

# Estructura del lenguaje javascript.



## Caso práctico

En **BK Programación** han decidido ponerse manos a la obra y **Antonio** comienza a estudiar los **fundamentos del lenguaje JavaScript**, (siempre bajo la tutoría de **Juan**, que estará ahí para ayudarle en todo momento).

En la unidad anterior **Antonio** analizó las posibilidades de los lenguajes de script, decidió qué lenguaje emplearía para la programación en el entorno cliente y vio cómo insertar en la página HTML dicho lenguaje de script.

Cómo han decidido emplear el lenguaje Javascript, lo primero que hará **Antonio** es ver los fundamentos de dicho lenguaje, y cuál es la estructura básica para comenzar a programar lo más pronto posible.

**Ada** está muy entusiasmada con este proyecto y está convencida de que **Antonio** será capaz de hacer ese trabajo sin ningún problema.



En esta unidad se hace una introducción a los fundamentos de JavaScript. Para ello se hará hincapié en la definición de variables, los tipos de datos soportados en JavaScript, conversiones, los operadores y terminaremos con las estructuras de control y bucles.

Al final de la unidad realizaremos nuestro primer ejemplo de JavaScript poniendo en práctica algunos de los contenidos aquí vistos.



[Ministerio de Educación y Formación Profesional](#) (Dominio público)

**Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.**

[Aviso Legal](#)

# 1.- Fundamentos de javascript.



## Caso práctico

**Juan** habla con **Antonio** y le recomienda leer las bases y fundamentos de JavaScript, en los cuáles se explica cómo se ponen comentarios en el código, cómo se definen variables, estructuras de control, etc.

Éste va a ser uno de los primeros pasos que **Antonio** tendrá que dar para el estudio de cualquier lenguaje de programación, y en especial de JavaScript.



Como creo que a cualquiera, a mí me gusta saber de dónde venimos y, en la medida de lo posible, hacia dónde nos dirigimos. Por ello, te muestro esta línea de tiempo en la que se incluyen los principales hitos en el desarrollo y estandarización de JavaScript.

[Mostrar](#)

### Evolución de JavaScript

Mayo de 1995

Septiembre de 1995

Año 1997

Año 1998

Año 1999

Año 2011

Año 2015

Y ECMAScript no para de evolucionar y ya se está intentando estandarizar la versión 7. Pero nosotros nos atendremos a la versión ES6 que es la que actualmente soportan todos los navegadores modernos.

Pero, seguramente, algunos os estaréis preguntando: ¿qué es JavaScript o ECMAScript y qué nos aporta?

La respuesta es fácil: es simplemente otra capa más en el ciclo de vida del desarrollo de nuestras páginas web. Como bien ya conocéis, hasta ahora estabais utilizando una capa para la estructura y otra para la presentación; pues JavaScript añade otra capa para el desarrollo web que es la del **comportamiento**.

- ✓ **HTML**: es el lenguaje que define la estructura para dar un sentido al contenido web,.
- ✓ **CSS**: es el lenguaje utilizado para formatear cada uno de los elementos o grupos de elementos.
- ✓ **JavaScript**: Es un lenguaje de programación que te permite controlar el comportamiento de tu página web. ¡No sabes lo que podrás lograr utilizando JavaScript!

Pasemos ya a ver cómo utilizar JavaScript, que seguro que estás impaciente.



## Autoevaluación

¿Quién desarrolló JavaScript?

- ☐ Microsoft.
- ☐ Oracle Corporation.
- ☐ Brendan Eich.

No es correcta, porque Microsoft creó JScript el cuál está basado en JavaScript.

Incorrecta, porque Oracle es la empresa propietaria de la licencia de JavaScript actualmente.

Muy bien. El nombre original fue Mocha, luego LiveScript, para pasar a ser finalmente JavaScript.

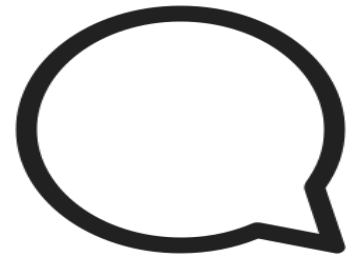
### Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta


## 1.1.- Comentarios en el código.

---

A la hora de programar en cualquier lenguaje de programación, es muy importante que comentes tu código. Aunque hoy día se aconseja que el propio código sea el que esté autodocumentado. Es decir, si una serie de sentencias realizan una determinada acción o función, es mejor y más claro crear una función para dicho propósito con un identificador lo suficientemente claro, que poner el comentario en el código. Pero todo eso lo veremos más adelante.



[Lycopene579](#) (Dominio público)

Los comentarios son sentencias que el  intérprete de JavaScript ignora. Sin embargo, estas sentencias permiten a los desarrolladores dejar notas sobre cómo funcionan las cosas en sus scripts.

Los comentarios ocupan espacio dentro de tu código de JavaScript, por lo que cuando alguien se descargue vuestro código necesitará más o menos tiempo, dependiendo del tamaño de vuestro fichero.

JavaScript permite dos estilos de comentarios: de una sola línea o de varias. Ejemplos de comentarios de una única línea o de varias líneas:

```
// Mi primer comentario

/*
    Mi segundo comentario,
    éste un poco más largo.
*/
```

## 1.2.- Constantes y variables.

Una **variable** es un contenedor de un valor de un tipo dado al que le damos un identificador. El valor de una variable puede cambiar en el tiempo. También existen las **constantes** que, al contrario de las variables, su valor no puede cambiar.

Para nombrar una variable o constante sólo podemos utilizar caracteres alfanuméricos [a-zA-Z0-9] y el caracter de subrayado `_`. No debe comenzar por un número. Tampoco podemos utilizar como identificador una palabra reservada ya que nos daría error. Es conveniente que los nombres, o los identificadores que les damos, sean descriptivos e informen del cometido de dicha variable para que nuestro código sea legible y entendible. También es conveniente seguir la convención [lower camel case](#) en la que se utilizan minúsculas para nombrarlas y si es la unión de varias palabras éstas se unen y se distingue entre palabras utilizando la primera letra en mayúscula.



[Ciker-Free-Vector-Images \(Pixabay License\)](#)

Antes de utilizar una variable deberemos declararla y luego darle un valor, ya que si no su valor será `undefined`.

Para declarar constantes y variables debemos utilizar las palabras reservadas `const`, `var` y `let` (la primera para las constantes y las otras para las variables en las que ahora nos detendremos).

```
const pi = 3.14;
var edad = 18;
let nombre = "Bob Esponja";
```

La palabra **var** declara una variable y se sigue utilizando por motivos históricos ya que genera algunos problemas que viene a solucionar **let**.

Debido al **hoisting**, era posible declarar una variable con `var` después de ser utilizada y eso funcionaba. Por ejemplo:

```
edad = 18;
...
...
var edad;
```

Ese código era válido, pero la verdad es que se hace confuso y más complejo de entender. Pues el **hoisting** ya no funciona con `let`, por lo que para utilizar una variable que vamos a declarar con `let`, primero debemos declararla o de lo contrario nos dará un error.

```
let edad;
...
edad = 18;
```

Además con el uso de `var`, una variable se podía declarar varias veces y era correcto, pero como ves en el ejemplo vuelve a ser confuso.

```
var edad = 18;  
....  
var edad = 20;
```

Eso con `let` ya no funciona y habría que hacerlo de la siguiente forma:

```
let edad = 18;  
....  
edad = 20;
```

JavaScript es un lenguaje "débilmente tipado", lo cual significa que, a diferencia de otros lenguajes, no es necesario especificar el tipo de dato que vamos a almacenar en una variable y además éste puede cambiar. Esto deberíamos evitarlo para evitar confusiones.

```
let edad;  
edad = '18';    // En este caso contiene una cadena  
edad = 18;      // En este caso contiene un número
```



## Autoevaluación

Para definir una variable hoy día es más conveniente utilizar `var` que `let`. ¿Verdadero o falso?

☐ Verdadero ☐ Falso

**Falso**

Es mejor que nos acostumbremos a utilizar siempre `let` por las ventajas que hemos mencionado.

## 1.3.- Tipos de datos.

Las variables en JavaScript podrán contener cualquier tipo de dato. A continuación, se muestran los tipos de datos soportados en JavaScript:

- ✓ **Números:** Pueden almacenar tanto números enteros como números decimales. Para expresar el valor de un número decimal se utiliza el separador ..
- ✓ **Cadenas:** Las cadenas de texto son una secuencia de caracteres. Para expresar su valor, éstas se pueden encerrar entre comillas simples ' o comillas dobles ''.
- ✓ **Valores lógicos:** Pueden almacenar los valores `true` o `false` y se suelen utilizar en las sentencias condicionales o ser el resultado de una operación de comparación como luego veremos.
- ✓ **Arrays:** son objetos que contienen diferentes valores, que pueden ser de diferentes tipos, indexados por su índice, comenzando en 0. Para declararlos utilizamos los corchetes y para acceder a sus valores utilizamos su índice encerrado entre corchetes:



[Everaldo Coelho](#) (GNU/GPL)

```
let personajes = ['Bob Esponja', 'Calamardo', 'Patricio'];
console.log(personajes[0]); // Muestra por consola: Bob Esponja
```

Ya hablaremos más de ellos.

- ✓ **Objetos:** Por ahora nos quedaremos con que un objeto es un tipo de dato que puede contener diferentes propiedades que lo definen, aunque esto no es del todo cierto. Para definir un objeto utilizamos las llaves y expresamos cada uno de sus atributos indicando el nombre y su valor separados por dos puntos.

```
let personaje = { nombre: 'Bob Esponja', edad: 5 };
console.log(personaje.nombre); // Muestra por consola: Bob Esponja
console.log(personaje[nombre]); // Muestra por consola: Bob Esponja
```

- ✓ **Funciones:** Esto lo veremos más adelante, así que por ahora no os liaré mas.



### Autoevaluación

Los arrays en javascript comienzan en índice 1. ¿Verdadero o falso?

☐ Verdadero ☐ Falso

**Falso**

Los arrays en JavaScript comienzan en el índice 0.

## 1.3.1.- Conversiones de tipos de datos.

Aunque los tipos de datos en JavaScript son muy sencillos, a veces te podrás encontrar con casos en los que las operaciones no se realizan correctamente, y eso es debido a la conversión de tipos de datos. JavaScript intenta realizar la mejor conversión cuando realiza esas operaciones, pero a veces no es el tipo de conversión que a ti te interesaría.



Por ejemplo cuando intentamos sumar dos números:

```
console.log(4 + 7); // Muestra por consola: 11
```

[Everaldo Coelho](#) (GNU/GPL)

Si uno de esos números está en formato de cadena de texto, JavaScript lo que hará es intentar convertir el otro número a una cadena y los concatenará, por ejemplo:

```
console.log(4 + '7'); // Muestra por consola: 47 (como cadena)
```

Otro ejemplo podría ser:

```
console.log(1 + 2 + '3'); // Muestra por consola: 33 (como cadena)
```

Esto puede resultar ilógico pero sí que tiene su lógica. La expresión se evalúa de izquierda a derecha. La primera operación funciona correctamente devolviendo el valor de 3 pero al intentar sumarle la cadena de texto 3, JavaScript lo que hace es convertir ese número a una cadena de texto y se lo concatenará al otro 3.

Para convertir cadenas a números dispones de las funciones: `parseInt()` y `parseFloat()`.

Por ejemplo:

```
console.log(parseInt('34')); // Muestra por consola: 34
console.log(parseInt('89.76')); // Muestra por consola: 89
console.log(parseFloat('34')); // Muestra por consola: 34
console.log(parseFloat('3.14')); // Muestra por consola: 3.14
console.log(1 + 2 + parseInt('3')); // Muestra por consola: 6
```




Si lo que deseas es realizar la conversión de números a cadenas, es mucho más sencillo, ya que simplemente tendrás que concatenar una cadena vacía al principio, y de esta forma el número será convertido a su cadena equivalente:

```
console.log('' + 2020); // Muestra por consola: 2020 (como cadena);
```



## 1.4.- Operadores.

JavaScript es un lenguaje rico en operadores: símbolos y palabras que realizan operaciones sobre uno o varios valores, para obtener un nuevo valor.

Cualquier valor sobre el cuál se realiza una acción (indicada por el operador), se denomina un operando. Una **expresión** puede contener un operando y un operador (denominado operador , como por ejemplo en `b++`, o bien dos operandos, separados por un operador (denominado operador , como por ejemplo en `a + b`). Incluso existe algún operador , como luego veremos.



[Everaldo Coelho](#) (GNU/GPL)

### Categorías de operadores en JavaScript

Tipo	Qué realizan
Comparación.	<p>Comparan los valores de 2 operandos, devolviendo un resultado de <code>true</code> o <code>false</code> (se usan extensivamente en sentencias condicionales).</p> <pre>== != === !== &gt; &gt;= &lt; &lt;=</pre>
Aritméticos.	<p>Unen dos operandos para producir un único valor que es el resultado de una operación aritmética u otra operación sobre ambos operandos.</p> <pre>+ - * / % ++ -- +valor -valor</pre>
Asignación.	<p>Asigna el valor a la derecha de la expresión a la variable que está a la izquierda.</p> <pre>= += -= *= /= %= &lt;&lt;= &gt;= &gt;&gt;= &gt;&gt;&gt;= &amp;=  = ^= []</pre>
Lógicos.	<p>Realizan operaciones lógicas sobre uno o dos operandos lógicos.</p> <pre>&amp;&amp;    !</pre>
Bit a Bit.	<p>Realizan operaciones aritméticas o de desplazamiento de columna en las representaciones binarias de dos operandos.</p> <pre>&amp;   ^ ~ &lt;&lt; &gt;&gt; &gt;&gt;&gt;</pre>

Tipo	Qué realizan
Objeto.	<p>Ayudan a los scripts a evaluar la herencia y capacidades de un objeto particular antes de que tengamos que invocar al objeto y sus propiedades o métodos.</p> <pre>. [] () delete in instanceof new this</pre>
Otros.	<p>Operadores que tienen un comportamiento especial.</p> <pre>, ?: typeof void</pre>



## Debes conocer

En el siguiente enlace podrás encontrar más información sobre los operadores de JavaScript, por lo que no nos detendremos más en ellos, aunque sí veremos algunos ejemplos que merecen la pena destacar

[Operadores en JavaScript](#)

## 1.4.1.- Algunos operadores de interés.

### Plantillas de cadenas (template string)

Con ES6 podemos definir cadenas de una forma más sencilla que como estábamos haciendo hasta ahora. Fíjate en este ejemplo, en el que para utilizar la plantilla de cadenas utilizamos la comilla invertida ``` y ponemos las variables encerradas entre `${}`.



[Everaldo Coelho](#) (GNU/GPL)

```
let nombre = 'Bob Esponja';
console.log(`Hola ${nombre}`); // Muestra por consola: Hola Bob Esponja
```

En ES6 también podemos definir cadenas en varias líneas, sin necesidad de utilizar el operador de concatenación `+`.

### Desestructuración

En ES6 también tenemos nuevas formas de asignar valores a variables a partir de un array.

```
let [personaje1, personaje2] = ['Bob Esponja', 'Calamardo'];
console.log(personaje1); // Muestra por consola: Bob Esponja
console.log(personaje2); // Muestra por consola: Calamardo
```

Esa misma desestructuración también la podemos hacer con objetos.

```
let personaje = { nombre: 'Bob esponja', edad: 18 };
let { nombre, edad } = personaje;
console.log(nombre); // Muestra por consola: Bob Esponja
console.log(edad); // Muestra por consola: 18
```

### ?:

Este operador condicional es la forma reducida de la expresión `if - else`.

La sintaxis formal para este operador condicional es:

`(condicion) ? expresionV : expresionF;`

Si la condición evalúa a `true` devuelve `expresionV` y en caso contrario devuelve `expresionF`.

```
let sexo = 'V';
console.log(`Su sexo es ${sexo ? 'varón' : 'hembra'}`);
```

### typeof

Este operador unario se usa para identificar el tipo de una variable o expresión.

```
let indefinido;
let cadena = '5'
let numero = 5;
```

```
console.log(typeof indefinido);    // Muestra por consola: undefined
console.log(typeof cadena);        // Muestra por consola: string
console.log(typeof numero);        // Muestra por consola: number
```

## 1.5.- Condiciones y bucles.

---

En esta sección te mostraremos cómo los programas pueden tomar decisiones, y cómo puedes lograr que un script repita un bloque de instrucciones las veces que quieras.

Cuando te levantas cada día tomas decisiones de alguna clase; muchas veces ni te das cuenta de ello, pero lo estás haciendo. Por ejemplo, imagínate que vas a hacer la compra a un supermercado; desde el momento que entras en el supermercado ya estás tomando decisiones: ¿compro primero la leche o compro la verdura?, ¿ese precio es barato o es caro?, ¿el color de ese tinte es azul claro u oscuro?, ¿tengo suficiente dinero para pagar o no?, ¿me llegan estos kilogramos de patatas o no?, ¿pago en efectivo o con tarjeta?, etc.



[Everaldo Coelho](#) (GNU/GPL)

Otras veces repites un ejercicio físico un número determinado de veces para estar en forma. O realizas cambios a un documento hasta que crees que está perfecto. O preguntas a tus amigos mientras uno no te responda que él es el que tiene ese libro que tanto querías leer.

Es decir, tomamos innumerables decisiones a lo largo del día y la mayor parte de las veces no nos damos ni cuenta de ello. Pero también realizamos acciones repetitivas sin casi darnos ni cuenta.

En las siguientes secciones, verás cómo puedes ejecutar unas u otras instrucciones, dependiendo de ciertas condiciones, y cómo puedes repetir una o varias instrucciones, las veces que te hagan falta.

## 1.5.1.- Estructuras de control.

En los lenguajes de programación, las instrucciones que te permiten controlar las decisiones y bloques de ejecución, se denominan "Estructuras de Control". Una estructura de control, dirige el flujo de ejecución a través de una secuencia de instrucciones, basadas en decisiones simples y en otros factores.

Una parte muy importante de una estructura de control es la "condición". Cada condición es una expresión que se evalúa a `true` o `false`.

### Sentencia `if`

La decisión más simple que podemos tomar en un programa, es la de seguir una rama determinada si una determinada condición evalúa a `true`.



[Peggy\\_Marco \(Pixabay License\)](#)

```
if (condicion) {  
    sentencias;  
}
```

Si la condición evalúa a `true` se ejecutarán las sentencias dentro del bloque encerrado entre llaves.

```
if (edad >= 18) {  
    alert('Ya eres mayor de edad');  
}
```

### Sentencia `if - else`

En este tipo de construcción, decidimos si ejecutar una secuencia u otro en función de que la condición evalúe a `true` o `false`.

```
if (condicion) {  
    sentenciasV;  
} else {  
    sentenciasF;  
}
```

Si la condición evalúa a `true` ejecutamos `sentenciasV` y en caso contrario ejecutamos `sentenciasF`.

```
if (edad >= 18) {  
    alert('Ya eres mayor de edad');  
} else {  
    alert('Aún eres menor de edad');  
}
```

Recuerda que también existía el `operador ternario`.



## Autoevaluación

Es obligatorio que en una sentencia **if** siempre utilicemos la cláusula **else**.  
¿Verdadero o falso?

☐ Verdadero ☐ Falso

**Falso**

La cláusula **else** puede estar presente o no.

## 1.5.2.- Bucles.

Los bucles son estructuras repetitivas, que se ejecutarán un número de veces fijado expresamente, o que dependerá de si se cumple una determinada condición.

### Bucle `for`.

Este tipo de bucle te deja repetir un bloque de instrucciones un número limitado de veces.



[OpenClipart-Vectors \(Pixabay License\)](#)

```
for (expresión inicial; condicion; incremento) {  
    sentencias;  
}
```

Primero se ejecuta la expresión inicial o de inicialización. En cada repetición del bucle se evalúa la condición y si ésta evalúa a `true`, se ejecutan las instrucciones, se ejecuta la sentencia de incremento y se repite el proceso de evaluación, ejecución e incremento hasta que la condición evalúe a `false`.

```
let numero = 5;  
for (let i = 0; i <= 10; i++) {  
    console.log(numero * i);  
}
```

### Bucle `while`.

Este tipo de bucles se utilizan cuando queremos repetir la ejecución de unas sentencias un número indefinido de veces, siempre que se cumpla una condición. Es más sencillo de comprender que el bucle `for`, ya que no incorpora en la misma línea la inicialización de las variables, su condición para seguir ejecutándose y su actualización. Sólo se indica, como veremos a continuación, la condición que se tiene que cumplir para que se realice una iteración o repetición.

```
while (condicion) {  
    sentencias;  
}
```

Se ejecutan las sentencias mientras la condición evalúe a `true`. Por tanto, dentro de dichas sentencias deberá darse la situación para que en algún momento la condición evalúe a `false` o de lo contrario estaríamos ante un **bucle infinito**.

```
let numero = 5;  
let i = 0;  
while (i <= 10) {  
    console.log(numero * i);  
    i++;  
}
```

### Bucle `do - while`.

Este tipo de bucle es la última de las estructuras para implementar repeticiones de las que dispone JavaScript, y es una variación del bucle `while` visto anteriormente. Se utiliza generalmente, cuando no



sabemos el número de veces que se habrá de ejecutar el bucle. Es prácticamente igual que el bucle `while`, con la diferencia, de que sabemos seguro que el bucle por lo menos se ejecutará una vez.

```
do {  
    sentencias;  
} while (condicion);
```

Al igual que en el caso anterior, siempre debe darse que las sentencias en el algún momento hagan que la condición evalúe a `false`, para evitar el bucle infinito.

```
let numero = 5;  
let i = 0;  
do {  
    console.log(numero * i);  
    i++;  
} while (i <= 10);
```

También existen otras funciones para recorrer colecciones u objetos iterables que veremos más adelante (`foreach`, `map`, `filter`, `reduce`, ...).



## Autoevaluación

**Un bucle `for` es imposible que se convierta en un bucle infinito ya que lleva implícita la sentencia de incremento. ¿Verdadero o falso?**

☐ Verdadero ☐ Falso

**Falso**

Eso depende de la condición y la sentencia de incremento y si no lo hacemos bien sí es posible que se convierta en un bucle infinito.

## 1.6.- Ejemplo sencillo con JavaScript.

En este apartado pretendo mostrarte un ejemplo completo y totalmente funcional de lo visto hasta ahora.

Cierto es, que para entender el ejemplo tendrás que hacer algunos actos de fe, ya que hay cosas que aún no hemos explicado, pero que creo que son fácilmente entendibles. La idea es que vayas familiarizándote con el entorno que utilices, con la forma de trabajar y sobre todo con la forma de pensar.

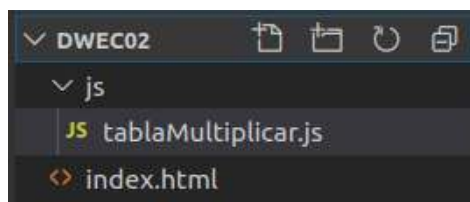


[Everaldo Coelho \(GNU/GPL\)](#)

Si hay alguna cosa que aún no entiendas no te desespere, pregunta en el foro, que encantado te responderé a tus dudas. Pero lo que sí quiero es que te centres en las cosas que ya conoces y cómo y por qué las he utilizado. Os animo a todos y todas a que implementéis este ejemplo, utilizando el editor o entorno de vuestra preferencia, lo visualices en el navegador o navegadores que tenéis instalados y que vayáis familiarizándolos con todo.

El ejemplo es muy simple. Un formulario en el que se pide al usuario que introduzca un número entre 0 y 10, ambos inclusive, y una vez que envíe el formulario le muestra la tabla de multiplicar para dicho número introducido.

Veamos primero cómo he estructurado el proyecto.



Visual Code Studio (Elaboración propia)

Una vez vista la estructura, pasemos a ver el formulario y la inclusión de nuestro fichero .js en el archivo `index.html`.

```
1 <!DOCTYPE html>
2 <html lang="es-ES">
3   <head>
4     <meta charset="UTF-8">
5     <title>Tabla de multiplicar</title>
6     <script src="js/tablaMultiplicar.js"></script>
7   </head>
8   <body>
9     <h1>
10      Tabla de multiplicar
11    </h1>
12    <form onsubmit="mostrarTabla()">
13      <label for="numero">Introduce el número:</label>
14      <input id="numero" type="number" required/>
15      <input type="submit" value="Mostrar tabla" />
16    </form>
17    <div id="tabla">
18    </div>
19  </body>
20 </html>
```

Como podéis apreciar, simplemente incluimos el archivo `tablaMultiplicar.js` para que su contenido esté accesible. Creamos la estructura del documento, en la que básicamente creamos dos elementos:

- ✔ Un formulario para que el usuario introduzca el número del que quiere visualizar su tabla de

multiplicar. Este formulario, cuando es enviado (aunque no es enviado a ningún lado, simplemente se valida en lado del cliente), mediante el evento `onSubmit` del mismo, simplemente llama a la función `mostrarTabla()` que está definida en nuestro archivo `tablaMultiplicar.js`.

- ✓ Un bloque vacío cuyo `id` es `tabla` y en el que luego mostraremos los resultados.

Por otro lado la función `mostrarTabla()` del archivo `tablaMultiplicar.js`.

```
1  const mostrarTabla = () => {
2    this.event.preventDefault();
3    const numero = Number(document.getElementById('numero').value);
4    if (numero >= 0 && numero <= 10) {
5      let tabla = document.getElementById('tabla');
6      let tablaMultiplicar = `<h2>Tabla de multiplicar del número ${numero}</h2>`;
7      tablaMultiplicar += '<ul>';
8      for (let i = 0; i <= 10; i++) {
9        tablaMultiplicar += `<li>${numero} * ${i} = ${numero * i}</li>`;
10     }
11     tablaMultiplicar += '</ul>';
12     tabla.innerHTML = tablaMultiplicar;
13   } else {
14     alert('El número introducido debe estar entre 0 y 10 (ambos inclusive');
15     document.getElementById("numero").value = '';
16   }
17 }
```

Esta función realiza las siguientes acciones.

- ✓ Primero de todo previene que el evento se propague. Todo eso lo entenderás mejor más adelante, pero si no lo hiciese, al terminar limpia los campos y no nos deja ver el resultado (te animo a que comentes dicha línea -línea número 2- para que veas lo que sucede).
- ✓ Pasa el valor del campo de texto a número y luego comprueba mediante una sentencia condicional si está entre los valores aceptados o no.
  - ➡ Si lo está, rellena lo que se quiere mostrar en el bloque cuyo `id` es `tabla` mediante un bucle.
  - ➡ Si no lo está, alerta al usuario del error y elimina el contenido del campo de texto.

La visualización del mismo en el navegador, es el que muestro a continuación, aunque te animo a que lo visualices tú en el tuyo.

## Tabla de multiplicar

Introduce el número:

### Tabla de multiplicar del número 5

- 5 \* 0 = 0
- 5 \* 1 = 5
- 5 \* 2 = 10
- 5 \* 3 = 15
- 5 \* 4 = 20
- 5 \* 5 = 25
- 5 \* 6 = 30
- 5 \* 7 = 35
- 5 \* 8 = 40
- 5 \* 9 = 45
- 5 \* 10 = 50



## Autoevaluación

El uso del bucle `for` en el ejemplo es el más adecuado. ¿Verdadero o falso?

☒ Verdadero ☐ Falso

### Verdadero

En este caso sabemos las iteraciones justas que debe dar el bucle, por que sí sería el más adecuado, aunque se podría haber llevado a cabo con cualquiera de los otros.

