

## EXERCISES QUE TRABAJAREMOS EN EL CUE

- EXERCISE 1: FUNCIONES DE JAVASCRIPT - ¿COMO SON? ¿CÓMO SE USAN?
- EXERCISE 2: RETORNANDO VALORES MEDIANTE UNA FUNCIÓN EN JAVASCRIPT.

## EXERCISE 1: FUNCIONES DE JAVASCRIPT - ¿COMO SON? ¿CÓMO SE USAN?

En el siguiente ejemplo construiremos funciones en JavaScript. Para empezar, abriremos nuestro editor de texto, y crearemos un nuevo archivo HTML en el que incrustaremos nuestro código JavaScript.

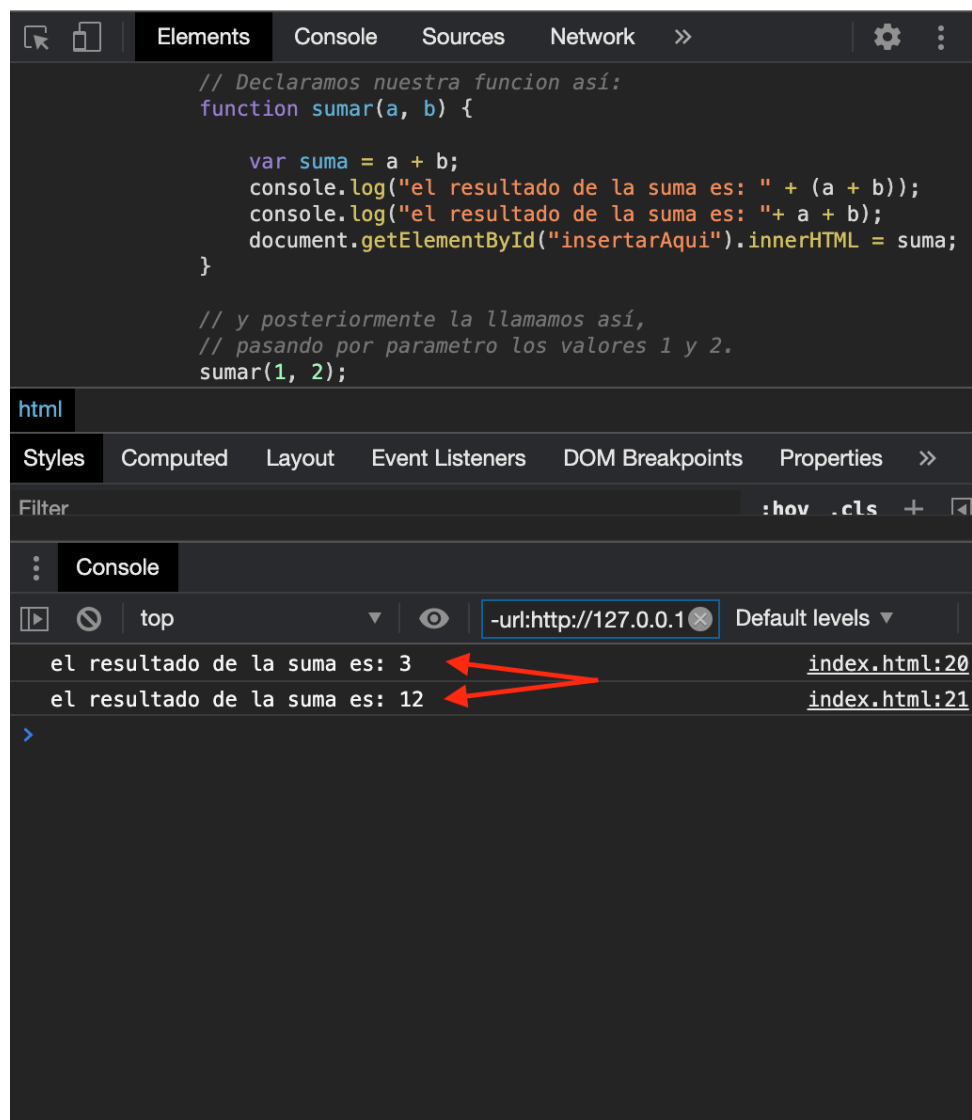
### SINTAXIS DE UNA FUNCIÓN

Para empezar a construir nuestra función en JavaScript, debemos considerar que en este lenguaje las funciones se definen con la palabra clave: **"function"**, seguido de un nombre junto a unos paréntesis **"( )"**. Los nombres de funciones pueden contener letras, dígitos, subrayados y meta caracteres. Dichos paréntesis pueden incluir nombres de parámetros separados por comas, y las instrucciones de la función se coloca entre llaves: **"{ }"**

A continuación, mostraremos una función de nombre **sumar()**, la cual realiza la operación aritmética de sumar dos valores que se introducen en la función por parámetro.

```
1 <body>
2   <h1 id="insertarAqui"></h1>
3   <script>
4
5       // Declaramos nuestra funcion así:
6       function sumar(a, b) {
7
8           var suma = a + b;
9           console.log("el resultado de la suma es: " + (a + b));
10          // console.log("el resultado de la suma es: "+ a + b);
11          document.getElementById("insertarAqui").innerHTML = suma;
12      }
13      // y posteriormente la llamamos así,
14      // pasando por parametro los valores 1 y 2.
15      sumar(1, 2);
16  </script>
17 </body>
18
```

En esta porción del código, podemos destacar bastantes cosas. Para empezar, notará que nuestra función recibe por parámetro un valor **a**, y un valor **b**. Dentro de ésta, declaramos una variable de nombre “suma”, con el valor de la suma aritmética de estos dos valores. Posteriormente, utilizamos el objeto **console.log()**, para mostrar el resultado por consola. Justo debajo de esa línea, encontramos código similar que está comentado, con la diferencia de que la suma no está entre paréntesis; esto es para observar que si lo hubiésemos escrito como muestra el código comentado, entonces, el resultado sería muy distinto a una suma. Para entender la diferencia, mostraremos ambos resultados por consola:



The screenshot shows a web browser's developer console with the following content:

```
// Declaramos nuestra funcion así:
function sumar(a, b) {

    var suma = a + b;
    console.log("el resultado de la suma es: " + (a + b));
    console.log("el resultado de la suma es: "+ a + b);
    document.getElementById("insertarAqui").innerHTML = suma;
}

// y posteriormente la llamamos así,
// pasando por parametro los valores 1 y 2.
sumar(1, 2);
```

The console output shows two log messages:

- el resultado de la suma es: 3 (index.html:20)
- el resultado de la suma es: 12 (index.html:21)

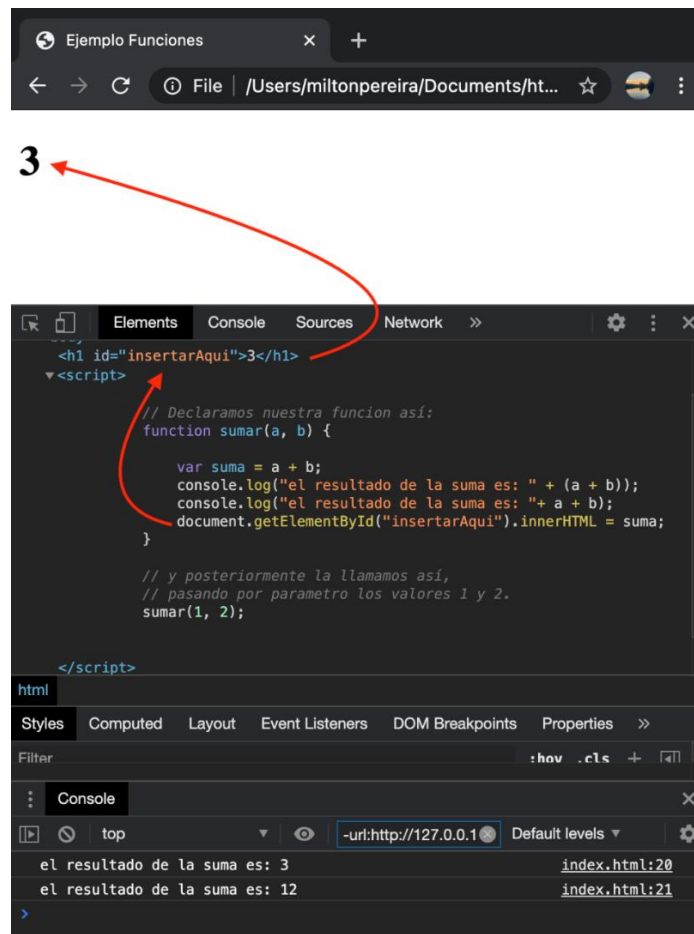
Two red arrows point from the text "muy distinto a una suma" in the previous block to the two log messages in the console, highlighting the difference between the two results.

Notarás que, al incluir unos paréntesis en nuestra suma, muestra el resultado adecuado que es tres; pero si no incluimos los paréntesis, solamente concatena el valor al **String** antepuesto, para dar un resultado equivocado.

Continuando con nuestro análisis del contenido de esta función, podemos ver que la última línea dentro del cuerpo de nuestra función es:

```
1 document.getElementById("insertarAqui").innerHTML = suma;
```

Con ella, logramos tomar el valor de la variable “suma”, y ponerlo en nuestro HTML en el elemento que tiene el **id=“insertarAqui”**. Éste es el que encontramos en el body o cuerpo de nuestro HTML, el cual es un **h1** con el **id=“insertarAqui”**. Al intentar escribir el resultado de nuestra suma dentro del HTML, podemos ver un resultado fuera de la consola, como lo muestra la siguiente imagen:



Acá se puede ver como desde una línea de código de nuestra función, podemos ingresar un valor a un elemento del HTML, que posteriormente se reflejará en el navegador.

A continuación, lograremos un resultado similar, pero de otra forma. Ingresando los valores a mano, y principalmente valiéndonos de un **event listener** (en castellano “detector de eventos”, o “oidor de eventos”). Éstos son unos procedimientos o funciones que esperan a que ocurra un evento. Ejemplos de ellos, son el usuario haciendo **CLIC** o moviendo el mouse, presionando una tecla en el teclado, entre otros. En otras palabras, el “oidor” o **listener**, está programado para reaccionar a una entrada o señal. En nuestro caso usaremos el **event listener onclick**, el cual nos permite invocar un método con el simple hecho de hacerle **CLIC** a un elemento.

Realizaremos algunos cambios a nuestro HTML, y el resultado del código es el siguiente:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-
6 scale=1.0">
7     <script src="script.js"></script>
8     <link rel="stylesheet" href="styles.css">
9     <title>Ejemplo Funciones</title>
10 </head>
11 <body>
12     <h3>Ingrese dos valores para sumar.</h3>
13     <!-- input para el valor de 'a' -->
14     <label for="a">Valor 1:</label>
15     <input name="a" id="a"><br>
16     <!-- input para el valor de 'b' -->
17     <label for="b">Valor 2:</label>
18     <input 5name="b" id="b"><br>
19     <button
20 onclick="sumar(document.getElementById('a').value,document.getElementById
21 ('b').value)">Sumar</button>
22     <script>
23         // Declaramos nuestra funcion así:
24         function sumar(a, b) {
25             var suma = parseInt(a) + parseInt(b);
26             // alert("la suma de " + a + " y " + b + "es: " + suma);
27             alert(`la suma de ${a} y ${b} es: ${suma}`);
28         }
29     </script>
30 </body>
31 </html>
```

Partiendo en nuestro análisis, desde el body del HTML, el elemento **button** es el que contiene el **event listener onclick**, que a su vez invoca a la función **sumar()**, pasando por parámetro los valores de los **inputs** por **id="a"** y **"b"**. Al recibir estos dos, nuestra función parsea los valores de tipo **String** convirtiéndolos al tipo **int**, para realizar la suma y finalizar con un **alert** conteniendo el resultado de la operación. De esta manera, hemos mostrado como regularmente se escribe una concatenación de **Strings** y variables, pero en este caso queda en evidencia que podemos utilizar el acento grave **" ` "**, para simplificar la sintaxis de nuestro **alert**.

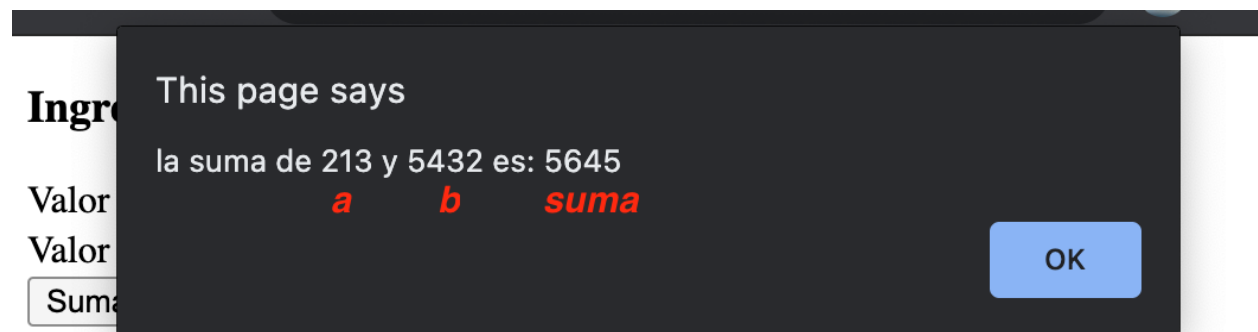
La vista de nuestro HTML es la siguiente:

## Ingrese dos valores para sumar.

Valor 1:

Valor 2:

Al momento de ingresar dos valores al azar, y hacerle **CLIC** al botón que dice sumar, se invoca nuestra función "sumar", la cual nos devuelve el siguiente resultado:

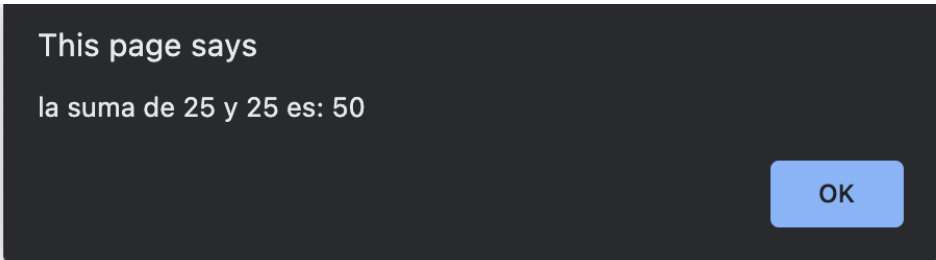


En rojo hemos indicado qué número corresponde a cuál variable en nuestro código JavaScript, pero más que mostrar el resultado de una suma, este **alert** permite ver cómo podemos utilizar las funciones de JavaScript para incrementar la funcionalidad de nuestras páginas, basándonos en la interacción del usuario.

Ahora, para ver cómo funciona la manera en que retornamos valores de funciones, llamaremos al método o la función dentro de las etiquetas script, y realizaremos un **console.log**:

```
1 <script>
2   // Declaramos nuestra funcion así:
3   function sumar(a, b) {
4       var suma = parseInt(a) + parseInt(b);
5       // alert("la suma de " + a + " y " + b + "es: " + suma);
6       alert(`la suma de ${a} y ${b} es: ${suma}`);
7   }
8   //invocamos el metodo y mostramos el resultado por consola:
9   console.log(`25 + 25 es: ${sumar(25, 25)}`)
10 </script>
```

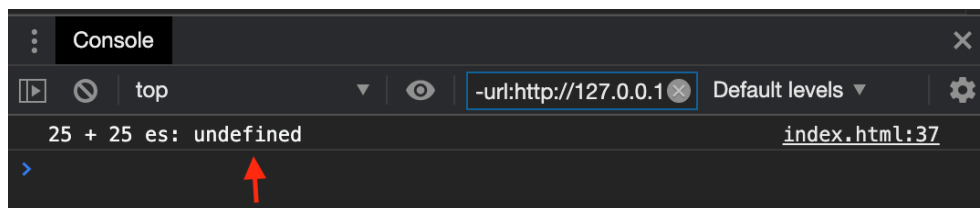
¿El resultado? El **alert** de la función despliega la información de manera correcta:



This page says  
la suma de 25 y 25 es: 50

OK

Pero nuestro **console.log()**:



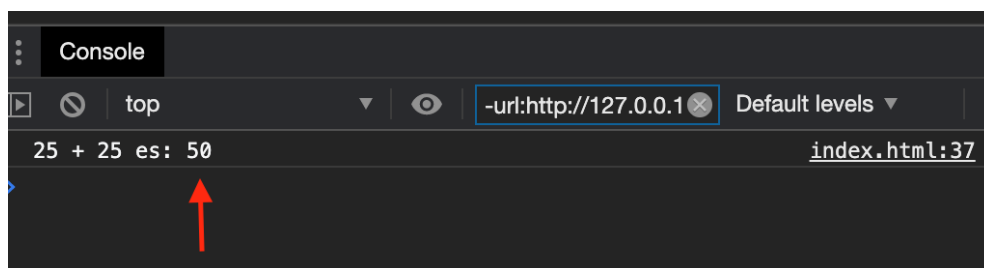
¿Por qué da este resultado?: se debe a que llamamos al método, pero éste no tiene ningún valor de retorno, por lo que la consola está esperando un valor, y como no retorna ninguna información, se despliega que es indefinido o, en inglés, **"undefined"**.

Para obtener el resultado adecuado, agregaremos un valor de retorno o, en otras palabras, una información que devuelve el método. Las siguientes líneas muestran que únicamente va a estar con una pequeña diferencia:

```

1 // Declaramos nuestra funcion así:
2 function sumar(a, b) {
3     var suma = parseInt(a) + parseInt(b);
4     // alert("la suma de " + a + " y " + b + "es: " + suma);
5     alert(`la suma de ${a} y ${b} es: ${suma}`);
6     return suma;
7 }
  
```

Al agregar la línea de código que especifica como retornaremos el resultado de la variable por nombre **suma**, podemos refrescar la página, y veremos lo siguiente por consola:



Ahora sí tenemos el resultado esperado, pues nos valemos de la variable **suma**, retornada por la función **sumar()**.

## EXPRESIONES

Existe otra manera para llegar al mismo resultado, y es mediante las expresiones de funciones; ¿en qué consisten?

```

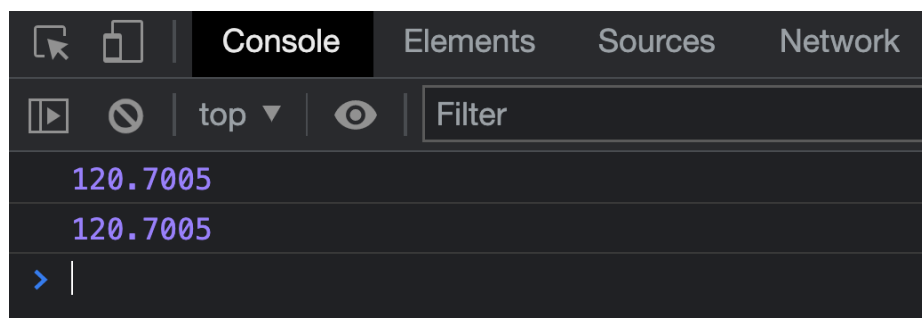
1 //la siguiente funcion se puede almacenar en una variable:
2 function multiplicar(a, b) {
3     return a * b
4 };
5 //Aquí tenemos la misma función almacenada en una variable:
6 var x = function (a, b) { return a * b };
7 // Aquí llamamos a la expresion:
8 console.log(x(3,3)) // 9
  
```

Cómo vemos en la línea 7 del código, podemos almacenar toda una función dentro de una variable. Una vez almacenada la variable, se puede utilizar como función. Las funciones almacenadas en variables no necesitan nombres de función, dado que, siempre se invocan utilizando el suyo propio.

Tomemos por ejemplo la siguiente función, que de manera sencilla convierte un valor de millas a kilómetros.

```
1 // con nombre de funcion
2 var km = function millas_a_km(millas) { return millas * 1.60934 };
3 console.log(km(75))
4
5 // sin nombre de funcion
6 var km = function (millas) { return millas * 1.60934 };
7 console.log(km(75))
```

Hemos declarado la misma expresión dos veces, con y sin nombre, para demostrar que el resultado es el mismo. La lógica de estas expresiones es que introducimos por parámetro un valor de distancia en millas, y en base a ese valor, se retornará convertido a kilómetros. Después de declarar nuestras funciones, mostramos los valores por consola, y el resultado es el siguiente:



De esta manera, queda comprobado como las funciones en JavaScript pueden ser declaradas, instanciadas, y almacenadas en una variable. Las oportunidades que esto nos ofrece, amplía la gama de implementaciones que le podemos dar a este lenguaje.



## EXERCISE 2: RETORNANDO VALORES MEDIANTE UNA FUNCIÓN EN JAVASCRIPT

Hasta este punto, hemos examinado la amplia gama de formas en que podemos usar funciones en JavaScript. Una de sus características, es que pueden devolver o retornar un valor. Este aspecto será el punto focal del siguiente ejemplo.

Para comenzar, debemos considerar que cuando se utiliza una declaración de retorno en el cuerpo de una función, se detiene su ejecución. Si se especifica, se devuelve un valor dado al llamador de la función. Tomemos, por ejemplo, la siguiente función. Aquí tenemos una con un nombre genérico: **"nombreFuncion"**, que en su cuerpo contiene el valor de una suma simple almacenada en la variable **"x"**. Esta variable luego se devuelve usando la palabra clave **return**, que nos permite emplear el valor que resulta en la ejecución completa de esta función fuera de ella.

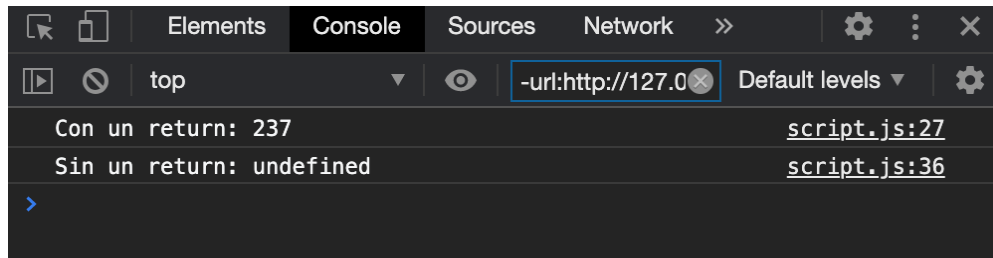
```
1 function nombreFuncion() {  
2     var x = a + b;  
3     return x; // declaramos valor por retornar  
4 }  
5 console.log(nombreFuncion()); //La consola mostrará el valor de x.
```

Dado que la función tiene un valor de retorno establecido, podemos usarlo fuera de ella, como en un **console.log()**. El resultado que esto muestra mediante la consola, es el valor de la variable **x**, que devolvemos en la función **nombreFuncion()**.

Pero, ¿qué mostraría la consola si nuestra función no tuviera una declaración de retorno? Averigüemos probando las siguientes dos funciones. Como se puede ver, una tiene una declaración de retorno, y la otra no.

```
1 function funcionReturn() {  
2     var a = 3;  
3     var b = 234;  
4     var x = a + b;  
5     return x; // declaramos valor por retornar  
6 }  
7 console.log(`Con un return: ${funcionReturn()}`); //La consola mostrará el  
8 valor de x.  
9 function funcionSinReturn() {  
10    var a = 3;  
11    var b = 234;  
12    var x = a + b;  
13 }  
14 console.log(`Sin un return: ${funcionSinReturn()}`); //¿Qué mostrará la  
15 consola?
```

¿Cuál fue el resultado? La consola muestra lo siguiente:



Al final, usar una declaración de retorno nos permitió emplear la variable `x` en nuestro `console.log()`, mientras que al intentar con una función similar sin una declaración de retorno, resultó en una variable indefinida. Esto nos entrega una lección importante: al incluir declaraciones de retorno, podemos crear funciones que se utilicen transversalmente en todo nuestro script.

También podemos utilizar esta propiedad de las funciones para ser creativos, y usar funciones dentro de otras. Considera el siguiente caso:

```

1 function ancho() {
2   var x = window.innerWidth;
3   return x;
4 }
5 function altura() {
6   var y = window.innerHeight;
7   return y;
8 }
9
10 function area() {
11   console.log(`El area de esta ventana es de ${ancho() * altura()}
12   pixeles cuadrados.`)
13   var a = `El area de esta ventana es de ${ancho() * altura()} pixeles
14   cuadrados.`
15   return a;
16 }
17 function mostrarArea() {
18   document.getElementById('area').innerHTML = area();
19 }
  
```

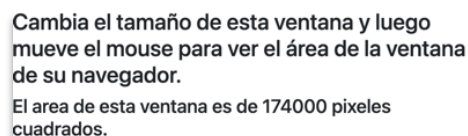
Como puedes ver, tenemos cuatro funciones: `ancho()`, `altura()`, `area()` y `mostrarArea()`. El primer método usa una función definida `"innerWidth"` del objeto `"window"` en JS, para calcular el ancho de la ventana de nuestro navegador. Asimismo, la segunda función hace algo similar, calculando la altura de la

ventana. Luego, nuestro tercer método calcula el área (ancho x alto) de la ventana, y devuelve un valor de cadena. Por último, la cuarta función llamada `mostrarArea()`, recupera el valor devuelto por `area()` y coloca ese valor dentro de un elemento HTML, con el `id` de `"area"`. Dicho elemento está contenido en el siguiente código:

```
1 <!DOCTYPE html>
2 <html lang="en" onmousemove="mostrarArea();">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-
6 scale=1.0">
7     <!-- Latest compiled and minified CSS -->
8     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
9 beta1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-
10 giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKhr8RbDVddVHYTfAAsrekWkmP1"
11 crossorigin="anonymous">
12     <script src="script.js"></script>
13     <meta name="theme-color" content="#7952b3">
14     <title>Ejemplo Funciones</title>
15 </head>
16 <body>
17     <h3>Cambia el tamaño de esta ventana y luego mueve el mouse para
18 ver el área de la ventana de su navegador.</h3>
19     <h5 id="area"></h5>
20 </body>
21 </html>
```

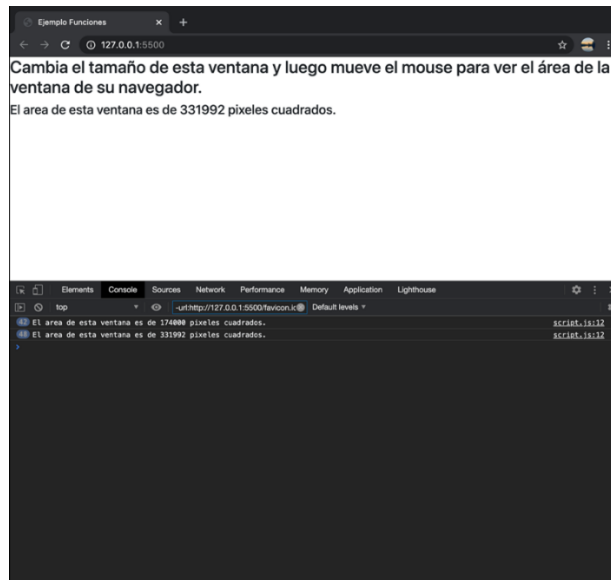
Aquí usamos el detector de eventos `"onmousemove"`, para llamar a nuestra última función cada vez que movemos el mouse sobre la página. Dado que colocamos este detector de eventos como un atributo en la etiqueta HTML, tendrá lugar tan pronto como se cargue el archivo HTML en el navegador.

Aquí podemos ver el resultado en una ventana de navegador de tamaño pequeño:

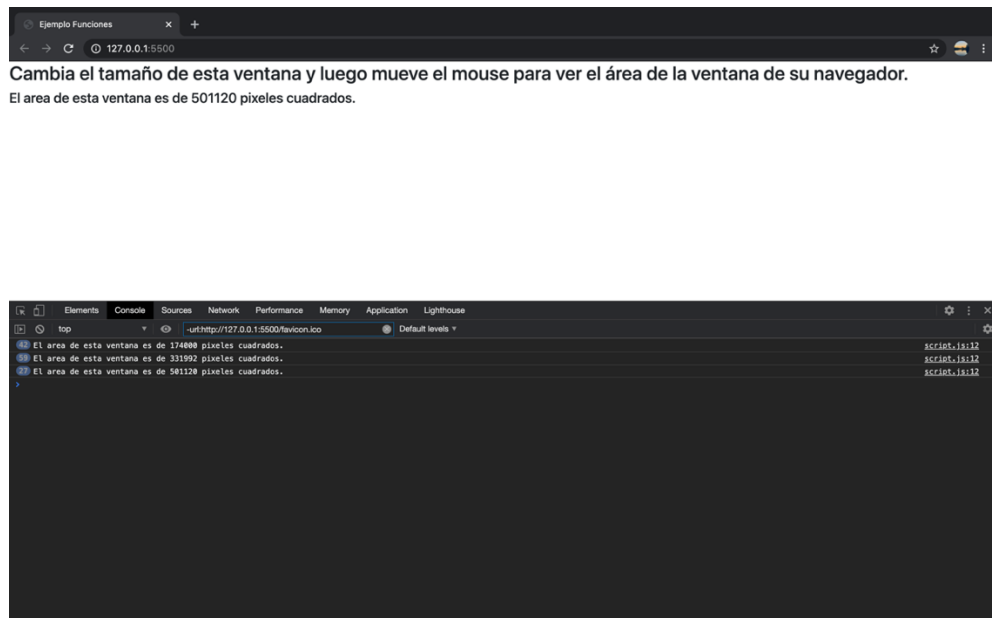




Aquí podemos ver el resultado en una ventana de navegador de tamaño mediano:



Aquí podemos ver el resultado en una ventana de navegador de tamaño completo:



Como podemos observar, las declaraciones de retorno abren otra capa de lo que es posible con los elementos más importantes de JavaScript: las funciones.