element. If the specified class already exist, the class will not be added
contains(*class*)	Returns a Boolean value, indicating whether an element has the specified class name. Possible values: • true - the element contains the specified class name • false - the element does not contain the specified class name
item(*index*)	Returns the class name with a specified index number from an element. Index starts at 0. Returns *null* if the index is out of range
remove(*class1, class2, ...*)	Removes one or more class names from an element. **Note:** Removing a class that does not exist, does NOT throw an error
toggle(*class*, true|false)	Toggles between a class name for an element.

Technical Details

Return Value:	A DOMTokenList, containing a list of the class name(s) of an element

More Examples

https://www.w3schools.com/jsref/prop_element_classlist.asp

preventDefault() Event Method

Definition and Usage

The preventDefault() method cancels the event if it is cancelable, meaning that the default action that belongs to the event will not occur.

For example, this can be useful when:

- Clicking on a "Submit" button, prevent it from submitting a form
- Clicking on a link, prevent the link from following the URL

Note: Not all events are cancelable. Use the cancelable property to find out if an event is cancelable.

Note: The preventDefault() method does not prevent further propagation of an event through the DOM. Use the

Syntax

```
event.preventDefault()
```

Parameters

None

Technical Details

Return Value:	No return value
DOM Version:	DOM Level 2 Events

Example

Prevent a link from opening the URL:

```
document.getElementById("myAnchor").addEventListener("click", function(event){
    event.preventDefault()
});
```

Window scrollBy() Method

Definition and Usage

The scrollBy() method scrolls the document by the specified number of pixels.

Note: For this method to work, the visible property of the window's scrollbar must be set to true!

Syntax

```
window.scrollBy(xnum, ynum)
```

Parameter Values

Parameter	Type
xnum	Number
ynum	Number

Example

Scroll the document by 100px horizontally:

```
window.scrollBy(100, 0); // Scroll 100px to the right
```

More Examples

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_win_scrollby3

Window pageXOffset and pageYOffset Properties

Where is my scroll pixels.

```
if(window.pageYOffset){

}
```

```
if(window.pageXOffset){

}
```

Syntax

```
window.pageXOffset
window.pageYOffset
```

Definition and Usage

The pageXOffset and pageYOffset properties returns the pixels the current document has been scrolled from the upper left corner of the window, horizontally and vertically.

The pageXOffset and pageYOffset properties are equal to the scrollX and scrollY properties.

These properties are read-only.

Element offsetTop Property

Syntax

Return the top offset position:

```
object.offsetTop
```

Click the button to get offsetTop for the test div.

Try it

offsetTop is:

Definition and Usage

The offsetTop property returns the top position (in pixels) relative to the top of the offsetParent element.

The returned value includes:

- the top position, and margin of the element
- the top padding, scrollbar and border of the offsetParent element

Note: The offsetParent element is the nearest ancestor that has a position other than static.

Tip: To return the left position of an element, use the offsetLeft property.

```
if (window.pageYOffset >= posters[i].offsetTop
```

Ex 69_scrollby :

JS scrollTo.js ▸ ...

```js
let titles = document.querySelectorAll('nav > li');
let posters = document.querySelectorAll('section');
let nav = document.querySelector('nav');

let heightToExecute = nav.clientHeight + 250;

window.addEventListener('scroll', function(ev) {

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

    if (window.pageYOffset >= posters[i].offsetTop - heightToExecute && window.pageYOffset
      for (let j = 0; j < titles.length; j++) {
        if (titles[j].classList.contains('changeColor')) {
          titles[j].classList.remove('changeColor');
          break;
        }
      }
      titles[i].classList.add('changeColor');
      posters[i].style.opacity = '1';
    }

  }

})
```

HTML DOM clientHeight Property

Definition and Usage

The clientHeight property returns the viewable height of an element in pixels, including padding, but not the border, scrollbar or margin.

Syntax

```
element.clientHeight
```

```
let nav = document.querySelector('nav');

let heightToExecute = nav.clientHeight + 250;
```

MouseEvent clientX Property

Definition and Usage

The clientX property returns the horizontal coordinate (according to the client area) of the mouse pointer when a mouse event was triggered.

The client area is the current window.

Tip: To get the vertical coordinate (according to the client area) of the mouse pointer, use the clientY property.

Note: This property is read-only.

Syntax

```
event.clientX
```

Example

Output the coordinates of the mouse pointer while the mouse pointer moves over an element:

```
var x = event.clientX;
var y = event.clientY;
var coor = "X coords: " + x + ", Y coords: " + y;
document.getElementById("demo").innerHTML = coor;
```

```
<div onmousemove="showCoords(event)" onmouseout="clearCoor()">
</div>

<p>Mouse over the rectangle above to get the horizontal and
vertical coordinates of your mouse pointer.</p>

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

<script>
function showCoords(event) {
    var x = event.clientX;
    var y = event.clientY;
    var coor = "X coords: " + x + ", Y coords: " + y;
    document.getElementById("demo").innerHTML = coor;
}

function clearCoor() {
    document.getElementById("demo").innerHTML = "";
}
</script>

</body>
</html>
```

Mouse over the rectangle above to get the horizontal and vertical coordinates of your mouse pointer.

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_event_mouse_clientxy2

currentTarget Event Property

```
let link = document.querySelector("a");
let select = document.querySelector('select');

select.addEventListener('change', function(ev){
   link.href=`https://www.${ev.currentTarget.value}.com`; // ev.currentTarget = select

   link.innerHTML=`Go to <b>${select.value}</b>`;

});
```

CSS z-index Property

Definition and Usage

The `z-index` property specifies the stack order of an element.

An element with greater stack order is always in front of an element with a lower stack order.

Note: `z-index` only works on positioned elements (position:absolute, position:relative, or position:fixed).

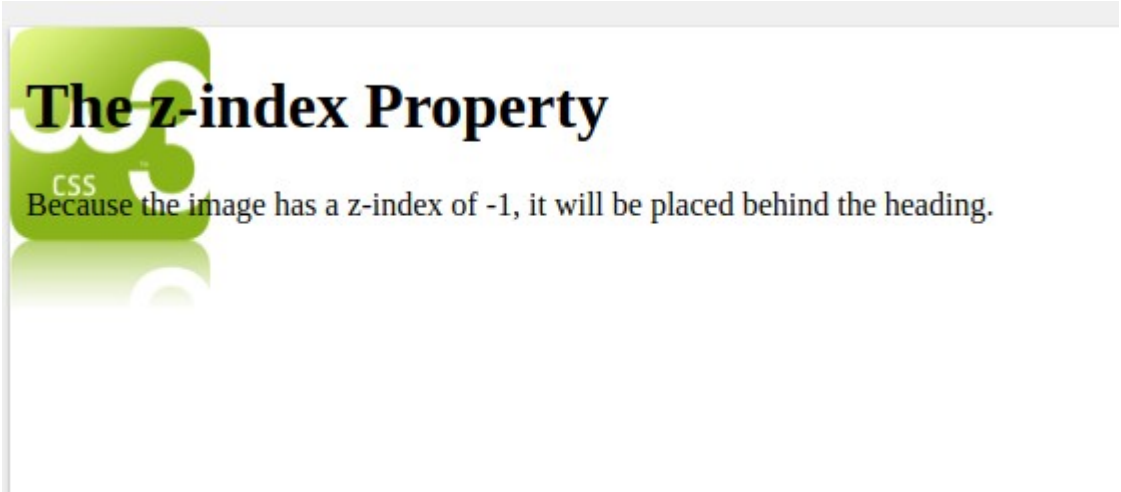
Note: `z-index` only works on positioned elements (position:absolute, position:relative, or position:fixed).

Note: `z-index` only works on positioned elements (position:absolute, position:relative, or position:fixed).

Example

Set the z-index for an image:

```html
<!DOCTYPE html>
<html>
<head>
<style>
img {
    position: absolute;
    left: 0px;
    top: 0px;
    z-index: -1;
}
</style>
</head>

<body>
<h1>The z-index Property</h1>

<img src="w3css.gif" width="100" height="140">

<p>Because the image has a z-index of -1, it will be placed behind the heading.</p>

</body>
</html>
```



WHEN NO POSITION, **z-index** will not effect anything to image.

```
<!DOCTYPE html>
<html>
<head>
<style>
img {
    //position: absolute;
    left: 0px;
    top: 0px;
    z-index: -1;
}
</style>
</head>

<body>
<h1>The z-index Property</h1>

<img src="w3css.gif" width="100" height="140">

<p>Because the image has a z-index of -1, it will be placed behind
the heading.</p>

</body>
</html>
```

The z-index Property

Because the image has a z-index of -1, it will be placed behind the heading.

document.addEventListener ('DOMContentLoaded', function (ev) {

your js code

})

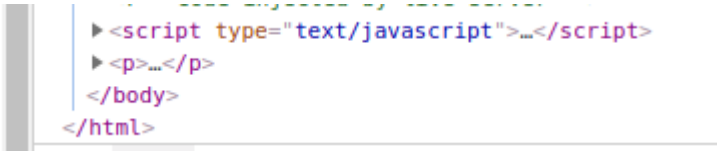
-The HTML DOM (Document Object Model) = to access all Html elements after loaded.

-When a web page is loaded, the browser creates a Document Object Model of the page.

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

domContentLoaded is the point when both the DOM is ready and there are no stylesheets that are blocking JavaScript execution.

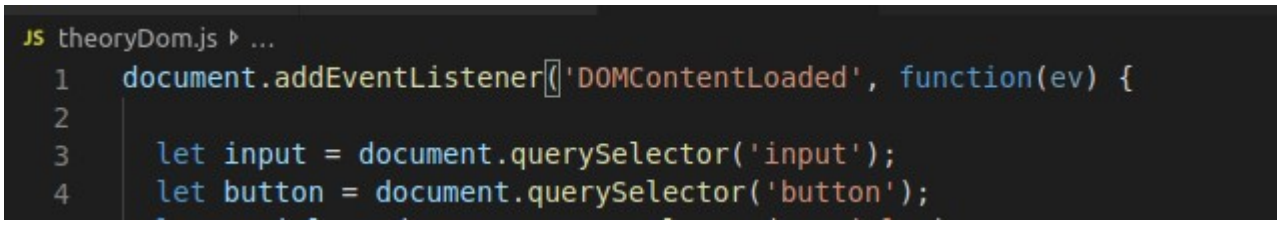
- (Scenario 1) ; when to add a new element on the fly by js with the document method like **document.createElement('P'),**
so the new element will be after JavaScript line (big problem).



- (Scenario 2 : to solve the problem of Scenario 1) ;

To move the js line to head, so the Html will start with js code , so the js code will call the html DOM eventListeners but the html DOM is not loaded yet. To solve this , it will set "**document.addEventListener ('DOMContentLoaded', function (ev) {"** in the first line in js, so now the the js code will compile after the HTML DOM is loaded and ready.

So finally now : the element that is added on the fly by js with the document method like **document.createElement('P'),** will load first then js.



Input text box

Write your thoughts!

```
<input type="text" maxlength="20" minlength="5" placeholder="Write your thoughts!"/>
```

Input password box

••••••••••

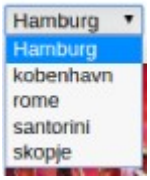
```
<input type="password" maxlength="20" minlength="5" placeholder="Write your password!"/>
```

Input checkbox

Hide what you write

```
<input type="checkbox" />Hide what you write
```

Select bar




```
<select>
  <option>Hamburg</option>
  <option>kobenhavn</option>
  <option>rome</option>
  <option>santorini</option>
  <option>skopje</option>

</select>
```

textarea

Put your commment here!

```
<textarea placeholder="Put your commment here!"></textarea><br />
```

keyup event

The keyup event is fired when a key is released.

يتم تشغيل حدث keyup بعد كتابة حرف و ترك زر الحرف.

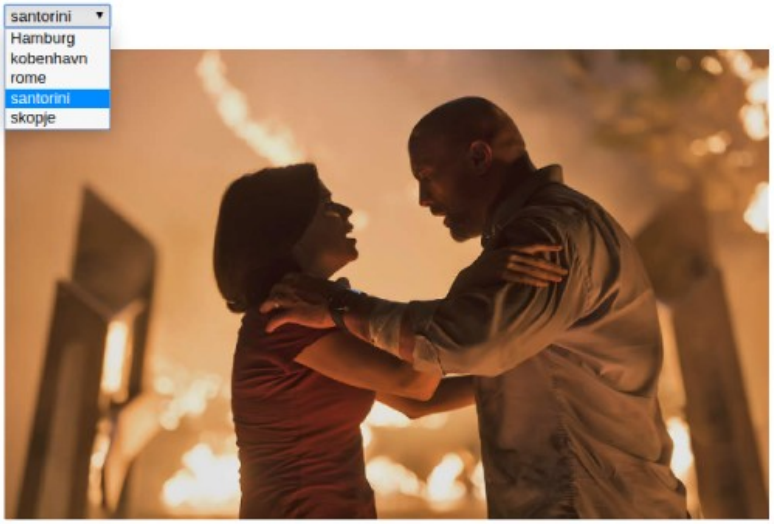
```
input.addEventListener('keyup', function(ev) {
  let val = input.value;
  val ? article.innerText = val : article.innerText = input.placeholder
})
```

mohammed
mohammed

Select to change img

```
let image = document.querySelector('img');
let selectFields = document.querySelectorAll('select');
```

```
selectFields[0].addEventListener('change', function(ev) {
  let selectedValue = selectFields[0].value;
  image.src = `../images/${selectedValue}.jpg`;
})
```



To add a new element on the fly by JavaScript

1- let var = document.createElement('P'); // musr inside ‘ ‘ a capital Letter
2- document.body.appendChild(var); // or element.appendChild(var);

```
button.addEventListener('click', function(ev) {
  let text = textarea.value;
  let newParagraph = document.createElement('P');
  let spanCloseParagaph = document.createElement('SPAN');
  newParagraph.innerText=text;
  spanCloseParagaph.innerHTML = '&times;';

  newParagraph.appendChild(spanCloseParagaph);
  document.body.appendChild(newParagraph);
  textarea.value = '';
```

To remove a element on the fly by JavaScript

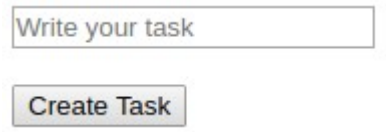
Element.removeChild(elementToRemove);

ex:


```
  spanInsideParagaph.addEventListener('click', function (ev) {
  let paragraphToBeRemoved = ev.currentTarget.parentElement;
  // ev.currentTarget=spanInsideParagaph
  paragraphToBeRemoved.parentElement.removeChild(paragraphToBeRemoved);
  // paragraphToBeRemoved.parentElement.removeChild()=document.body.removeChild();
})
```

submit event , it is only with form element

It mean when I press enter in input box, the event will trigger.
It muss to set `<button type="submit">` to button, in order to also the button triggers the submit event of form.




```
<form>
  <input type="text" placeholder="Write your task" minlength="2"/><br /><br />
  <button type="submit">Create Task</button>
</form>
```

```
form.addEventListener('submit', function(event) {
  let text = input.value;
  let paragraph = document.createElement('P');
  paragraph.innerText = text;
  section.appendChild(paragraph);
  input.value = '';
})
```

Write your task

Create Task

45454X

Window localStorage Property

Definition and Usage

The localStorage and sessionStorage properties allow to save key/value pairs in a web browser.

The localStorage object stores data with no expiration date. The data will not be deleted when the browser is closed, and will be available the next day, week, or year.

The localStorage property is read-only.

Tip: Also look at the sessionStorage property which stores data for one session (data is lost when the browser tab is closed).

Syntax

```
window.localStorage
```

Syntax for SAVING data to localStorage:

```
localStorage.setItem("key", "value");
```

Syntax for READING data from localStorage:

```
var lastname = localStorage.getItem("key");
```

Syntax for REMOVING data from localStorage:

```
localStorage.removeItem("key");
```

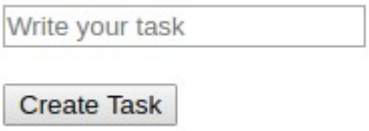
Technical Details

Return Value:	A Storage object

The line of example :

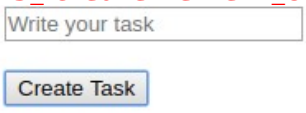
https://www.w3schools.com/jsref/prop_win_localstorage.asp

how to use it :




```
let counterID = localStorage.length;

// Check if local storage has something and render them before everything!
if (localStorage.length > 0) {

  for (let task in localStorage) {

    // to get only the storge value, because the localstroage has stotge proto also
    if (typeof localStorage[task] !== 'string') {
      break;
    }

    //your here code to get values from  localStorage

  }
}


form.addEventListener('submit', function(event) {
  counterID++;

  //your here code to save values to  localStorage
  let text = input.value;
  localStorage.setItem(`task-${counterID}`, text);


})
```

ex:
73_ createElement_on_the_fly with LocalStorg



xxcxX

xcxcX

nmnX

25785X

hnhjhX

jjhjX

The <fieldset> tag draws a box around the related elements.

<fieldset>
<legend>my title</legend>

..... your code

</fieldset>


```
<form>

 <fieldset>
  <legend>Personalia:</legend>

  Name: <input type="text"><br>
  Email: <input type="text"><br>
  Date of birth: <input type="text">

 </fieldset>

</form>
```

JavaScript Regular Expressions

A regular expression is a sequence of characters that forms a search pattern.

The search pattern can be used for text search and text replace operations.

What Is a Regular Expression?

A regular expression is a sequence of characters that forms a **search pattern**.

When you search for data in a text, you can use this search pattern to describe what you are searching for.

A regular expression can be a single character, or a more complicated pattern.

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

Syntax

```
/pattern/modifiers;
```

To check regex pattern: **www.regex101.com**

Example

```
var patt = /w3schools/i;
```

Example explained:

/w3schools/i is a regular expression.

w3schools is a pattern (to be used in a search).

i is a modifier (modifies the search to be case-insensitive).

n$	Matches any string with *n* at the end of it
^n	Matches any string with *n* at the beginning of it

(x|y)	Find any of the alternatives specified

\s	Find a whitespace character

n?	Matches any string that contains zero or one occurrences of *n*

n+	Matches any string that contains at least one *n*

Brackets

Brackets are used to find a range of characters:

Expression	Description
[abc]	Find any character between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find any character between the brackets (any digit)
[^0-9]	Find any character NOT between the brackets (any non-digit)
(x|y)	Find any of the alternatives specified

n{X}	Matches any string that contains a sequence of *X* *n*'s

n{X,Y}	Matches any string that contains a sequence of X to Y *n*'s

\w	Find a word character

\d	Find a digit

.	Find a single character, except newline or line terminator

. is everything(word,numbers,sympols)

n+	Matches any string that contains at least one *n*

how to write / : slash + symbol = \ /

\b	Find a match at the beginning/end of a word

Self-Invoking Functions Invoking:استدعاء it used for security

It runs only one time and not allow to call it.

Function expressions can be made "self-invoking".

A self-invoking expression is invoked (started) automatically, without being called.

Function expressions will execute automatically if the expression is followed by ().

You cannot self-invoke a function declaration.

You have to add parentheses around the function to indicate that it is a function expression:

Example

```
(function () {
    var x = "Hello!!";      // I will invoke myself
})();
```

Try it Yourself »

The function above is actually an **anonymous self-invoking function** (function without name).

```
// Testing code
const checkAddition = function() {
  const expectedResult = 25;
  const actualResult = addition(2, 23);

  if (expectedResult == actualResult) {
    console.log('Your test passed!');
  } else {
    console.log('Your test has failed! This is the checkAdditon function!');
  }
}();

const additionGoneWrong = (() => {
  const expectedResult = 'Sorry, the operation is not possible!';
  const actualResult = addition(false, '23');

  if (expectedResult == actualResult) {
    console.log('Your test passed!');
  } else {
    console.log('Your test has failed! This is the additionGoneWrong function');
  }
})();
```

ECMAScript 6 - ECMAScript 2015

What is ECMAScript 6?

ECMAScript 6 is also known as ES6 and ECMAScript 2015.

Some people like to call it JavaScript 6.

This chapter will introduce some of the new features in ES6.

- JavaScript let
- JavaScript const
- Exponentiation (**)
- Default parameter values
- Array.find()
- Array.findIndex()

https://www.w3schools.com/js/js_es6.asp

Default Values in functions' parameters

Default Values

```
7   // Ecma Script 6
8
9   // Default values on function parameters
10  let printName = function(name = 'jake') {
11    console.log(name);
12  }
13
14  printName('Mauro'); // Prints Mauro
15  printName('Daniel'); // Prints Daniel
16  printName(); // Prints jake
17
```

Arrow Functions

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**.

Example

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

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

Try it Yourself »

```
let root= a => a**(1/2);

console.log(root(9));
```

```
let AmultiB = (a,b) => { if(a%b==0)return true; else return false;};
console.log(AmultiB(9,3));
console.log(AmultiB(9,4));
```

```
let button= document.querySelector('button');
button.addEventListener("click", ev => {
   console.log(ev.type);
})
```

Set.prototype Constructor (unique values of any type)

The **`Set`** object lets you store unique values of any type, whether primitive values or object references.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set

JavaScript Demo: Set.prototype Constructor

```
let set1 = new Set([1, 2, 3, 4, 5, 5, 2]);

console.log(set1.has(1));
// expected output: true

console.log(set1.has(5));
// expected output: true

console.log(set1.has(6));
// expected output: false

```

Run ›

Reset

```
> true
> true
> false
```

Syntax

```
new Set([iterable]);
```

Parameters

`iterable`

If an iterable object is passed, all of its elements will be added to the new `Set`. If you don't specify this parameter, or its value is `null`, the new `Set` is empty.

Return value

A new `Set` object.

Set

Properties

- `Set.prototype`
- `Set.prototype.size`
- `get Set[@@species]`

Methods

- `Set.prototype.add()`
- `Set.prototype.clear()`
- `Set.prototype.delete()`
- `Set.prototype.entries()`
- `Set.prototype.forEach()`
- `Set.prototype.has()`
- `Set.prototype.values()`
- `Set.prototype[@@iterator]()`

```
// Find Unique Names
// Given that you have
let names = [
   'Mauro',
   'Eugen',
   'Mauro',
   'Meir',
   'Eugen',
   'Jens',
   'Jens',
   'Jake',
   'Mohammed',
   'Mauro',
   'Mohammed',
   'Marcelo',
   'Sue',
   'Murhaf',
   'Jens'
 ];
```

```
let uniqueNames= new Set(names);
console.log(uniqueNames);
```

```
▼Set(9) {"Mauro", "Eugen", "Meir", "Jens", "Jake", …}
   size: (...)
  ▶__proto__: Set
  ▼[[Entries]]: Array(9)
    ▶0: "Mauro"
    ▶1: "Eugen"
    ▶2: "Meir"
    ▶3: "Jens"
    ▶4: "Jake"
    ▶5: "Mohammed"
    ▶6: "Marcelo"
    ▶7: "Sue"
    ▶8: "Murhaf"
     length: 9
```

Class based Inheritance

JavaScript classes, introduced in ECMAScript 2015, are primarily syntactical sugar over JavaScript's existing prototype-based inheritance

```
class Human {
    constructor (){
        this.species="human";
        this.eyes=2;
        this.legs=2;
        this.hands=2;
    }
}
```

```
class SouthAmer extends Human{
    constructor(dance,drink,lang){
      super();
      this.canDance=dance;
      this.loveFottball=true;
      this.drink=drink;
      this.lang=lang;
    }
}
```

```
class Brazilians extends SouthAmer{
    constructor(){
      super(['Samba','salsa'],"piscola",'portuguese');
      this.music=true;
    }
}
```

```
class Chileans extends SouthAmer{
        constructor(hobbies,profession){
        super('everything',"vodca",'English');
        this.hobbies=hobbies;
        this.profession=profession;
        }
}
```

```
let Mauro= new Chileans(['fishing','Music'],'sprot');
let bobo= new Chileans();
```

console.table() Method

Definition and Usage

The console.table() method writes a table in the console view.

```
console.table(Mauro);
```

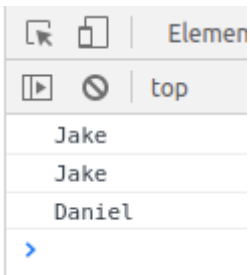
(index)	Value
species	"human"
eyes	2
legs	2
hands	2
canDance	"everything"
loveFottball	true
drink	"vodca"
lang	"English"
hobbies	
profession	"sprot"

▶ Chileans

Asynchronous Code

it will execute alone in the end of the js file, it will not effect on the main code after it.

```
let name = `Jake`;

setTimeout(function() {
    name = `Daniel`;
    console.log(name);
},0)

console.log(name);
console.log(name);
```



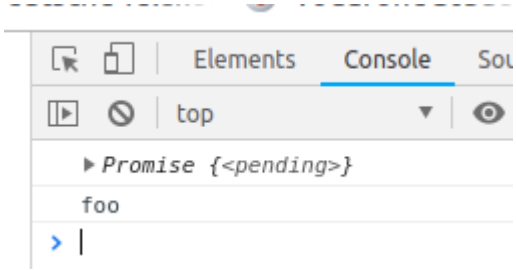
Promises , to create a Asynchronous function

The **`Promise`** object represents the eventual completion (or failure) of an asynchronous operation, and its resulting value.

يمثل الكائن Promise الانتهاء (أو الفشل) في نهاية المطاف لعملية غير متزامنة ، والقيمة الناتجة عنها.

promise object means the 2 option:
completion : ==> resolve ==> .then
failur : ==> reject ==> .catch

```
var promise1 = new Promise(function(resolve, reject) {   // asynchronous operation
    setTimeout(function() {
      resolve('foo');
    }, 300);
  });

  promise1.then(function(value) {                        // asynchronous operation
    console.log(value);
    // expected output: "foo"
  });

  console.log(promise1);
  // expected output: [object Promise]
```



tips to create asynchronous function:

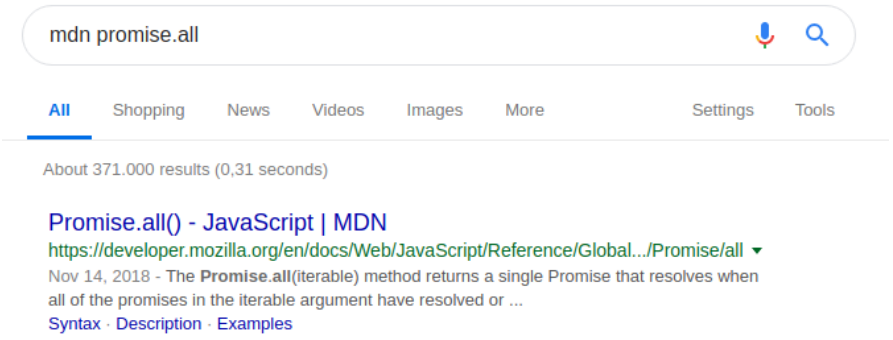
1. create a function with argument(s)
2. create a promise object inside that function
3. resolve (value), reject (message)
4. return promise object to that function

```
let firstStudent = 'Meir';

let changeStudent = name => {  // 1. create a function with argument(s)
  let promisedStudent = new Promise((resolve, reject) => {  //  2. create a promise object inside that function
    if (name.length === 4) {      // 3. reject (message)
      reject('Too short name!');
    }
    resolve(name);              // 3. resolve (value),
  });

  return promisedStudent;       // 4. return promise object to that function
}
```

```
let printAnotherStudent = newStudent => {
  let nameLower = newStudent.toUpperCase();
  firstStudent = nameLower;
  console.log(firstStudent);
}
```

```
console.log(firstStudent);

changeStudent('Jens')
  .then(printAnotherStudent)
  .catch(nostradamus => {
    console.warn(nostradamus);
  })

console.log(firstStudent);
```

advance of promise: .all .race



https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/all

f you want to work with a library that deals with promises. Bit advanced to understand though: http://bluebirdjs.com/docs/features.html

fetch

it is ready function to return promise object (.then , .catch).

```
// Use the fetch api to brng data

fetch('../images/home.jpg')
  .then(response => {
    console.log(response.url);
    let image = document.createElement('IMG');
    image.src = response.url;
    document.body.appendChild(image);
  })
  .catch(errMessage => {
    let createParagraph = document.createElement('P');
    createParagraph.innerText = `Something went terribly wrong!`;
    document.body.appendChild(createParagraph);
  })
```

status:
200 success
401 client-error Authorization failed
404 client-error page not found
501 server down

JSON

It is encapsulation the data in “ “ then send , so recive the data then decapsulation

> JSON: **J**ava**S**cript **O**bject **N**otation.
>
> JSON is a syntax for storing and exchanging data.
>
> JSON is text, written with JavaScript object notation.

Exchanging Data

When exchanging data between a browser and a server, the data can only be text.

JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server.

We can also convert any JSON received from the server into JavaScript objects.

This way we can work with the data as JavaScript objects, with no complicated parsing and translations.

Try to Catch

JavaScript Errors - Throw and Try to Catch

❮ Previous Next ❯

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

The **catch** statement lets you handle the error.

The **throw** statement lets you create custom errors.

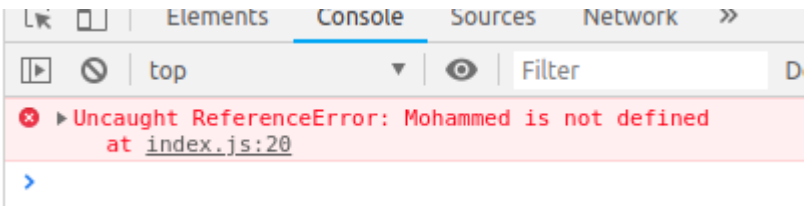
The **finally** statement lets you execute code, after try and catch, regardless of the result.

Errors Will Happen!

When executing JavaScript code, different errors can occur.

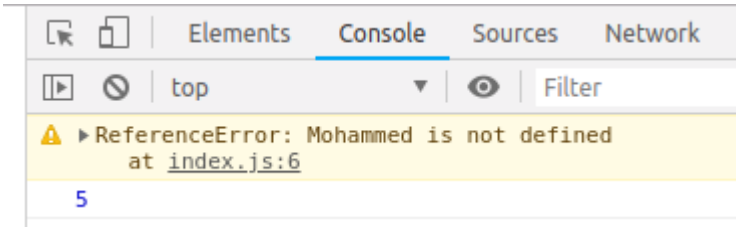
Errors can be coding errors made by the programmer, errors due to wrong input, and other unforeseeable things.

```
20  console.log(Mohammed);
21  console.log(5);
```



Look the error will prevent to continue the code in line 21.


//with try catch
try{
console.log(Mohammed);
}
catch(er)
{
console.warn(er);
}
console.log(5);



Look the error will **not** prevent to continue the code in line 14.

async + await , to create a Asynchronous function

it is instead of 'Promises , to create a Asynchronous function'

```
// async + await
let getWeather = async url => { //1.write 'async' to do async function

  try {//2. this is like .then
    //4. write "await" to every async fun as fetch and json,
    // it will not go the next until the responce to come
    // if no responce so will go to catch
    let res =  await fetch(url);
    let data = await res.json();
    console.log(data);
  }
  catch(error) {  //3.this is like .catch
    console.warn(error);
  }

}
```

API

API (**A**pplication **P**rogramming **I**nterface).

> An API is a set of methods and tools that can be used for building software applications.

focus() Method

Definition and Usage

The focus() method is used to give focus to an element (if it can be focused).

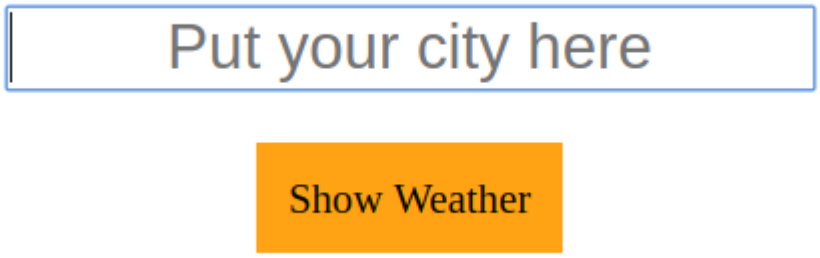
Tip: Use the blur() method to remove focus from an element.

Syntax

```
HTMLElementObject.focus()
```

Return Value:	No return value

```
inputField.focus();
```



to use asynchronous function with delay timeout , Promise.all

folder: 86_to use asynchronous function with delay timeout

```
//to use asynchronous function with dealy timeout

const imageNames = ['a1', 'a2', 'a3', 'a4'];
//names of images
const button = document.querySelector('button');
const container = document.querySelector('#container');

let bringImages = array => {
  let promiseImages = [];

  for (imageName of array) {
    let url = `../images/${imageName}.jpg`;
    let imagePromise = new Promise((resolve, reject) => {
      resolve(fetch(url));
      // return promise
    })
    promiseImages.push(imagePromise);
    // to push all promise to this array (that is more requst and store)
  }
  console.log(promiseImages);

  return Promise.all(promiseImages);
  //Promise.all ??
  // I can to return dirctally the promises array "promiseImages",
  // but may will return a rejected promise.

  // so I will use Promise.all(promiseImages) to return all promise as
  //resolved not one  rejected.
}
```

```
let showImages = async ev => {
  let responses = await bringImages(imageNames);
  let counter = 0;

  let myInterval = setInterval(() => {
    if (counter === responses.length - 1) {
      clearInterval(myInterval);
    }
    let newImage = document.createElement('IMG');
    newImage.src = responses[counter].url;
    container.appendChild(newImage);
    counter++;
  }, 200)

}

button.addEventListener('click', showImages);
```

---

jQuery

$(this).
In JQ ; $(this) = .currentTarget

-not who is called the function in JS
-not working with arrow function =>

append(), prepend()

to add children.

.after , .before()

To add siblings.

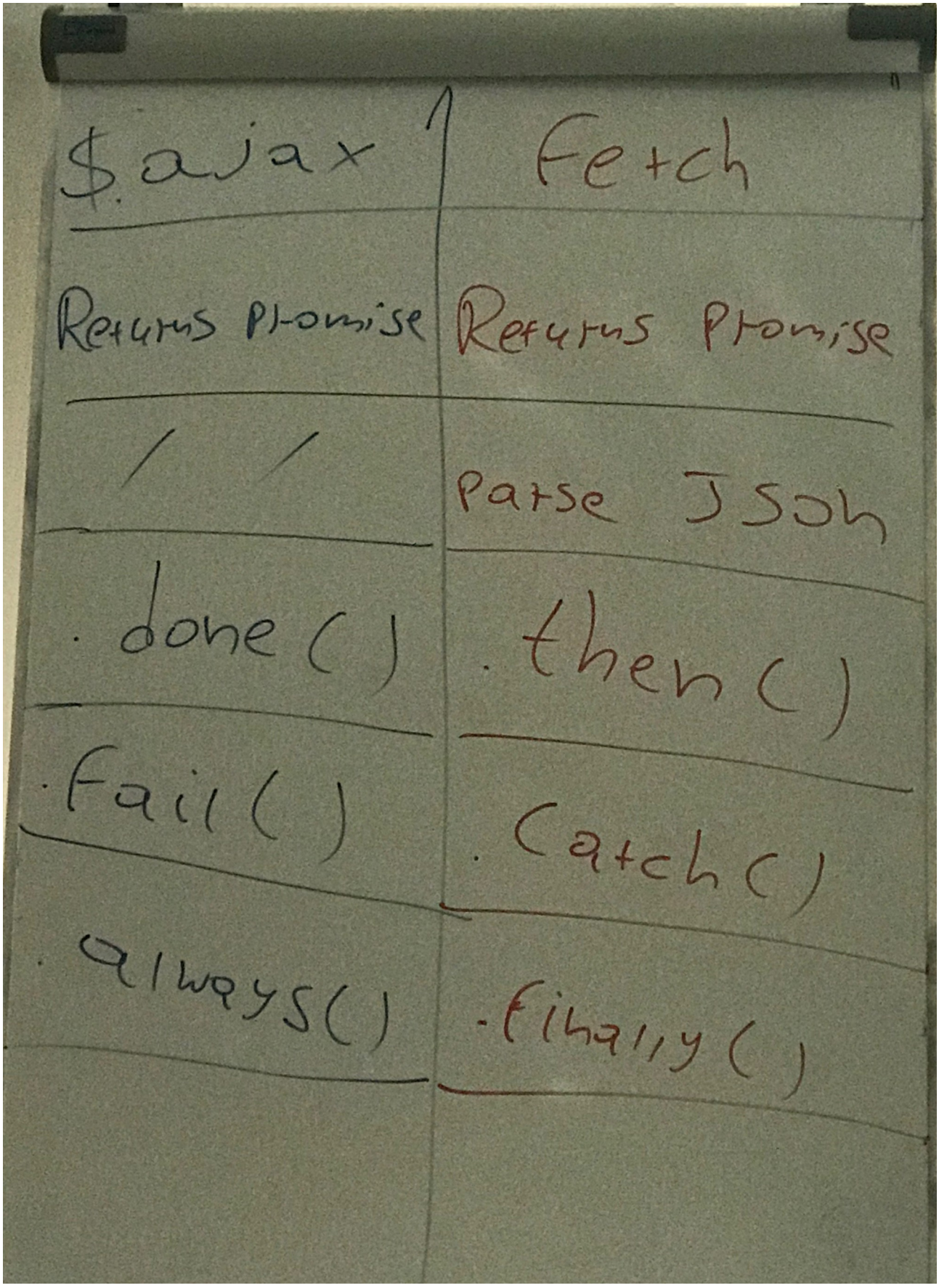
.remove()

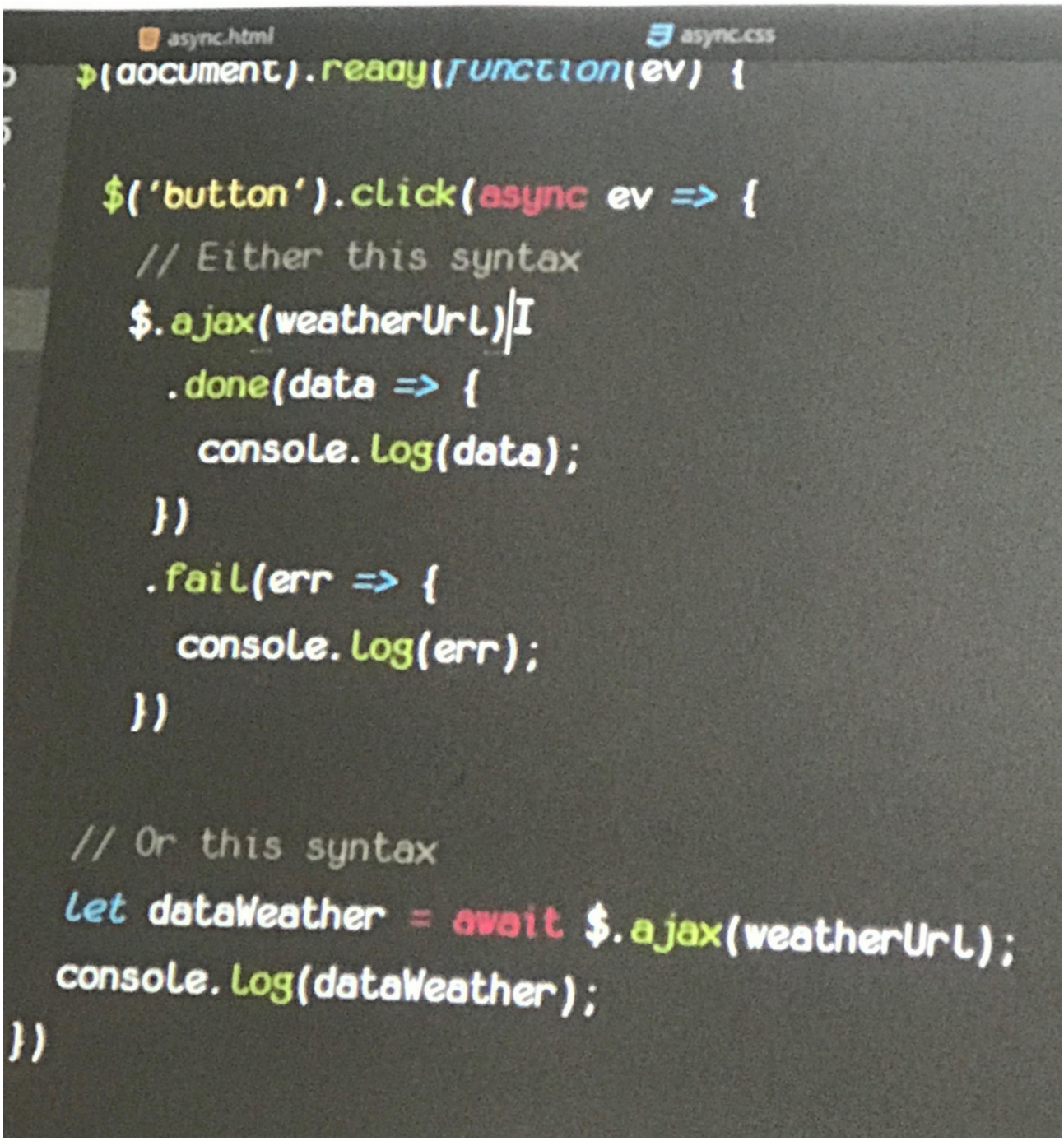
remove the element itself

vanilla js vs jQuery

innerText ==> text()
innerHtml ==> html()
Value ==> val()

$.ajax(in jQuery) vs Fetch (in vanilla js)



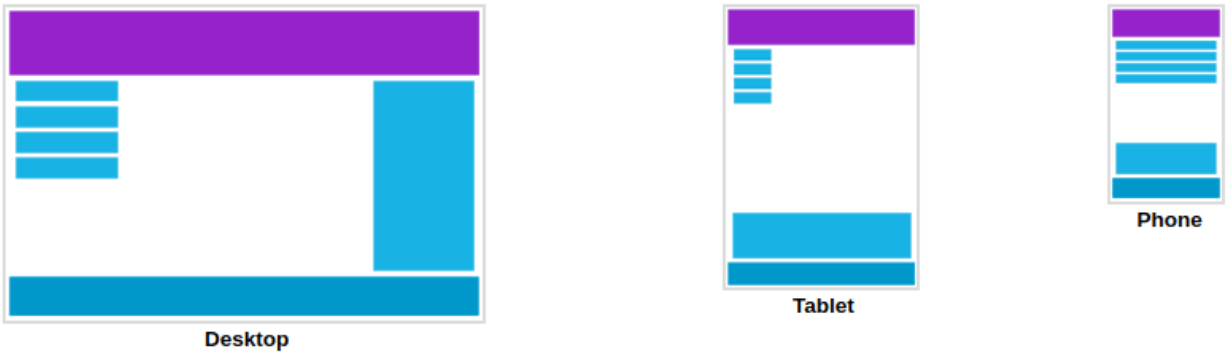


https://jqueryui.com/

jQuery UI is a curated set of user interface interactions, effects, widgets, and themes built on top of the jQuery JavaScript Library. Whether you're building highly interactive web applications or you just need to add a date picker to a form control, jQuery UI is the perfect choice.

Media Queries

Responsive Web Design - Media Queries




```
/* Extra small devices (phones, 600px and down) */
@media only screen and (max-width: 600px) {...}

/* Small devices (portrait tablets and large phones, 600px and up) */
@media only screen and (min-width: 600px) {...}

/* Medium devices (landscape tablets, 768px and up) */
@media only screen and (min-width: 768px) {...}

/* Large devices (laptops/desktops, 992px and up) */
@media only screen and (min-width: 992px) {...}

/* Extra large devices (large laptops and desktops, 1200px and up) */
@media only screen and (min-width: 1200px) {...}
```

Try it Yourself »

https://www.w3schools.com/css/css_rwd_mediaqueries.asp

export import js

1. do this for source and destination

delete this

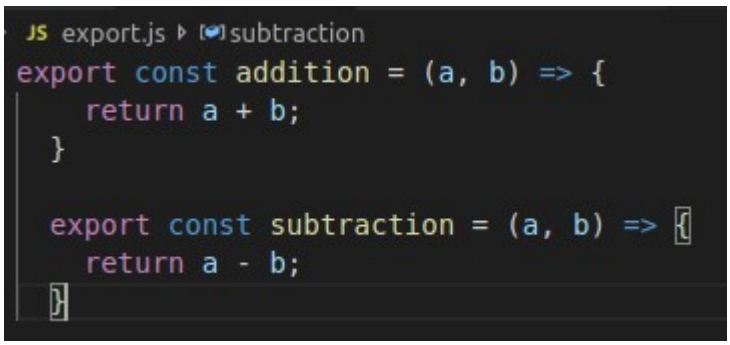
```
<script src="../scripts/index.js" type="text/javascript"></script>
```

add this

```
<script src="../scripts/index.js" type="module"></script>
```

2.

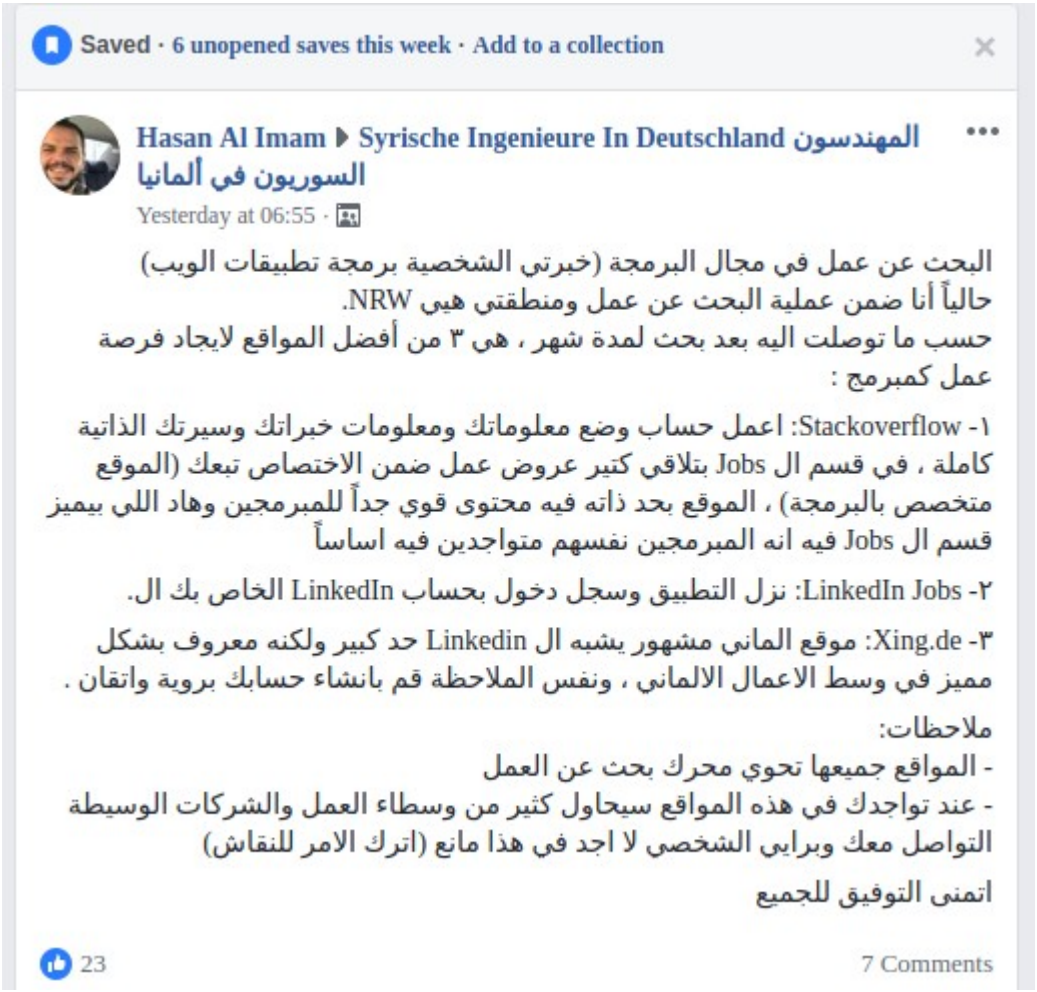
3.



4.

css Example
Css Example

css Example
Css Example



https://www.w3schools.com/js/js_arithmetic.asp