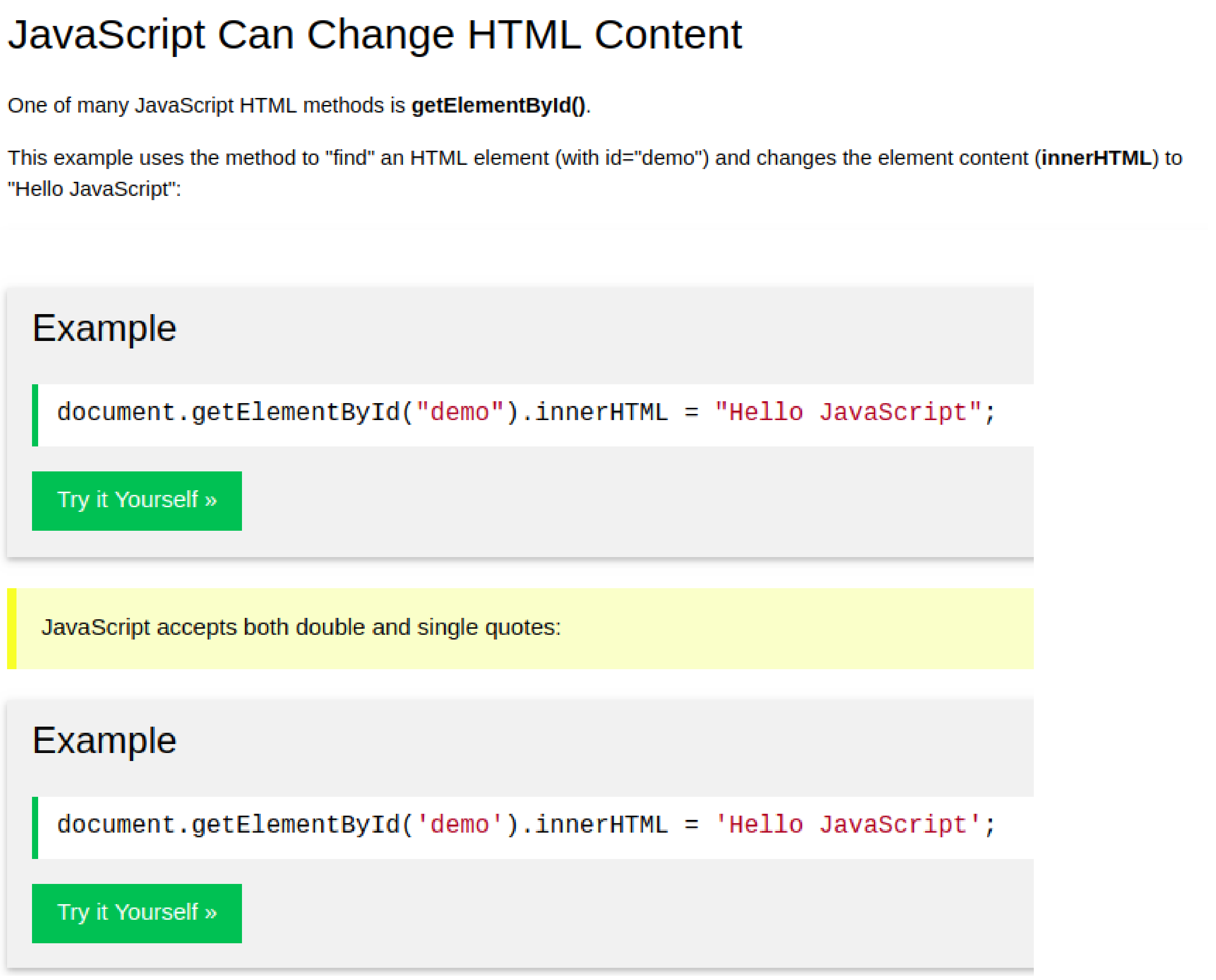
JavaScript is the programming language of HTML and the Web.



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

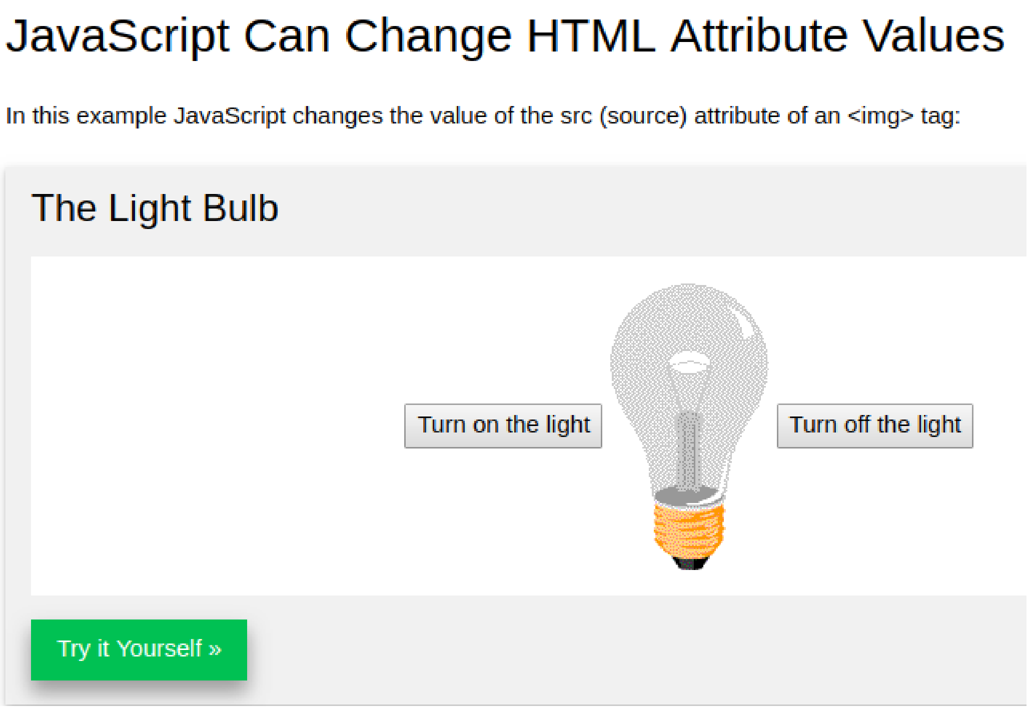
```
document.getElementById("demo").innerHTML = "Hello JavaScript";
```

JavaScript accepts both double and single quotes:

Example

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

Nice Example

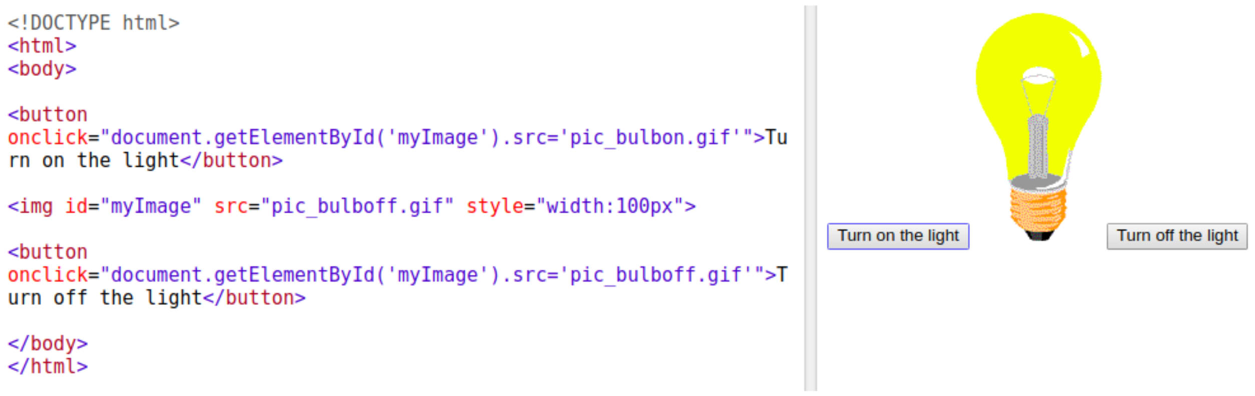


JavaScript Can Change HTML Attribute Values

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

The Light Bulb

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

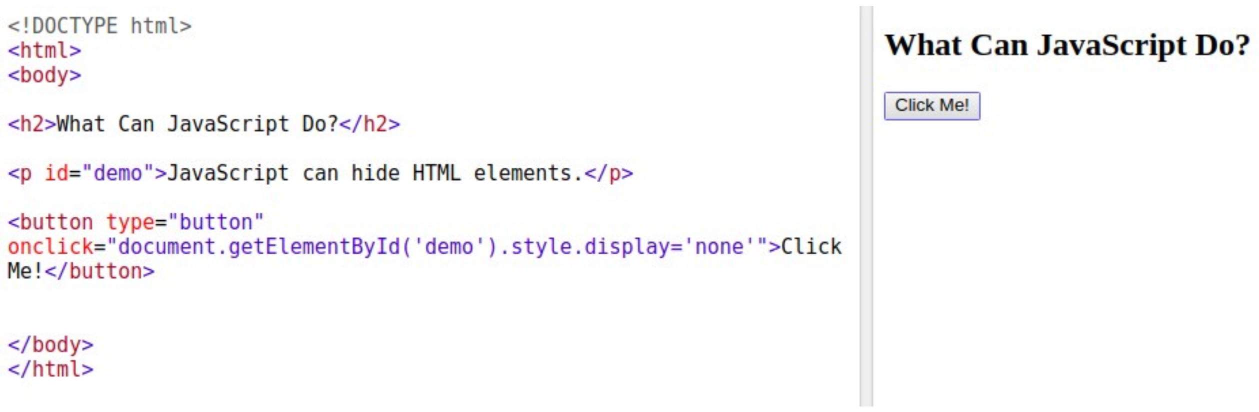


```
<!DOCTYPE html>
<html>
<body>

<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>
```

What Can JavaScript Do?

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>
```

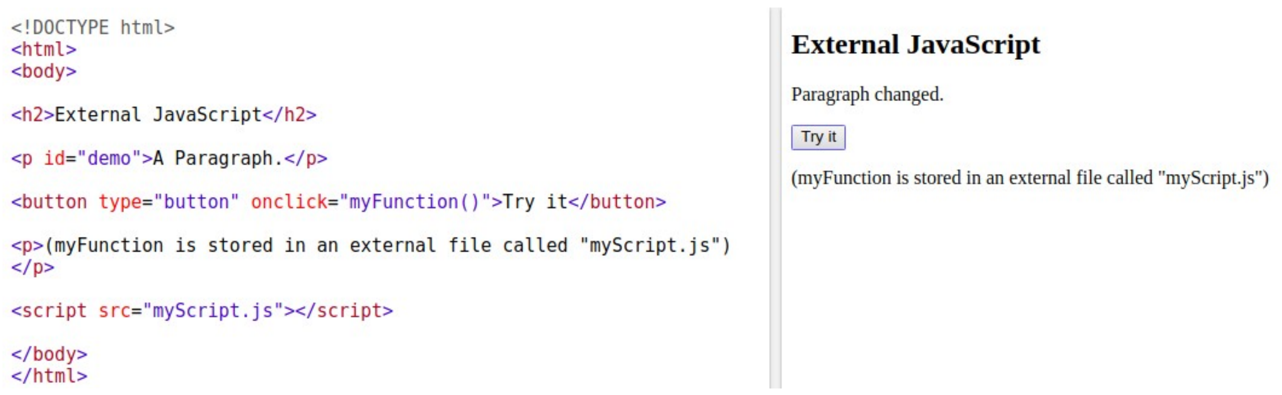
```
<html>
<body>

<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

Paragraph changed.

(myFunction is stored in an external file called "myScript.js")

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:

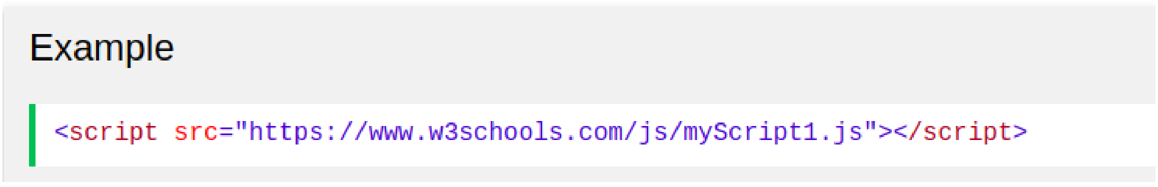
Example

```
<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:



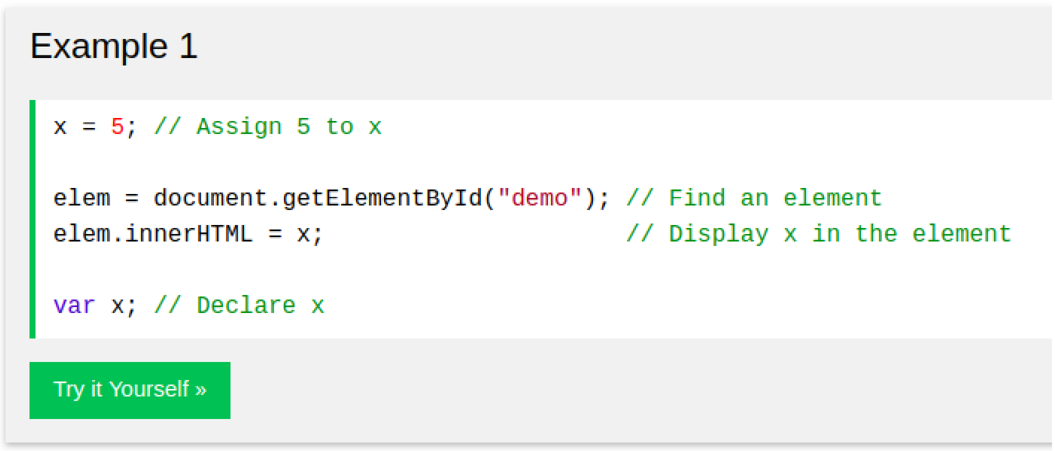
Example

```
<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 .

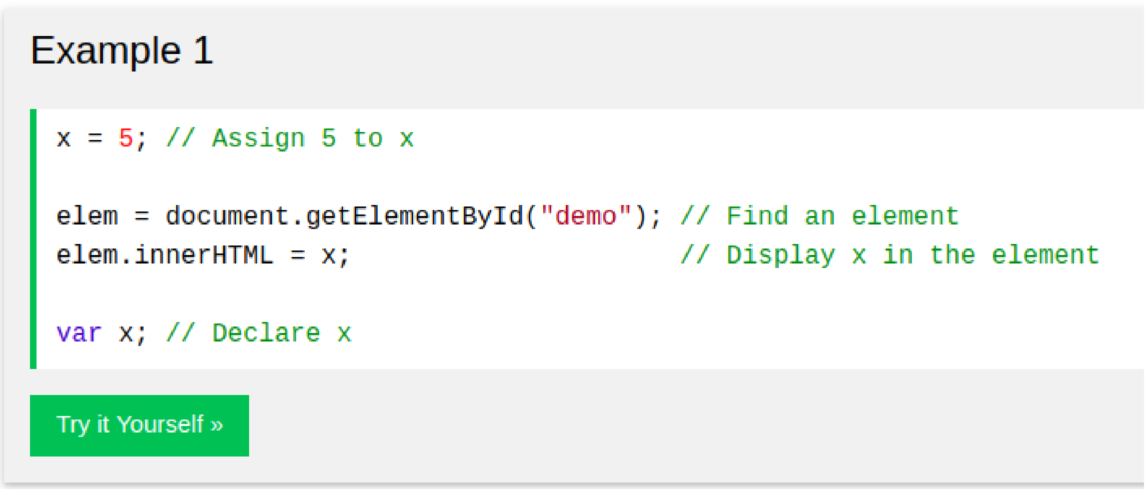
Example 1 gives the same result as **Example 2**:



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
```

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



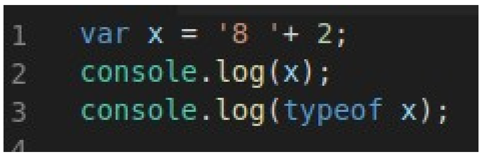
Example 1

```
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
```

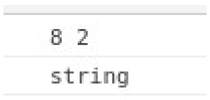
Hoisting is JavaScript's default behavior of moving all declarations to the top of the current scope (to the top of the current script or the current function).

The math operations with Converting the ‘string’

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

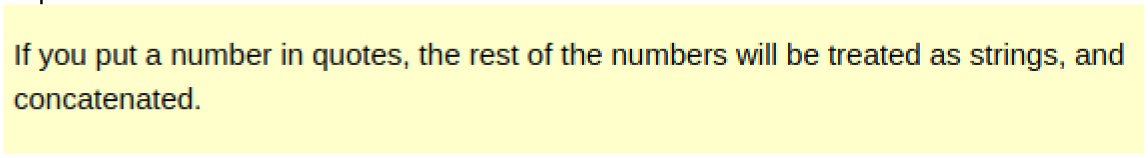


```
var x = '8 '+ 2;
console.log(x);
console.log(typeof x);
```

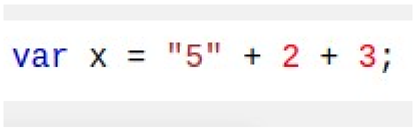


```
8 2
string
```

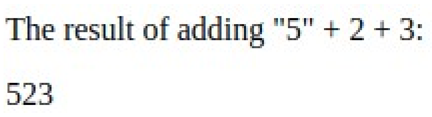
Important Note 1:



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



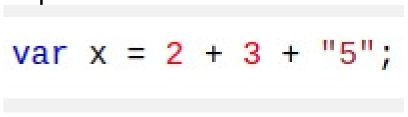
```
var x = "5" + 2 + 3;
```



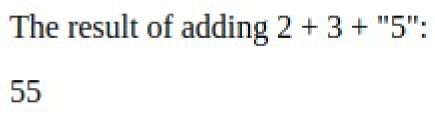
The result of adding "5" + 2 + 3:

523

Important Note 2:



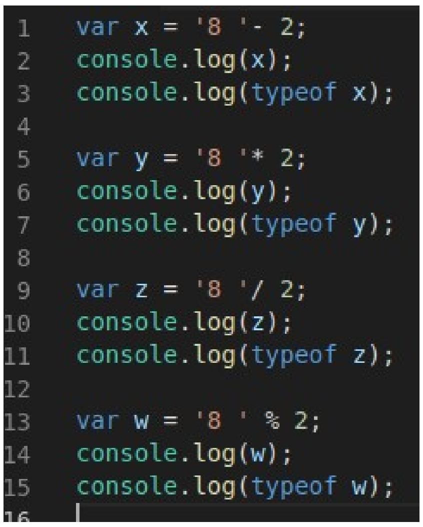
```
var x = 2 + 3 + "5";
```



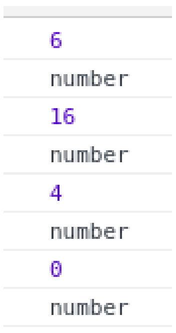
The result of adding 2 + 3 + "5":

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(){});
```

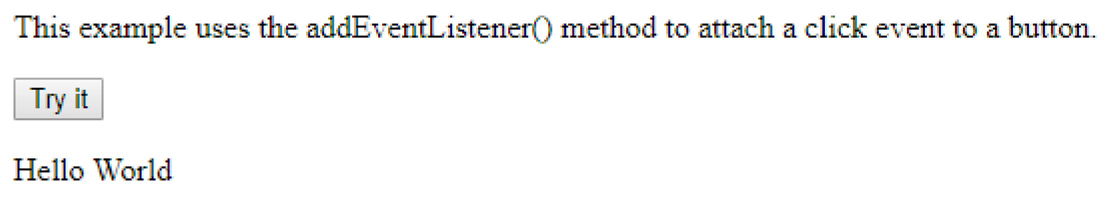
- **Tip:** Use the removeEventListener() method to remove an event handler that has been attached with the addEventListener() method.

-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



This example uses the addEventListener() method to attach a click event to a button.

Try it

Hello World

Logical operation ! && ||

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

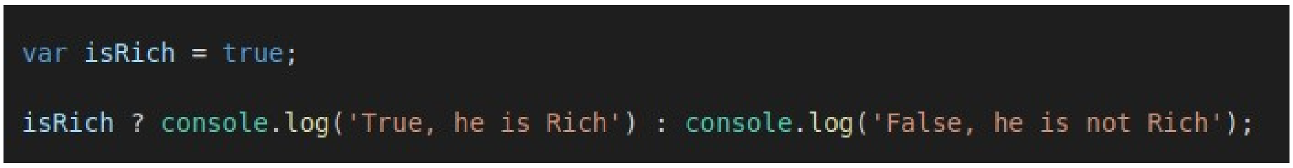
Given that **x = 6** and **y = 3**, the table below explains the logical operators:

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 :

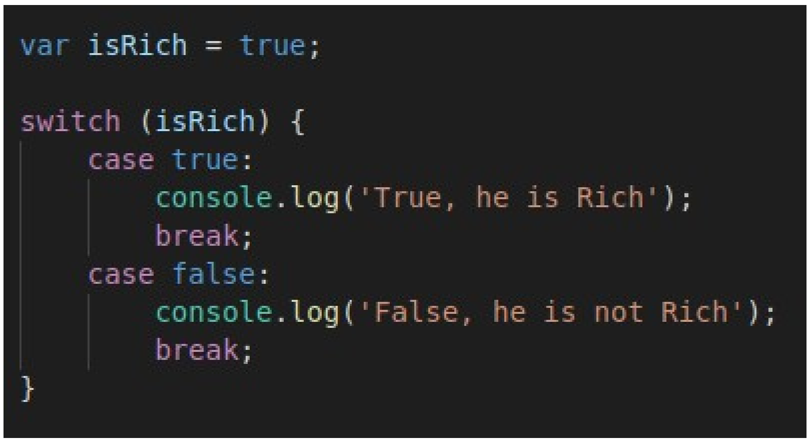
1 **not !**
2 **and &&**
3 **or ||**

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.

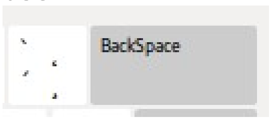
https://developer.mozilla.org/en-US/docs/eb/JavaScript/Reference/Operators/Operator_Precedence

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:

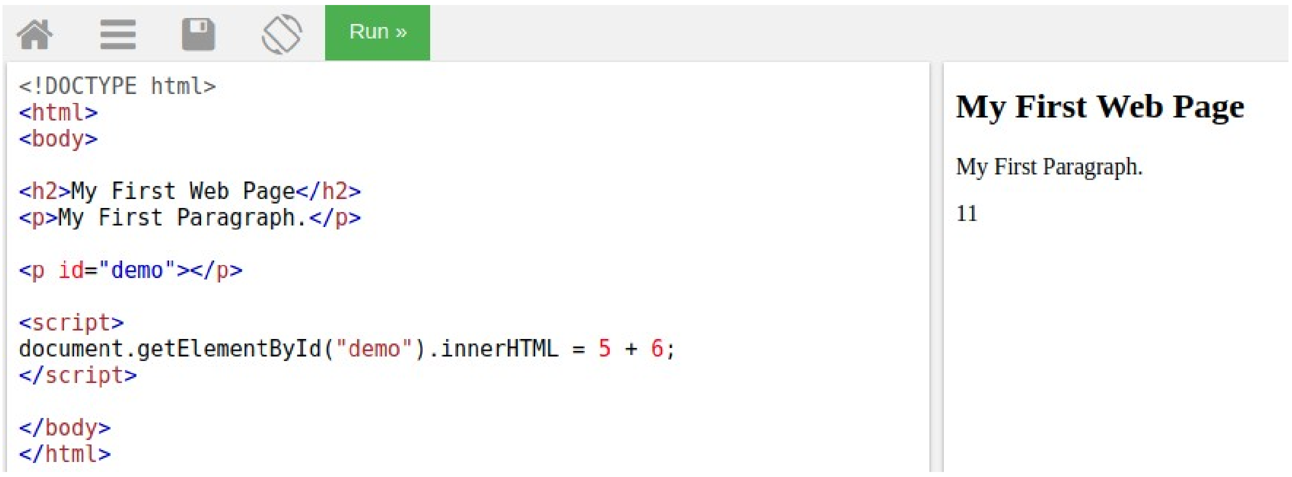
- Writing into an HTML element, using **innerHTML**.
- Writing into the HTML output using **document.write()**.
- Writing into an alert box, using **window.alert()**.
- Writing into the browser console, using **console.log()**.

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:

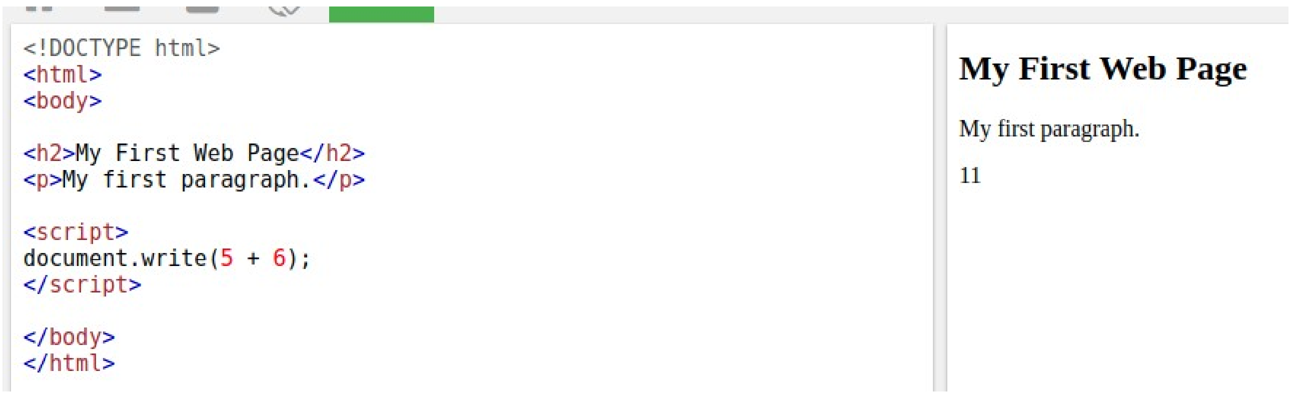


```
<!DOCTYPE html>
<html>
<body>

<h2>My First Web Page</h2>
<p>My First Paragraph.</p>

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

<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>

</body>
</html>
```

```
My First Web Page

My First Paragraph.

11
```

Using document.write()

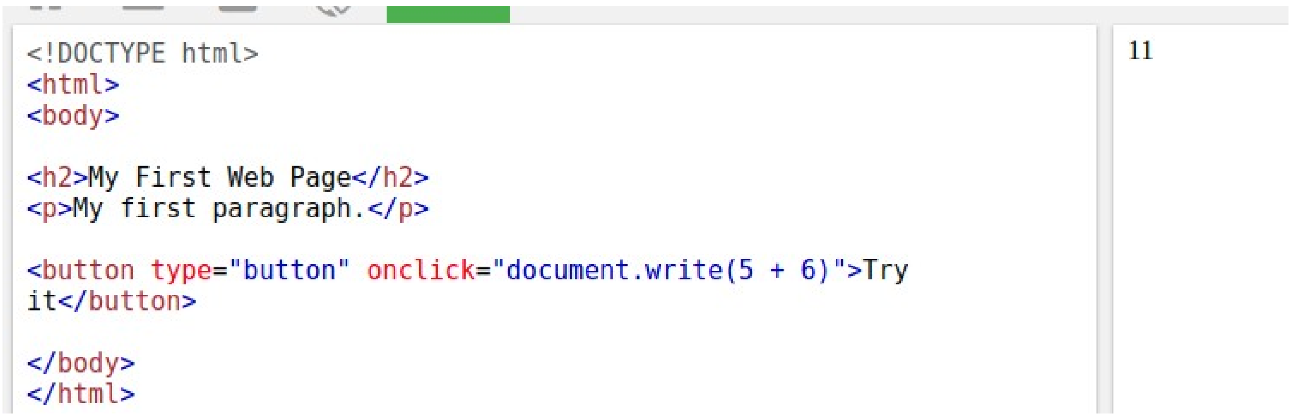
For testing purposes, it is convenient to use **document.write()**:



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

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

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

</body>
</html>
```

```
My First Web Page

My first paragraph.

11
```

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



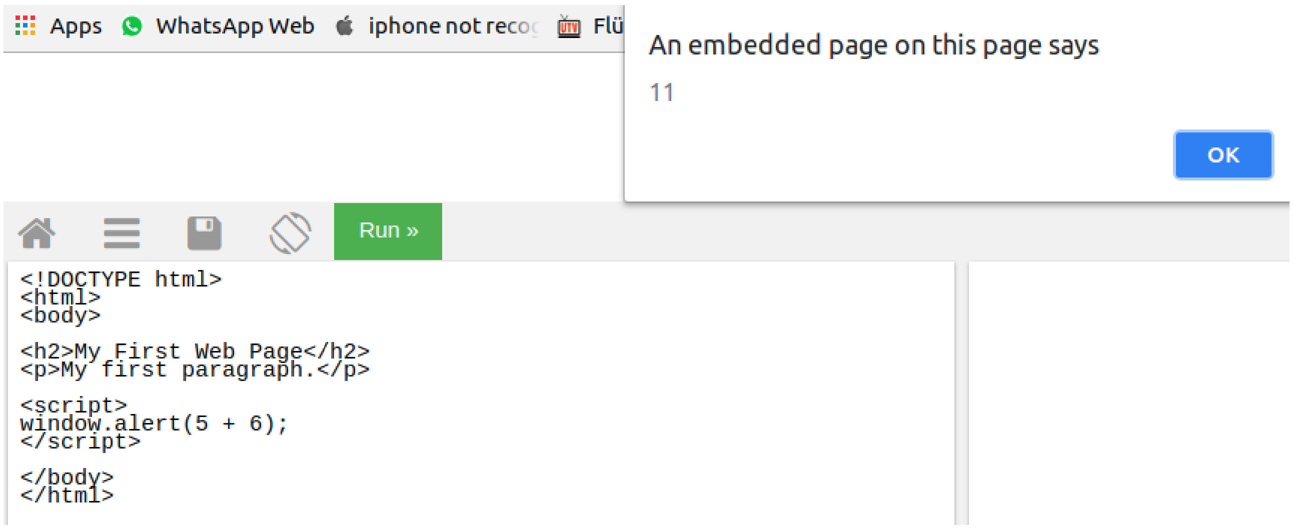
```
<!DOCTYPE html>
<html>
<body>

<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:



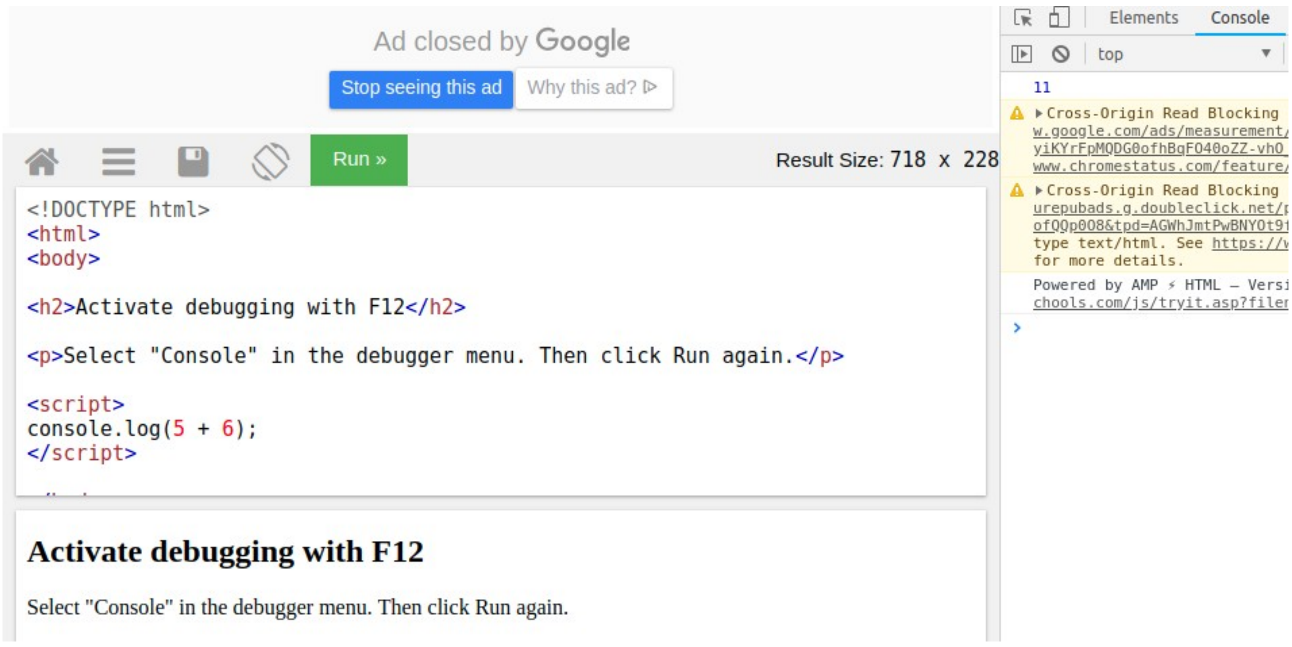
```
An embedded page on this page says
11
```

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

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

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

</body>
</html>
```

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.



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

<h2>Activate debugging with F12</h2>

<p>Select "Console" in the debugger menu. Then click Run again.</p>

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

Activate debugging with F12

Select "Console" in the debugger menu. Then click Run again.

```
11
▸Cross-Origin Read Blocking
w.google.com/ads/measurement/
yiKYrFpMQDG0ofhBqFO40oZZ-vhO_
www.chromestatus.com/feature/
▸Cross-Origin Read Blocking
urepubads.g.doubleclick.net/p
ofQQp0O8&tpd=AGWhJmtPwBNYOt9f
type text/html. See https://w
for more details.
Powered by AMP ⚡ HTML – Versi
chools.com/js/tryit.asp?filen
>
```

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";
```

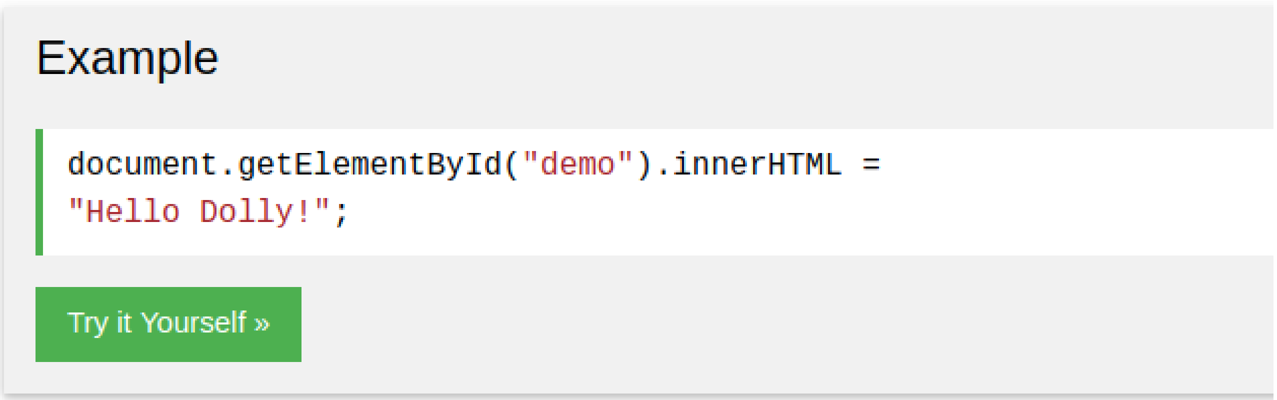
A good practice is to put spaces around operators (= + - * /):

```
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:



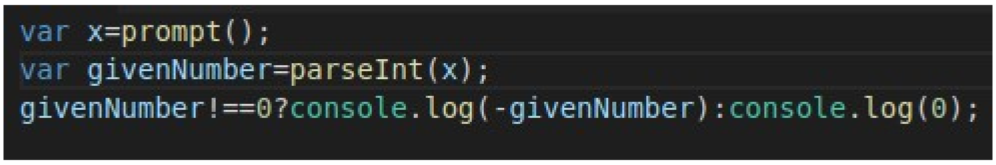
Example

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

Variable = Variable + ‘   Hallo ‘

Variable += ‘ Hallo ‘ ;

Classy if – one line



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

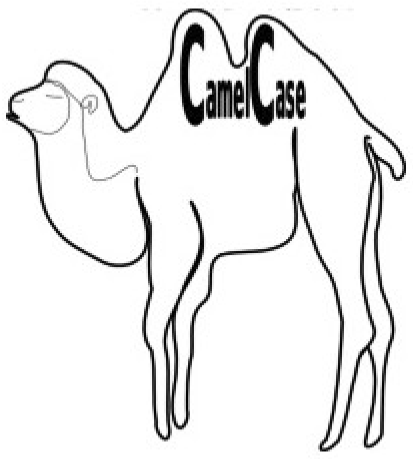
```
var x=prompt();
var givenNumber=parseInt(x);
givenNumber!==0?console.log(-givenNumber):console.log(0);
```

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.

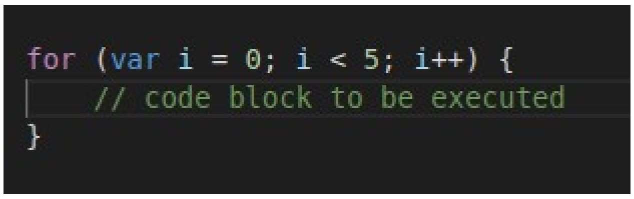
Underscore:

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

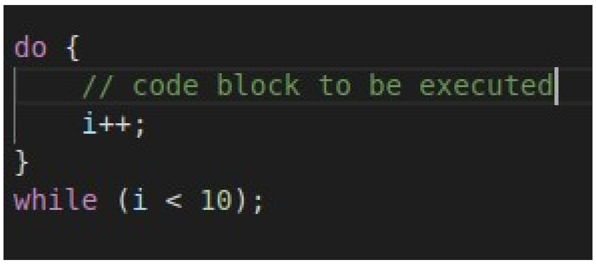
The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

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

Example

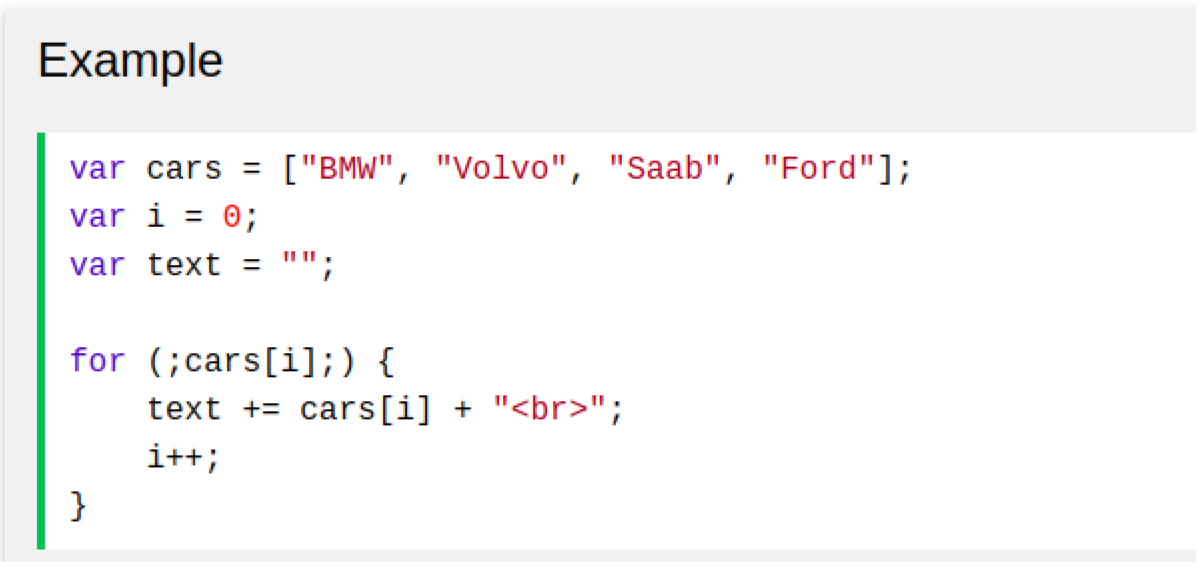
The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:



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

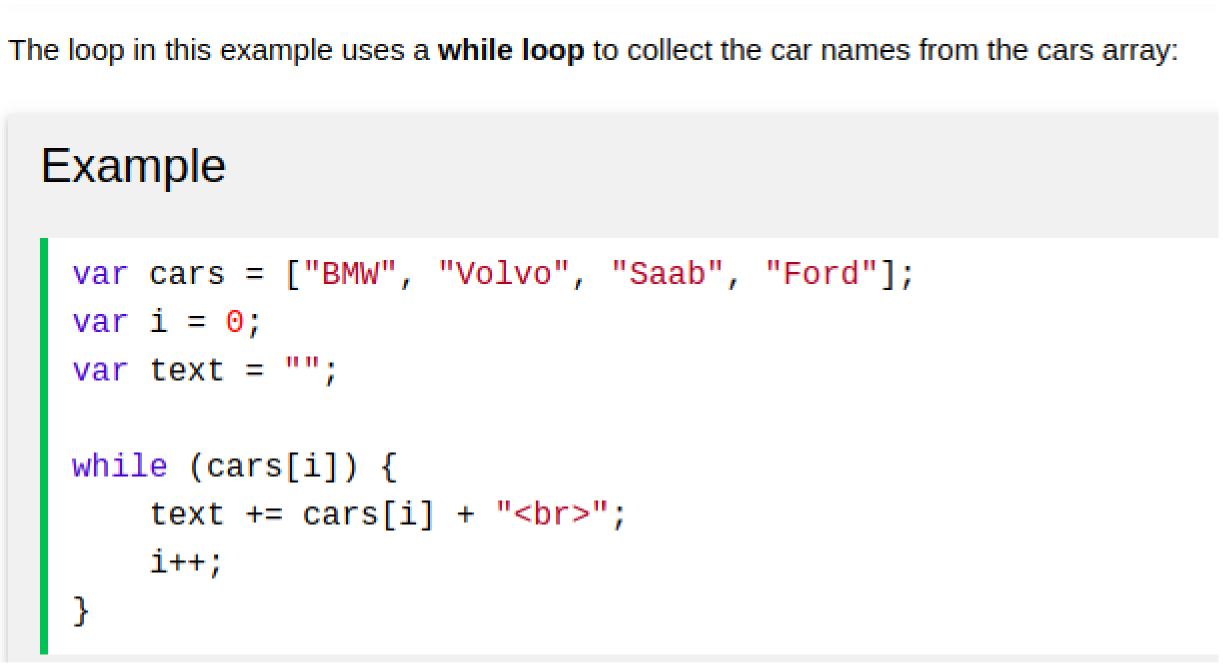
Comparing For and While

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

```
function myFunction(var p1,var | p2) {
    return p1 * p2;
}
```

Ex:

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

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

----

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.

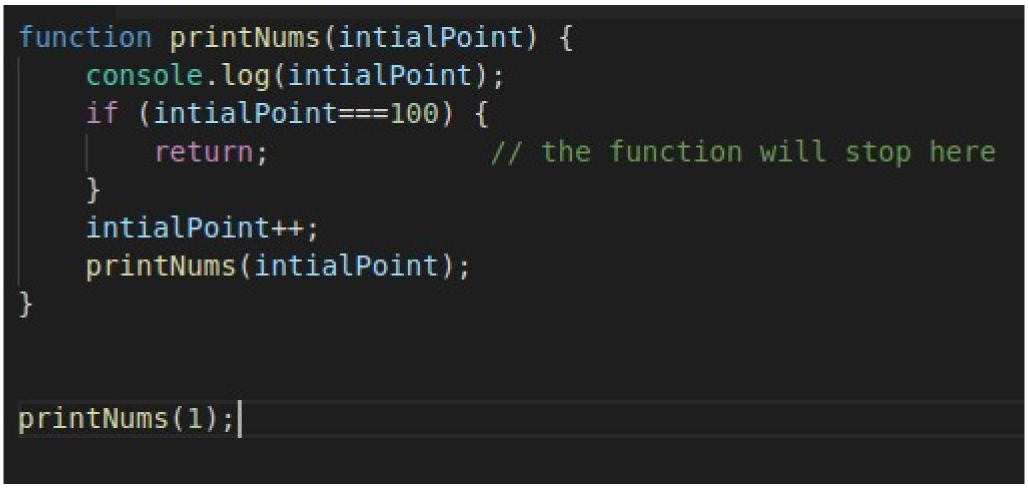
Local variables are created when a function starts, and deleted when the function is completed.

Note: js function is hoisted

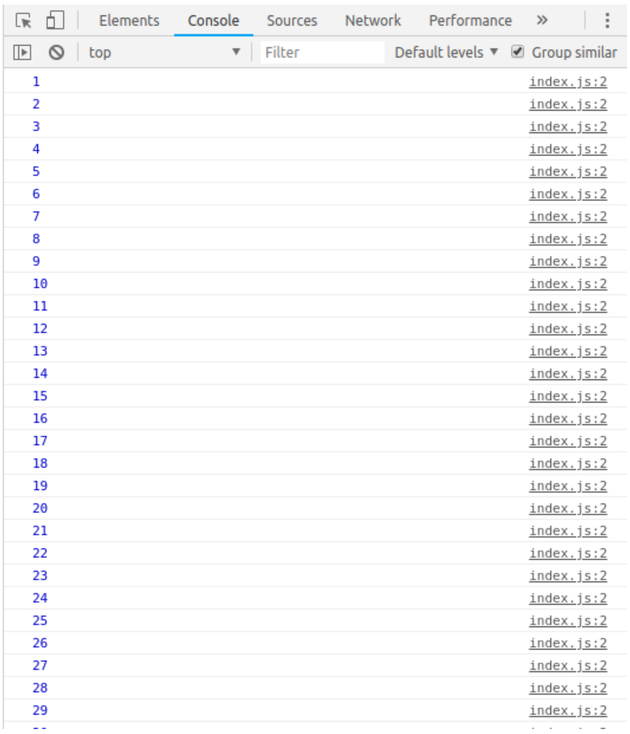
(Important) Function Return

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);
```



```
1                                  index.js:2
2                                  index.js:2
3                                  index.js:2
4                                  index.js:2
5                                  index.js:2
6                                  index.js:2
7                                  index.js:2
8                                  index.js:2
9                                  index.js:2
10                                 index.js:2
11                                 index.js:2
12                                 index.js:2
13                                 index.js:2
14                                 index.js:2
15                                 index.js:2
16                                 index.js:2
17                                 index.js:2
18                                 index.js:2
19                                 index.js:2
20                                 index.js:2
21                                 index.js:2
22                                 index.js:2
23                                 index.js:2
24                                 index.js:2
25                                 index.js:2
26                                 index.js:2
27                                 index.js:2
28                                 index.js:2
29                                 index.js:2
```

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.

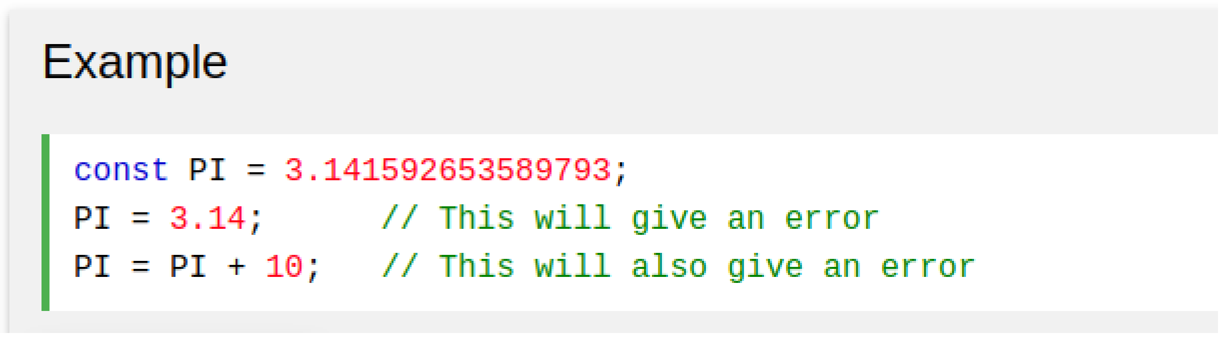
The variable is in a "temporal dead zone" from the start of the block until it is declared:

Example

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

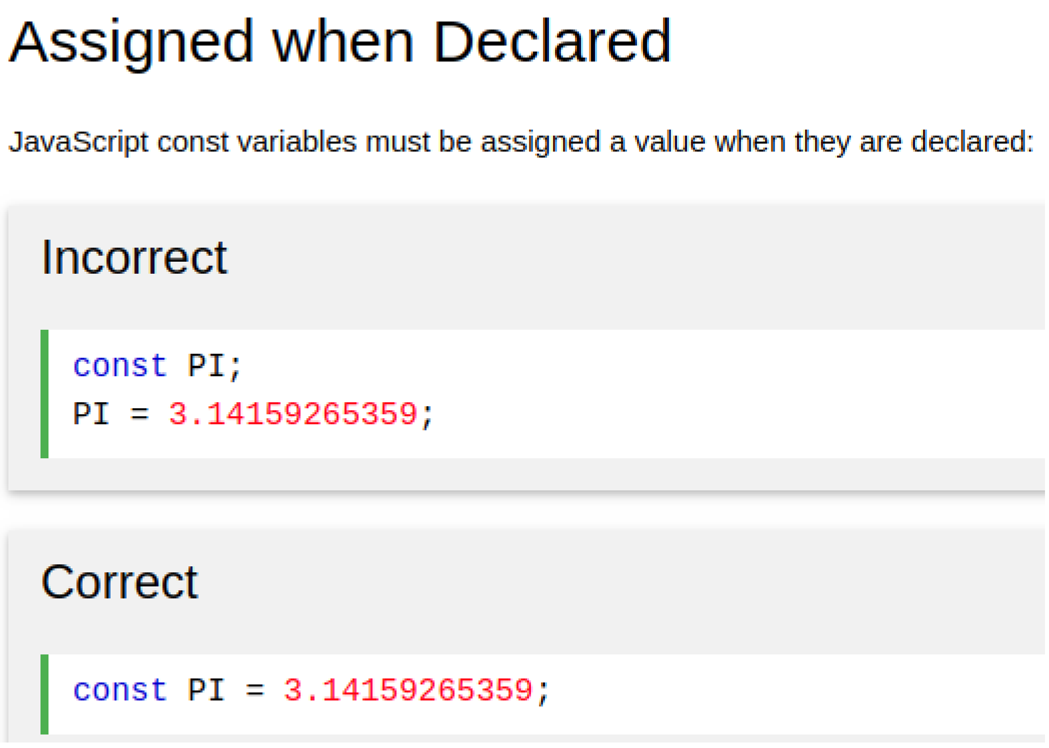
JavaScript Const

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

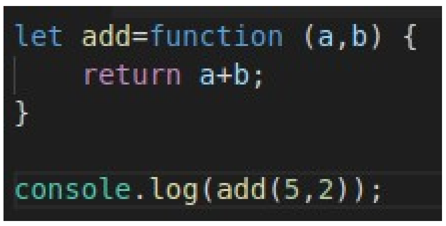
```
const PI;
PI = 3.14159265359;
```

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.

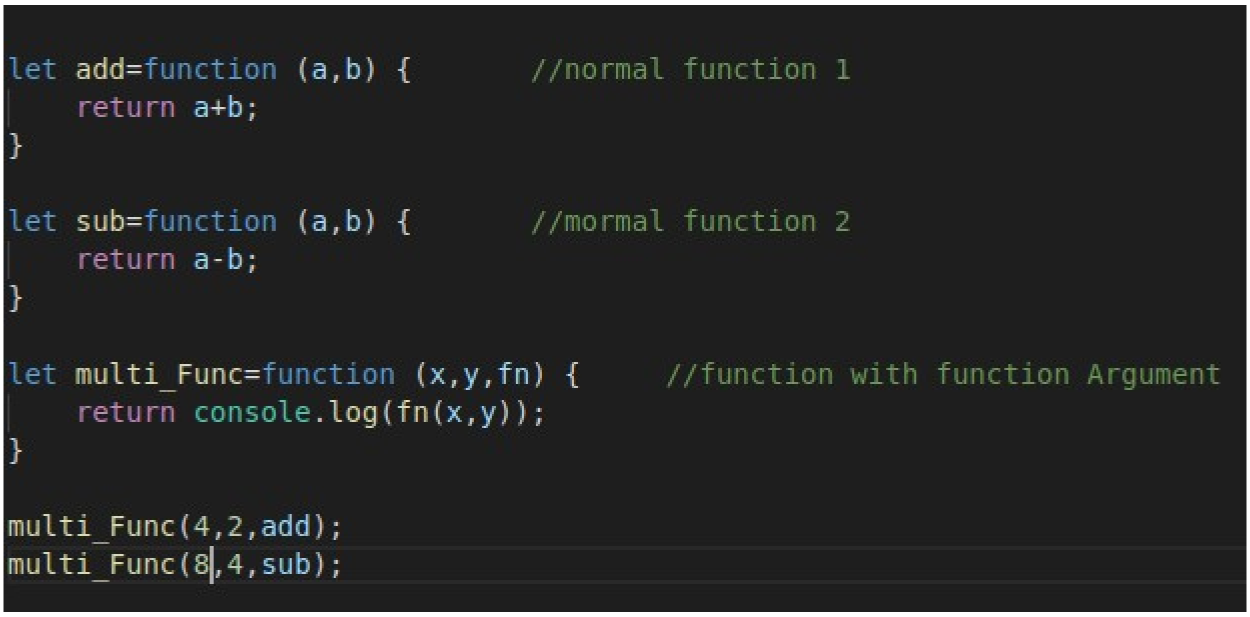


```
let add=function (a,b) {
    return a+b;
}

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

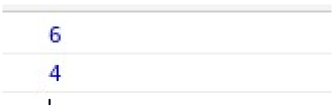
Note: The IIFE's Function is not Hoisting

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
```

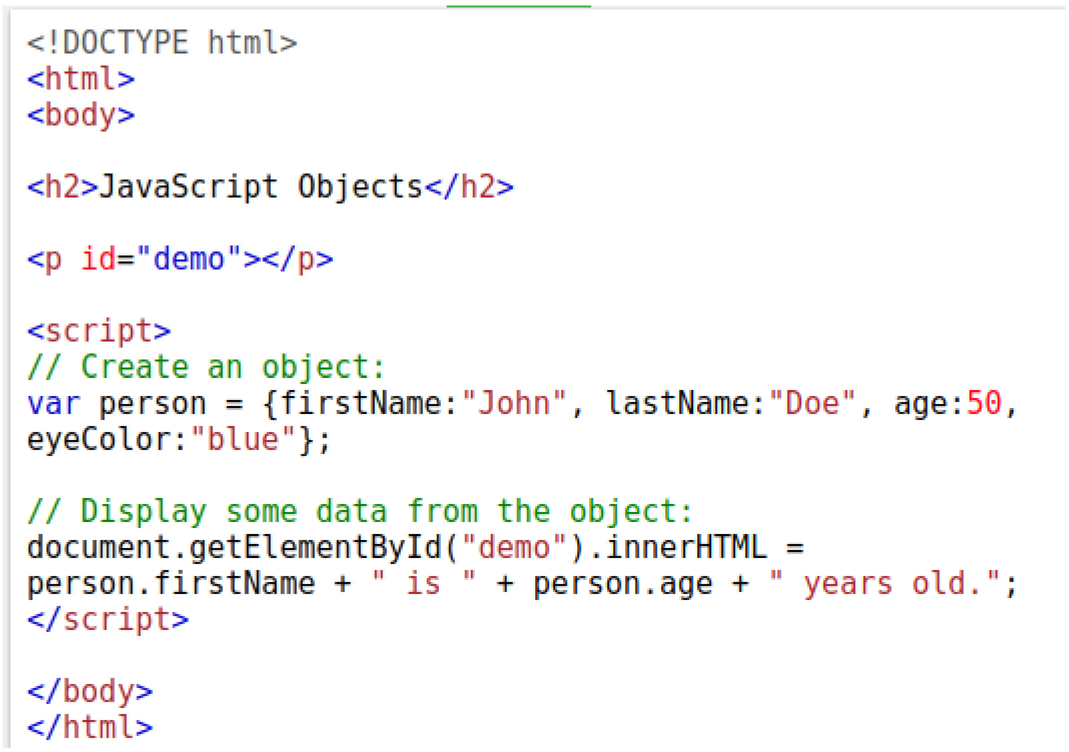
Objected-oriented programming

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

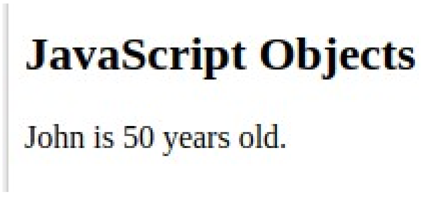
The values are written as **name:value** pairs (name and value separated by a colon).

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;
    }
};
```



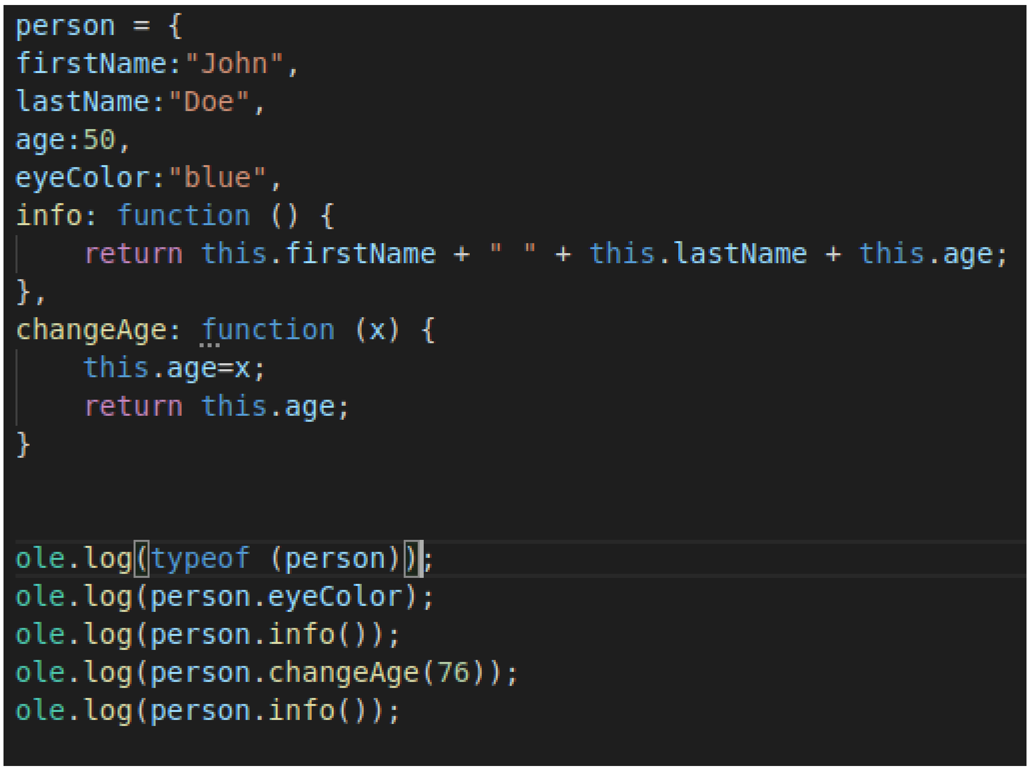
```
<!DOCTYPE html>
<html>
<body>

<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>
```



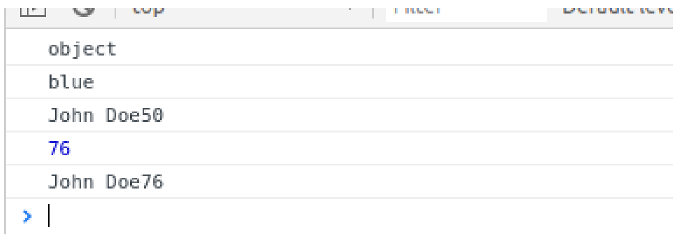
JavaScript Objects

John is 50 years old.

Ex2:

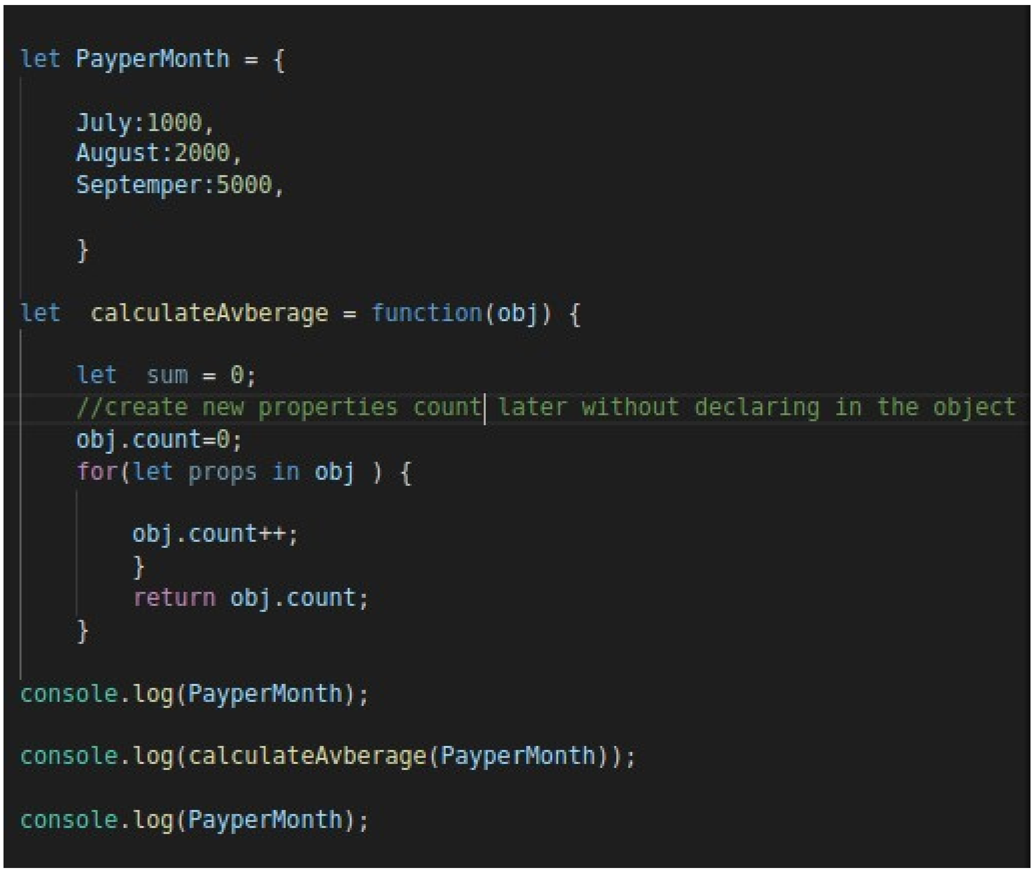


```
person = {
firstName:"John",
lastName:"Doe",
age:50,
eyeColor:"blue",
info: function () {
    return this.firstName + " " + this.lastName + this.age;
},
changeAge: function (x) {
    this.age=x;
    return this.age;
}


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

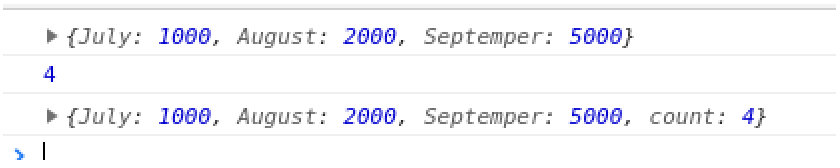


```
object
blue
John Doe50
76
John Doe76
>
```

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

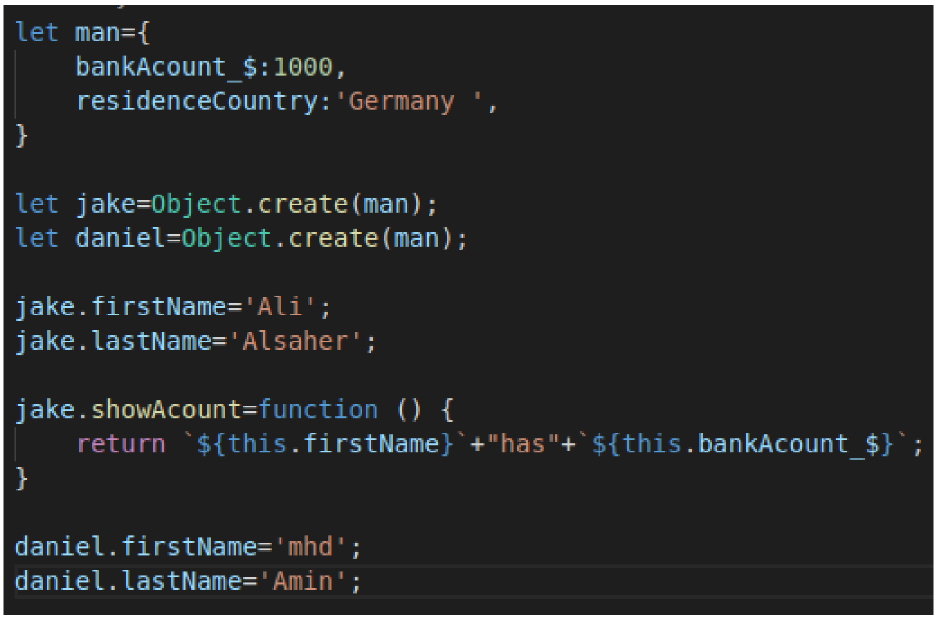


```
let PayperMonth = {

    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);
```

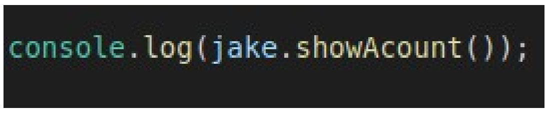


```
▶{July: 1000, August: 2000, Septemper: 5000}
4
▶{July: 1000, August: 2000, Septemper: 5000, count: 4}
>
```

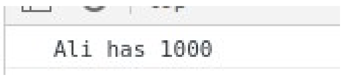
Object Inheritance



```
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';
```

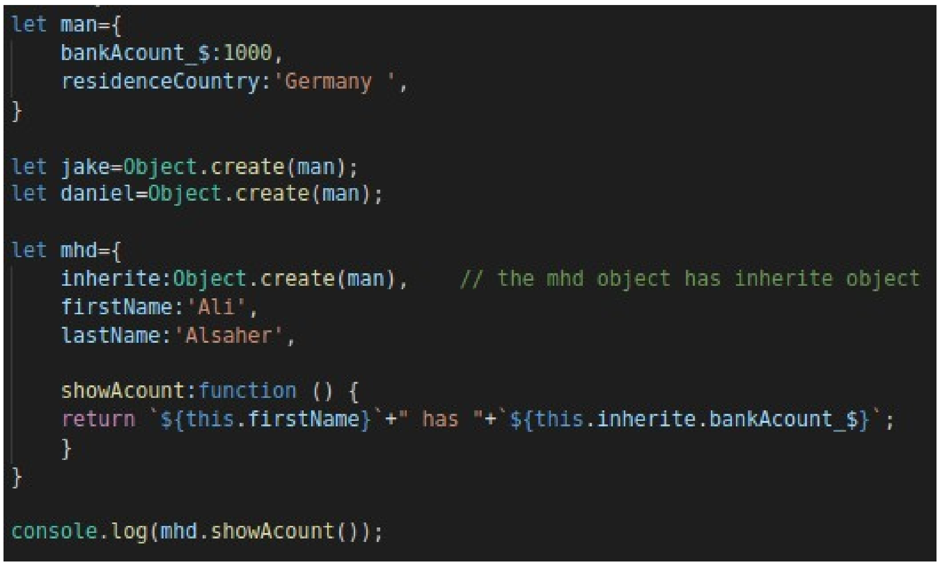


```
console.log(jake.showAcount());
```



```
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());
```

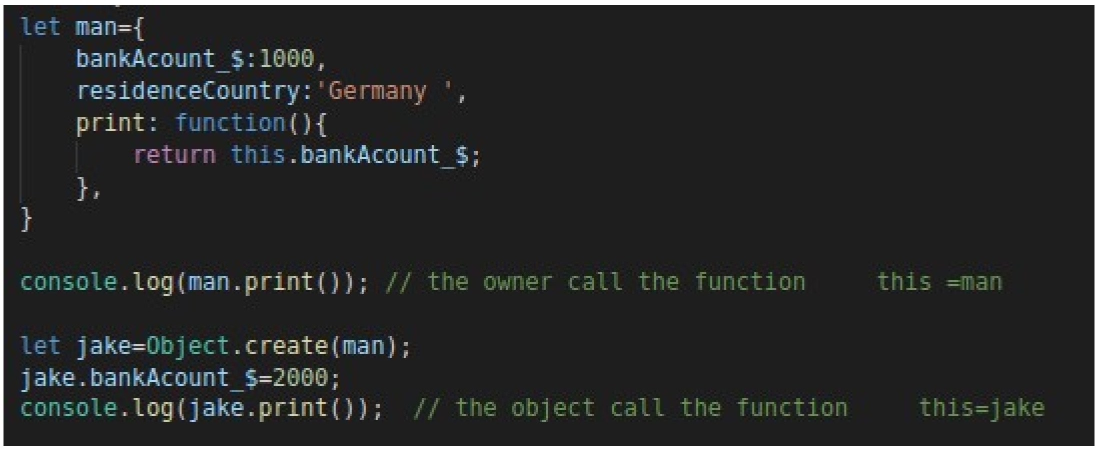
this

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

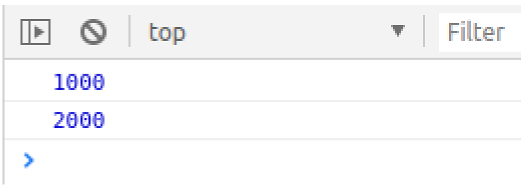
1 - This.numOfsales +=;   ===> this=**owner** the function

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
```



```
top
1000
2000
>
```

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:

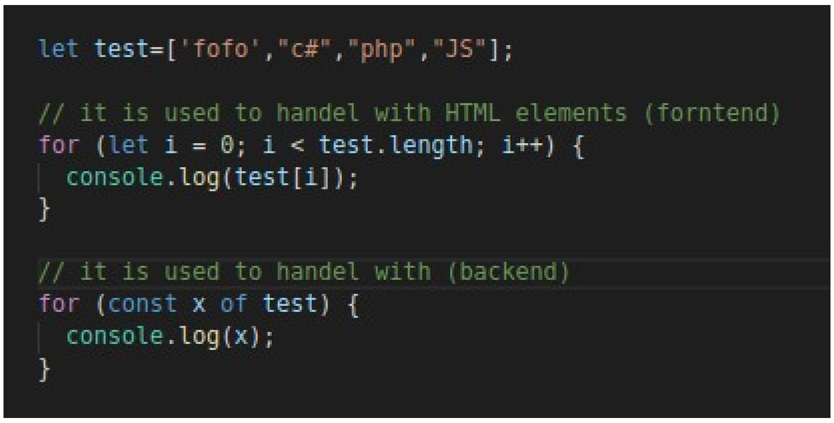
```
var array_name = [item1, item2, ...];
```

Example

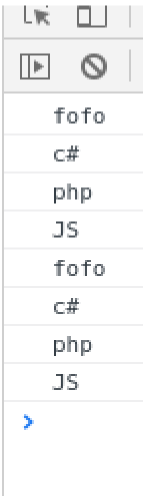
```
var cars = ["Saab", "Volvo", "BMW"];
```

the data type of array is : object

ex:

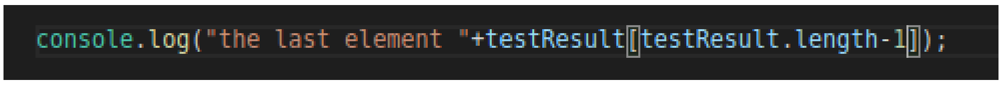


```
let test=['fofo',"c#","php","JS"];

// 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);
}
```



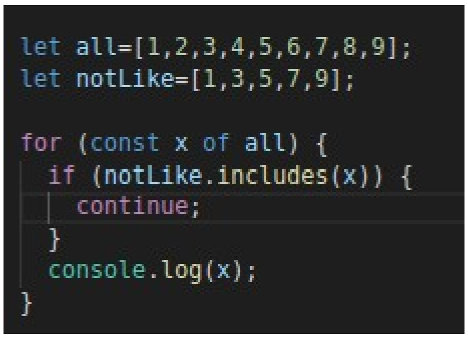
```
fofo
c#
php
JS
fofo
c#
php
JS
>
```

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

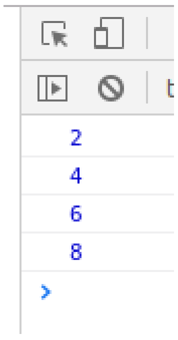


```
console.log("the last element "+testResult[testResult.length-1]);
```

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);
}
```



```
2
4
6
8
>
```

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)

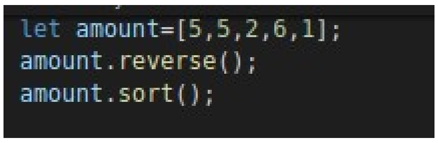
```
var arr1 = ["Cecilie", "Lone"];
var arr2 = ["Emil", "Tobias", "Linus"];
var arr3 = ["Robin", "Morgan"];
var myChildren = arr1.concat(arr2, arr3);     // Concatenates arr1 with arr2 and
arr3
```

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

Example (Merging an Array with Values)

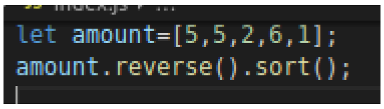
```
var arr1 = ["Cecilie", "Lone"];
var myChildren = arr1.concat(["Emil", "Tobias", "Linus"]);
```

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

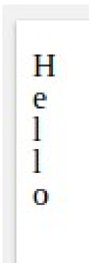
If the separator is omitted, the returned array will contain the whole string in index [0].

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

Example

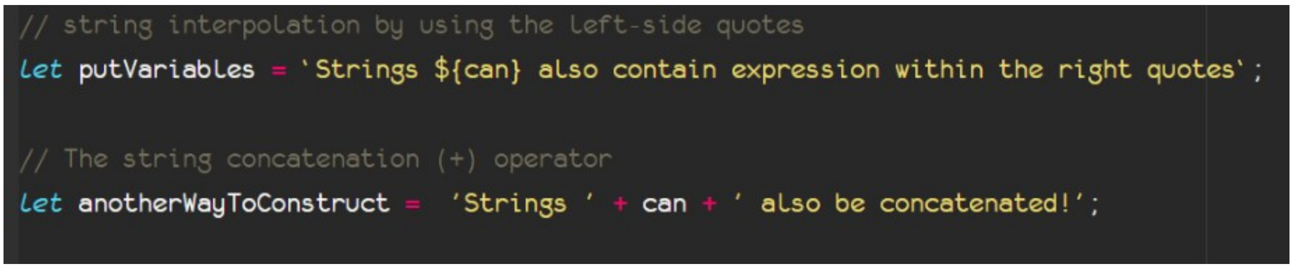
```
var txt = "Hello";       // String
txt.split("");           // Split in characters
```

```
<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>
```

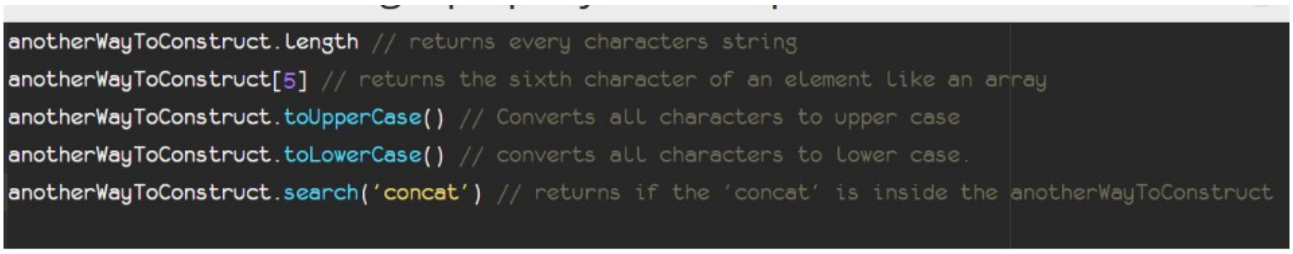


```
H
e
l
l
o
```

String Variable is like array

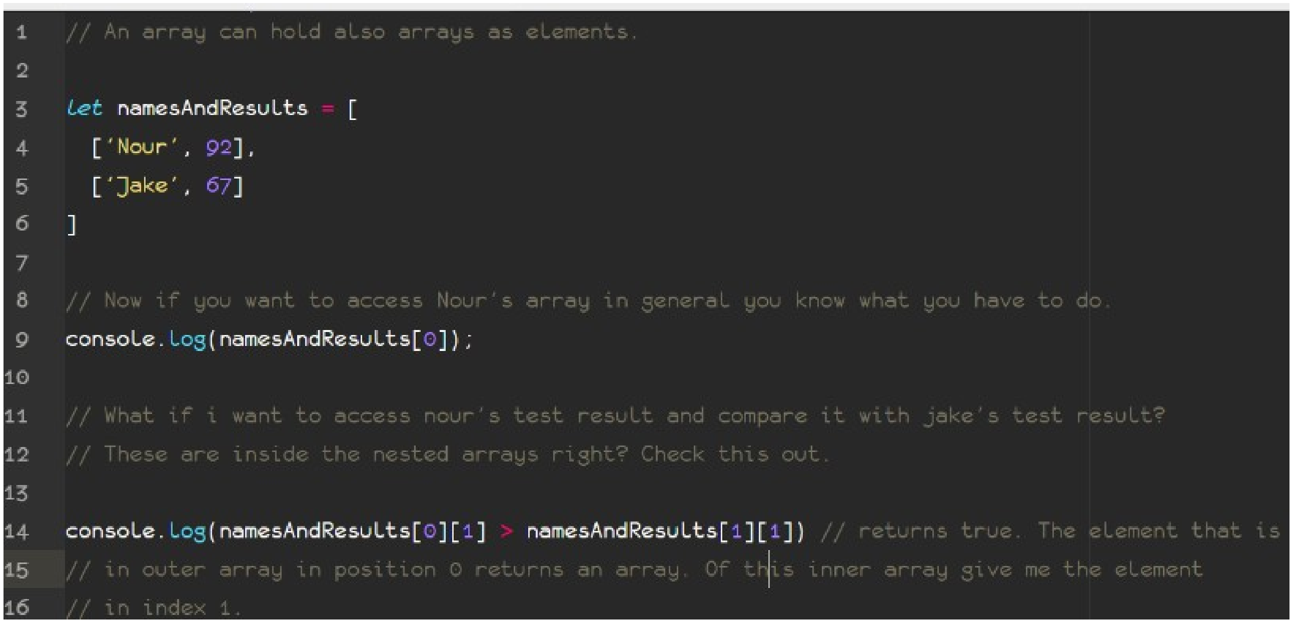


```
// string interpolation by using the left-side quotes
let putVariables = `Strings ${can} also contain expression within the right quotes`;

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



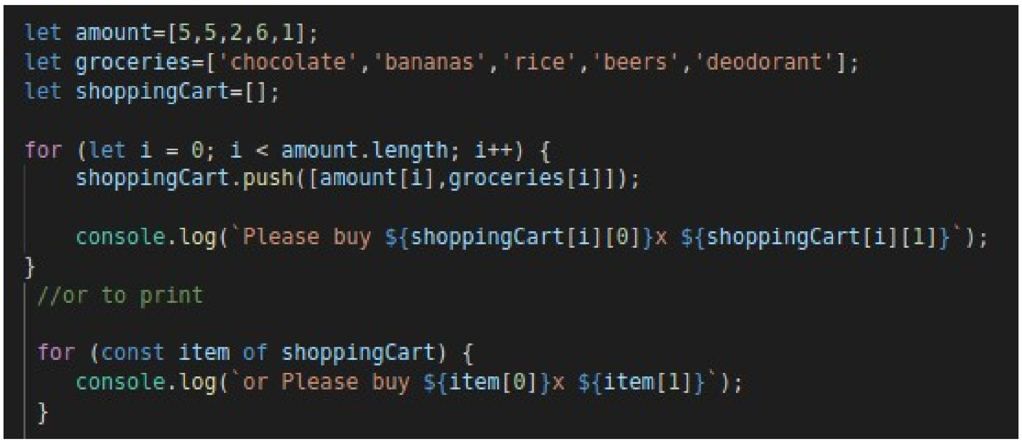
```
anotherWayToConstruct.length // returns every characters string
anotherWayToConstruct[5] // returns the sixth character of an element like an array
anotherWayToConstruct.toUpperCase() // Converts all characters to upper case
anotherWayToConstruct.toLowerCase() // converts all characters to lower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

Nested Arrays

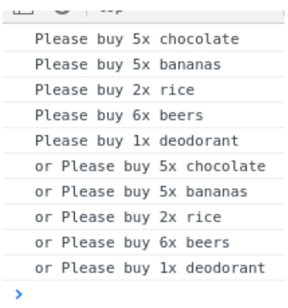


```
// An array can hold also arrays as elements.

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]}`);
 }
```

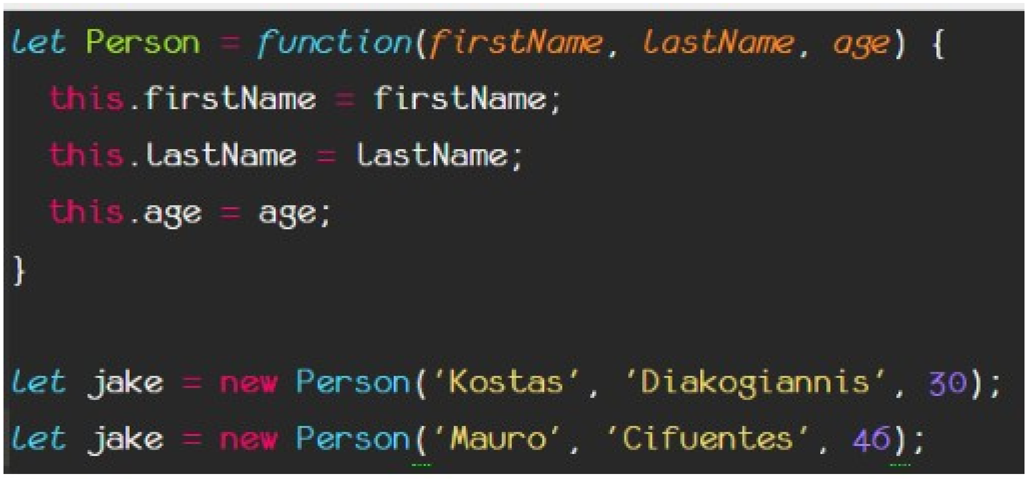


```
Please buy 5x chocolate
Please buy 5x bananas
Please buy 2x rice
Please buy 6x beers
Please buy 1x deodorant
or Please buy 5x chocolate
or Please buy 5x bananas
or Please buy 2x rice
or Please buy 6x beers
or Please buy 1x deodorant
>
```

inheritance (Constructor+Prototype)

1-New Constructor منشئ جديد

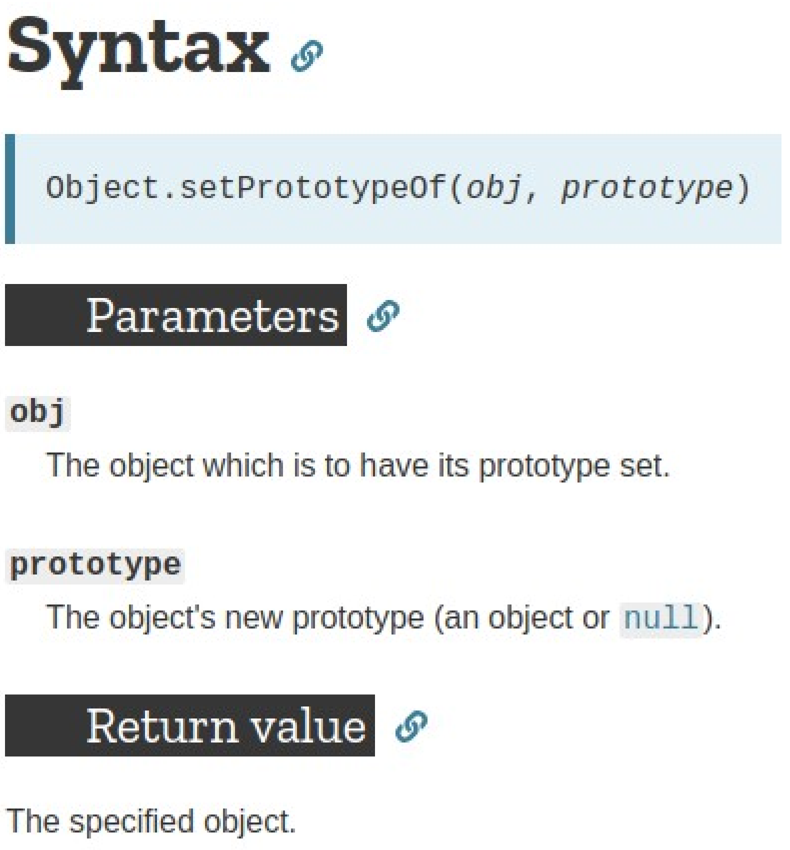
Another way to create objects is by creating a constructor pattern and then fill the properties with the values as shown



```
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`

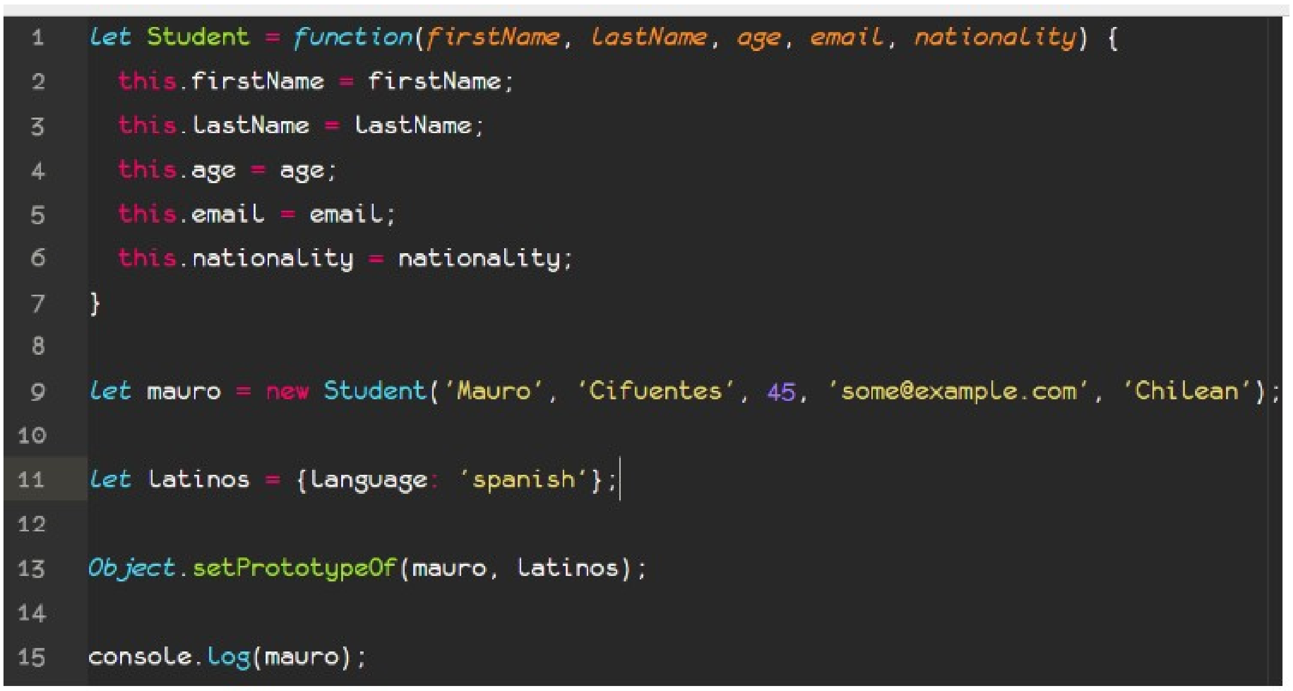
The object which is to have its prototype set.

`prototype`

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

Return value

The specified object.



```javascript
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

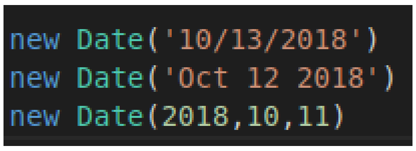
```
var d = new Date(2018, 11, 24, 10, 33, 30, 0);
```

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.

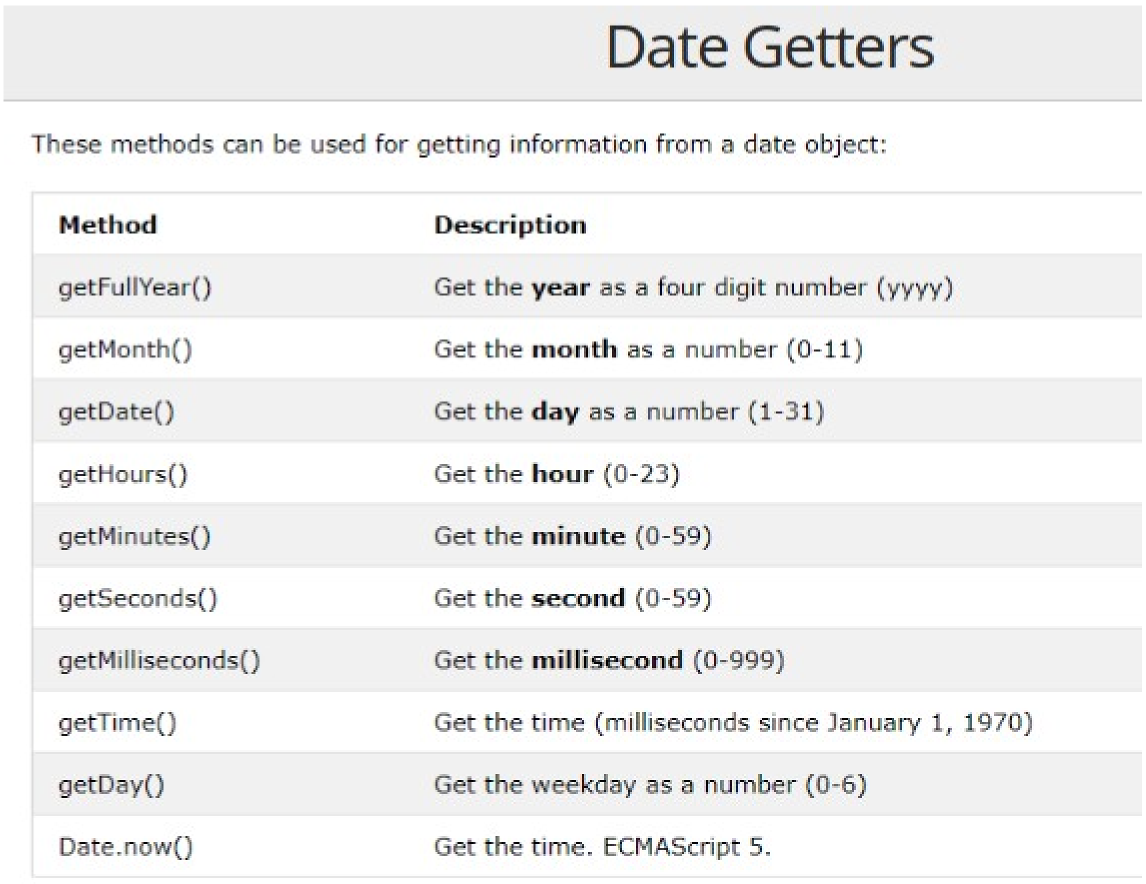
JavaScript Date Input

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.

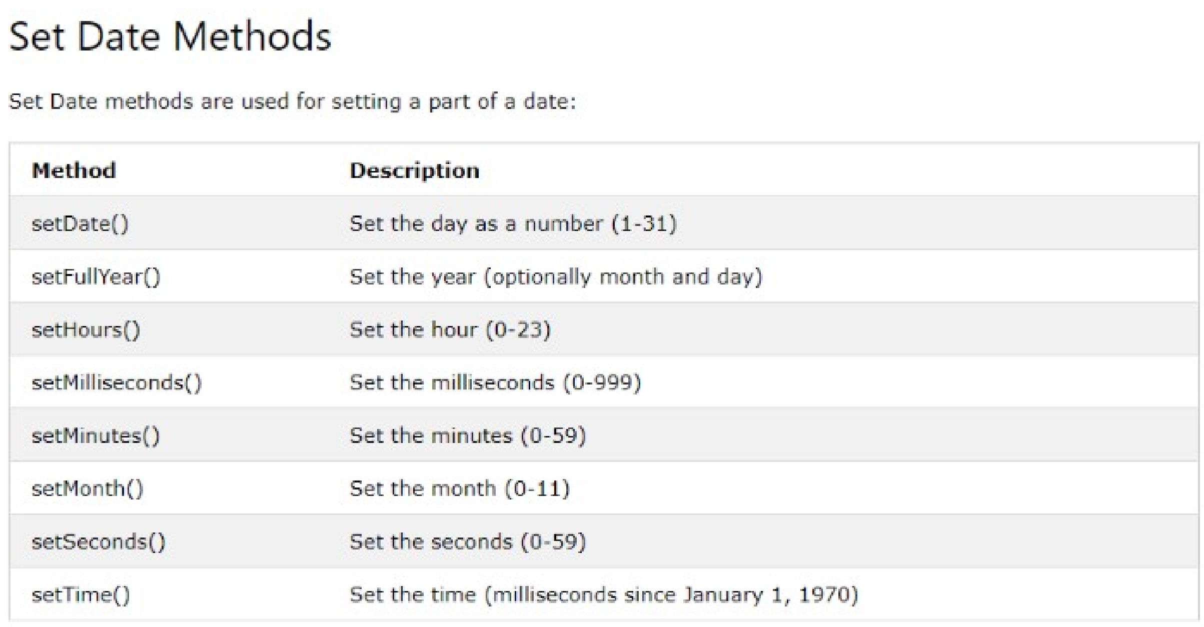
The other formats are not so well defined and might be browser specific.



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)

css Example
Css Example

css Example
Css Example

https://www.w3schools.com/js/js_operators.asp
error in function