



Javascript S Intro | Review

Después de esta lección podrás:

1. Ejecutar Js en tu página de HTML.
2. Ejecutar Js desde Node.
3. Declarar variables en JavaScript.
4. Usar operadores booleanos: **and**, **or** y **not**.
5. Comprender y aplicar los condicionales en código.
6. Aplicar **bucles** y entender la necesidad en código.
7. Manipulación de Arrays.
8. Entender qué es una función

¿Cómo añadir javascript a una página HTML?

La manera de **incluir** y ejecutar **JavaScript** en nuestra página es mediante la etiqueta `<script>`. Podemos incluir estas etiquetas en cualquier lugar de nuestra página y el navegador ejecutará las sentencias JavaScript dentro de ellas, en cuanto las lea.

```
<!DOCTYPE html>
<head>
  <meta charset="utf-8">
  <title>Es es mi blog</title>
  <link rel="stylesheet" type="text/css" href="fichero.css">
  <!-- En la proxima línea vamos a ejecutar código javascript en nuestro HTML -->
  <script type="text/javascript">console.log("Hola clase!");</script>
</head>
<body>
  <p class="estilo1">Hola soy un párrafo con estilo1</p>
</body>
</html>
```

Mediante la **etiqueta `<script>`** que acabamos de ver, podremos escribir lenguaje JavaScript directamente, aunque lo mejor es **cargar un fichero externo con el código JavaScript**. Esto nos permite

tener nuestro código más organizado.

```
<!DOCTYPE html>
<head>
  <meta charset="utf-8">
  <title>Es mi blog</title>
  <link rel="stylesheet" type="text/css" href="fichero.css">
  <!-- En la proxima linea vamos a cargar un fichero con código -->
  <script type="text/javascript" src="codigo.js"></script>
</head>
<body>
  <p class="estilo1">Hola soy un párrafo con estilo 1</p>
</body>
</html>
```

Dentro de **codigo.js** podemos incluir código JavaScript, por ejemplo:

```
var mensaje = "Hola, esto es un mensaje para imprimir en log";
console.log(mensaje);
// Todo lo que vaya precedido por '//' serán comentarios.
```

Trabajando con la consola de Chrome

Para entender cómo se ejecuta este código Javascript en el navegador **Chrome**, lo mejor es abrir la propia consola que incorpora, para poder "jugar" ahí dentro.

Debemos hacer "click" en la parte superior derecha sobre los tres puntos consecutivos en vertical: ⏮ →
Más herramientas → Herramientas para desarrolladores.



INICIA TU ESCALADA

Hacia una de las **profesiones digitales más demandadas**. En la era de la transformación digital, muchos puestos del sector tecnológico quedan vacantes cada año.

Este texto ha sido generado automáticamente por un programa de IA. No es necesario que sea completamente preciso.

Node y cómo ejecutar Javascript

Para esta sección, **crearemos** un fichero `ejemplo.js` en nuestro escritorio, lo abriremos con nuestro editor de código ([Visual Studio Code](#) si hemos seguido la guía de instalación de entorno) y **escribiremos** nuestra primera función:

```
console.log('mi primera ejecución con Node');
// Todo lo que vaya precedido por '//' serán comentarios, que no se ejecutan en JavaScript
```

Guardamos el fichero y **abrimos un terminal**, nos posicionamos en el escritorio y ejecutamos `node ejemplo.js` y ¡magia! 🎉

Declarando una variable en Javascript

Veamos un ejemplo en el que creamos variables y las imprimimos por pantalla, que podremos copiar a un `fichero.js`, para probarlo en nuestro ordenador:

```
var name = "Luis Garcia"; // Creamos una variable y la inicializamos
console.log(name);
var edad; // Creamos una variable sin inicializarla con un valor
console.log(edad); // Imprime por pantalla: undefined --> Porque la variable existe, pero no tiene valor dentro
edad = 33; // Ahora le damos valor
console.log(edad); // Imprime por pantalla: 33
```

Que no nos asuste la palabra `undefined`, simplemente es lo que nos muestra JavaScript cuando una variable todavía no tiene valor dentro, no está definido.



The screenshot shows a dark-themed instance of Visual Studio Code. In the top left, there's a tab labeled 'JS codigo.js'. The main editor area contains the following code:

```
1 var name = "Luis Garcia"; // Creamos una variable y la inicializamos
2 console.log(name);
3 var edad; // Creamos una variable sin inicializarla con un valor
4 console.log(edad); // Imprime por pantalla: undefined --> Porque la variable existe, pero no tiene valor dentro
5 edad = 33; // Ahora le damos valor
6 console.log(edad); // Imprime por pantalla: 33
```

Below the editor is a terminal window titled 'zsh'. It shows the command `node codigo.js` being run, followed by the output:
Luis Garcia
undefined
33

Operadores

Como en otras ocasiones vamos a comprobar en nuestro `fichero.js` el resultado de operar con ellos. Añadimos dos variables, asignamos un valor y realizamos tantas operaciones como queramos. Siempre dentro del `console.log` que nos imprimirá en pantalla la respuesta.

```
var numa = 5;
var numb = 2
console.log(numa + numb); // Imprime 7
console.log(numa - numb); // Imprime 3
console.log(numa * numb); // Imprime 10
console.log(numa / numb); // Imprime 2.5
```

Operador de asignación

Veamos un bloque de código con diferentes ejemplos (probar a copiarlo y ejecutarlo en vuestro [fichero.js](#)):

```
var num1 = 5;
var num2 = 10;

// asignación
num1 = num2;
console.log(num1); // num1 pasa a valer 10

// asignación y suma (num1 ya no vale el 5 inicial, recordar que ahora vale 10)
num1 += num2;
console.log(num1); // num1 pasa a ser 20

// asignación y resta
num1 -= num2;
console.log(num1); // num1 pasa a ser 10

// asignación y multiplicación
num1 *= num2;
console.log(num1); // num1 pasa a ser 100

// asignación y división
num1 /= num2;
console.log(num1); // num1 pasa a ser 10
```

Operador OR (||)

El operador **or**, representado por `||`, devuelve verdadero si uno de los valores combinados es verdadero.

```
var tengoEfectivo = true;
var tengoTarjeta = false;
var puedoPagar = tengoEfectivo || tengoTarjeta;
console.log(puedoPagar); // Devuelve true, porque tengo efectivo
```

Operador AND (&&)

El operador **and**, representado por `&&`, devuelve verdadero solo si todos los valores combinados son verdaderos.

```
var tengoCoche = false;
var tengoCarnetDeConducir = true;
var puedoConducir = tengoCoche && tengoCarnetDeConducir;
console.log(puedoConducir); // Devuelve false, porque no tengo coche
```

Operador NOT (!)

Por último, pero no menos importante, tenemos el operador **not** de negación `!`. Se usa para negar el valor de una expresión (darle el valor opuesto).

```
!true      // => false
!false     // => true
```

Ejercicio con operadores lógicos

Ahora que ya hemos visto todos los operadores, vamos a realizar algunas comprobaciones, para ello debemos copiar este código en nuestro fichero de **codigo.js** y probar a cambiar los valores de las variables, para ver las posibles combinaciones.

```
var tengoDinero = true;
var meDaMiedoVolar = true;

// Puedo ir a Mexico si tengo dinero y NO me da miedo volar
var puedoIrAMexico = tengoDinero && !meDaMiedoVolar;
console.log(puedoIrAMexico);

var meTomoUnTranquilizante = true;
// Puedo ir a Mexico si tengo dinero y NO me da miedo volar o me tomo un tranquilizante
var puedoIrAMexico = tengoDinero && (!meDaMiedoVolar || meTomoUnTranquilizante);
console.log(puedoIrAMexico);
```

Operadores de comparación

Ahora que ya sabemos cómo combinar valores booleanos, vamos a ver como compararlos entre sí. Podemos comprobar si dos valores booleanos son:

- `==` iguales
- `!=` desiguales
- `===` iguales estrictamente
- `!==` desiguales estrictamente

Para finalizar el tema de comparaciones, también podremos comparar números:

- `<` menor que ...
- `>` mayor que ...
- `<=` menor o igual que ...
- `>=` mayor o igual que ...

if - if..else

Nos va a permitir comparar **si** se cumple una condición, para tomar un camino, y **sino** tomar otro. Tan simple como eso, gracias a esta sentencia podemos dividir nuestro código en dos caminos, uno para el supuesto verdadero y otro para el falso:

```
// Edad que tenemos
var age = 15;
// Si soy mayor de edad, entonces puedo ser un Vengador
if (age < 18) {
    // Si mi edad es menor de 18
    console.log ("Vaya tendrás que ir con Spedy a jugar al parque");
} else {
    // Si mi edad es mayor de 18
    console.log ("Bienvenido Vengador");
}
```

while

```
var i = 0;

// Mientras la variable "i" sea menor o igual que 100
while (i <= 100) {
    console.log (i);
    i = i + 1; // Suma 1 a la variable i
}
```

El bucle sigue las siguientes reglas:

- Comprueba si el valor de `i` es menor o igual que 100.
- Imprime en la consola el valor de `i`.
- Incrementa el valor de la `i` en 1.

for

En caso de necesitar un bucle algo más robusto y sofisticado, usaremos la instrucción `for`. Con ella crearemos un bucle con tres valores diferentes separados por punto y coma: **inicialización**, **condición** y **expresión de incremento**. Es un poco complicado al principio pero a través de un ejemplo sencillo lo verás mejor.

Vamos a hacer el mismo ejercicio que antes, esta vez usando una declaración `for`. Imprimamos en la consola los números del 0 al 100. El código es:

```
// Inicializacion: var i = 0
// Condición: si i es menor o igual que 100
// Incremento: por cada iteración, sumale 1 al valor de 'i'
// Pista extra: i++ es lo mismo que: i = i + 1
for (var i = 0; i <= 100; i++) {
    console.log(i);
}
```

Declarar y acceder a los elementos de un Array

Pensemos en el array como una lista de elementos, cada uno con su posición (**empezando por la 0**). Para acceder a la primera posición del array, basta con poner entre corchetes el número 0.

```
var avengers = ["Hulk", "SpiderMan", "AntMan"];
```

```
var avenger = avengers[0]; // Probad a cambiar este numero ;
console.log(avenger) // Devuelve "Hulk"
```

Funciones de utilidad en los Arrays

Los array tienen distintas funciones de utilidad, que nos van a facilitar trabajar con ellos, algunos de ellos los detallaremos, aunque otros los iremos descubriendo a lo largo del bootcamp.

Propiedad length:

Si queremos conocer el número de elementos del array `avengers` se puede consultar mediante la propiedad `length`. Os dejamos un ejemplo:

```
avengers.length; // Devuelve 3
```

Manipular el array:

Al final el array actua **como una pila**, que te permite añadir datos o sacarlos. Al ser una pila de elementos, siempre se añaden o eliminan por el final:

```
var avengers = ["Hulk", "Thor", "Iron-Man"];
// Iron-Man    <-- Posicion: 2 (Elemento más alto de la pila)
// Thor        <-- Posicion: 1
// Hulk         <-- Posicion: 0 (Elemento más bajo de la pila)
```

La inserción de datos se puede hacer mediante el método **push** y la extracción mediante el método **pop**.

Método Push en Array:

Para añadir elementos en un array, usamos el método `push`, que inserta un nuevo elemento:

```
var avengers = ["Hulk", "Thor", "Iron-Man"];
avengers.push("SpiderMan", "AntMan");
console.log(avengers);
// Imprime ["Hulk", "Thor", "Iron-Man", "SpiderMan", "AntMan"]
```

A screenshot of the Visual Studio Code interface. The left sidebar shows icons for file operations like copy, search, and refresh. The main editor window has a dark theme and displays the following code:

```
var avengers = ["Hulk", "Thor", "Iron-Man"];
avengers.push("SpiderMan", "AntMan");
console.log(avengers);
// Imprime ["Hulk", "Thor", "Iron-Man", "SpiderMan", "AntMan"]
```

The bottom bar shows tabs for COMMENTS, PROBLEMS, TERMINAL, and others. The TERMINAL tab is active, showing the command `node codigo.js` and its output: `['Hulk', 'Thor', 'Iron-Man', 'SpiderMan', 'AntMan']`.

Método Pop en Array:

Para eliminar elementos de un array, usamos el método `pop`, que elimina el último elemento:

```
var avengers = ["Hulk", "Thor", "Iron-Man", "SpiderMan", "AntMan"];
avengers.pop();
console.log(avengers);
// Imprime ["Hulk", "Thor", "Iron-Man", "SpiderMan"]
```

A screenshot of the Visual Studio Code interface. The left sidebar shows icons for file operations like copy, search, and refresh. The main editor window has a dark theme and displays the following code:

```
var avengers = ["Hulk", "Thor", "Iron-Man", "SpiderMan", "AntMan"];
avengers.pop();
console.log(avengers);
// Imprime ["Hulk", "Thor", "Iron-Man", "SpiderMan"]
```

The bottom bar shows tabs for COMMENTS, PROBLEMS, TERMINAL, and others. The TERMINAL tab is active, showing the command `node codigo.js` and its output: `['Hulk', 'Thor', 'Iron-Man', 'SpiderMan']`.

Método Sort:

Este método se utiliza para **ordenar** un array. En caso de que tengamos un array con elementos string, pues estos serán ordenados **alfabéticamente**:

```
var avengers = ["Hulk", "Thor", "Iron-Man", "SpiderMan", "AntMan"];
avengers.sort();
console.log(avengers);
// Imprime ['AntMan', 'Hulk', 'Iron-Man', 'SpiderMan', 'Thor']
```

The screenshot shows a dark-themed instance of Visual Studio Code. A file named 'codigo.js' is open in the editor, containing the following code:

```
1 var avengers = ["Hulk", "Thor", "Iron-Man", "SpiderMan", "AntMan"];
2 avengers.sort();
3 console.log(avengers);
4 // Imprime ['AntMan', 'Hulk', 'Iron-Man', 'SpiderMan', 'Thor']
```

The terminal at the bottom shows the command 'node codigo.js' being run, with the output '['AntMan', 'Hulk', 'Iron-Man', 'SpiderMan', 'Thor']' displayed.

En caso de ser números, lo ordenará de menor a mayor:

The screenshot shows a dark-themed instance of Visual Studio Code. A file named 'codigo.js' is open in the editor, containing the following code:

```
1 var numbers = [2, 8, 6, 1, 5];
2 numbers.sort();
3 console.log(numbers);
```

The terminal at the bottom shows the command 'node codigo.js' being run, with the output '[1, 2, 5, 6, 8]' displayed.

Trabajando con arrays, bucles y condicionales

Hemos visto cómo manipular un array, pero siempre desde el último elementos. Pero ¿y si queremos manipular otro dato que no sea el último?

Pues para eso os vamos a enseñar a recorrer un Array, haciendo uso de los bucles aprendidos anteriormente, en concreto un bucle **for**:

```
var avengers = ["Thor", "Spiderman", "Iron-Man", "Hulk"];  
  
for (var i = 0; i <= avengers.length; i++) {  
    console.log(avengers[i]); // Imprime los nombres de nuestros vengadores según la posición de i  
}
```

Ahora introduzcamos un condicional (también aprendido en la sección anterior). Recorramos el array y imprimamos un mensaje, pero solo para **Hulk**:

```
var avengers = ["Thor", "Spiderman", "Iron-Man", "Hulk"];  
  
for (var i = 0; i <= avengers.length; i++) {  
    if(avengers[i] == 'Hulk') {  
        console.log('Este vengador se llama' + avengers[i] + ' y es el más fuerte');  
    };  
}
```

Y finalmente, vamos a sustituir a **Spiderman** por otro vengador:

```
var avengers = ["Thor", "Spiderman", "Iron-man", "Hulk"];  
  
for (var i = 0; i <= avengers.length; i++) {  
    if(avengers[i] == 'Spiderman') {  
        avengers[i] = 'AntMan';  
    };  
}  
  
console.log(avengers);
```



```
codigo.js  
JS codigo.js ×  
Users > marioantoniopolodaza > Desktop > JS codigo.js > ...  
1 var avengers = ["Thor", "Spiderman", "Iron-man", "Hulk"];  
2  
3 for(var i = 0; i <= avengers.length; i++) {  
4     if(avengers[i] == 'Spiderman') {  
5         avengers[i] = 'AntMan';  
6     };  
7 }  
8  
9 console.log(avengers);  
  
COMMENTS PROBLEMS TERMINAL ... 1: zsh + ┌ ┐ ^ x  
... ┌ ┐ 14:48 100% ┌ ┐  
└ ┘ node codigo.js └ ┘ [ 'Thor', 'AntMan', 'Iron-man', 'Hulk' ]
```

Qué es una función?

Una función se define para a resolver una **tarea concreta**.

```
console.log('mi primera función');
```

En el ejemplo anterior, vemos una función para **imprimir** un mensaje. Sencillo, ¿no? 🎉

Pensad cuando tratamos de resolver un ejercicio de programación, escribiendo código en varias líneas... ¿y si existiera alguna manera de **dividir** en "trocitos" todo ese código? (para que no esté todo en un bloque inmenso). Pues en esta sección aprenderemos a **separar** nuestro código por funcionalidades.

Volviendo al ejemplo inicial, tenemos la función `console.log` para imprimir por pantalla, y la podemos usar todas las veces que queramos. Imaginad que ahora quisiéramos una función para sumar dos números:

```
// Vamos a crear nuestra primera función!
function sum(num1, num2) {
  return num1 + num2;
}
// Ya tenemos declarada nuestra función para sumar dos números

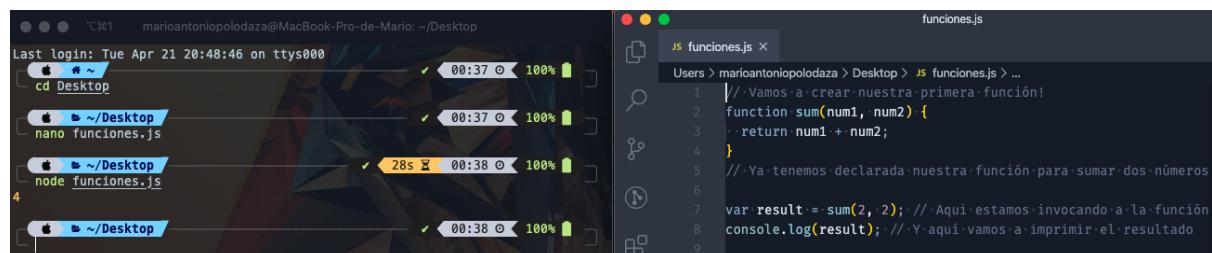
var result = sum(2, 2); // Aquí estamos invocando a la función (ejecutándola)
console.log(result); // Y aquí vamos a imprimir el resultado
```

Analicemos este código:

- `function` es la palabra mágica (sentencia reservada) para crear **funciones**
- `sum` es el **nombre** que le doy a la función (hay que procurar darle un nombre descriptivo)
- `(num1, num2)` a esto se le llama **parámetros**, y es lo que **entra** en nuestra función
- `return` es la palabra mágica (reservada) para **devolver** el resultado, la **salida** de la función
- `{ }` entre los corchetes irá el cuerpo de la función, donde operamos

Tras la declaración de la función, vemos la línea en la que hacemos uso de ella `sum(2, 2)`, es fácil, basta con escribir su **nombre** y ponerle los **paréntesis** con los valores de **entrada**.

Ahora copia y pega el bloque de código en tu fichero `funciones.js` y ejecútalo con **Node**. El resultado es 4 (hemos conseguido sumar 2 + 2 😊).



Asignación de funciones

Existe otra forma de crear y nombrar funciones, que es **asignando** funciones a **variables**. Aquí tenemos un ejemplo:

```
var sayHello = function() {
    return "Hello!";
};

sayHello(); // Devuelve "Hello!"
```

Funciones definidas en Javascript

En anteriores secciones ya hemos usado algunas funciones, lo que pasaba es que NO éramos conscientes de ello, pero ¿cómo es esto posible, si yo no recuerdo haber creado ninguna? 🤯

Es fácil, existe un conjunto de funciones que ya vienen definidas en Javascript, existen en el estándar. Al principio de esta sección usábamos la función `console.log()`, y si nos paramos a pensar, es la función con nombre **log** creada dentro de la consola de Javascript, que sirve para imprimir por pantalla.

Os invitamos a repasar anteriores bloques, en búsqueda de funciones que hemos ido usando, aunque para hacéroslo más fácil, os dejamos aquí descripciones algunas de las usadas:

```
var test = "Test";
console.log(test); // Imprime por pantalla "Test"
var res = test.indexOf("T"); // Devuelve la posición de la letra "T" en el string "Test" -> 0
console.log(res);
var res2 = test.includes("st"); // Indica si existe la cadena "st" en el string "Test" -> true
console.log(res2);
```

