JavaScript

Curso Completo por freeCodeCamp

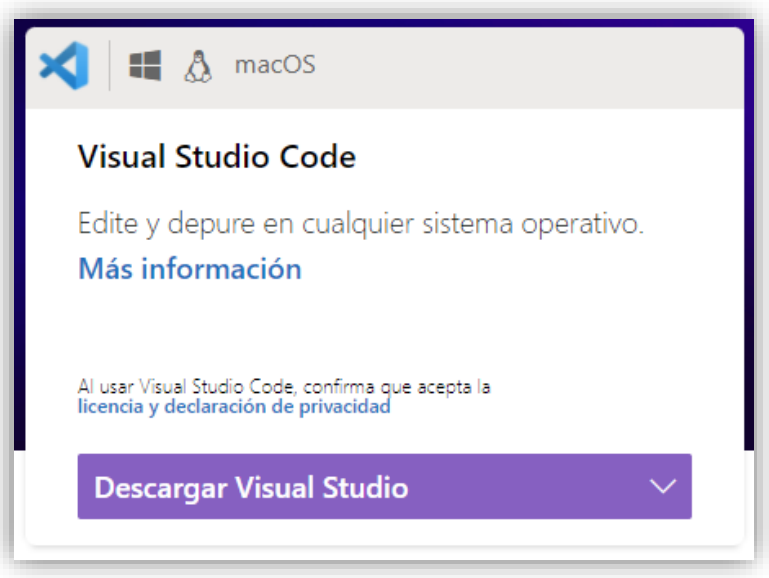
PERE MANEL VERDUGO ZAMORA
pereverdugo@gmail.com

¿Cómo ejecutar el JavaScript?

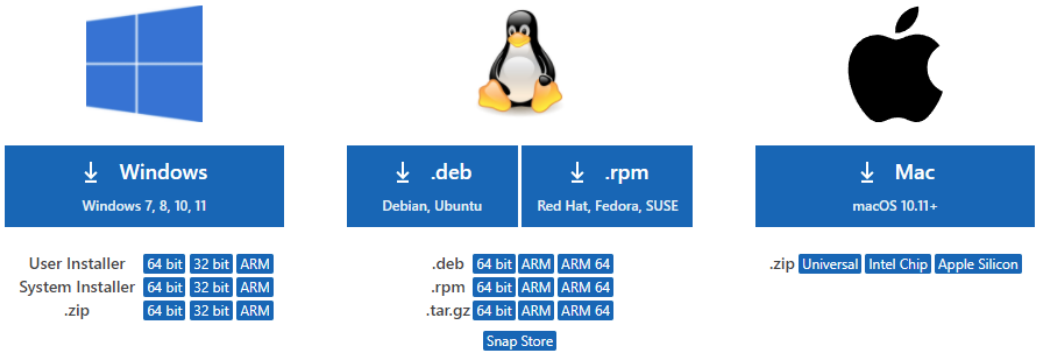
Para ejecutar JavaScript no necesitas descargar ningún programa, ya que este se ejecuta en los navegadores de Internet.

Una alternativa para editar este código es Visual Studio Code una aplicación gratuita que podrás descargar desde el siguiente enlace:

https://visualstudio.microsoft.com/es/



Esta disponible para Windows, Mac y Linux.



Cuando lo tengas descargado:

Lo ejecutas y así se iniciará el proceso de instalación.

Para poder comprobar nuestros proyectos vamos a realizar los siguientes pasos:

Primero vamos a crear una pequeño documento htlm.

*index.html: Bloc de notas

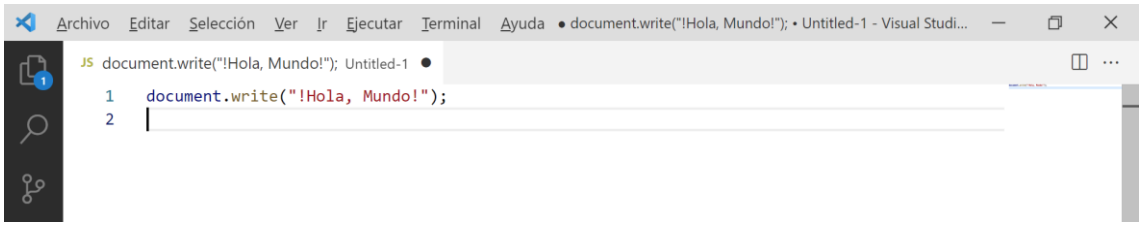
Archivo Edición Formato Ver Ayuda

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<TITLE>Prueba</TITLE>
</HEAD>
<BODY bgcolor=white>
<SCRIPT type="text/javascript" src="Nombre_proyecto.js">
//estoy incluyendo el fichero "Nombre_proyecto.js"
</SCRIPT>
</BODY>
</HTML>
```

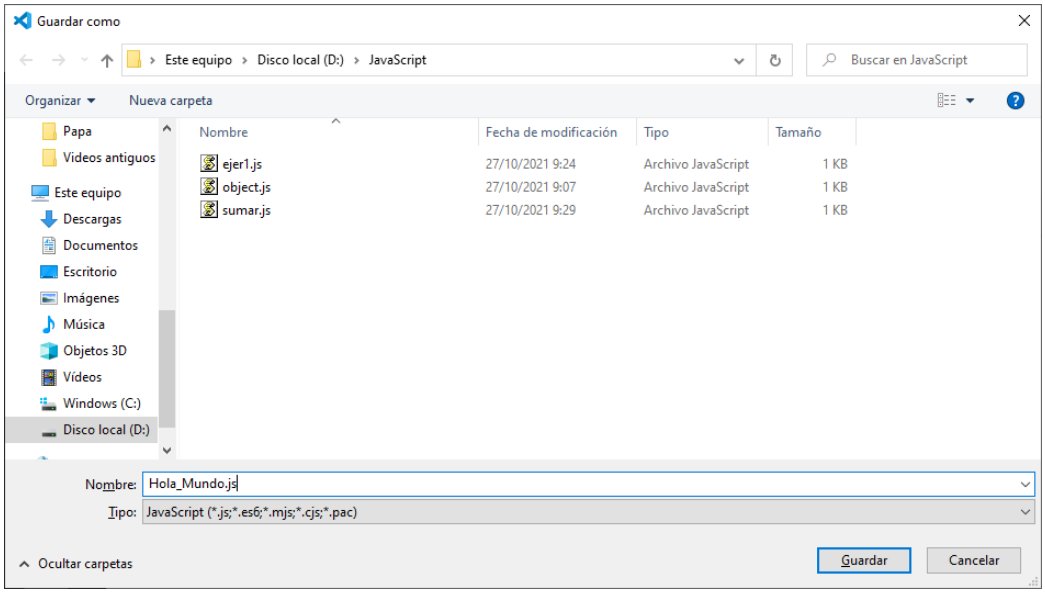
Nosotros cuando realicemos con Visual Studio Code nuestro proyecto este lo tenemos que guardar con una extensión .js, que guardaremos en la misma carpeta que contiene archivo .html.

En la parte subrayada del documento HTML cambiaremos el nombre del proyecto, lo guardaremos y a continuación lo ejecutaremos.

Vamos a ver un ejemplo:



Hemos creado nuestro primer proyecto que guardaremos con el nombre de Hola_Mundo.js.



A continuación abrimos el archivo html con el bloc de notas.

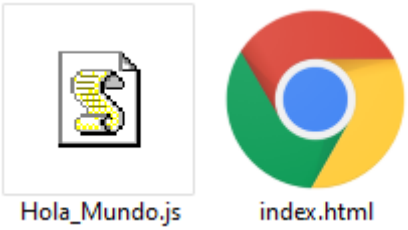
v

index.html: Bloc de notas

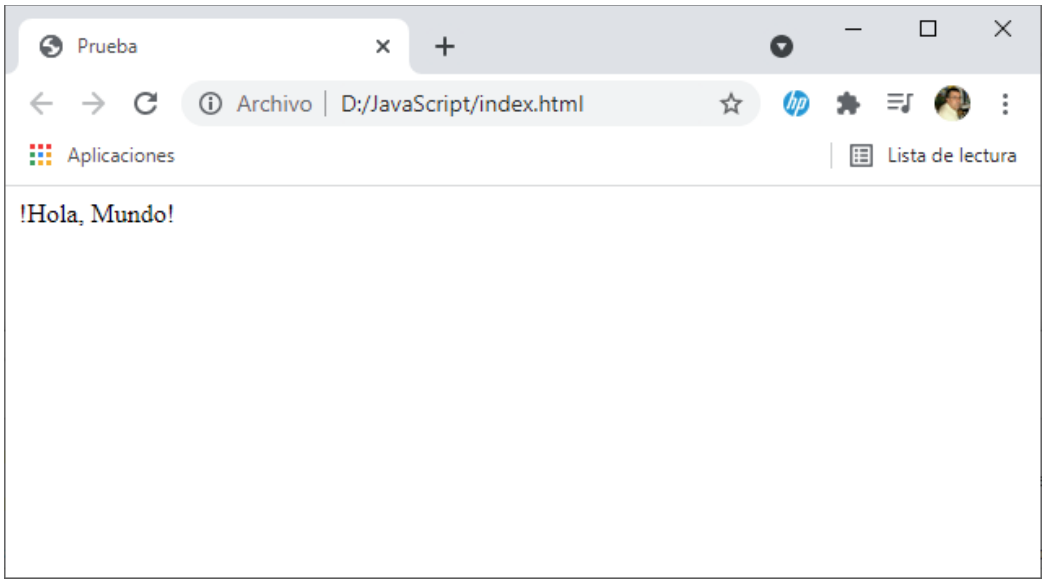
Archivo Edición Formato Ver Ayuda

```
<HTML>
<HEAD>
<TITLE>Prueba</TITLE>
</HEAD>
<BODY bgcolor=white>
<SCRIPT type="text/javascript" src="Hola_mundo.js">
//estoy incluyendo el fichero "Hola_mundo.js"
</SCRIPT>
</BODY>
</HTML>
```

Guardamos los cambios y cerramos dicho documento.



Ahora tenemos en la misma carpeta el proyecto en JavaScript y el documento index.html, vamos a hacer doble clic sobre él.



Este será el resultado.

Recuerda cada vez que realices un proyecto nuevo tendrás que cambiar el nombre del archivo.js de documento index.html.

Comentarios

Los comentarios son comentarios del código que no son parte del código, es decir a la hora de ejecutar el código los comentarios son ignorados, este se inicia con un //.

```
// Comentario de una sola línea.
```

También puede haber una línea con parte de código y parte de comentario.

```
var x = 5; // Comentario de una sola línea.
```

Si queremos realizar comentarios de más de una línea.

```
/* Este es un comentario de varias líneas
se abre con una barra diagonal y asterísco
para iniciar el comentario y un asterisco y
barra diagonal para cerrar comentario. */
```

También se puede hacer de la siguiente forma:

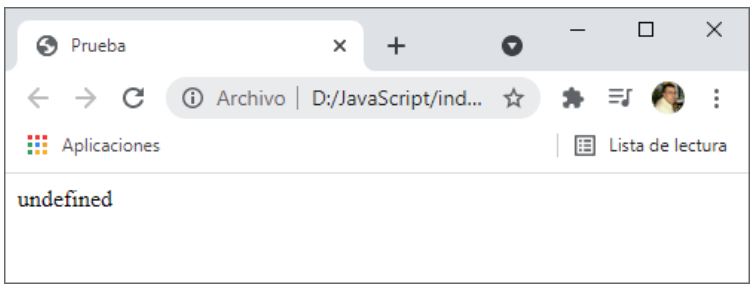
```
/*
Este es un comentario de varias líneas
se abre con una barra diagonal y asterísco
para iniciar el comentario y un asterisco y
barra diagonal para cerrar comentario.
*/
```

Variables

```
/*
Tipos de datos:
undefined, null, boolean, string, symbol, number y object.
*/
```

undefined: Cuando a una variable no se le asigna valor.

```
var x;
document.write(x);
```

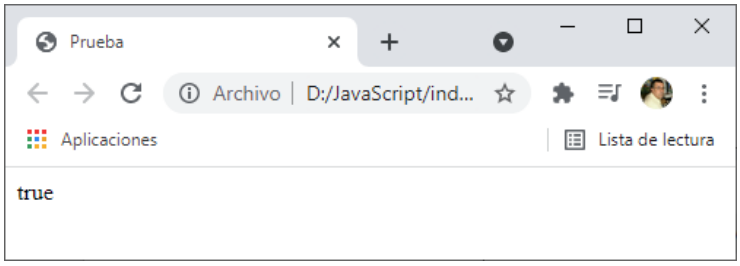


null: Cuando la variable no existe.

boolear: Cuando la variable es verdadero o falso.

```
6    var x = true;
7    document.write(x);
```

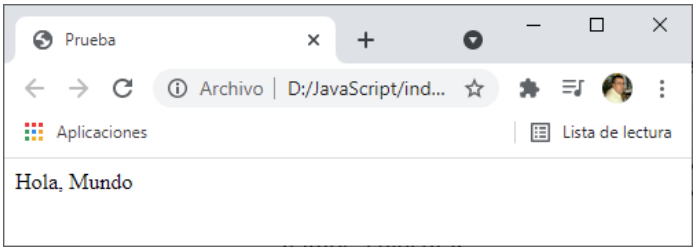
Vamos a ejecutar.



string: Cadena de caracteres.

```
6    var x = "Hola, Mundo";
7    document.write(x);
```

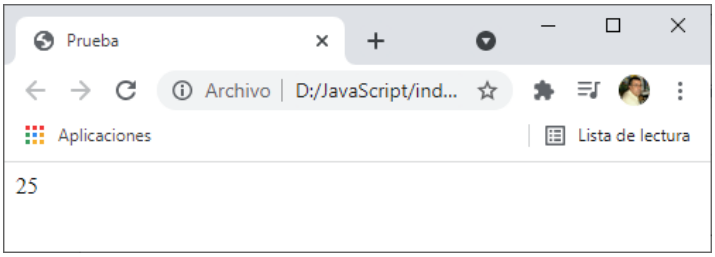
Vamos a ejecutar.



number: Para almacenar un valor numérico.

```
6    var x = 25;
7    document.write(x);
```

Vamos a ejecutar:



object: se usa para representar un objeto en una estructura que nos permitirá relacionar propiedades.

Para crear una variable tenemos que utilizar la palabra reservada var seguido del nombre de la variable (puedes utilizar cualquier palabra que no esté reservada a JavaScript) a continuación el operador de asignación '=' seguido del valor que le queremos dar.

Se aconseja que el nombre de la variable sea descriptivo según el contenido de la misma.

A la hora de dar nombre a la variable una regla es que empiece en minúscula y si esta consta de varias palabras no se pueden separar pero la primera letra de cada palabra ponerla en mayúsculas, por ejemplo: miNombre, fechaDeNacimeinto, etc.

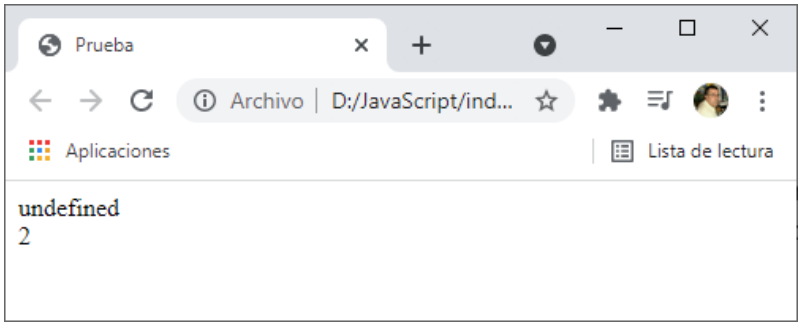
Operadores de asignación

D: > JavaScript > JS operadores.js > ...

```
1   var a;
2   var b = 2;
3   document.write(a + "<br/>" + b);
```

Hemos definido la variable a, a la que no le hemos asignado ningún valor, en cambio a la variable b la hemos definido y además le hemos asignado el valor de 2, que lo hacemos mediante el signo igual '='.

Vamos a ejecutar.

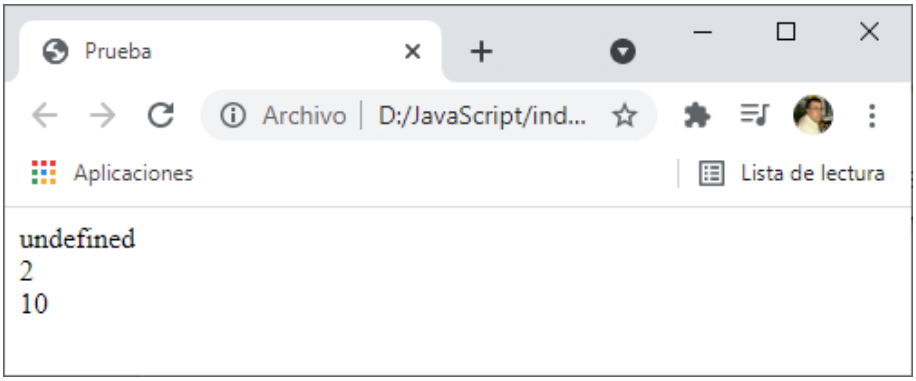


La primera variable no está definida y la segunda tiene el valor 2.

D: > JavaScript > JS operadores.js > ...

```
1   var a;
2   var b = 2;
3   document.write(a + "<br/>" + b);
4   a = 10;
5   document.write("<br/>" + a);
```

En la siguiente línea le asignamos un valor a la variable a que después queremos mostrar.



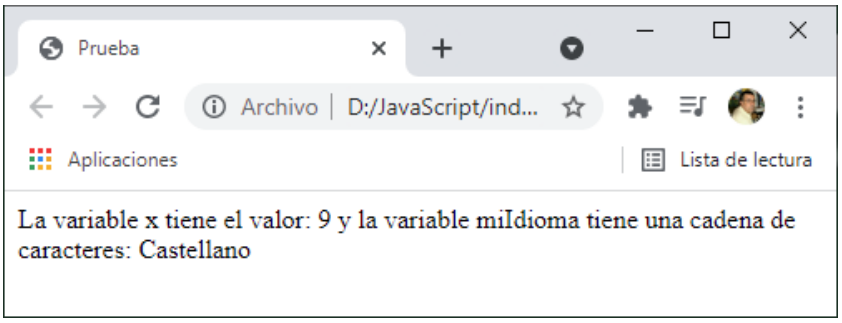
Inicializar variables

D: > JavaScript > JS operadores.js > mildioma

```
1   var x = 9; // Inicializar la variable.
2   var miIdioma = "Castellano";
3
4   document.write("La variable x tiene el valor: " + x);
5   document.write(" y la variable miIdioma tiene una cadena de caracteres: " + miIdioma);
```

En este capítulo inicializamos las dos variables, la x con un valor numérico y la variable mildioma con una cadena de caracteres.

Este será el resultado:

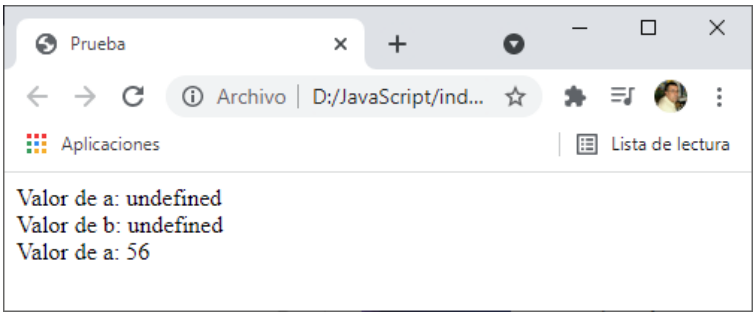


Variable no inicializada

D: > JavaScript > JS operadores.js > ...

```
1   // Variable no inicializada
2
3   var a;
4   var b;
5   document.write("Valor de a: " + a);
6   document.write("<br\>");
7   document.write("Valor de b: " + b);
8   a = 56;
9   document.write("<br\>");
10  document.write("Valor de a: " + a);
```

En este ejemplo definimos dos variables sin asignarles valor, las mostramos y después la variable a le asignamos el valor 56, este será el resultado:

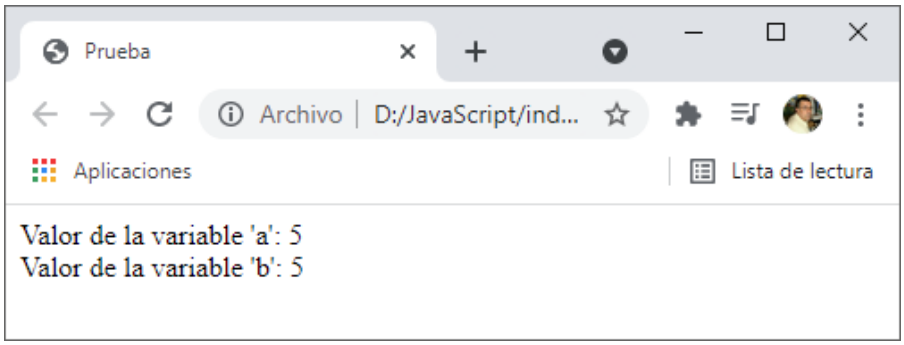


Asignar el valor de una variable a otra variable

D: > JavaScript > JS operadores.js > ...

```
1   var a = 5;
2   var b = a;
3   document.write("Valor de la variable 'a': " + a + "<br\>" + "Valor de la variable 'b': "
    + b);
```

A la variable a le asignamos el valor de 5 y a la variable b le asignamos el valor de la variable a. Este será el resultado:



D: > JavaScript > JS operadores.js > ...

```
1   var a = 5;
2   var b;
3   b = a;
4   document.write("Valor de la variable 'a': " + a + "<br\>" + "Valor de la variable 'b': "
    + b);
```

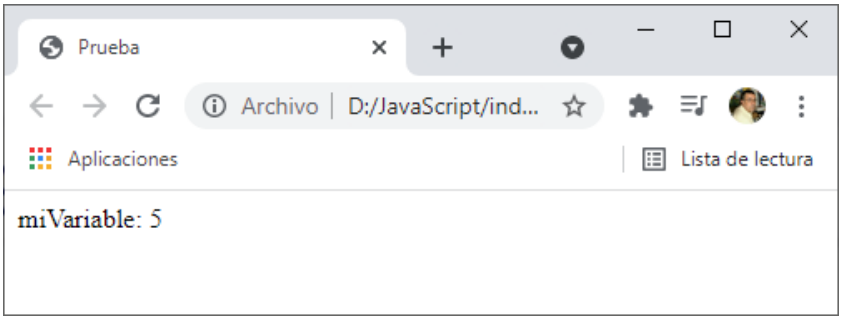
En la línea 2 definimos la variable b pero no le asignamos ningún valor, es en la línea 3 donde le asignamos el valor de la variable a.
Si ejecutamos el resultado será el mismo.

Mayúsculas y minúsculas

D: > JavaScript > JS operadores.js > ...

```
1   var miVariable = 5;
2   document.write("miVariable: " + miVariable + "<br\>");
3   document.write("MiVariable: " + MiVariable + "<br\>");
4   document.write("MIVARIABLE: " + MIVARIABLE + "<br\>");
5   document.write("mivariable: " + mivariable );
```

A la hora de ejecutar solo reconoce la primera variable, el cambio de una variable con letras mayúsculas o minúsculas hace que esta sea otra variable, pero como no las hemos definido estas líneas dan error (variable no definida) y no se emprimen, este es el resultado.



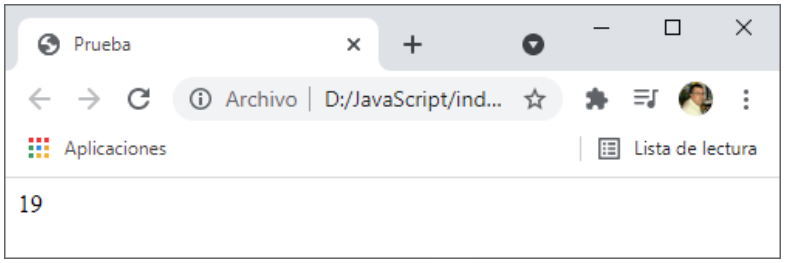
Esto se denomina Case-sensitive.

Suma

D: > JavaScript > JS operadores.js > ...

```
1   var suma = 7 + 12;
2   document.write(suma);
```

Este será el resultado:

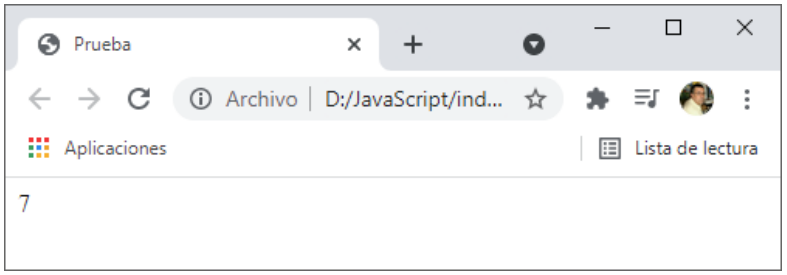


Resta

D: > JavaScript > JS operadores.js > ...

```
1   var resta = 12 - 5;
2   document.write(resta);
```

Este es el resultado:

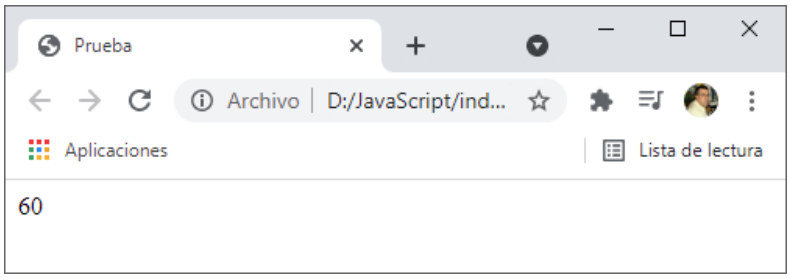


Multiplicación

D: > JavaScript > JS operadores.js > ...

```
1   var multiplicacion = 12 * 5;
2   document.write(multiplicacion);
```

Este es el resultado:

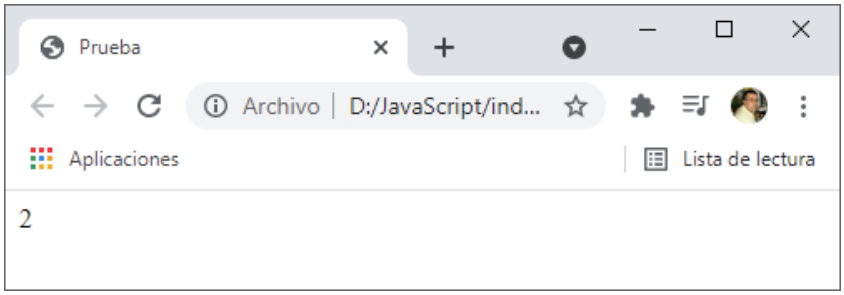


División

D: > JavaScript > JS operadores.js > ...

```
1    var division = 12 / 6;
2    document.write(division);
```

El resultado será:

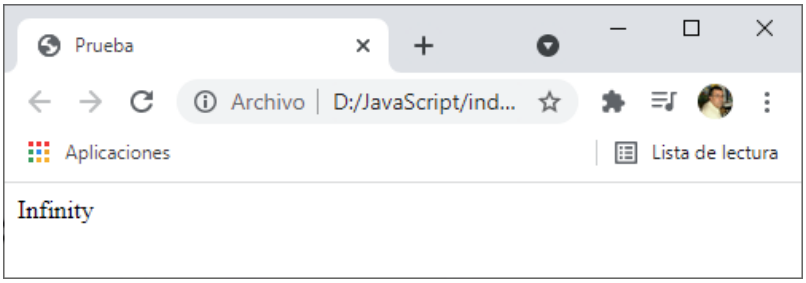


Vamos a dividir entre 0.

D: > JavaScript > JS operadores.js > division

```
1    var division = 12 / 0;
2    document.write(division);
```

Este es el mensaje de error:

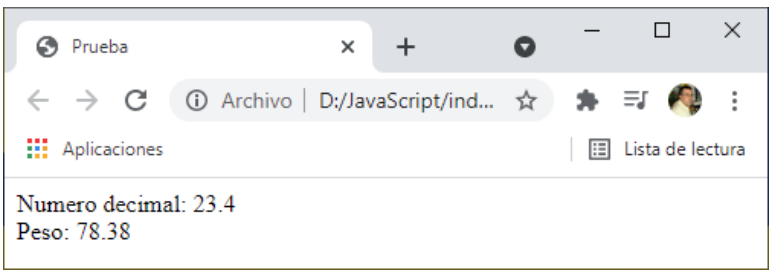


Números decimales

D: > JavaScript > JS operadores.js > ...

```
1    var numeroDecimal = 23.4;
2    var peso = 78.38;
3    document.write("Numero decimal: " + numeroDecimal + "<br\>");
4    document.write("Peso: " + peso);
```

Este será el resultado:

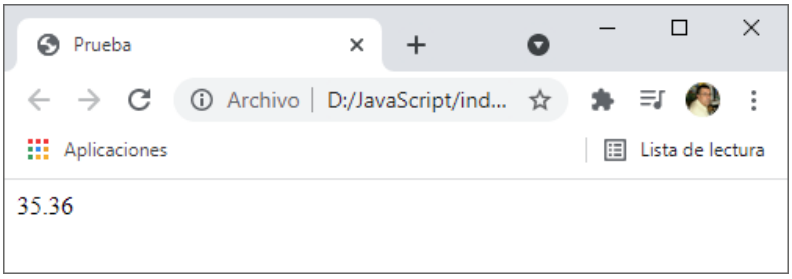


Multiplicar números decimales

D: > JavaScript > JS operadores.js > ...

```
1    var producto = 3.4 * 10.4;
2    document.write(producto);
```

Este será el resultado:

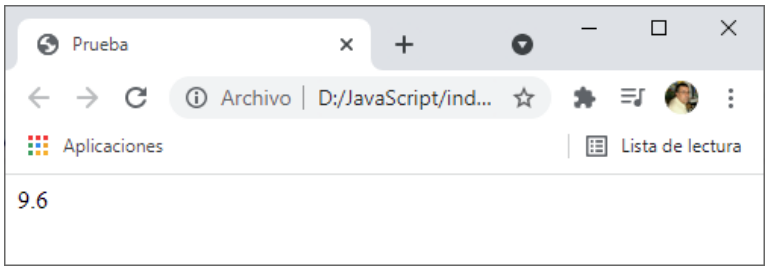


Podemos multiplicar un número decimal con otro de tipo entero, nos retornará un número decimal.

D: > JavaScript > JS operadores.js > ...

```
1    var producto = 2.4 * 4;
2    document.write(producto);
```

Este será el resultado:

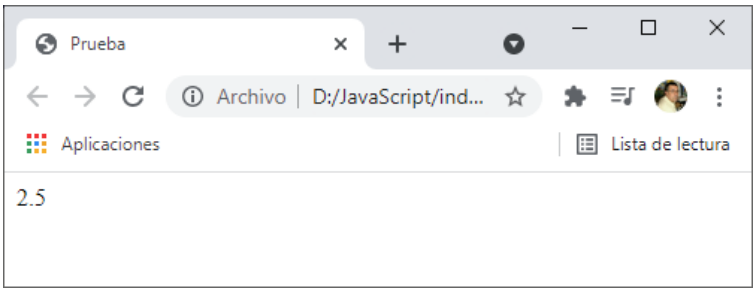


Dividir números decimales

D: > JavaScript > JS operadores.js > ...

```
1    var cociente = 5.0 / 2.0;
2    document.write(cociente);
```

Este será el resultado:

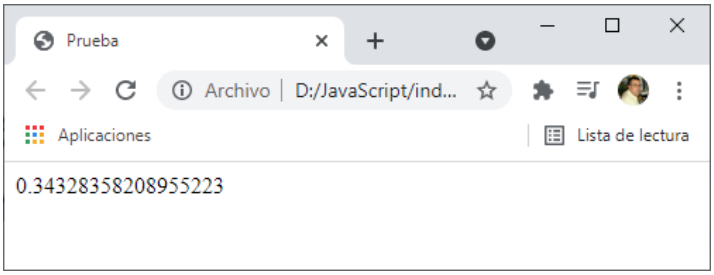


Otro ejemplo:

D: > JavaScript > JS operadores.js > cociente

```
1   var cociente = 2.3 / 6.7;
2   document.write(cociente);
```

Este será el resultado:

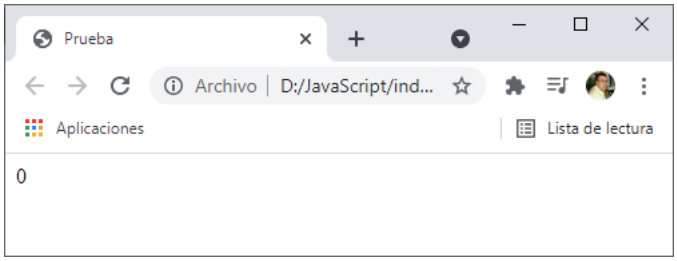


Resto de la división

D: > JavaScript > JS operadores.js > ...

```
1   var resto = 15 % 5;
2   document.write(resto);
```

Este será el resultado:

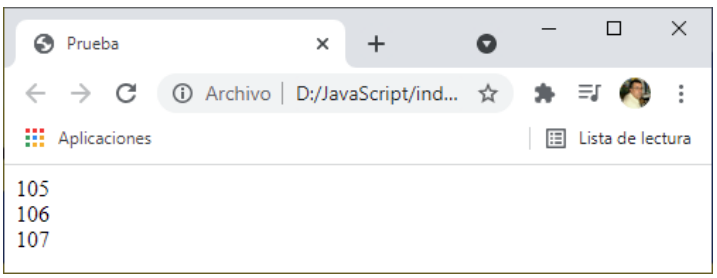


Incrementar el valor de una variable

D: > JavaScript > JS operadores.js > ...

```
1   var librosComprados = 105;
2   document.write(librosComprados + "<br\>");
3
4   librosComprados = librosComprados + 1; // Se incrementa en 1.
5   document.write(librosComprados + "<br\>");
6
7   librosComprados++; // incrementar en 1.
8   document.write(librosComprados);
```

Este es el resultado:

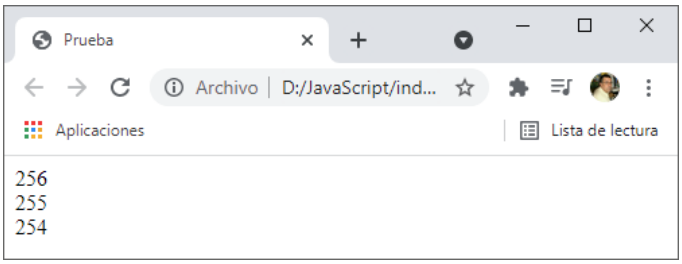


Reducir el valor de una variable

D: > JavaScript > JS operadores.js > ...

```
1   var numeroDeEstudiantes = 256;
2   document.write(numeroDeEstudiantes + "<br\>");
3
4   numeroDeEstudiantes = numeroDeEstudiantes - 1; // Se decrementa en 1.
5   document.write(numeroDeEstudiantes + "<br\>");
6
7   numeroDeEstudiantes--; // decrementar en 1.
8   document.write(numeroDeEstudiantes);
```

Este es el resultado:

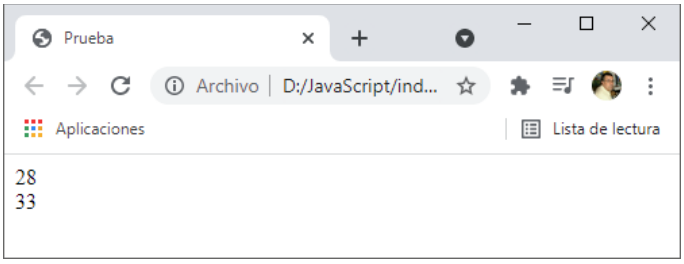


Asignación de suma

D: > JavaScript > JS operadores.js > ...

```
1   var a = 23;
2   a = a + 5; // incrementamos en 5
3   document.write(a + "<br\>");
4   a += 5; // También incrementamos en 5
5   document.write(a);
```

Este es el resultado:

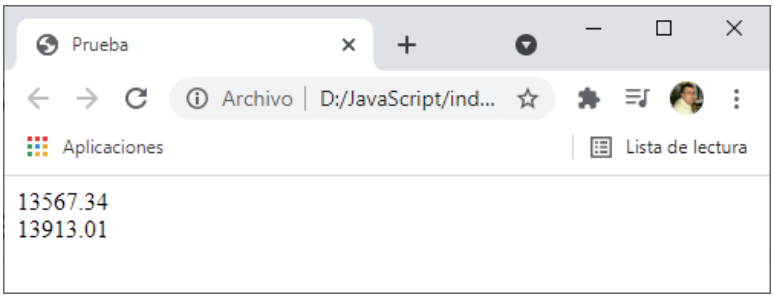


Otro ejemplo:

D: > JavaScript > JS operadores.js > ...

```
1   var totalVentas = 13567.34;
2   document.write(totalVentas + "<br\>");
3   totalVentas += 345.67;
4   document.write(totalVentas);
```

Este será el resultado:

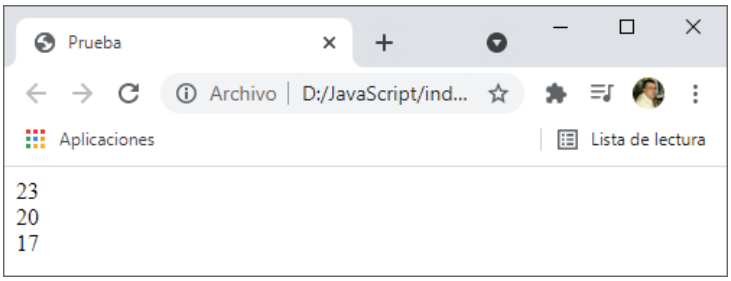


Asignación de Resta

D: > JavaScript > JS operadores.js > ...

```
1    var b= 23;
2    document.write(b + "<br\>");
3    b = b - 3; // Resta 3 a la variable b
4    document.write(b + "<br\>");
5    b -= 3; // También resta 3 a la variable b
6    document.write(b)
```

Este será el resultado:

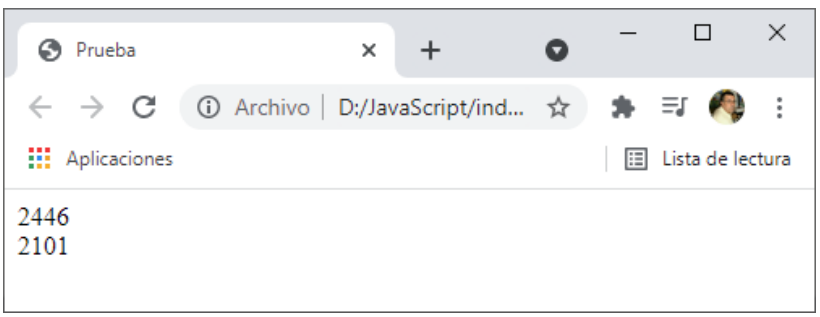


Otro ejemplo:

D: > JavaScript > JS operadores.js > ...

```
1    var totalDeuda= 2446;
2    document.write(totalDeuda + "<br\>");
3    totalDeuda -= 345;
4    document.write(totalDeuda);
```

Este será el resultado:

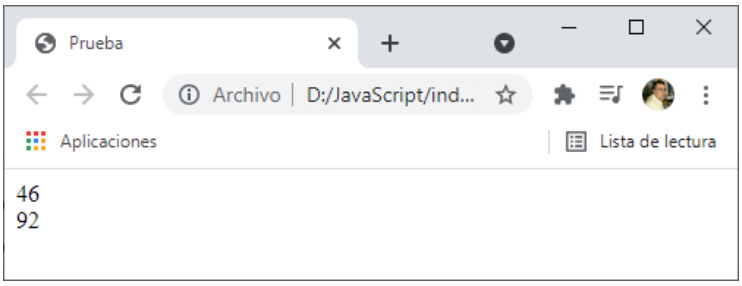


Asignación de multiplicación

D: > JavaScript > JS operadores.js > ...

```
1    var c = 23;
2    c = c * 2;
3    document.write(c + "<br\>");
4    c *= 2;
5    document.write(c);
```

Este será el resultado:

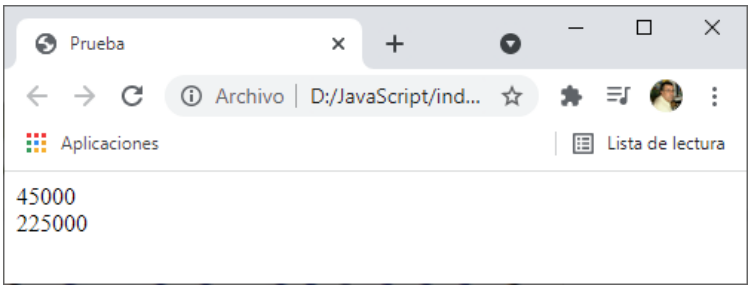


Otro ejemplo:

D: > JavaScript > JS operadores.js > ...

```
1    var salario = 45000;
2    document.write(salario + "<br\>");
3    salario *= 5; // Lo quintuplicamos
4    document.write(salario);
```

Este es el resultado:



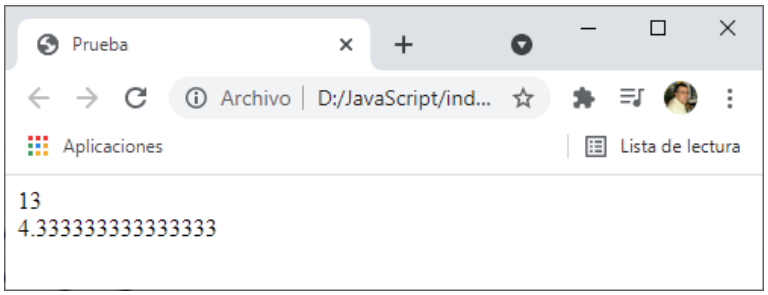
Asignación de la división

D: > JavaScript > JS operadores.js > ...

```
1    var d = 39;
2    d = d / 3;
3    document.write(d + "<br\>");
```

```
4    d /= 3;
5    document.write(d);
```

Este será el resultado:

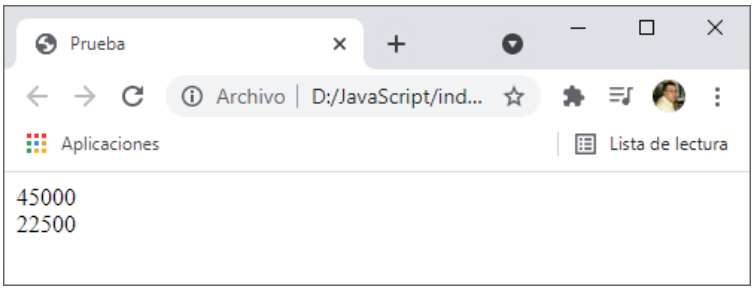


Otro ejemplo:

D: > JavaScript > JS operadores.js > ...

```
1    var salario = 45000;
2    document.write(salario + "<br\>");
3    salario /= 2;
4    document.write(salario);
```

Este será el resultado:



Variables con cadena de caracteres

D: > JavaScript > JS operadores.js > ...

```
1    var nombre = "Alan"; // Tambien es correcto 'Alan'
2    // Esto no es correcto "Alan'
3    var animal = "perro'
```

D: > JavaScript > JS operadores.js > [0] animal

```
1    var nombre = "Alan"
2    // Esto no es corre
3    var animal = "perro'
```

Literal de cadena sin terminar. ts(1002)

Ver el problema No hay correcciones rápidas disponibles

D: > JavaScript > JS operadores.js > ...

```
1    var nombre = "Alan"; // Tambien es correcto 'Alan'
2    // Esto no es correcto "Alan'
3    var animal = 'perro';
4    document.write("Mi nombre es " + nombre + " y tengo un " + animal + ".");
```

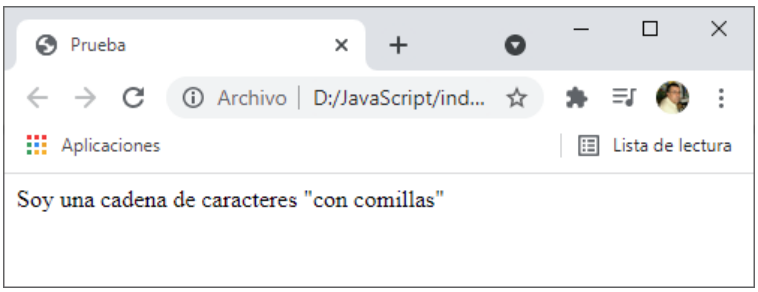
Este será el resultado:

Escapar comillas en cadena de caracteres

D: > JavaScript > JS operadores.js > ...

```
1   // var miCadena = "Soy una cadena de caracteres "con comillas"";
2   // para insertar una cadena con comillas.
3   var miCadena = "Soy una cadena de caracteres \"con comillas\"";
4   document.write(miCadena);
```

Este es el resultado:

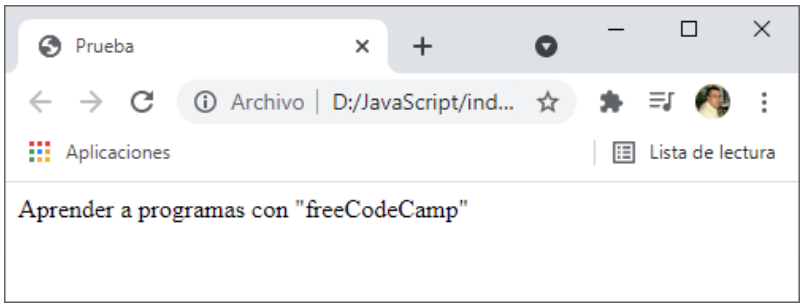


Cadena de caracteres con comillas simples

D: > JavaScript > JS operadores.js > ...

```
1   var miMeta;
2   miMeta = "Aprender a programas con \"freeCodeCamp\"";
3   miMeta = 'Aprender a programas con "freeCodeCamp"';
4   document.write(miMeta);
```

Este es el resultado:



También es correcto:

```
3   miMeta = "Aprender a programas con 'freeCodeCamp'";
```

Secuencias de escape

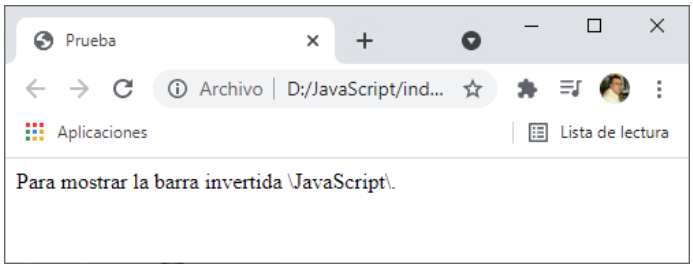
Código	Resultado
\’	Comillas simples
\”	Comillas dobles
\\	Barra invertida
\n	Línea nueva
\r	Retorno de carro
\t	Tabulación
\b	Retroceso
\f	Salto de página

Barra invertida

D: > JavaScript > JS operadores.js

```
1   document.write("Para mostrar la barra invertida \\JavaScript\\.");
```

Este será el resultado:



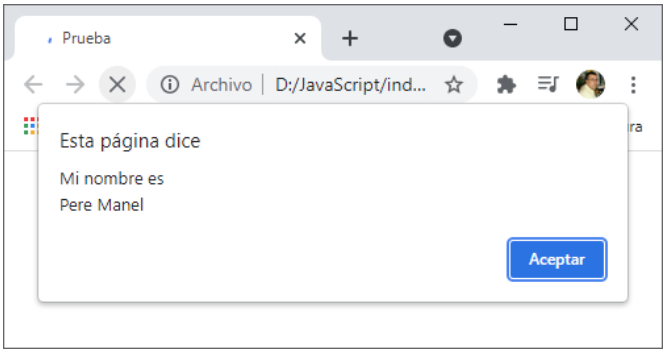
Línea nueva:

Para este ejemplo vamos a utilizar la sentencia Alert, ya que en document.write no funciona hay que utilizar
 como en html.

D: > JavaScript > JS operadores.js

```
1   alert("Mi nombre es \nPere Manel");
```

Este será el resultado.

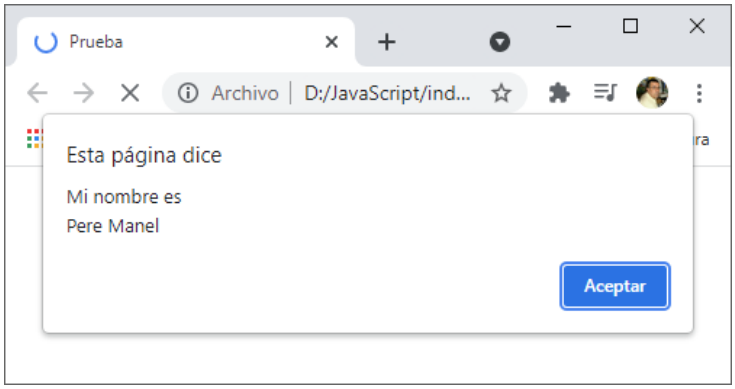


Retorno de carro:

D: > JavaScript > JS operadores.js

```
1    alert("Mi nombre es \rPere Manel");
```

Este es el resultado:



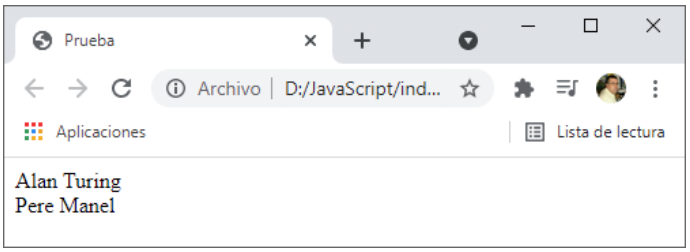
Algunos me funcionan y otros no.

Concatenar cadena de caracteres

D: > JavaScript > JS operadores.js > ...

```
1    var nombreCompleto1 = "Alan " + "Turing";
2    var nombreCompleto2 = "Pere" + " " + "Manel"
3    document.write(nombreCompleto1 + "<br>");
4    document.write(nombreCompleto2);
```

Este será el resultado:

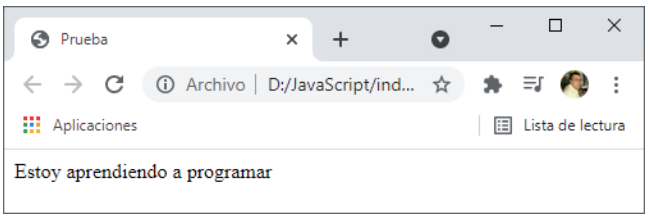


Construir cadenas con variables

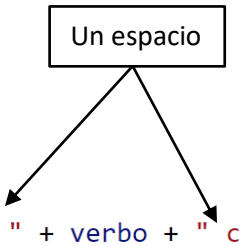
D: > JavaScript > JS operadores.js > ...

```
1    var verbo = "programar";
2    var mensaje ="Estoy aprendiendo a " + verbo;
3    document.write(mensaje);
```

Este es el resultado:



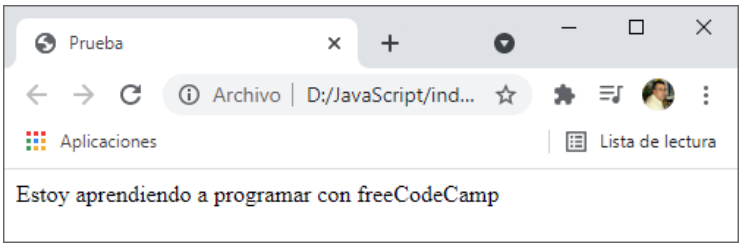
Podemos concatenar más texto:



D: > JavaScript > JS operadores.js > ...

```
1   var verbo = "programar";
2   var mensaje ="Estoy aprendiendo a " + verbo + " con freeCodeCamp";
3   document.write(mensaje);
```

Este será el resultado:

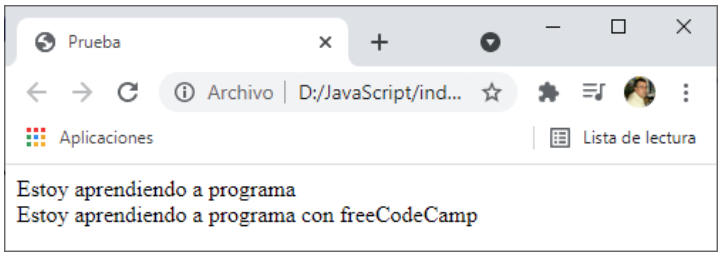


Agregar variables a Cadena de caracteres

D: > JavaScript > JS operadores.js > ...

```
1   var mensajeCompleto = "Estoy aprendiendo a programa ";
2   var parteFinal = "con freeCodeCamp";
3   document.write(mensajeCompleto + "<br>");
4   mensajeCompleto += parteFinal;
5   document.write(mensajeCompleto);
```

Este será el resultado:

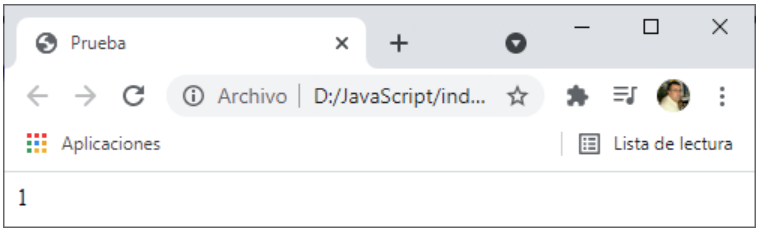


Longitud de una cadena de caracteres

D: > JavaScript > JS operadores.js > ...

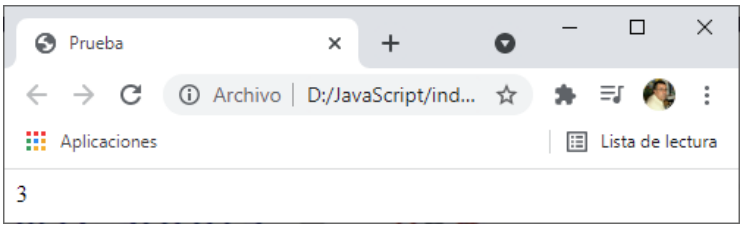
```
1   var miCadena = "A";
2   document.write(miCadena.length);
```

Este será el resultado:




```
1   var miCadena = "A B";
```

Vamos a cambiar el valor de la cadena, este será el resultado:



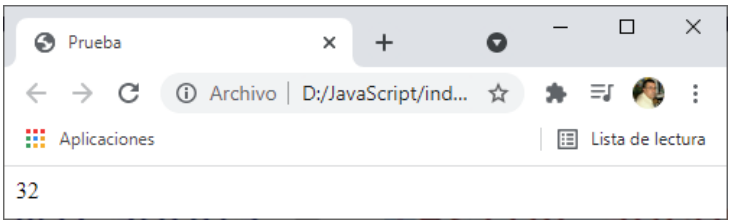
Los espacios en blanco también cuentan.

Otro ejemplo:

D: > JavaScript > JS operadores.js > ...

```
1   var miCadena = "¡Estoy aprendiendo a programar!";
2   document.write(miCadena.length);
```

Este será el resultado:

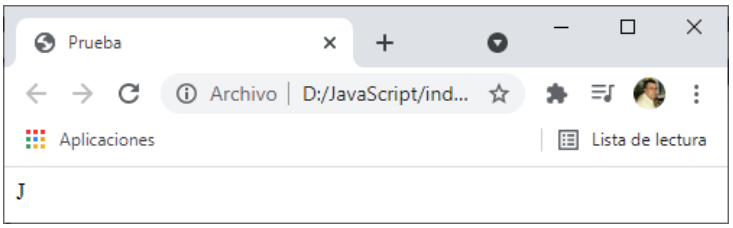


Notación de corchetes: Primer carácter

D: > JavaScript > JS operadores.js > ...

```
1   var lenguajeDeProgramacion = "JavaScript";
2   /*
3   Cadena:   J a v a S c r i p t
4   Índices:  0 1 2 3 4 5 6 7 8 9
5   */
6   document.write(lenguajeDeProgramacion[0]);
```

Este será el resultado:



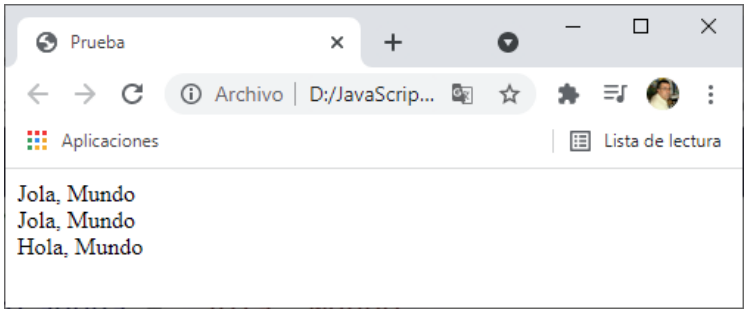
Inmutabilidad de cadena de caracteres

D: > JavaScript > JS operadores.js > ...

```
1   // Inmiutabilidad
2   var miCadena = "Jola, Mundo";
3   document.write(miCadena + "<br>");
4   miCadena[0] = "H";
5   document.write(miCadena + "<br>");
```

```
6   /* Error no se puede cambiar un carácter
7      se pude cambiar toda la cadena.
8   */
9   miCadena = "Hola, Mundo";
10  document.write(miCadena);
```

Este será el resultado:



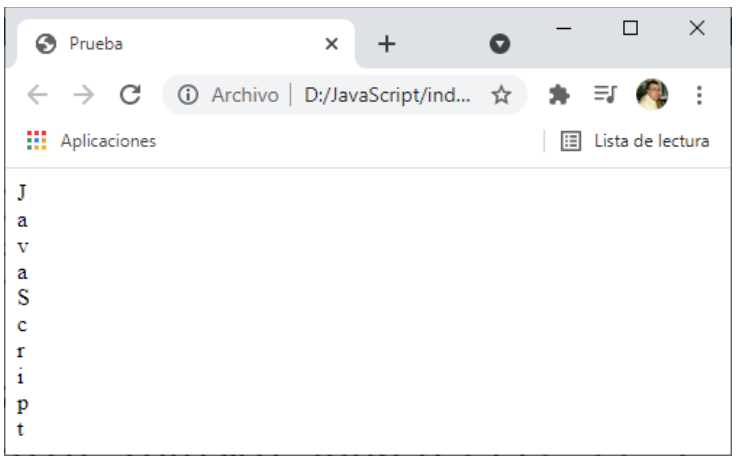
No se puede modificar ningún carácter individualmente, tiene que ser toda la cadena.

Notación de corchetes: Enésimo Carácter

D: > JavaScript > JS operadores.js > ...

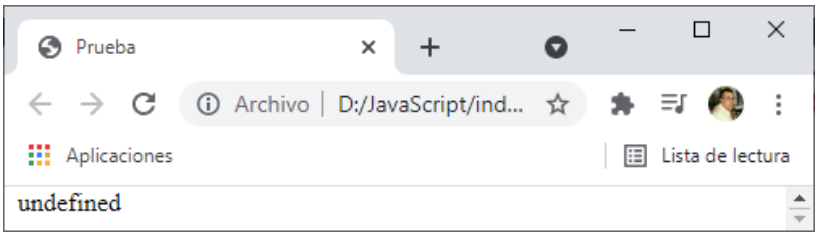
```
1   var miCadena = "JavaScript";
2   // Cadena:    J a v a S c r i p t
3   // Índice:    0 1 2 3 4 5 6 7 8 9
4   document.write(miCadena[0] + "<br>"); // J
5   document.write(miCadena[1] + "<br>"); // a
6   document.write(miCadena[2] + "<br>"); // v
7   document.write(miCadena[3] + "<br>"); // a
8   document.write(miCadena[4] + "<br>"); // S
9   document.write(miCadena[5] + "<br>"); // c
10  document.write(miCadena[6] + "<br>"); // r
11  document.write(miCadena[7] + "<br>"); // i
12  document.write(miCadena[8] + "<br>"); // p
13  document.write(miCadena[9] + "<br>"); // t
```

Este será el resultado:



Que pasa si buscamos un carácter que no está en la cadena.

```
13    document.write(miCadena[10] + "<br>");
```



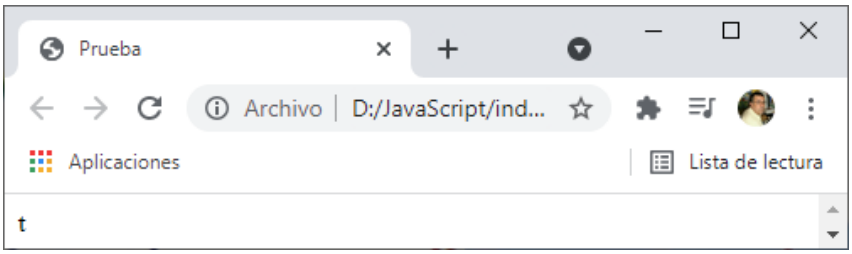
Este carácter no está definido.

Notación de corchetes: Último carácter

D: > JavaScript > JS operadores.js > ...

```
1    var miCadena;
2    /*
3    El último índice siempre el la longitud -1 porque comenzamos
4    a contar desde 0.
5    miCadena.length para "JavaScript" es 10.
6    El último índice es 9.
7    Cadena:   J a v a S c r i p t
8    Índices:  0 1 2 3 4 5 6 7 8 9
9    */
10   miCadena = "JavaCript";
11   document.write(miCadena[miCadena.length - 1]);
```

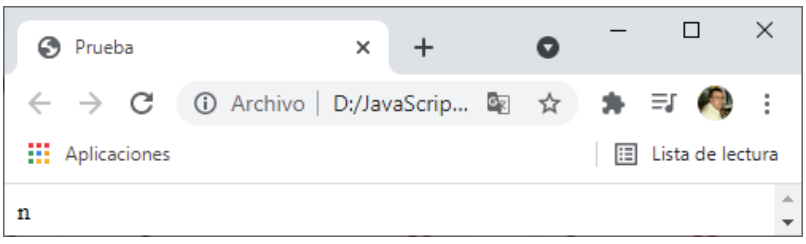
Este será el resultado:



Podemos cambiar el valor de la cadena:

```
10    miCadena = "Python";
```

Ahora este será el resultado:

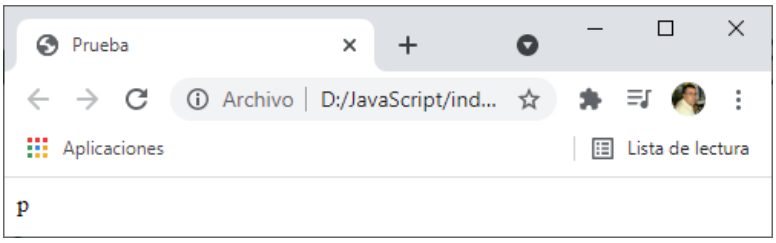


Notación de corchetes: De derecha a izquierda

D: > JavaScript > JS operadores.js > ...

```
1   var miCadena;
2   /*
3   El penúltimo índice es longitud - 2 porque comenazamos
4   a contar desde 0.
5   miCdena.length es 10. El penúltimo es 8.
6   Cadena:     J a v a S c r i p t
7   Índices:    0 1 2 3 4 5 6 7 8 9
8   */
9   miCadena = "JavaScript";
10  document.write(miCadena[miCadena.length -2]);
```

Este será el resultado:

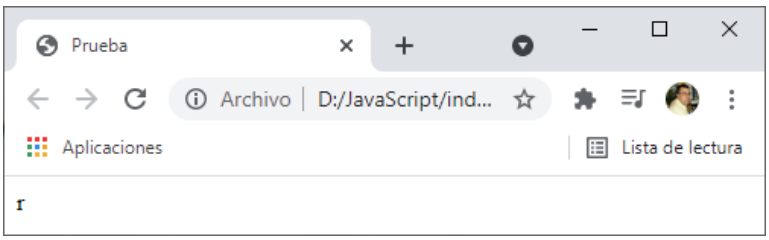


Lo podemos hacer declarando una variable.

D: > JavaScript > JS operadores.js > ...

```
1   var miCadena;
2   var n;
3   /*
4   El penúltimo índice es longitud - 2 porque comenazamos
5   a contar desde 0.
6   miCdena.length es 10. El penúltimo es 8.
7   Cadena:     J a v a S c r i p t
8   Índices:    0 1 2 3 4 5 6 7 8 9
9   */
10  miCadena = "JavaScript";
11  n=4;
12  document.write(miCadena[miCadena.length -n]);
```

Este será el resultado:



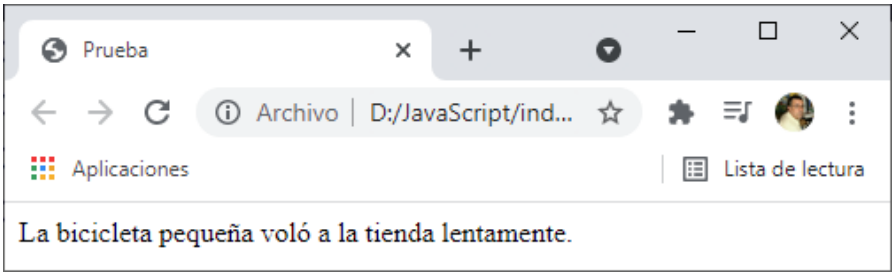
Posición 4 por la derecha.

Palabras en blanco

D: > JavaScript > JS operadores.js > ...

```
1   var miSustantivo = "bicicleta";
2   var miAdjetivo = "pequeña";
3   var miVerbo = "voló";
4   var miAdverbio = "lentamente";
5
6   /* Concatena las cadenas para crear una cadena que muestre un mensaje.
7   Puedes cambiar los valores de las variables.
8
9   Por ejemplo: El perro negro corrio rapidamente a la tienda.
10                  La bicicleta pequeña voló a la tienda lentamente.
11  */
12  var palabrasEnBlanco = "La " + miSustantivo + " " + miAdjetivo + " " + miVerbo +  " " +
    "a la tienda " + miAdverbio +".";
13  document.write(palabrasEnBlanco);
```

Este será el resultado:

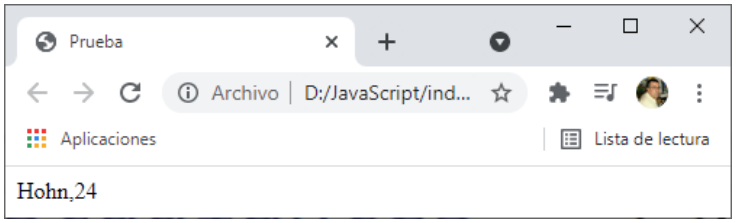


Arreglos (Arrays)

D: > JavaScript > JS operadores.js > ...

```
1   var miArreglo = ["Hohn", 24];
2   document.write(miArreglo);
```

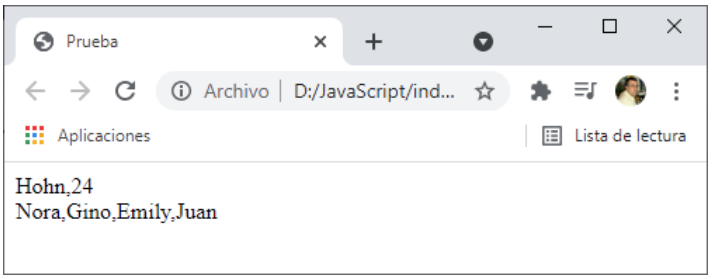
Este será el resultado:



D: > JavaScript > JS operadores.js > ...

```
1   var miArreglo = ["Hohn", 24];
2   document.write(miArreglo);
3   document.write("<br>");
4   var estudiantes = ["Nora", "Gino", "Emily", "Juan"];
5   document.write(estudiantes);
```

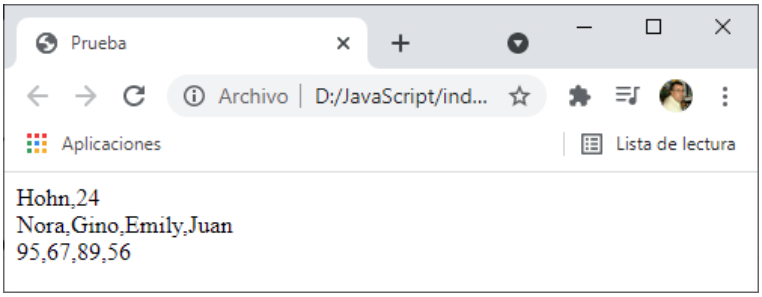
Este será el resultado:



D: > JavaScript > JS operadores.js > ...

```
var miArreglo = ["Hohn", 24];
document.write(miArreglo);
document.write("<br>");
var estudiantes = ["Nora", "Gino", "Emily", "Juan"];
document.write(estudiantes);
var notas = [95, 67, 89, 56];
document.write("<br>");
document.write(notas);
```

Este será el resultado:

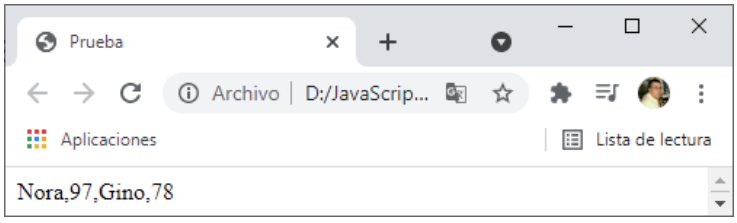


Arreglos anidados

D: > JavaScript > JS operadores.js > ...

```
var listaDeEstudiantes = [["Nora", 97], ["Gino", 78]];
document.write(listaDeEstudiantes)
```

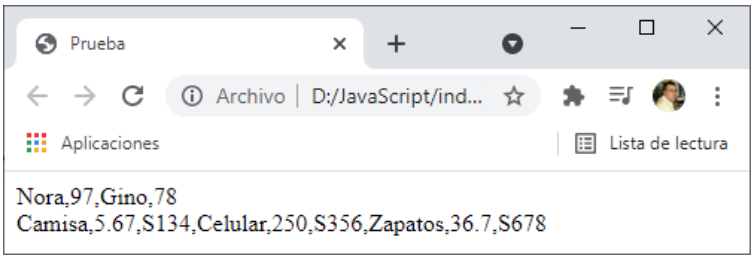
Este será el resultado:



D: > JavaScript > JS operadores.js > ...

```
var listaDeEstudiantes = [["Nora", 97], ["Gino", 78]];
document.write(listaDeEstudiantes)
document.write("<br>");
var listaDeProductos = [["Camisa", 5.67, "S134"],["Celular", 250, "S356"],["Zapatos", 36.
7, "S678"]];
document.write(listaDeProductos);
```

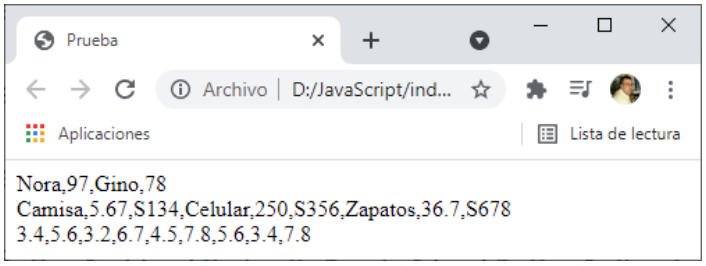
Este será el resultado:



D: > JavaScript > JS operadores.js > ...

```
1   var listaDeEstudiantes = [["Nora", 97], ["Gino", 78]];
2   document.write(listaDeEstudiantes)
3   document.write("<br>");
4   var listaDeProductos = [["Camisa", 5.67, "S134"],["Celular", 250, "S356"],["Zapatos", 36.
    7, "S678"]];
5   document.write(listaDeProductos);
6   document.write("<br>");
7   var datos = [[3.4, 5.6, 3.2],[6.7, 4.5, 7.8],[5.6, 3.4, 7.8]];
8   document.write(datos);
```

Este será el resultado:

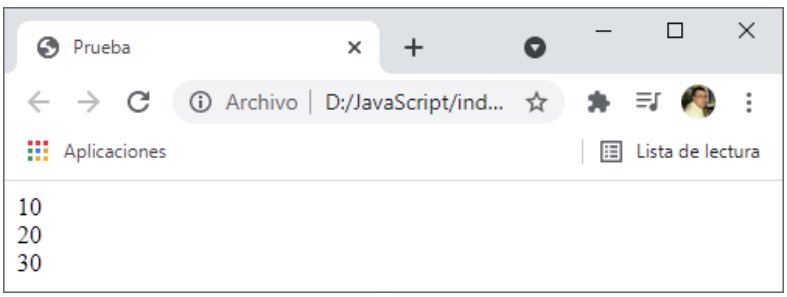


Acceder a los elementos de un arreglo

D: > JavaScript > JS operadores.js > ...

```
1   var miArreglo = [10, 20, 30];
2   /*
3   Arreglo:  [10, 20, 30]
4   Índices:    0   1   2
5   */
6   document.write(miArreglo[0]); // 10
7   document.write("<br>");
8   document.write(miArreglo[1]); // 20
9   document.write("<br>");
10  document.write(miArreglo[2]); // 30
```

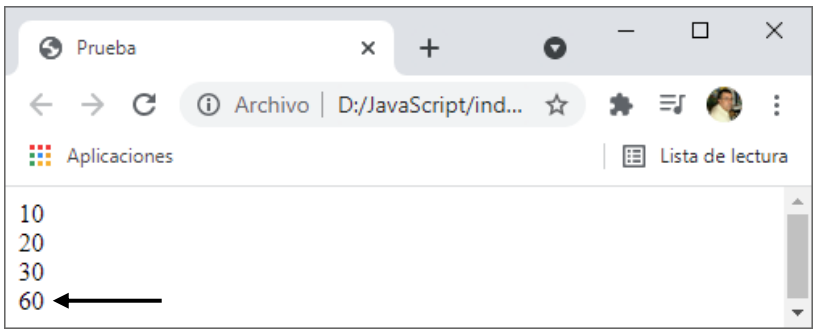
Este será el resultado:



D: > JavaScript > JS operadores.js > ...

```
1   var miArreglo = [10, 20, 30];
2   /*
3   Arreglo:  [10, 20, 30]
4   Índices:    0   1   2
5   */
6   document.write(miArreglo[0]); // 10
7   document.write("<br>");
8   document.write(miArreglo[1]); // 20
9   document.write("<br>");
10  document.write(miArreglo[2]); // 30
11  document.write("<br>");
12  var suma = miArreglo[0] + miArreglo[1] + miArreglo[2];
13  document.write(suma);
```

Este será el resultado:

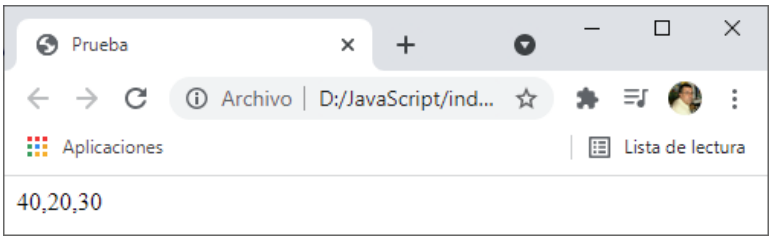


Modificar los datos de un arreglo

D: > JavaScript > JS operadores.js > ...

```
1   var miArreglo = [10, 20, 30];
2   miArreglo[0] = 40;
3   document.write(miArreglo);
```

Este es el resultado:

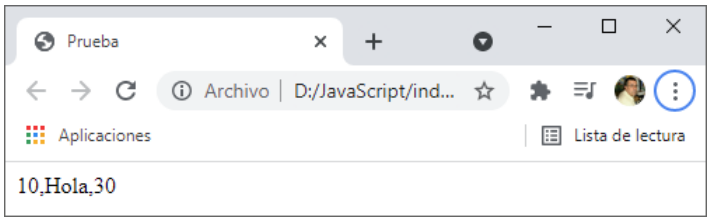


Los arreglos son mutables, podemos modificar un elemento de un arreglo.

D: > JavaScript > JS operadores.js > ...

```
1   var miArreglo = [10, 20, 30];
2   miArreglo[1] = "Hola";
3   document.write(miArreglo);
```

Este será el resultado:

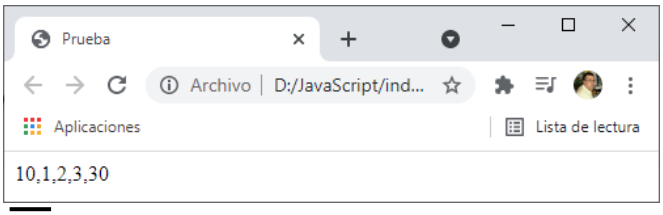


Podemos cambiar el tipo de valor de un elemento de un array.

D: > JavaScript > JS operadores.js > ...

```
1   var miArreglo = [10, 20, 30];
2   miArreglo[1] = [1, 2, 3];
3   document.write(miArreglo);
```

Este es el resultado:



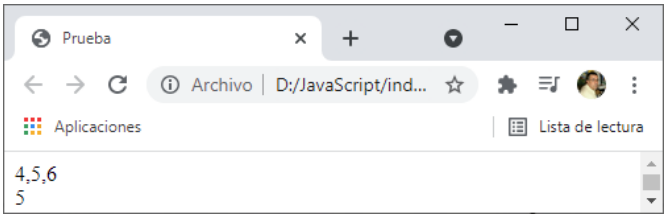
En un elemento de un array podemos almacenar otro array. [10, [1, 2, 3], 30].

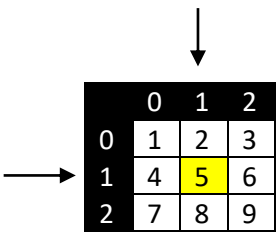
Acceder a Arreglos multidimensionales

D: > JavaScript > JS operadores.js > ...

```
1   var miArreglo = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];
2   /* Arreglo:          [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
3      Índices:               0          1          2
4      Índices internos:    0  1  2    0  1  2    0  1  2  */
5   document.write(miArreglo[1]);
6   document.write("<br>");
7   document.write(miArreglo[1][1]);
```

Este será el resultado:





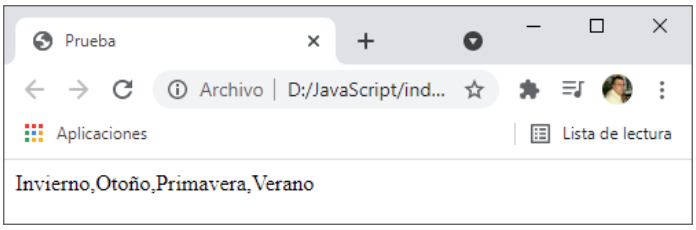
	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

Método .push()

D: > JavaScript > JS operadores.js > ...

```
1   var estaciones = ["Invierno", "Otoño", "Primavera"];
2   // Nos falta Verano
3   estaciones.push("Verano");
4   document.write(estaciones);
```

Si ejecutamos este será el resultado:



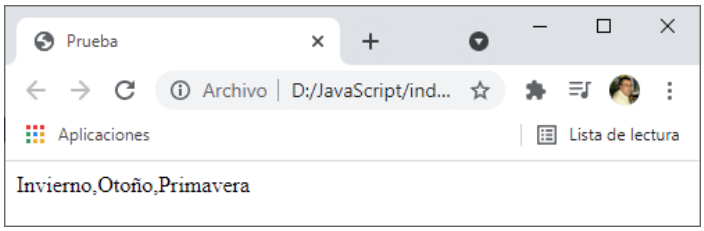
Este método nos permite añadir un elemento al final del array.

Método .pop()

D: > JavaScript > JS operadores.js > ...

```
1   var estaciones = ["Invierno", "Otoño", "Primavera", "Verano"];
2   // Nos falta Verano
3   estaciones.pop();
4   document.write(estaciones);
```

Este será el resultado:



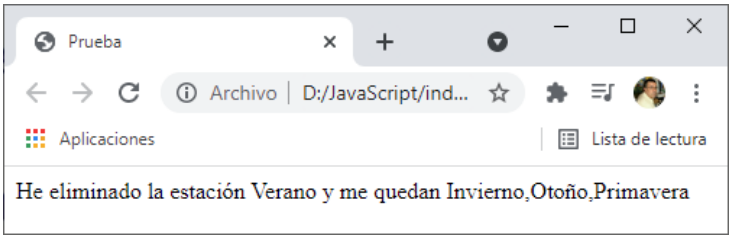
Este método elimina el último elemento del array.

D: > JavaScript > JS operadores.js > ...

```
1   var estaciones = ["Invierno", "Otoño", "Primavera", "Verano"];
2   // Nos falta Verano
3   var estacion = estaciones.pop();
4   document.write("He eliminado la estación " + estacion + " y me quedan " + estaciones);
```

El método .pop() retorna el valor eliminado.

Este será el resultado:



Método .shift()

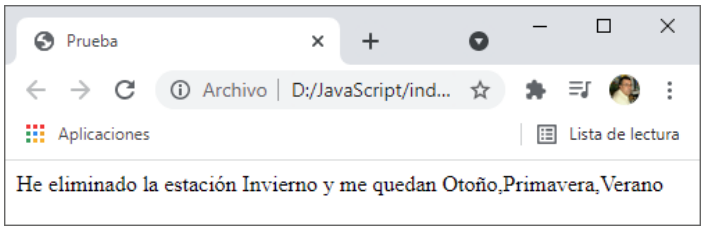
D: > JavaScript > JS operadores.js > ...

```
1   // Vamos a eliminar el primer elemento.
2   var estaciones = ["Invierno", "Otoño", "Primavera", "Verano"];
3   var estacionEliminar = estaciones.shift();
4   document.write("He eliminado la estación " + estacionEliminar + " y me quedan " + estaciones);
```

Este método elimina el primer elemento de una array.

El método .shift() retorna el valor eliminado.

Este será el resultado:



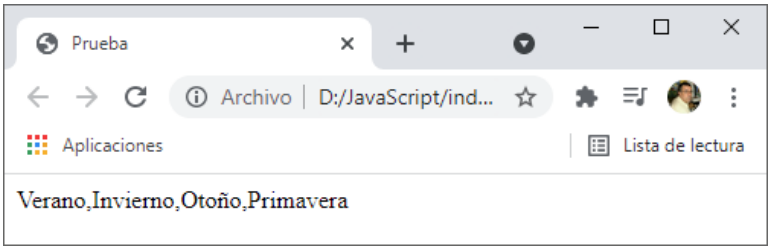
Método .unshift()

D: > JavaScript > JS operadores.js > ...

```
1   // Vamos a eliminar el primer elemento.
2   var estaciones = ["Invierno", "Otoño", "Primavera"];
3   estaciones.unshift("Verano");
4   document.write(estaciones);
```

Este método nos permite agregar un elemento al principio de la array.

Esta será el resultado:



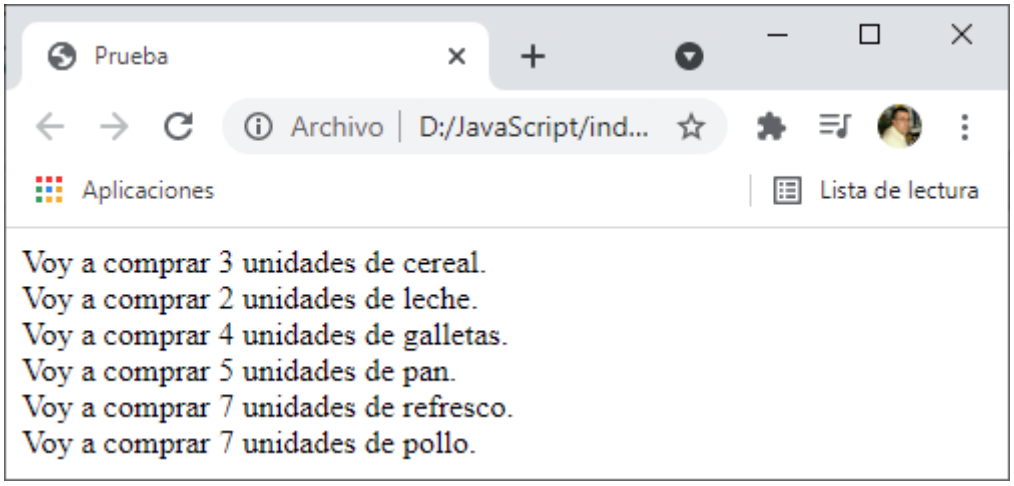
Lista de compras

D: > JavaScript > JS operadores.js > ...

```
1   var miListaDeCompras = [["cereal", 3], ["leche", 2], ["galletas", 4], ["pan",
    ["refresco", 7], ["pollo", 7]];
2   document.write("Voy a comprar " + miListaDeCompras[0][1] + " unidades de " +
    miListaDeCompras[0][0] + ".<br>");
3   document.write("Voy a comprar " + miListaDeCompras[1][1] + " unidades de " +
    miListaDeCompras[1][0] + ".<br>");
4   document.write("Voy a comprar " + miListaDeCompras[2][1] + " unidades de " +
    miListaDeCompras[2][0] + ".<br>");
5   document.write("Voy a comprar " + miListaDeCompras[3][1] + " unidades de " +
    miListaDeCompras[3][0] + ".<br>");
6   document.write("Voy a comprar " + miListaDeCompras[4][1] + " unidades de " +
    miListaDeCompras[4][0] + ".<br>");
```

```
7   document.write("Voy a comprar " + miListaDeCompras[5][1] + " unidades de " +
    miListaDeCompras[5][0] + ".<br>");
```

Este será el resultado:

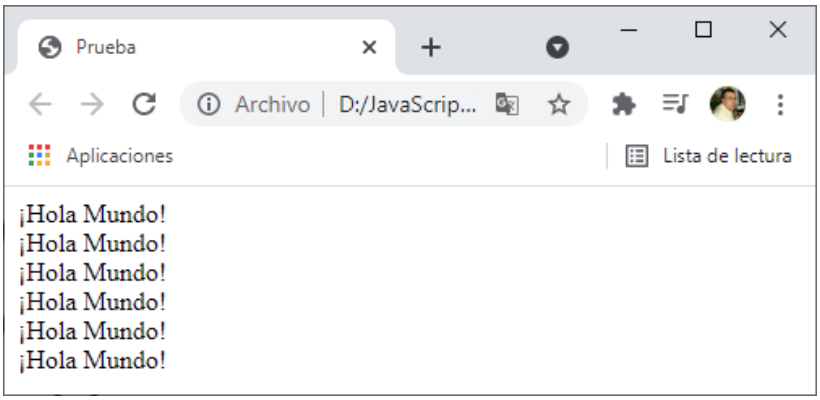


Funciones

D: > JavaScript > JS operadores.js > ...

```
1   function mostrarMensaje() {
2       document.write("¡Hola, Mundo! <br>");
3   }
4
5   mostrarMensaje();
6   mostrarMensaje();
7   mostrarMensaje();
8   mostrarMensaje();
9   mostrarMensaje();
10  mostrarMensaje();
```

Este será el resultado:



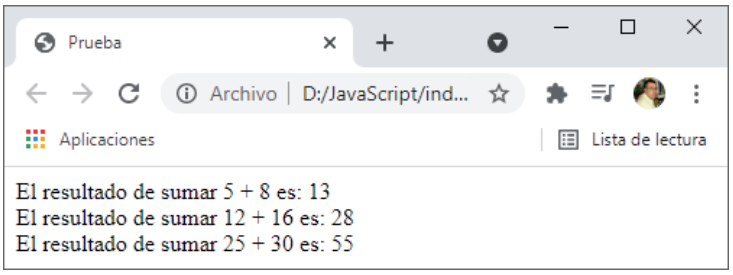
Una función en un grupo de codigo que se escribe solo una vez y se podrá reutilizar las veces que sea necesario.

Argumentos

D: > JavaScript > JS operadores.js > ...

```
1   function sumar(a, b) {
2       var suma = a + b;
3       document.write("El resultado de sumar " + a + " + " + b + " es: " + suma + "<br>");
4   }
5
6   sumar(5, 8);
7   sumar(12, 16);
8   sumar(25, 30);
```

Este será el resultado:

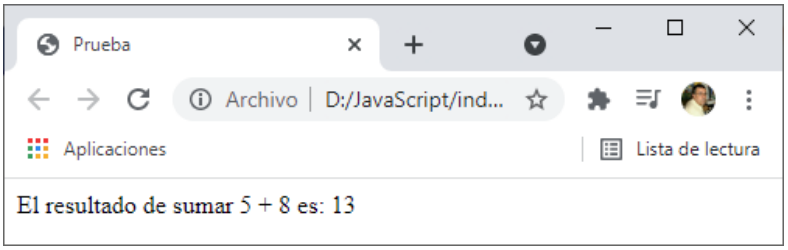


Esta función con parámetros permite realizar el mismo proceso las veces que sea necesario pero con argumentos distintos, ya que tu al llamar a la función también le pasas los valores que tiene que calcular.

D: > JavaScript > JS operadores.js > ...

```
1   function sumar(a, b) {
2       var suma = a + b;
3       document.write("El resultado de sumar " + a + " + " + b + " es: " + suma + "<br>");
4   }
5   var x = 5;
6   var y = 8;
7   sumar(x, y);
```

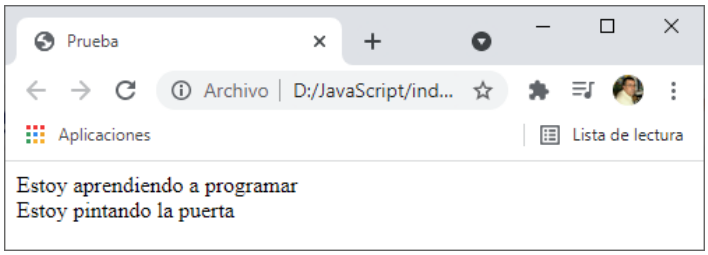
Los argumentos pueden ser variables, como en este ejemplo, este será el resultado:



D: > JavaScript > JS operadores.js > ...

```
1   function concatenarTresCadenas(cadena1, cadena2, cadena3) {
2       document.write(cadena1 + " " + cadena2 + " " + cadena3 + "<br>");
3   }
4
5   concatenarTresCadenas("Estoy", "aprendiendo", "a programar");
6   concatenarTresCadenas("Estoy", "pintando", "la puerta");
```

En este ejemplo estamos trabajando con caracteres, este será el resultado:



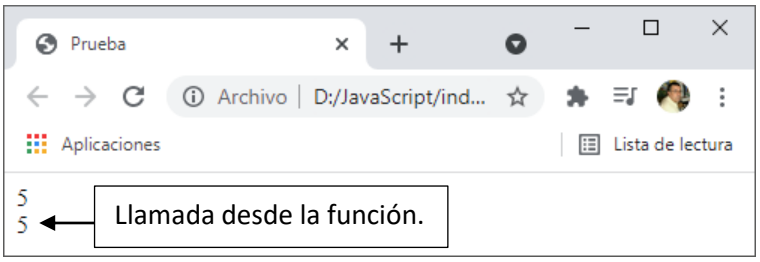
Ámbito global

Hay dos tipos de variables unas son de tipo global y otras son de tipo local.

D: > JavaScript > JS operadores.js > ...

```
1   var miVariableGlobal = 5;
2   document.write(miVariableGlobal);
3   document.write("<br>");
4   function miFuncion() {
5       document.write(miVariableGlobal);
6   }
7
8   miFuncion();
```

En este ejemplo tenemos una variable global llamada miVariableGlobal que le asignamos el valor de 5, mostramos su contenido, pero además para demostrar que dicha variable también la reconoce la función la mostramos también, este será el resultado:



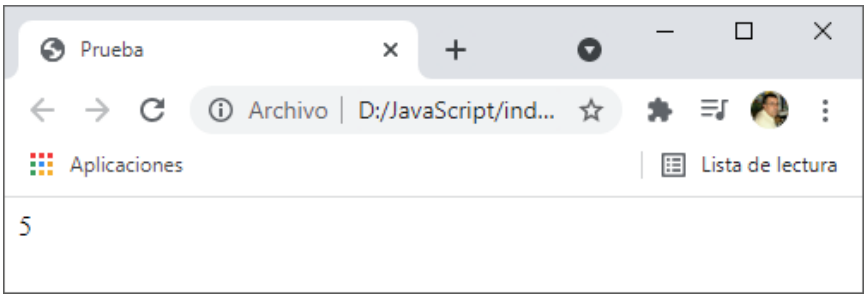
Ámbito local

D: > JavaScript > JS operadores.js > ...

```
1   function miFuncion() {
2       var miVariableLocal = 5;
3       document.write(miVariableLocal);
4       document.write("<br>");
5   }
6
7   miFuncion();
8   document.write(miVariableLocal);
```

Dentro de la función miFuncion definimos una variable llamada miVaraiableLocal con un valor inicial de 5, en la misma función la mostramos, después la queremos mostrar desde el

programa principal, esto quiere decir que la queremos mostrar dos veces, este será el resultado:



Se muestra solo una vez, ya que desde el programa principal no tenemos acceso a ella.

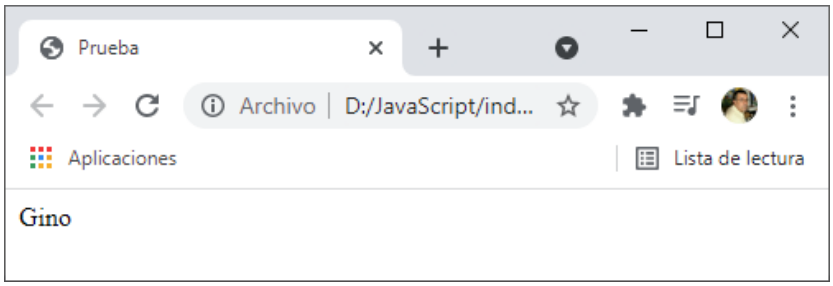
Se generará un error de miVariableLocal no está definida.

Ámbito Global vs. Ámbito Local

D: > JavaScript > JS operadores.js > ...

```
1   var miNombre = "Nora";
2   function mostrarMiNombre() {
3       var miNombre = "Gino";
4       document.write(miNombre);
5   }
6
7   mostrarMiNombre();
```

Este será el resultado:



Las variables locales tienen prioridad sobre las variables globales.

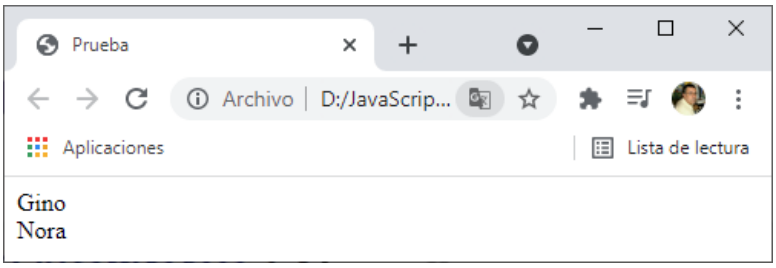
Vamos a mostrarla fuera de la función.

D: > JavaScript > JS operadores.js > ...

```
1   var miNombre = "Nora";
2   function mostrarMiNombre() {
3       var miNombre = "Gino";
4       document.write(miNombre);
5   }
```

```
6
7    mostrarMiNombre();
8
9    document.write("<br>");
10   document.write(miNombre);
```

Este será el resultado:



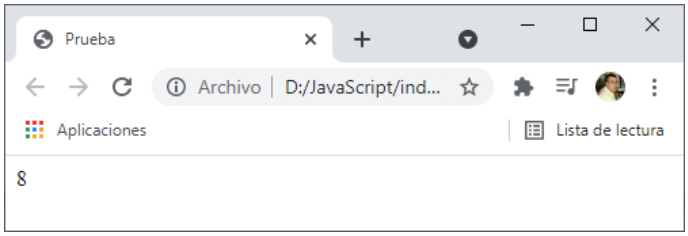
El segundo valor es de la variable global, ya que se llama desde el programa principal que se le asignaba Nora, a Gino solo se tiene acceso desde la función.

Retornar un valor

D: > JavaScript > JS operadores.js > ...

```
1    function sumar(a, b) {
2        return a + b;
3    }
4
5    document.write(sumar(5, 3));
```

Con la palabra reservada return nos permite que una función retorne un valor, en este caso no la mostramos desde la función sino que desde el programa principal la mostramos, este será el resultado:

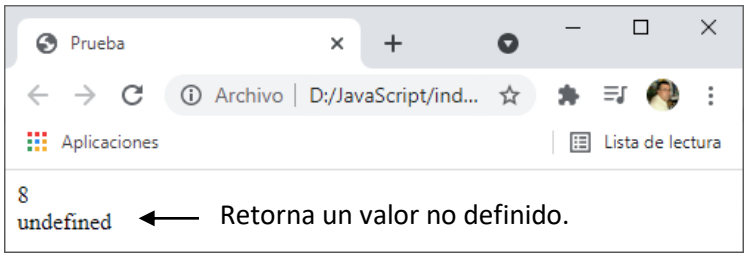


undefined

D: > JavaScript > JS operadores.js > ...

```
1    function sumar(a, b) {
2        document.write(a + b);
3    }
4    document.write("<br>" + sumar(5, 3));
```

Si mostramos el valor desde la función este se muestra sin problemas, cuando queremos mostrar el resultado de la función como no utilizamos la palabra return, nos muestra un mensaje de error, este será el resultado:

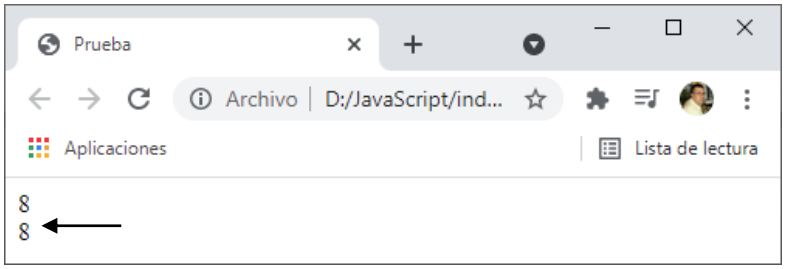


Vamos a modificar el código:

D: > JavaScript > JS operadores.js > ...

```
1   function sumar(a, b) {
2       document.write(a + b);
3       return a + b;
4   }
5   document.write("<br>" + sumar(5, 3));
```

Ahora este será el resultado:

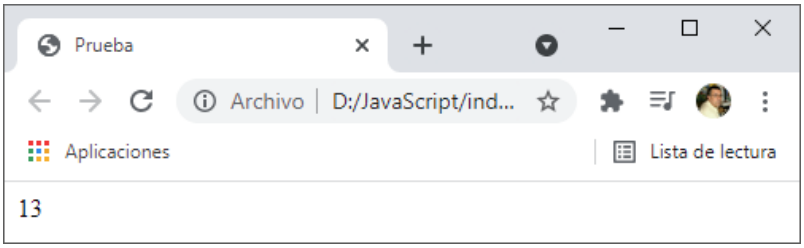


Asignar un valor retorno

D: > JavaScript > JS operadores.js > ...

```
1   function sumar(a, b) {
2       return a + b;
3   }
4   var resultado = sumar(5, 8);
5   document.write(resultado);
```

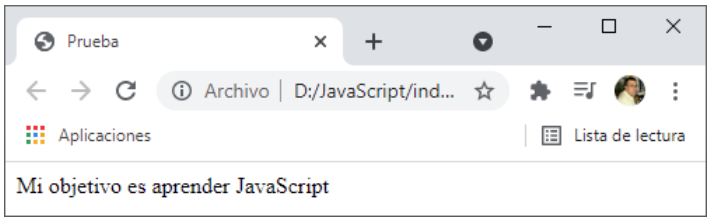
El retorno de una función se le puede asignar a una variable, y luego mostrarla o operar con ella cuando lo necesites, este será el resultado:



D: > JavaScript > JS operadores.js > ...

```
1   function crearCadena(leguajeDeProgramacion) {
2       return "Mi objetivo es aprender " + leguajeDeProgramacion;
3   }
4   var objetivo = crearCadena("JavaScript");
5   document.write(objetivo);
```

Creamos una función que retorna una cadena concatenando el valor que le pasas por parámetros. En la línea 4 llamamos a la función pasándole el argumento “JavaScript” y el resultado se lo pasamos a la variable “objetivo”, que en la siguiente línea mostramos, este será el resultado:



Permanece en fila

En informática una cola (queue) es una estructura de datos abstracta en la cual los elementos se mantienen en orden. Los nuevos elementos se pueden añadir al final de la cola y los elementos previos se retiran del principio de la cola.

Define una función proximoEnLaFila que tome un arreglo (arreglo) y un número (elemento) como argumentos. Agrega el número al final del arreglo y luego elimina el primer elemento del arreglo. La función proximoEnLaFila debe retornar el elemento que fue eliminado.

D: > JavaScript > JS operadores.js > ...

```
1   var arreglo = [1, 2, 3, 4, 5, 6];
2   function proximoEnLaFila(elemento){
3       arreglo.push(elemento);
4       return arreglo.shift();
5   }
6   document.write("Antes de llamar a la función: " + arreglo + "<br>");
7   var eliminado = proximoEnLaFila(7);
8   document.write("Es eliminado el " + eliminado + "<br>" );
9   document.write("Ahora es como está la fila: " + arreglo);
10  document.write("<br>");
11  var eliminado = proximoEnLaFila(8);
12  document.write("Es eliminado el " + eliminado + "<br>" );
13  document.write("Ahora es como está la fila: " + arreglo);
```

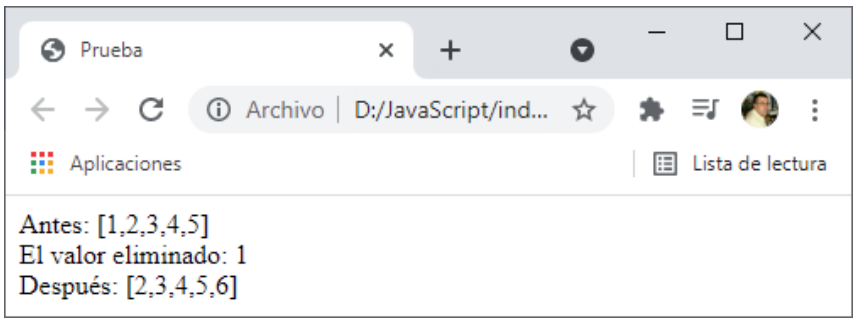
Este será el resultado:

Otra forma de hacerlo:

D: > JavaScript > JS operadores.js > ...

```
1   function proximoEnLaFila(arreglo, elemento) {
2       arreglo.push(elemento);   // Agregar al final del arreglo
3       return arreglo.shift();   // Eliminar el primer elemento
4   }
5   var miArreglo = [1, 2, 3, 4, 5];
6   document.write("Antes: " + JSON.stringify(miArreglo) + "<br>");
7   document.write("El valor eliminado: " + proximoEnLaFila(miArreglo, 6) + "<br>")
8   document.write("Después: " + JSON.stringify(miArreglo));
```

Este será el resultado:



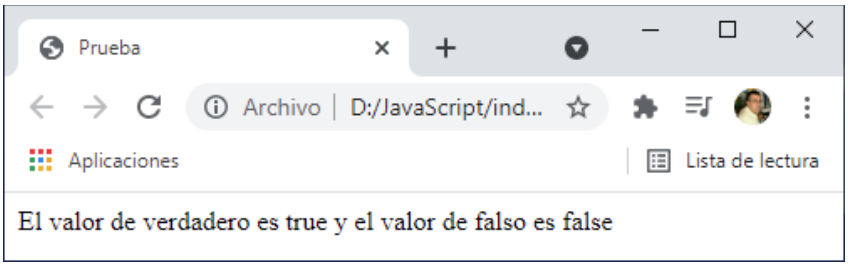
Valores Booleanos

Estos valores serán muy interesante cuando trabajemos con condicionales, operadores lógicos, etc.

D: > JavaScript > JS operadores.js > ...

```
1   // Sus valores son true y false.
2   // Escritos siempre en minúsculas.
3   var verdadero = true;
4   var falso = false;
5   document.write("El valor de verdadero es " + verdadero + " y el valor de falso es " + falso);
```

Este será el resultado:

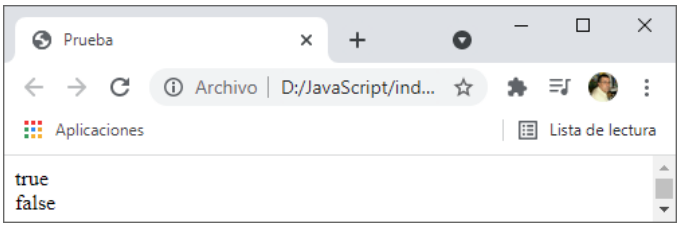


Operadores de igualdad

D: > JavaScript > JS operadores.js

```
1   document.write(5 == 5); // true
2   document.write("<br>");
3   document.write(6 == 5); // false
```

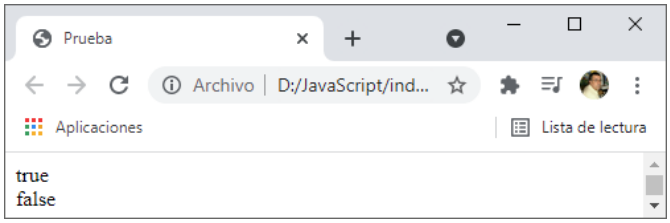
Si la condición se cumple retorna un true de lo contrario retorna un false, este será el resultado:



D: > JavaScript > JS operadores.js

```
1   document.write("Hola" == "Hola"); // true
2   document.write("<br>");
3   document.write("Hola" == "hola"); // false
```

Comparando con cadenas podemos observar que JavaScript diferencia las mayúsculas de las minúsculas, este será el resultado:

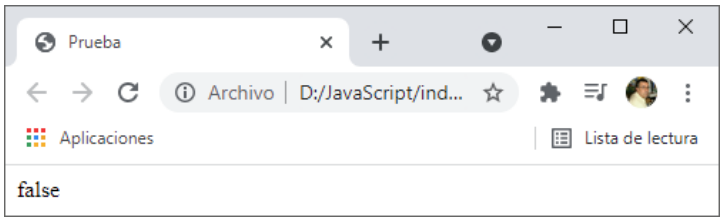


No se pueden comparar arreglos con este operador.

D: > JavaScript > JS operadores.js

```
1   document.write([1, 2, 3] == [1, 2, 3]);
```

Este será el resultado:



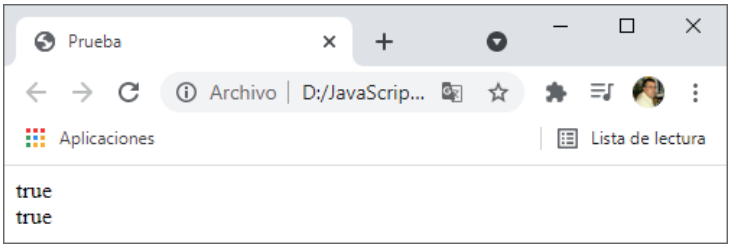
En este caso no compara valor por valor, sino compara elementos en la memoria.

Operadores de igualdad estricta

D: > JavaScript > JS operadores.js

```
1   document.write(9 ==9);
2   document.write("<br>");
3   document.write(9 =="9");
```

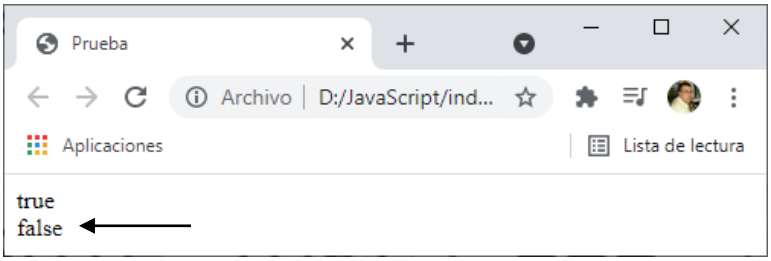
En el ejemplo de la línea 3 el dato es el mismo pero el tipo de variable no pero con este operador de igualdad no lo diferencia, este será el resultado:



D: > JavaScript > JS operadores.js

```
1   document.write(9 ===9);
2   document.write("<br>");
3   document.write(9 ==="9"); // Operador de igualdad estricta.
```

Este será el resultado:

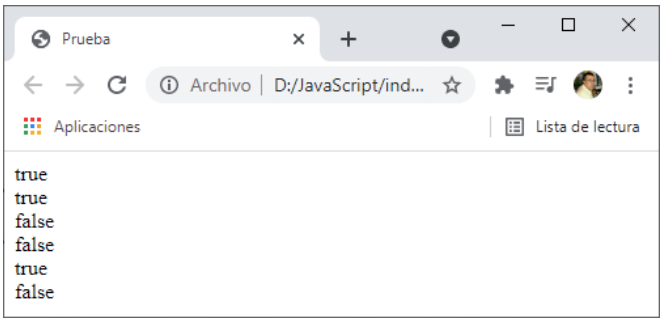


Practica cómo comparar valores

D: > JavaScript > JS operadores.js > ...

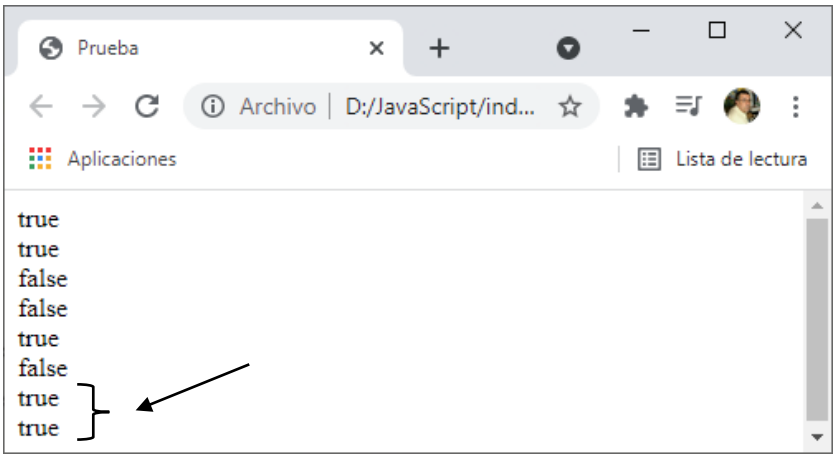
```
1   var a;
2   var b;
3   a = 5;
4   b = 5;
5   document.write(a == b); // true
6   document.write("<br>");
7   document.write(a === b); // true
8   document.write("<br>");
9   b = 8;
10  document.write(a == b); // false
11  document.write("<br>");
12  document.write(a === b); // false
13  document.write("<br>");
14  b = "5";
15  document.write(a == b); // true
16  document.write("<br>");
17  document.write(a === b); // false
18  document.write("<br>");
```

Este será el resultado:



```
19    a = "JavaScript";
20    b = "JavaScript";
21    document.write(a == b); // true
22    document.write("<br>");
23    document.write(a === b); // true
24    document.write("<br>");
```

Para comparar cadena de caracteres, este será el resultado:

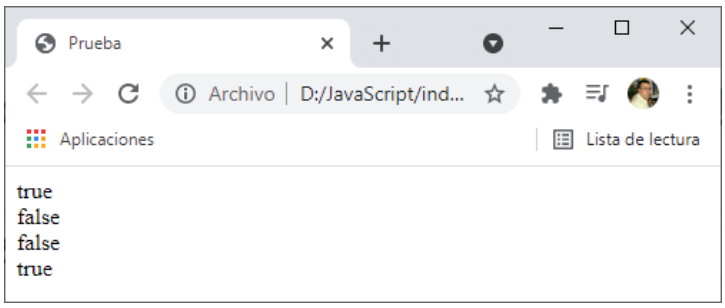


Operador de desigualdad

D: > JavaScript > JS operadores.js

```
1    document.write( 9 != 6); // No igual a
2    document.write("<br>");
3    document.write( 9 != 9);
4    document.write("<br>");
5    document.write( "JavaScript" != "JavaScript");
6    document.write("<br>");
7    document.write( [1, 2, 3] != [1, 2, 3]); // true son objetos independietes
```

Este será el resultado:

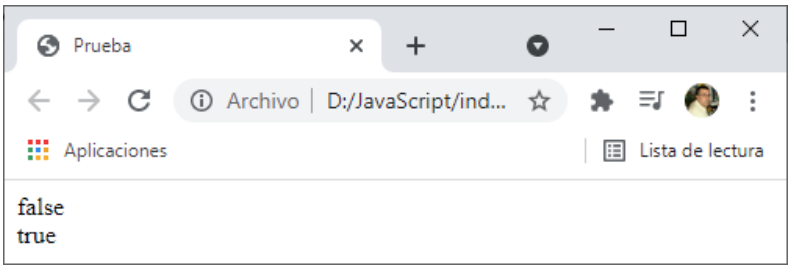


Operadores de desigualdad estricta

D: > JavaScript > JS operadores.js

```
1    document.write(1 != "1");
2    document.write("<br>");
3    document.write(1 !== "1"); // El resultado será verdadero.
```

Trabaja igual que el operador de igualdad estricta pero la distinto de, este será el resultado:

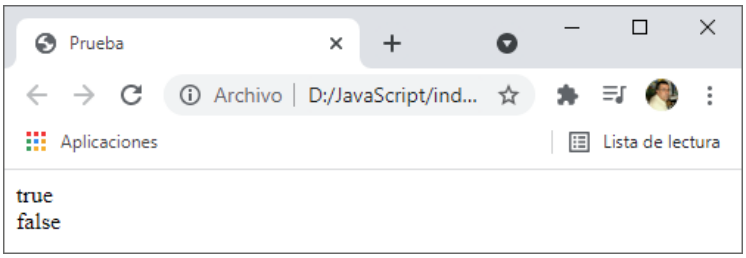


Operador mayor que

D: > JavaScript > JS operadores.js

```
1   // el operador mayor que se representa con '>'
2   document.write(6 > 5); // true
3   document.write("<br>");
4   document.write(3 > 10); // false
```

Este será el resultado:

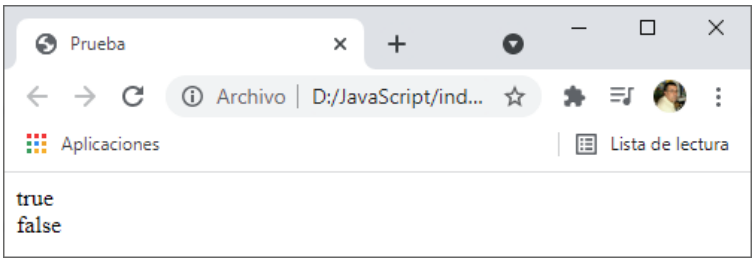


También compara las cadenas según el orden alfabético.

D: > JavaScript > JS operadores.js

```
1   // el operador mayor que se representa con '>'
2   document.write("C" > "A"); // true
3   document.write("<br>");
4   document.write("B" > "D"); // false
```

Este será el resultado:

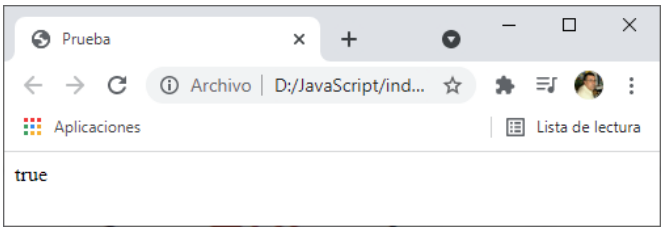


D: > JavaScript > JS operadores.js

```
1   // el operador mayor que se representa con '>'
2   document.write("ACB" > "ABC"); // true
```

Si es una cadena de caracteres va comparando una por una hasta dictaminar si es mayor o no.

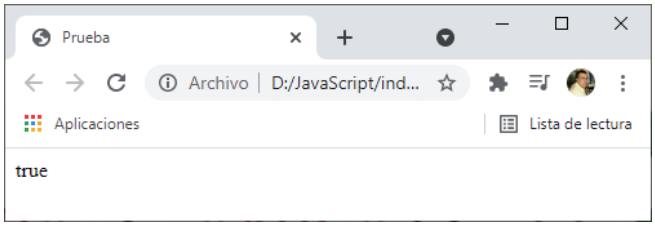
Este será el resultado:



D: > JavaScript > JS operadores.js

```
1   // el operador mayor que se representa con '>'
2   document.write("AB" > "A"); // true
```

Si una cadena empieza igual que la segunda, pero esta tiene más letras esta será mayor que la segundo, este será el resultado:

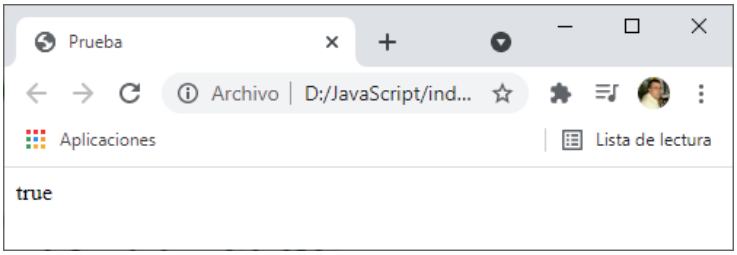


Podemos utilizar los operadores de condición con variables.

D: > JavaScript > JS operadores.js > ...

```
1   // el operador mayor que se representa con '>'
2   var palabra1 = "Mundo";
3   var palabra2 = "Hola";
4   document.write(palabra1 > palabra2);
```

Este será el resultado:

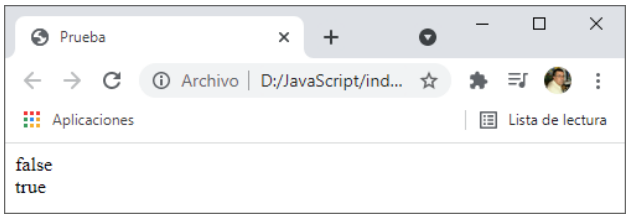


Operador mayor o igual que

D: > JavaScript > JS operadores.js

```
1   // el operador mayor igual que que se representa con '>='
2   document.write(5 > 5);
3   document.write("<br>");
4   document.write(5 >= 5);
```

La diferencia está en si queremos incluir además del mayor el igual, en este ejemplo se ve muy claro, también sirve para la cadena de caracteres, este es el resultado:

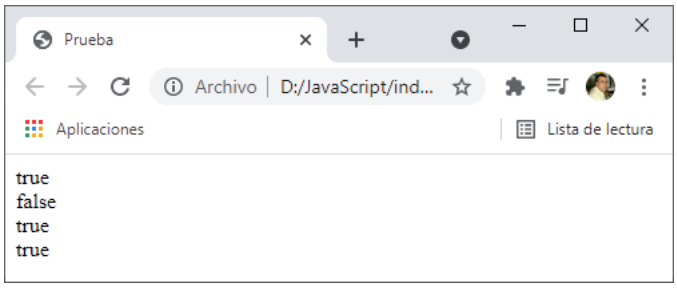


Operador menor que

D: > JavaScript > JS operadores.js

```
1   // el operador menor que que se representa con '<'
2   document.write(5 < 6); // true
3   document.write("<br>");
4   document.write(10 < 3);// false
5   document.write("<br>");
6   document.write("A" < "B"); // true
7   document.write("<br>");
8   document.write("ABC" < "ACB"); // true
```

Este será el resultado:

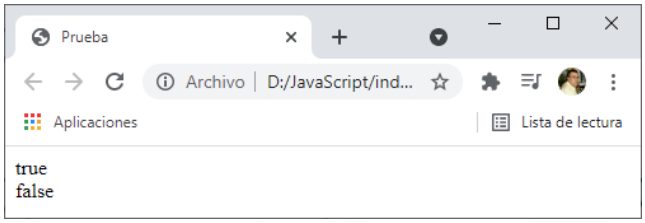


También podemos comparar los valores de las variables.

D: > JavaScript > JS operadores.js > ...

```
1   // el operador menor que que se representa con '<'
2   var a = 10;
3   var b = 15;
4   document.write(a < b); // true
5   document.write("<br>");
6   document.write(b < a); // false
```

Este será el resultado:

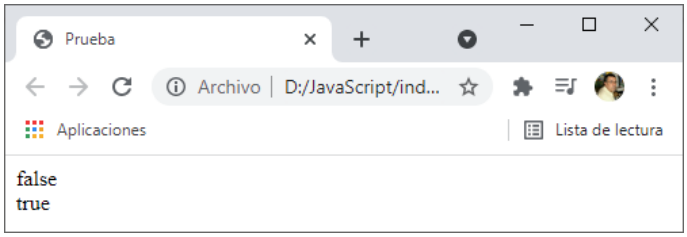


Operador menor o igual que

D: > JavaScript > JS operadores.js

```
1   // el operador menor o igual que que se representa con '<='
2   document.write(5 < 5); // false
3   document.write("<br>");
4   document.write(5 <= 5); // true
```

Este será el resultado:



Operador lógico and

Tabla de verdad del operador AND

Para X && Y

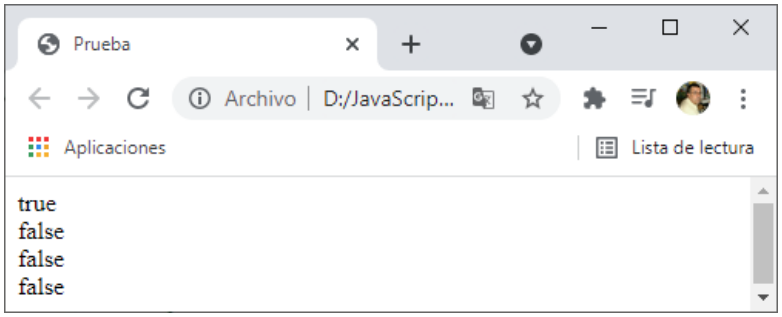
X	Y	X && Y
T	T	T
T	F	F
F	T	F
F	F	F

La expresión solo es verdadera cuando ambos operandos son verdaderos.

D: > JavaScript > JS operadores.js

```
1   document.write(true && true );
2   document.write("<br>")
3   document.write(true && false );
4   document.write("<br>")
5   document.write(false && true );
6   document.write("<br>")
7   document.write(false && false);
```

Este será el resultado:

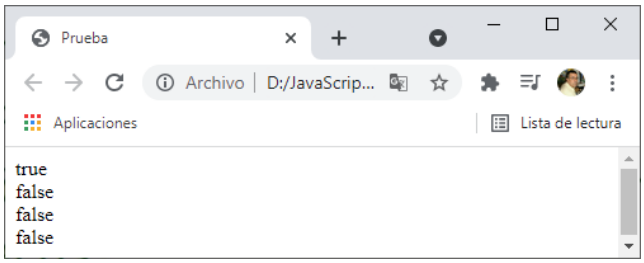


Vamos a escribir condiciones.

D: > JavaScript > JS operadores.js > ...

```
1   var a = 8;
2   document.write((a > 5) && (a < 10));
3   document.write("<br>")
4   var a = 3;
5   document.write((a > 5) && (a < 10));
6   document.write("<br>")
7   var a = 15;
8   document.write((a > 5) && (a < 10));
9   document.write("<br>")
10  var a = 1;
11  document.write((a > 5) && (a == 3));
```

Este será el resultado:



Operador lógico or

Tabla de verdad del operador OR

Para x || Y

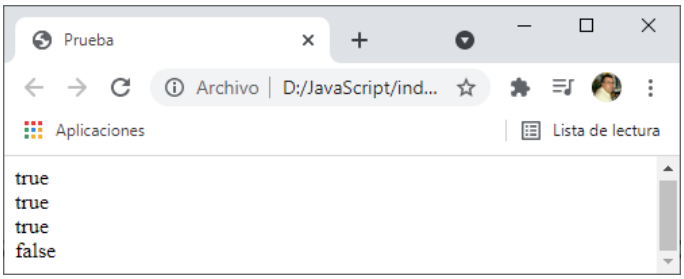
X	Y	X || Y
T	T	T
T	F	T
F	T	T
F	F	F

La expresión es verdad si alguno de los dos operadores o ambos son verdaderos.

D: > JavaScript > JS operadores.js

```
1   document.write(true || true);
2   document.write("<br>")
3   document.write(true || false);
4   document.write("<br>")
5   document.write(false || true);
6   document.write("<br>")
7   document.write(false || false);
```

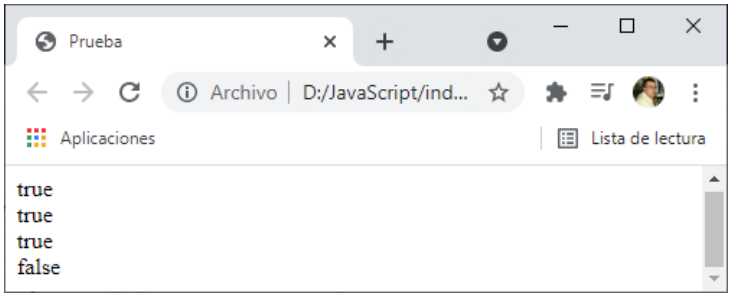
Este será el resultado:



D: > JavaScript > JS operadores.js > ...

```
1   var a = 2;
2   document.write((a < 5) || (a> 15));
3   document.write("<br>")
4   var a = 8;
5   document.write((a < 5) || (a == 8));
6   document.write("<br>")
7   var a = 20;
8   document.write((a < 5) || (a> 15));
9   document.write("<br>")
10  document.write((a == 5) || a < 15);
```

Este será el resultado:



Operador lógico not

Tabla de verdad del operador NOT

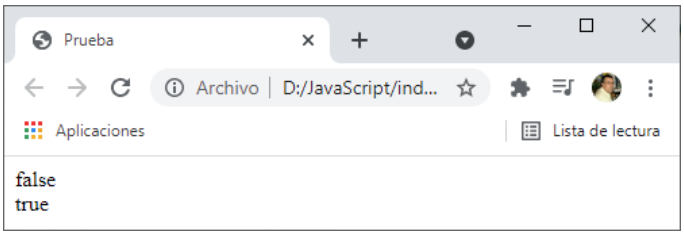
Para ¡X

X	¡X
T	F
F	T

D: > JavaScript > JS operadores.js

```
1   document.write(!true);
2   document.write("<br>");
3   document.write(!false);
```

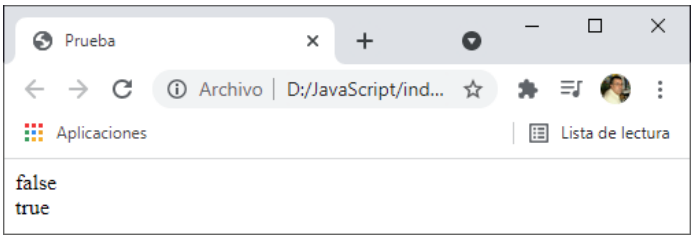
Este será el resultado:



D: > JavaScript > JS operadores.js > ...

```
var a = 8;
document.write(!(a > 5));
document.write("<br>");
document.write(!(a < 5));
```

Este será el resultado:

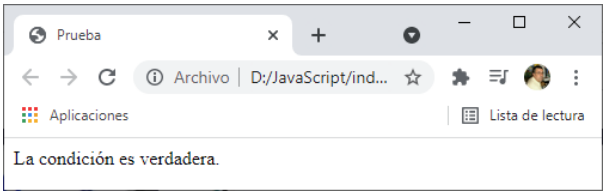


Sentencias condicionales

D: > JavaScript > JS operadores.js

```
if (true) {
    document.write("La condición es verdadera.");
}
```

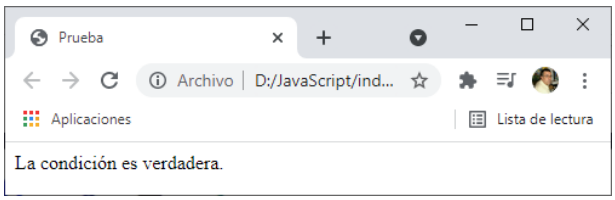
La sentencia if ejecuta el codigo que tiene entre llaves y la condición es verdadera, este será el resultado.



D: > JavaScript > JS operadores.js > ...

```
var x = 5;
if (x > 2) {
    document.write("La condición es verdadera.");
}
```

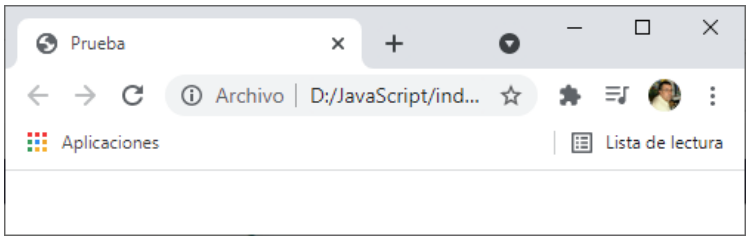
Si x es mayor de 2 que ejecute el texto que tiene entre llaves, este será el resultado.



D: > JavaScript > JS operadores.js > ...

```
1   var x = 5;
2   if (x < 2) {
3       document.write("La condición es verdadera.");
4   }
```

Este será el resultado:

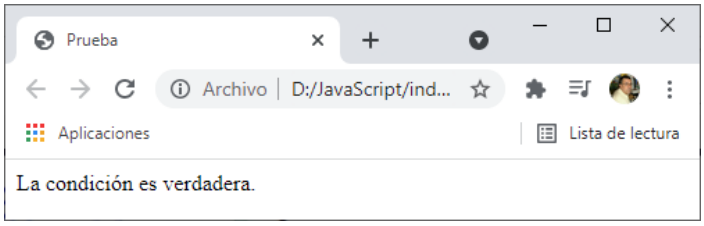


Al no cumplirse la condición la instrucción o instrucciones que hay en el if no se ejecutarán.

D: > JavaScript > JS operadores.js > ...

```
1   var x = 5;
2   if (x > 2 && x < 10) {
3       document.write("La condición es verdadera.");
4   }
```

En este ejemplo estamos trabajando con los operadores de comparación y lógicos, este será el resultado:

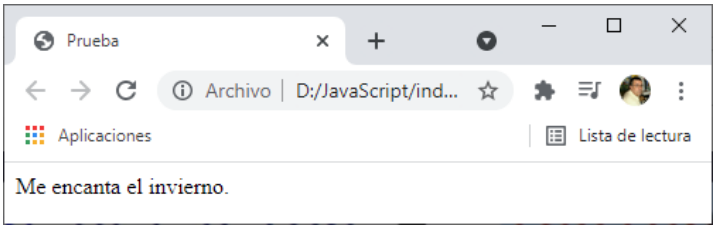


Otro ejemplo:

D: > JavaScript > JS operadores.js > ...

```
1   var estacion = "Invierno";
2   if(estacion == "Invierno") {
3       document.write("Me encanta el invierno.")
4   }
```

Este será el resultado:

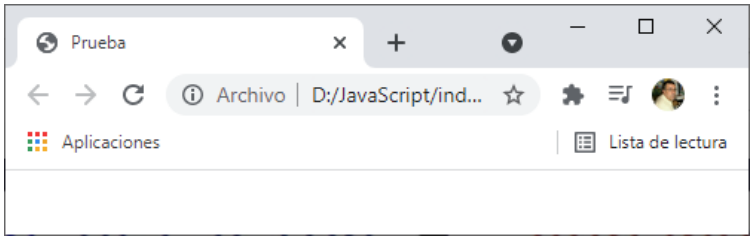


Ahora la variable estación será igual a Verano.

D: > JavaScript > JS operadores.js > ...

```
1   var estacion = "Verano";
2   if(estacion == "Invierno") {
3       document.write("Me encanta el invierno.")
4   }
```

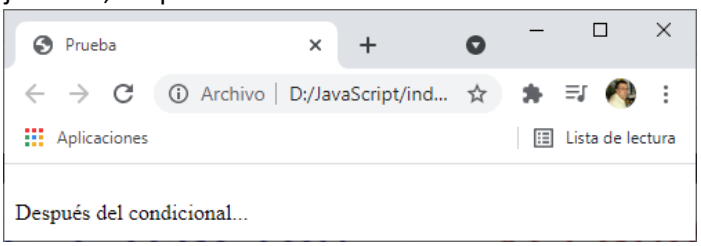
Este será el resultado:



D: > JavaScript > JS operadores.js > ...

```
1   var estacion = "Verano";
2   if(estacion == "Invierno") {
3       document.write("Me encanta el invierno.")
4   }
5   document.write("<br>");
6   document.write("Después del condicional...");
```

En este ejemplo observamos que después de hacer la comparación, si se cumple la condición o no continuará su ejecución, después del if.

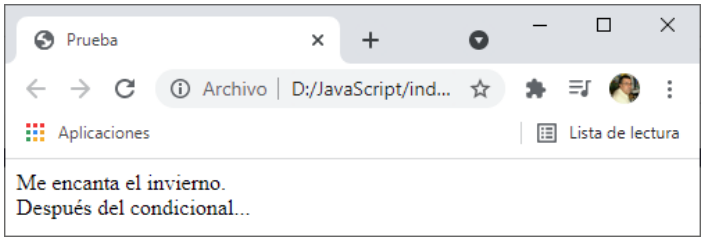


Si se cumple la condición:

D: > JavaScript > JS operadores.js > ...

```
1   var estacion = "Invierno";
2   if(estacion == "Invierno") {
3       document.write("Me encanta el invierno.")
4   }
5   document.write("<br>");
6   document.write("Después del condicional...");
```

Este será el resultado:



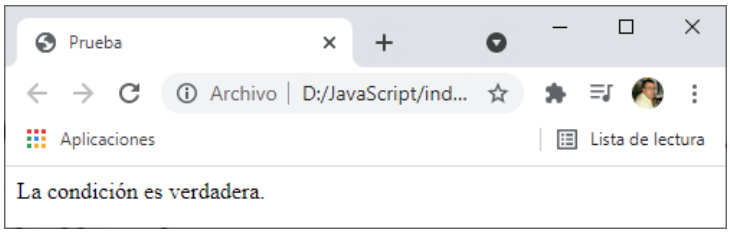
Cláusula “else”

D: > JavaScript > JS operadores.js

```
1   if (true) {
2       document.write("La condición es verdadera.");
3   } else {
4       document.write("La condición es falsa");
5   }
```

else se ejecuta cuando la condición es falsa.

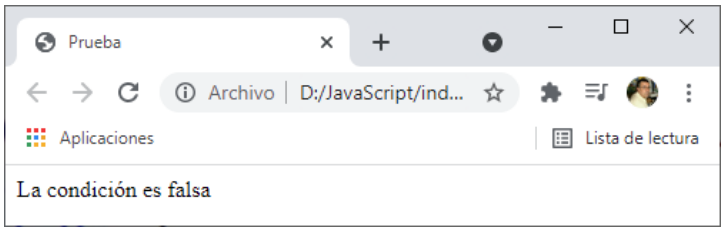
Si la condición es verdadera este será el resultado:



D: > JavaScript > JS operadores.js

```
1   if (false) {
2       document.write("La condición es verdadera.");
3   } else {
4       document.write("La condición es falsa");
5   }
```

Si la condición es falsa este será el resultado:

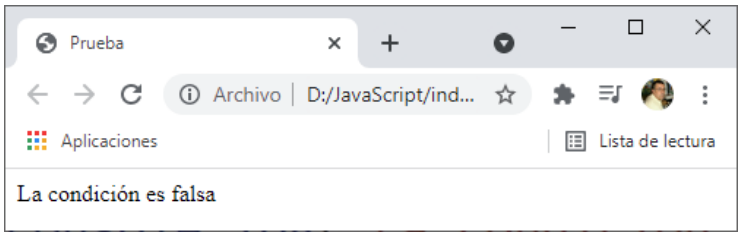


Un ejemplo:

D: > JavaScript > JS operadores.js > ...

```
1   var x = 5;
2   if (x < 2) {
3       document.write("La condición es verdadera.");
4   } else {
5       document.write("La condición es falsa");
6   }
```

Este será el resultado:

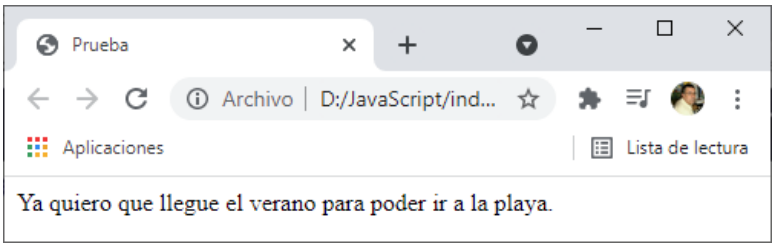


Otro ejemplo:

D: > JavaScript > JS operadores.js > ...

```
var estacion = "Invierno";
if (estacion === "Verano") {
    document.write("Comenzó el verano. Ya podemos ir a la playa.");
} else {
    document.write("Ya quiero que llegue el verano para poder ir a la playa.")
}
```

Este ejemplo lo hacemos con igualdad estricta, este será el resultado:



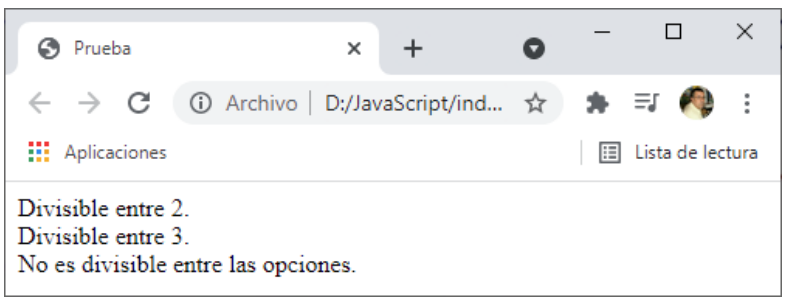
Cláusula “else if”

D: > JavaScript > JS operadores.js > ...

```
function clasificarValor(valor){
    if (valor % 2 == 0) {
        document.write("Divisible entre 2.<br>");
    } else if(valor % 3 == 0) {
        document.write("Divisible entre 3. <br>");
    } else {
        document.write("No es divisible entre las opciones.<br>");
    }
}

clasificarValor(2);
clasificarValor(3);
clasificarValor(5);
```

Este será el resultado:



Hemos creado una función que la llamaremos con varios valores, el primero es divisible por 2, el segundo es divisible por 3 el tercero al no ser divisible ni por 2 ni por tres, muestra el mensaje que se encuentra en else.

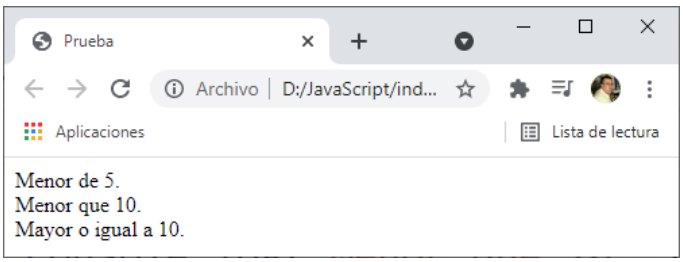
Condicionales: Orden lógico

D: > JavaScript > JS operadores.js > ...

```
1   function clasificarValor(valor){
2       if (valor < 5) {
3           document.write("Menor de 5. <br>");
4       } else if(valor < 10) {
5           document.write("Menor que 10. <br>");
6       } else {
7           document.write("Mayor o igual a 10. <br>");
8       }
9   }
10
11  clasificarValor(2);
12  clasificarValor(7);
13  clasificarValor(12);
```

A la primera llamada a la función que le pasamos el valor 2, podemos observar que cumple dos condiciones que es menor de 5 y que es menor de 10, pero al cumplir la primera condición ignora el resto de condiciones, en la llamada a la segunda función con el valor 7 la primera condición no la cumple, en cambio si cumple la segunda condición de este modo ignora el else y en la tercera llamada con el valor 12, al no cumplir ni la primera ni segunda condición ejecuta el codigo que se encuentra en el else condición que sirve como respaldo.

Este será el resultado:



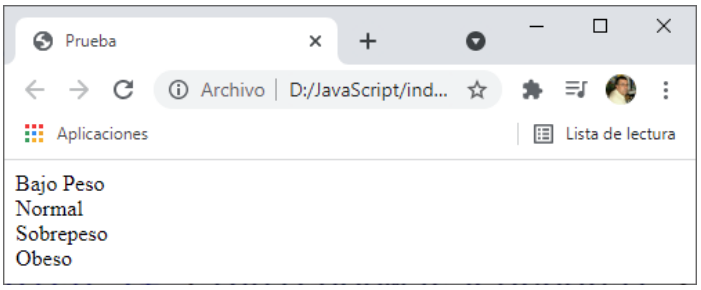
Encadenar sentencias “if...else”

D: > JavaScript > JS operadores.js > ...

```
1   function interpretarIMV(indiceDeMasaCorporal) {
2       if(indiceDeMasaCorporal < 18.5) {
3           document.write("Bajo Peso <br>");
4       } else if (indiceDeMasaCorporal <= 24.9) {
5           document.write("Normal <br>");
6       } else if(indiceDeMasaCorporal <= 29.9) {
7           document.write("Sobrepeso <br>");
8       } else {
9           document.write("Obeso <br>");
```

```
10      }
11  }
12
13  interpretarIMV(17.8);
14  interpretarIMV(22.2);
15  interpretarIMV(28.5);
16  interpretarIMV(32.2);
```

En este ejemplo queremos saber según el índice de masa corporal nos diga si estamos “Bajo en peso”, “Normal”, “Sobrepeso” y “Obeso”, hemos llamado a la función varias veces con distintos índices de masa corporal, este será el resultado:



Recuerda que cuando se cumpla una condición se ejecutará el código de dicha condición y el resto de condiciones ya no sé verifican.

Código de Golf

En el jugo de golf cada hoyo tiene un par que representa el número promedio de golpes que se espera que haga un golfista para introducir la pelota en el hoyo.

Hay un nombre diferente dependiendo de que tan por encima o por abajo del par estén sus golpes.

Tu función tomará los argumentos par y golpes.

Retornará la cadena correcta según esta tabla que muestra los golpes en orden de mayor a menor prioridad.

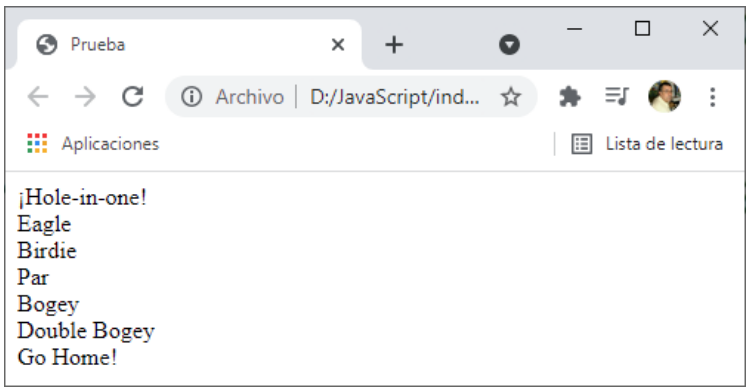
Golpes	Retorna
1	“Hole-in-one!”
<= par -2	“Eagle”
par -1	“Birdie”
par	“Par”
par + 1	“Bogey”
par + 2	“Double Bogey”
>= par + 3	“Go Home!”

Par y golpes siempre será numéricos y positivos.

D: > JavaScript > JS operadores.js > ...

```
1   function puntajeDeGolg(par, golpes) {
2       if(golpes == 1){
3           return "¡Hole-in-one!";
4       } else if(golpes <= par - 2) {
5           return "Eagle";
6       } else if(golpes == par -1) {
7           return "Birdie";
8       } else if(golpes == par) {
9           return "Par";
10      } else if(golpes == par + 1){
11          return "Bogey";
12      } else if(golpes == par + 2){
13          return "Double Bogey";
14      } else if(golpes >= par + 3) {
15          return "Go Home!"
16      }
17  }
18
19  document.write(puntajeDeGolg(3, 1) + "<br>");
20  document.write(puntajeDeGolg(5, 3) + "<br>");
21  document.write(puntajeDeGolg(7, 6) + "<br>");
22  document.write(puntajeDeGolg(4, 4) + "<br>");
23  document.write(puntajeDeGolg(5, 6) + "<br>");
24  document.write(puntajeDeGolg(7, 9) + "<br>");
25  document.write(puntajeDeGolg(10, 15) + "<br>");
```

Este será el resultado:



Sentencias Switch

D: > JavaScript > JS operadores.js > clasificarValor

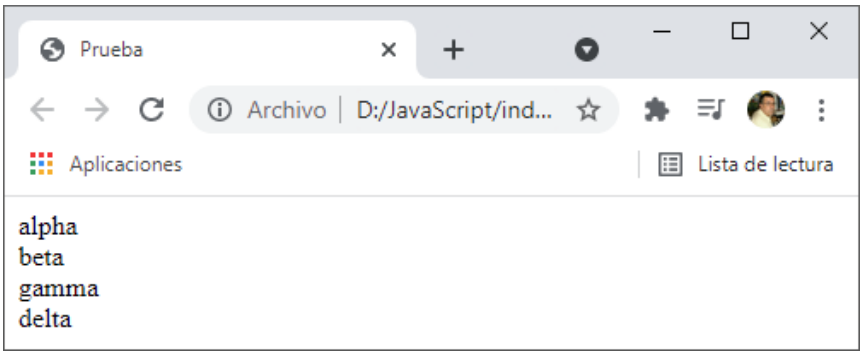
```
1   function clasificarValor(valor){
2       var respuesta;
3       switch (valor){
4           case 1:
5               respuesta = "alpha";
```

```
6           break;
7       case 2:
8           respuesta = "beta";
9           break;
10      case 3:
11          respuesta = "gamma";
12          break;
13      case 4:
14          respuesta = "delta";
15          break;
16  }
17  return respuesta;
18  }
19  document.write(clasificarValor(1) + "<br>");
20  document.write(clasificarValor(2) + "<br>");
21  document.write(clasificarValor(3) + "<br>");
22  document.write(clasificarValor(4) + "<br>");
```

Esta sentencia según el valor (1, 2, 3 o 4) podríamos decir según el caso 1, 2, 3 o 4 la variable respuesta asume el valor Alpha, beta, gamma o delta.

Ponemos la instrucción break (interrumpir) con el fin de que si algún caso se cumple no siga con la sentencia Swtch.

Al final esta función nos retornará dicho valor, este será el resultado y que llamamos a la función con 4 valores distintos, este será el resultado:



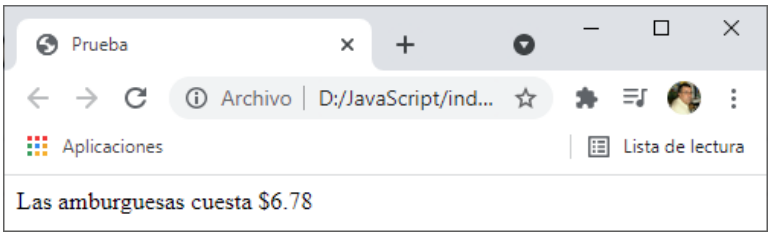
Nota: el último break se puede omitir.

Otro ejemplo:

D: > JavaScript > JS operadores.js > ...

```
1   var producto = "hamburguesa";
2   switch (producto) {
3       case "pizza":
4       document.write("La pizza básica cuesta $10.55");
5       break;
6       case "hamburguesa":
7       document.write("Las amburguesas cuesta $6.78");
8       break;
9       case "helado":
10      document.write("El helado cuesta $2.80");
11      break;
12  }
```

Este será el resultado:



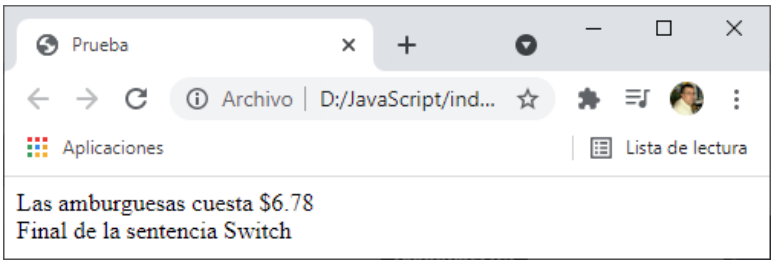
Cuando se cumple la condición la sentencia break hace que salgamos de switch, si hubiera más codigo después de esta sentencia switch seguiría con la ejecución.

D: > JavaScript > JS operadores.js > ...

```
var producto = "hamburguesa";
switch (producto) {
    case "pizza":
    document.write("La pizza básica cuesta $10.55");
    break;
    case "hamburguesa":
    document.write("Las amburguesas cuesta $6.78");
    break;
    case "helado":
    document.write("El helado cuesta $2.80");
    break;
}

document.write("<br>");
document.write("Final de la sentencia Switch");
```

Este será el resultado:



También podemos definir una opción predeterminada.

Sentencia Switch: Opción predeterminada

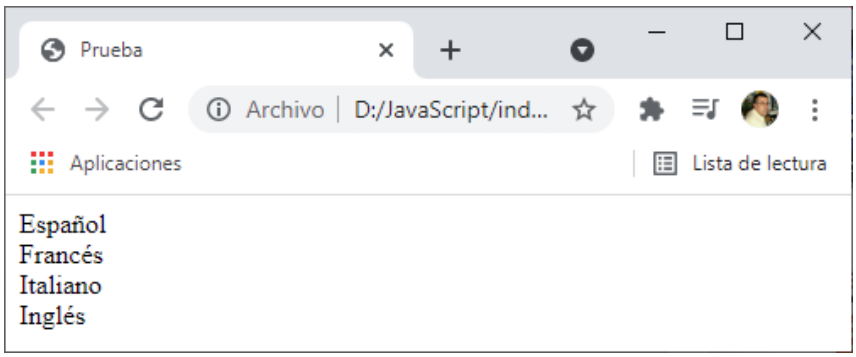
D: > JavaScript > JS operadores.js > ...

```
function seleccionarIdioma(valor) {
    var idioma;
    switch(valor){
        case 1:
            idioma = "Español";
            break;
        case 2:
            idioma = "Francés";
            break;
```

```
        case 3:
            idioma = "Italiano";
            break;
        default:
            idioma = "Inglés"
            break;
    }
    return idioma;
}

document.write(seleccionarIdioma(1) + "<br>");
document.write(seleccionarIdioma(2) + "<br>");
document.write(seleccionarIdioma(3) + "<br>");
document.write(seleccionarIdioma(4) + "<br>");
```

En este ejemplo después de todas las opciones case encontramos una opción llamada default esto significa que si no se cumple ninguna de las anteriores es el valor que tendrá por defecto, en este caso “Inglés”, este será el resultado:



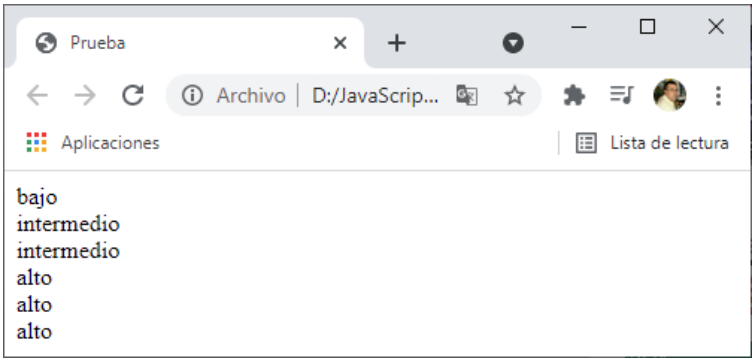
Sentencias Switch: Múltiples casos

D: > JavaScript > JS operadores.js > ...

```
function clasificarVolumen(valor) {
    var volumen;
    switch (valor) {
        case 1:
            volumen = "bajo";
            break;
        case 2:
        case 3:
            volumen = "intermedio";
            break;
        case 4:
        case 5:
        case 6:
            volumen = "alto";
            break;
    }
    return volumen;
}
```

```
19  document.write(clasificarVolumen(1) + "<br>");
20  document.write(clasificarVolumen(2) + "<br>");
21  document.write(clasificarVolumen(3) + "<br>");
22  document.write(clasificarVolumen(4) + "<br>");
23  document.write(clasificarVolumen(5) + "<br>");
24  document.write(clasificarVolumen(6) + "<br>");
```

Este será el resultado:



Reemplazar “if...else” por “switch”

D: > JavaScript > JS operadores.js > ...

```
1   function seleccionarIdioma(valor) {
2       var idioma;
3
4       if (valor == 1) {
5           idioma = "Español";
6       } else if(valor == 2) {
7           idioma = "Francés";
8       } else if(valor == 3) {
9           idioma = "Italiano";
10      } else {
11          idioma = "Inglés";
12      }
13      return idioma;
14  }
```

Vamos a reemplazar este condicional con la sentencia switch.

D: > JavaScript > JS operadores.js > ...

```
1   function seleccionarIdioma(valor) {
2       var idioma;
3
4       switch (valor) {
```

```
        case 1:
            idioma = "Español";
            break;
        case 2:
            idioma = "Francés";
            break;
        case 3:
            idioma = "Italiano";
            break;
        default:
            idioma = "Inglés";
            break;
    }
    return idioma;
}
```

Ya hemos modificado la sentencia 'if' por la sentencia 'switch'.

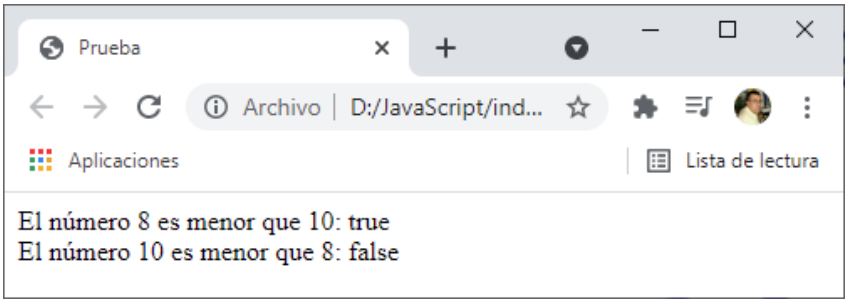
Retornar valores Booleanos

D: > JavaScript > JS operadores.js > ...

```
function esMenorQue(a, b) {
    if (a < b){
        return true;
    }else{
        return false;
    }
}

document.write("El número 8 es menor que 10: " + esMenorQue(8, 10) + "<br>");
document.write("El número 10 es menor que 8: " + esMenorQue(10, 8) + "<br>");
```

Este será el resultado:



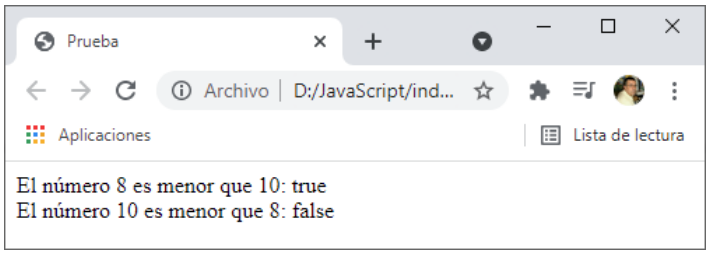
También se puede realizar de la siguiente forma:

D: > JavaScript > JS operadores.js > ...

```
function esMenorQue(a, b) {
    return a < b;
}
```

```
4
5   document.write("El número 8 es menor que 10: " + esMenorQue(8, 10) + "<br>");
6   document.write("El número 10 es menor que 8: " + esMenorQue(10, 8) + "<br>");
```

Este será el resultado:



Una comparación retorna un valor Booleano, true o false.

Patrón de retorno anticipado

D: > JavaScript > JS operadores.js > ...

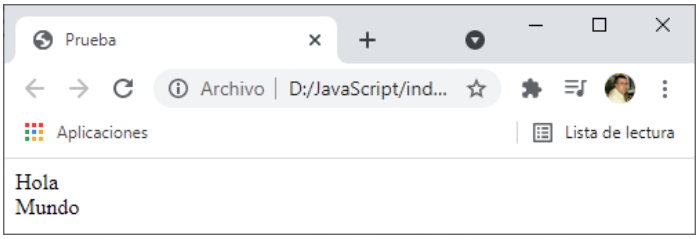
```
1   function miFuncion() {
2           document.write("Hola <br>");
3           return "Mundo";
4           document.write("Adiós");
5   }
```

Podrás observar que la última línea está más transparente, si nos colocamos encima de esta línea nos dice “Unreachable code detected” código que no se puede alcanzar durante la ejecución del programa.

Vamos a agregar una llamada a la función:

```
6   document.write(miFuncion());
```

Este será el resultado:



La última línea no llega a ejecutarse.

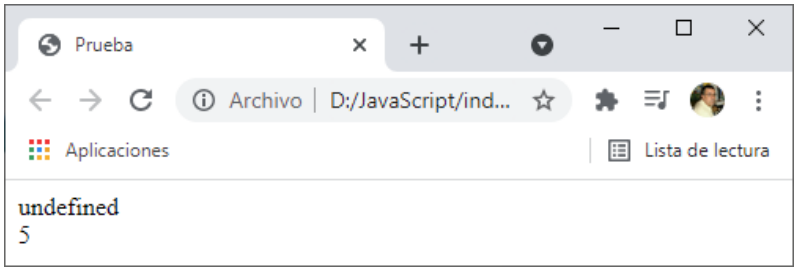
Vamos con otro ejemplo:

D: > JavaScript > JS operadores.js > ...

```
1   function calcularRaizCuadrada(num) {
2           if (num < 0) {
3               return undefined;
4           }
5           return Math.sqrt(num);
6   }
```

```
7   document.write(calcularRaizCuadrada(-5) + "<br>");
8   document.write(calcularRaizCuadrada(25));
```

Este será el resultado:

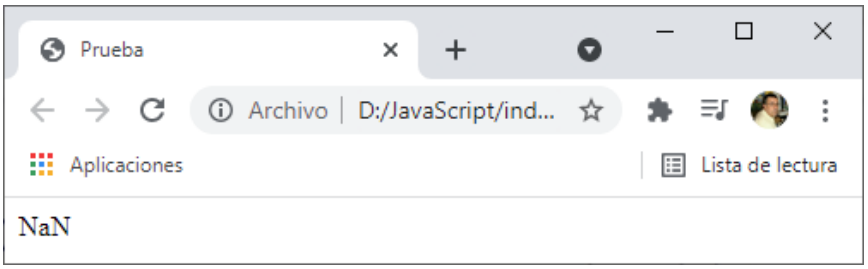


Si comentamos la condición que hace el retorno anticipado.

D: > JavaScript > JS operadores.js > ...

```
1   function calcularRaizCuadrada(num) {
2       /*     if (num < 0) {
3                 return undefined;
4           } */
5           return Math.sqrt(num);
6   }
7   document.write(calcularRaizCuadrada(-5));
```

Y lo ejecutamos con un valor negativo, este será el resultado:



Se muestra el siguiente error.

Conteo de cartas

En un juego de casino Blackjack el jugador puede sacarle ventaja a l casa llevando un registro de número relativo de cartas altas y bajas que quedan en la baraja.
Esto se llama “conteo de cartas”.
Tener más cartas altas en la baraja es una ventaja para el jugador. Se le asigna un valor a cada carta de acuerdo a la siguiente tabla.

- Cuando el conteo es positivo, el jugador debería apostar alto.
- Cuando el conteo en 0 o negativo, el jugador debería apostar bajo.

Cambio de conteo	Cartas
+1	2, 3, 4, 5, 6
0	7, 8, 9
-1	10, ‘J’, ‘Q’, ‘K’, ‘A’

Nuestro objetivo es definir una función para contar cartas.

La función debe tomar un parámetro carta que puede ser un número o una cadena de caracteres y luego aumentar o reducir el valor de las variable global conteo de acuerdo al valor de la carta (observa la tabla).

La función debe retornar una cadena de caracteres con el conteo actual y la cadena:

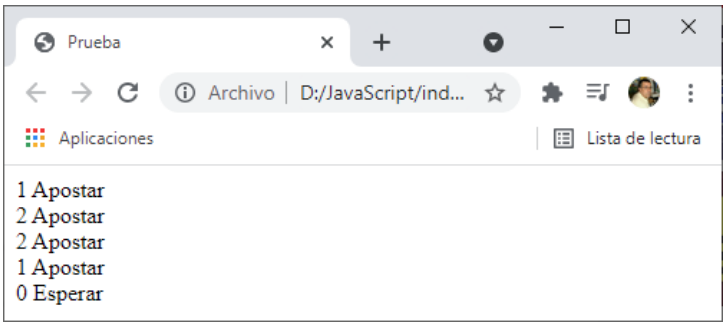
- “Apostar” si el conteo es positivo.
- “Esperar” si el conteo es cero o negativo.

El conteo actual y la decisión del jugador (“Apostar” o “Esperar” deben estar separados por un espacio.

D: > JavaScript > JS operadores.js > ...

```
1   var conteo = 0;
2   function contarCarta(carta){
3           var decision;
4           switch(carta) {
5                   case 2:
6                   case 3:
7                   case 4:
8                   case 5:
9                   case 6:
10                          conteo++;
11                          break;
12                  case 10:
13                  case "J":
14                  case "Q":
15                  case "K":
16                  case "A":
17                          conteo--;
18                          break;
19          }
20          if (conteo >0 ){
21                  decision = "Apostar";
22          }else {
23                  decision = "Esperar";
24
25
26          }
27         return conteo + " " + decision;
28  }
29  document.write(contarCarta(2) + "<br>");
30  document.write(contarCarta(3) + "<br>");
31  document.write(contarCarta(7) + "<br>");
32  document.write(contarCarta("K") + "<br>");
33  document.write(contarCarta("A") + "<br>");
```

Cuando ejecutemos este será el resultado:



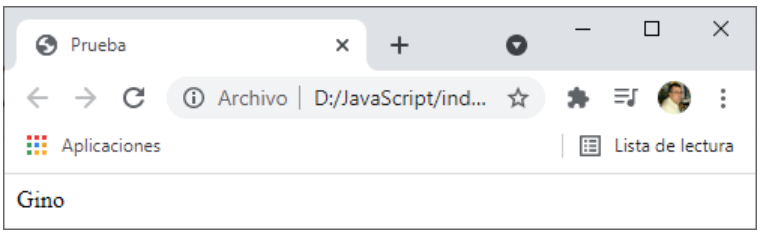
Crear Objetos

Los objetos nos permiten guardar un conjunto de propiedades que están relacionadas con sus correspondientes valores.

D: > JavaScript > JS operadores.js > ...

```
1   var miPerro = {
2       "nombre": "Gino",
3       "edad":  5,
4       "peso": 6,
5       "raza": "Beagle"
6   };
7   document.write(miPerro.nombre);
```

Este será el resultado:

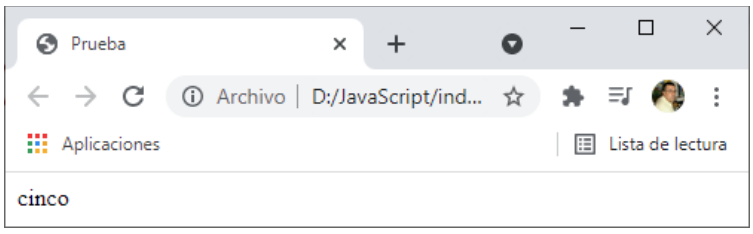


Otro ejemplo:

D: > JavaScript > JS operadores.js > ...

```
1   var miObjeto = {
2           5: "cinco"
3   };
4   document.write(miObjeto[5]);
```

Este será el resultado:

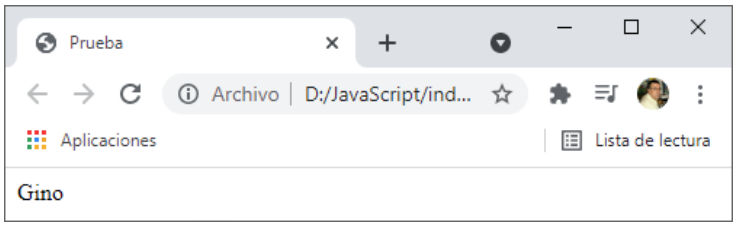


Acceder a Propiedades: Notación de Punto

D: > JavaScript > JS operadores.js > ...

```
1   var miPerro = {
2           nombre: "Gino",
3           edad: 5,
4           peso: 6,
5           raza: "Beagle"
6   };
7   document.write(miPerro.nombre);
```

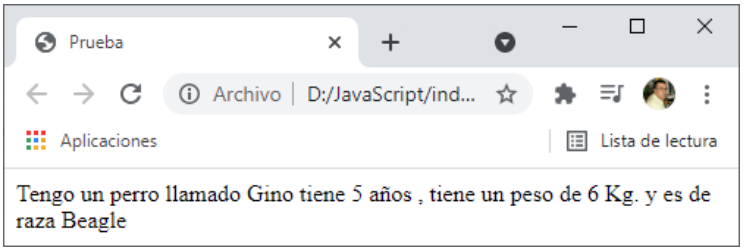
Este será el resultado:



D: > JavaScript > JS operadores.js > ...

```
1   var miPerro = {
2           nombre: "Gino",
3           edad: 5,
4           peso: 6,
5           raza: "Beagle"
6   };
7   document.write("Tengo un perro llamado " + miPerro.nombre + " tiene " + miPerro.edad + " años  , tiene
    un peso de " + miPerro.peso + " Kg. y es de raza " + miPerro.raza);
```

Este será el resultado:

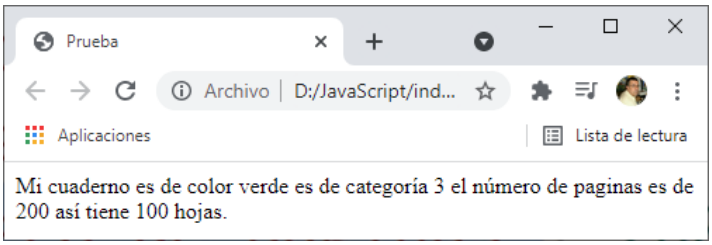


Acceder a Propiedades: Notación de corchetes

D: > JavaScript > JS operadores.js > ...

```
1   var miCuadereno = {
2           "color": "verde",
3           "categoria": 3,
4           "numero de paginas": 200,
5           "numero de hojas": 100
6   };
7
8   document.write("Mi cuaderno es de color " + miCuadereno.color + " es de categoría " + miCuadereno.
    categoria + " el número de paginas es de " + miCuadereno["numero de paginas"] + " así tiene " +
    miCuadereno["numero de hojas"] + " hojas.");
```

Este será el resultado:

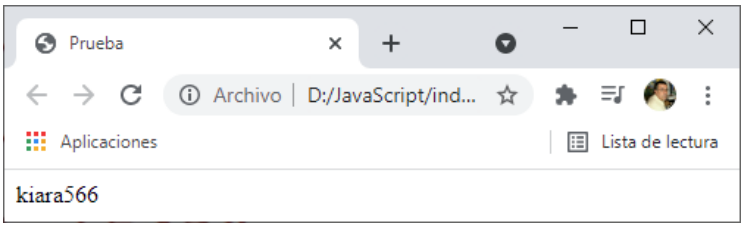


Acceder a propiedades: Variables

D: > JavaScript > JS operadores.js > ...

```
var resultados = {
    1: "nora256",
    2: "gino577",
    3: "estef543",
    4: "kiara566"
};
var posicion = 4;
document.write(resultados[posicion]);
```

Si queremos poner la variable en lugar de su posición esta tiene que ir entre corchetes, este será el resultado:



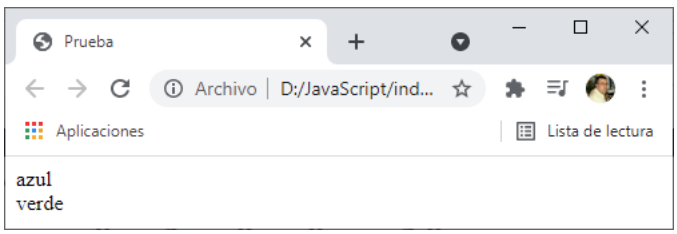
Actualizar propiedades

D: > JavaScript > JS operadores.js > ...

```
var mochila = {
    "color": "azul",
    "tamaño": "mediano",
    "contenido": ["botella de agua", "cuaderno"]
};

document.write(mochila.color + "<br>"); // azul
mochila.color = "verde";
document.write(mochila.color); // verde
```

Este será el resultado:

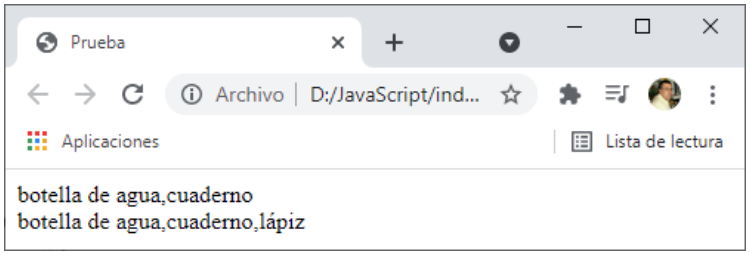


D: > JavaScript > JS operadores.js > ...

```
1  var mochila = {
2      "color": "azul",
3      "tamaño": "mediano",
4      "contenido": ["botella de agua", "cuaderno"]
5  };
6
7  document.write(mochila.contenido + "<br>");
8  mochila.contenido.push("lápiz");
9  document.write(mochila.contenido);
```

A la propiedad “contenido” queremos añadir un lápiz, lo haremos con el método push().

Este será el resultado antes y después de agregar el lápiz.

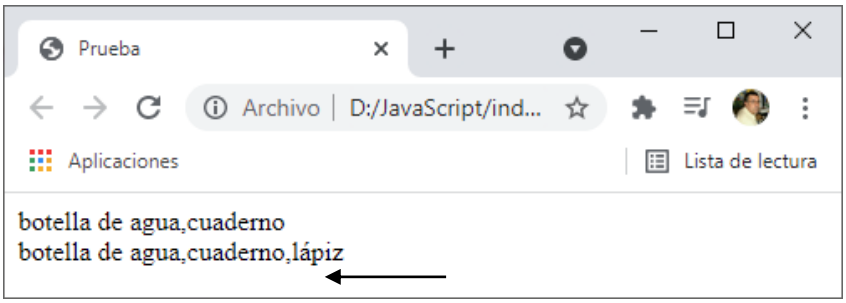


Ahora vamos vaciar la propiedad “contenido”.

D: > JavaScript > JS operadores.js > ...

```
1  var mochila = {
2      "color": "azul",
3      "tamaño": "mediano",
4      "contenido": ["botella de agua", "cuaderno"]
5  };
6
7  document.write(mochila.contenido + "<br>");
8  mochila.contenido.push("lápiz");
9  document.write(mochila.contenido + "<br>");
10 mochila.contenido = [];
11 document.write(mochila.contenido);
```

Este será el resultado:



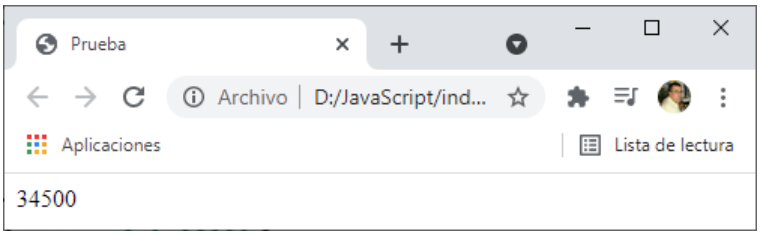
No muestra nada porque está vacía la propiedad “contenido”.

Agregar propiedades

D: > JavaScript > JS operadores.js > ...

```
1   var curso = {
2       "titulo": "Aprende JavaScript desde cero",
3       "idioma": "Español",
4       "duracion": 30
5   };
6
7   curso.vistas = 34500; // también puede ser -> curso["vistas"] = 34500;
8   document.write(curso.vistas);
```

Este será el resultado:

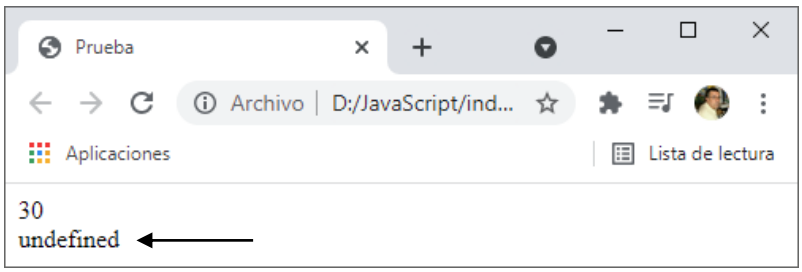


Eliminar propiedades

D: > JavaScript > JS operadores.js > ...

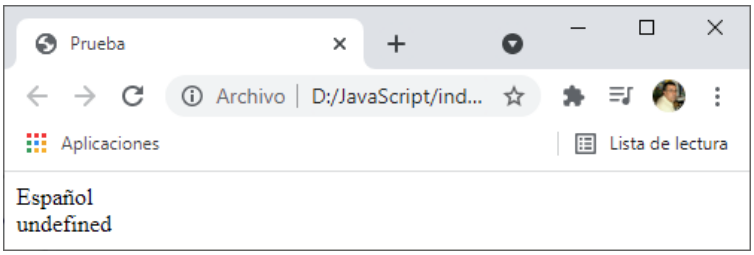
```
1   var curso = {
2       "titulo": "Aprende JavaScript desde cero",
3       "idioma": "Español",
4       "duracion": 30
5   };
6   document.write(curso.duracion);
7   delete curso.duracion;
8   document.write("<br>");
9   document.write(curso.duracion);
```

Este será el resultado:




```
6   document.write(curso.idioma);
7   delete curso.idioma;
8   document.write("<br>");
9   document.write(curso.idioma);
```

Vamos a modificar estas líneas, esto será el resultado.



Objetos para búsqueda

D: > JavaScript > JS operadores.js > ...

```
1   function buscarElementoQuimico(simbolo){
2       var elementoQuimico = "";
3       switch (simbolo){
4           case "Al":
5               elementoQuimico = "Aluminio";
6               break;
7           case "S":
8               elementoQuimico = "Azufre";
9               break;
10          case "Cl":
11              elementoQuimico = "Cloro";
12              break;
13          case "He":
14              elementoQuimico = "Helio";
15              break;
16          case "B":
17              elementoQuimico = "Boro";
18              break;
19          case "Li":
20              elementoQuimico = "Litio";
21              break;
22      }
23      return buscarElementoQuimico;
24  }
```

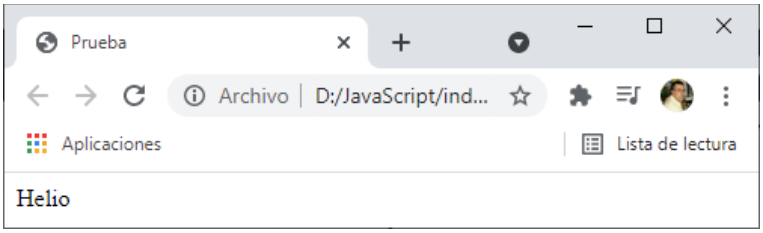
Este ejemplo es una función que introduciendo el símbolo del elemento químico te retorna el nombre del elemento químico utilizando switch, vamos a ver otro ejemplo que hace los mismo.

D: > JavaScript > JS operadores.js > ...

```
1   function buscarElementoQuimico(simbolo){
2       var simbolosQuimicos = {
3           "Al": "Aluminio",
4           "S": "Azufre",
5           "Cl": "Cloro",
6           "He": "Helio",
7           "B": "Boro",
```

```
        "li": "Litio"
    };
    return simbolosQuimicos[simbolo];
}
document.write(buscarElementoQuimico("He"));
```

Se crea un objeto con los símbolos y los elementos que luego se retorna como en los ejemplos anteriores con objetos, este será el resultado:



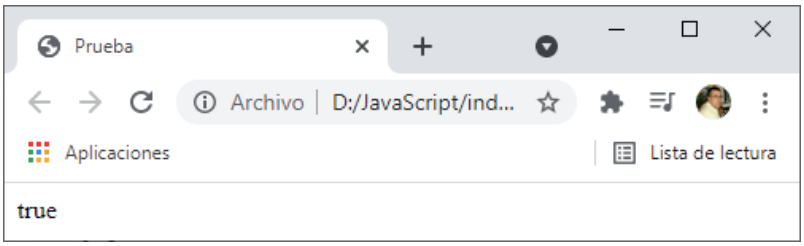
Verificar propiedades

D: > JavaScript > JS operadores.js > ...

```
var miCuaderno = {
    "color": "verde",
    "categoria": 3,
    "precio": 4.56
};

document.write(miCuaderno.hasOwnProperty("color"));
```

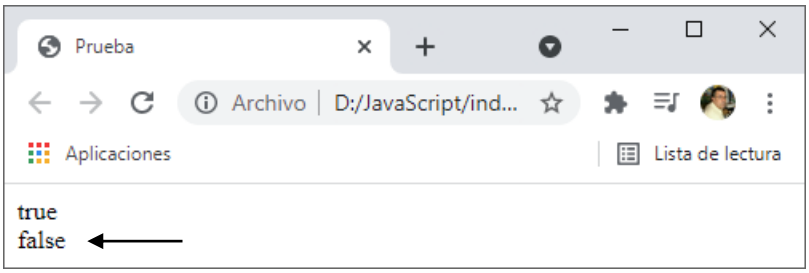
Con el método hasOwnProperty() podemos saber y un objeto tiene una determinada propiedad, en este ejemplo es el color, retorna un valor Booleano, este será el resultado.



Vamos a agregar la siguiente línea:

```
document.write("<br>");
document.write(miCuaderno.hasOwnProperty("origen"));
```

Este será el resultado:

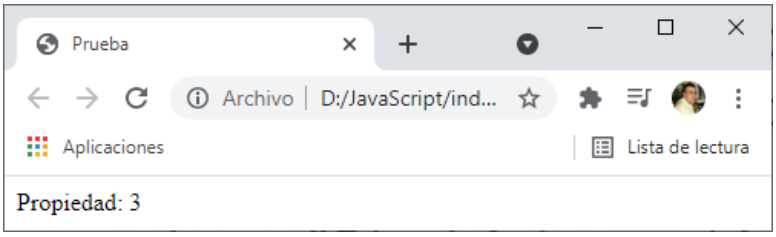


Este objeto no tiene la propiedad origen.

D: > JavaScript > JS operadores.js > ...

```
1  function verificarPropiedad(obj, propiedad){
2      if(obj.hasOwnProperty(propiedad)){
3          return "Propiedad: " + obj[propiedad];
4      } else {
5          return "El objeto no tiene esta propiedad";
6      }
7  }
8  var miCuaderno = {
9      "color": "verde",
10     "categoria": 3,
11     "precio": 4.56
12 };
13
14 document.write(verificarPropiedad(miCuaderno, "categoria"));
```

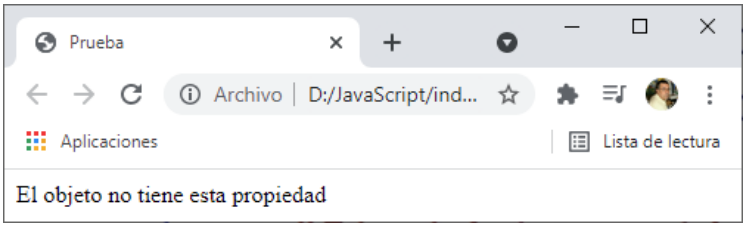
En este ejemplo hemos creado una función llamada verificarPropiedad que adminte dos parámetros obj y propiedad, sirve para verificar si una propiedad existe en un objeto, este será el resultado:



Vamos a modificar la línea 14.

```
14 document.write(verificarPropiedad(miCuaderno, "paginas"));
```

En este caso esta propiedad no la tiene, este será el resultado:



Objetos complejos

D: > JavaScript > JS operadores.js > ...

```
1  var ordenesDePizzas = [
2      {
3          "tipo": "margarita",
4          "tamaño": "Individual",
5          "precio": 5.67,
6          "toppings": [
7              "extra queso",
8              "champiñones",
9              "piña"
```

```
        ],
        "paraLlevar": true
    },
    {
        "tipo": "cuatro quesos",
        "tamaño": "familiar",
        "precio": 18.34,
        "toppings": [
            "extra queso",
            "pimentón"
        ],
        "paraLlevar": false
    }
];
```

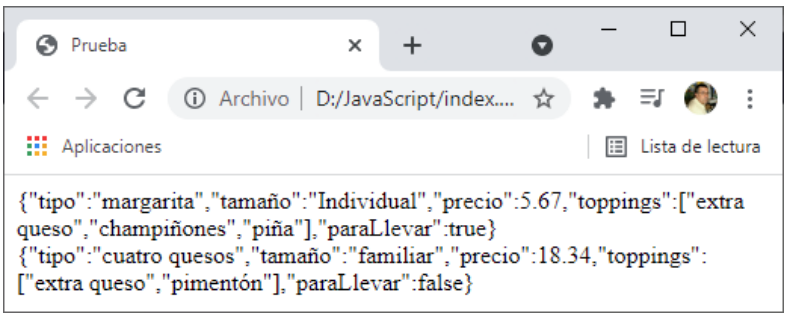
Lo vamos a tratar como un arreglo que tiene dos elementos con sus respectivas propiedades.

Vamos a añadir las siguiente líneas.

```
document.write(JSON.stringify(ordenesDePizzas[0]));
document.write("<br>");
document.write(JSON.stringify(ordenesDePizzas[1]));
```

Podremos ver los elementos de la array ‘ordenesDePizzas de los elementos 0 y 1.

Este será el resultado:



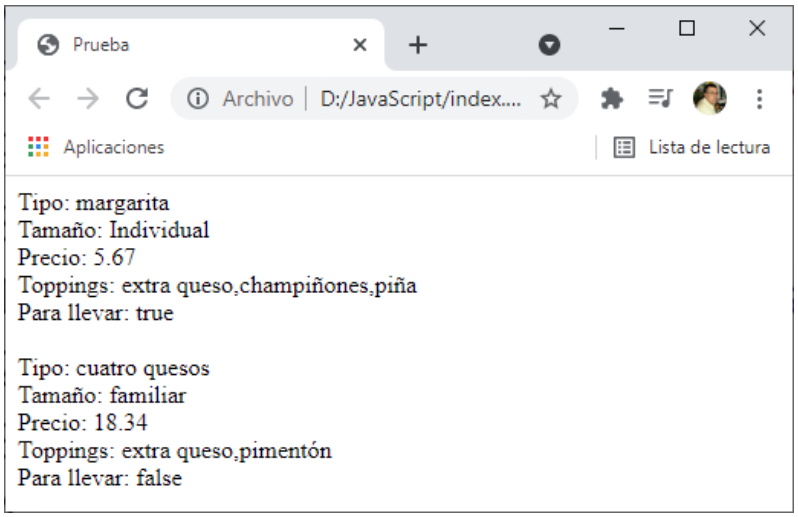
Vamos a acceder a toda las propiedades del elemento creando la siguiente función:

```
function consultarPedidos(num){
    pedido = "Tipo: " + ordenesDePizzas[num].tipo + "<br>";
    pedido = pedido + "Tamaño: " + ordenesDePizzas[num].tamaño + "<br>";
    pedido = pedido + "Precio: " + ordenesDePizzas[num].precio + "<br>";
    pedido = pedido + "Toppings: " + ordenesDePizzas[num].toppings + "<br>";
    pedido = pedido + "Para llevar: " + ordenesDePizzas[num].paraLlevar + "<br>";
    return pedido;
}
```

Que después las vamos a llamar:

```
document.write(consultarPedidos(0));
document.write("<br>");
document.write(consultarPedidos(1));
```

Este será el resultado:



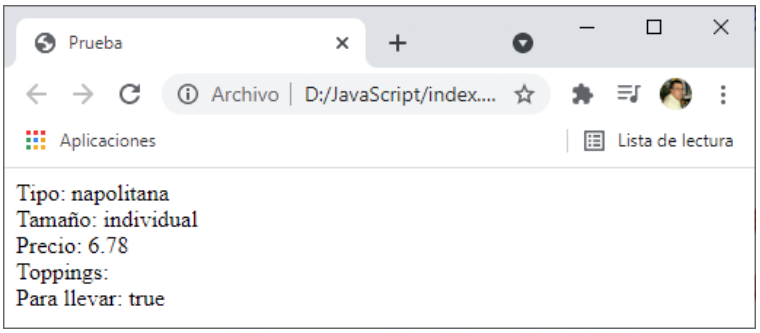
Ahora vamos a agregar un tercer elemento a la variable ordenesDePizza.

```
22      },
23      {
24          "tipo": "napolitana",
25          "tamaño": "individual",
26          "precio": 6.78,
27          "toppings": [],
28          "paraLlevar": true
29      }
```

Para consultar este elemento será el número 2.

```
42  document.write(consultarPedidos(2));
```

Este será el resultado:



Esta pizza no tiene Toppings (Extras).

Objetos anidados

D: > JavaScript > JS operadores.js > ...

```
1  var miReceta = {
2      "descripcion": "mi postre favorito",
3      "costo": 15.6,
4      "ingredientes": {
5          "masa": {
6              "harina": "100 grs",
```

```
            "sal": "1 cucharadita",
            "agua": "1 taza"
        },
        "cobertura": {
            "azucar": "120 grs",
            "chocolate": "4 cucharadas",
            "mantequilla": "200 grs"
        }
    }
};
```

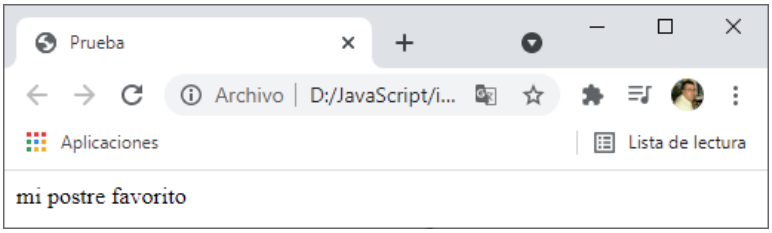
Objetos anidados son objetos que están metidos en otros objetos.

El objeto principal tiene 3 propiedades (descripción, costo e ingredientes), dentro de ingredientes hay 2 propiedades llamadas (masa y cobertura) con sus respectivas propiedades.

Vamos a ver cómo podemos acceder a un determinado elemento.

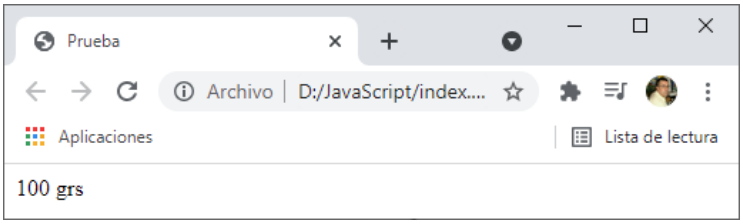
```
document.write(miReceta.descripcion);
```

Accedemos a la descripción de la receta:



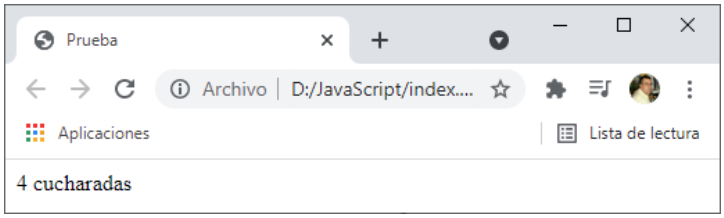

```
document.write(miReceta.ingredientes.masa.harina);
```

Queremos saber cuánta harina utilizamos.




```
document.write(miReceta.ingredientes.cobertura.chocolate);
```

Queremos saber cuánto chocolate se necesita.

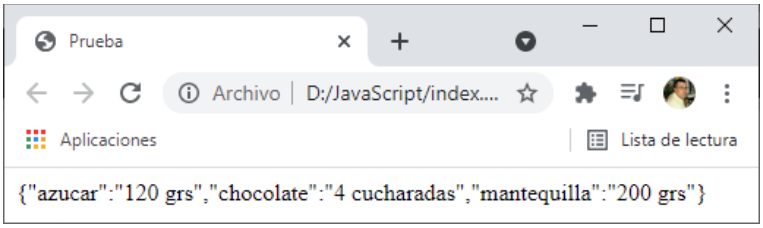



```
document.write(JSON.stringify(miReceta.ingredientes));
```

Queremos consultar por todos los ingredientes.

```
18    document.write(JSON.stringify(miReceta.ingredientes.cobertura));
```

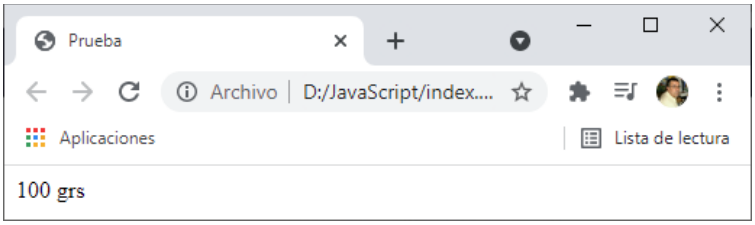
Ahora queremos consultar por la cobertura.



Recuerda que también podemos cambiar el punto por corchetes pero la propiedad tendrá que ir entre comillas.

```
18    document.write(miReceta["ingredientes"]["masa"]["harina"]);
```

Este será el resultado:



La ventaja de los corchetes es que podemos introducir una variable.

```
18    var ing = "ingredientes";
19    var ma = "masa";
20    var ha = "harina"
21    document.write(miReceta[ing][ma][ha]);
```

Obtendremos el mismo resultado.

Arreglos anidados

D: > JavaScript > JS operadores.js > ...

```
1    var misPlantas = [
2        {
3            tipo: "flores",
4            lista:[
5                "rosas",
6                "tulipanes",
7                "dientes de león"
8            ]
9        },
```

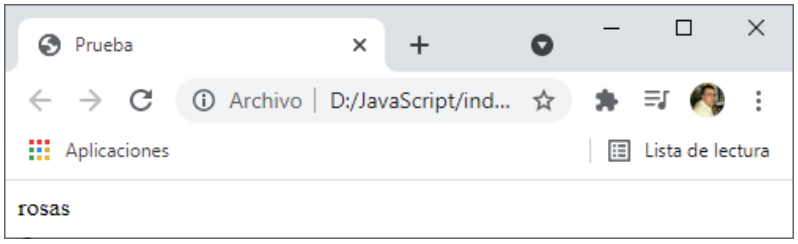
```
10      {
11          tipo: "árboles",
12          lista: [
13              "abeto",
14              "pino",
15              "abedul"
16          ]
17      }
18  ];
```

Los que están enmarcados son arreglos anidados porque están dentro de otro arreglo.

Si queremos acceder a la “rosas”.

```
19  var primeraFlor = misPlantas[0].lista[0]
20  document.write(primeraFlor);
```

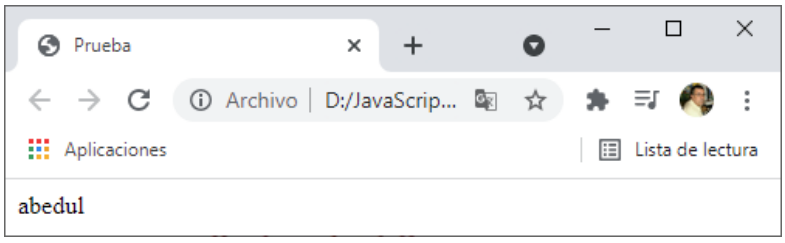
Este será el resultado:



Ahora vamos a buscar el árbol abedul.

```
19  var tercerArbol = misPlantas[1].lista[2]
20  document.write(tercerArbol);
```

Este será el resultado:



Colección de discos

Tenemos un objeto que representa parte de una colección de álbumes musicales.

Cada álbum tiene un número de identificación único (id) asociado a otras propiedades.

No todos los álbumes tienen la información completa.

```
1  var coleccionDeDiscos = {
2      7853: {
3          titulaDelAlbum: "bee Gees Greatest",
```

```
4           artista: "Bee Gees",
5           canciones: ["Stayin' Alive"]
6       },
7       5439:{
8           titulaDelAlbum: "ABBA Gold"
9       }
10  };
```

Define una función actulizarDiscos que tome los siguientes parámetros:

- discos (el objeto que representa la colección de discos).
- id.
- propiedad (“artista” o “canciones”).
- valor.

Tu objetivo es completar la función implementando las siguientes reglas para modificar el objeto pasado a la función:

- Si “valor” es una cadena vacía elimina la propiedad del álbum correspondiente.
- Si “propiedad” es igual a la cadena de caracteres “canciones” pero el álbum no tiene una propiedad llamada “canciones”, crea un arreglo vacío agrega “valor” a ese arreglo.
- Si “propiedad” es igual a la cadena de caracteres “canciones” y “valor” no es una cadena vacía, agrega “valor” al final del arreglo de canciones del álbum correspondiente.
- Sí “valor” no es una cadena vacía y “propiedad” no es igual a “canciones”, asigna el valor del parámetro “valor” a la propiedad. Si la propiedad existe. Debes crearla y asignarle este valor.

```
13  function actualizarDiscos(discos, id, propiedad, valor){
14      if (valor === "") {
15          delete discos[id][propiedad];
16      }else if(propiedad === "canciones"){
17          discos[id][propiedad] = discos[id][propiedad] || [];
18          discos[id][propiedad].push(valor);
19      }else{
20          discos[id][propiedad] = valor;
21      }
22  }
```

Línea 13 creamos una función llamada ‘actualizarDiscos(con cuatro parámetros, discos, id, propiedad y valor)’.

Línea 14 si valor es igual a cadena vacía entonces.

Línea 15 del array según el id y la propiedad (artista o canciones) la borramos.

Línea 16 Sino si propiedad es igual a “canciones” entonces.

Línea 17 disco[id][propiedad] es igual disco[id][propiedad] o un elemento nuevo [],

Línea 18 a disco[id][propiedad] al final se le agrega el valor.

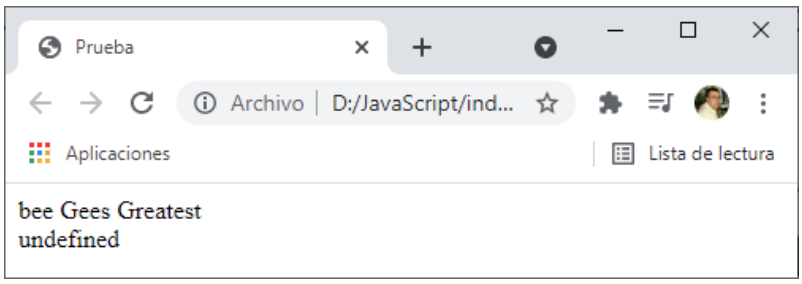
Línea 19 sino

Línea 20 a disco[id][propiedad] se le asigna el valor. Vamos a realizar varias situaciones:

Vamos a eliminar la canción del disco 7853, mostramos primero la canción, llamamos a la función que elimina la canción y por último observamos si la canción se ha borrado.

```
24    document.write(coleccionDeDiscos[7853].titulaDelAlbum);
25    actualizarDiscos(coleccionDeDiscos,7853, "titulaDelAlbum", "");
26    document.write("<br>");
27    document.write(coleccionDeDiscos[7853].titulaDelAlbum);
```

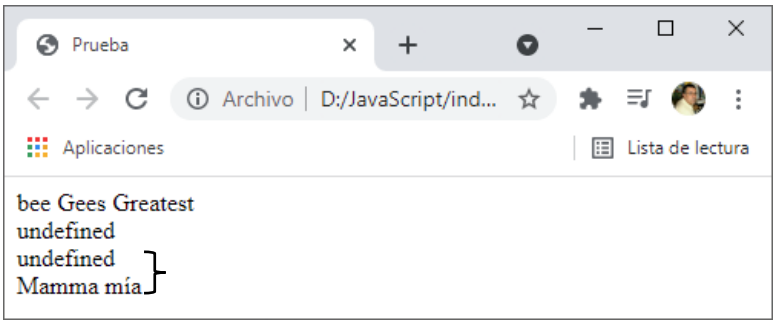
Este será el resultado:



Ahora queremos agregar de nuevo el título del album 5439.

```
29    document.write(coleccionDeDiscos[5439].canciones);
30    actualizarDiscos(coleccionDeDiscos,5439, "canciones", "Mamma mía");
31    document.write("<br>");
32    document.write(coleccionDeDiscos[5439].canciones);
```

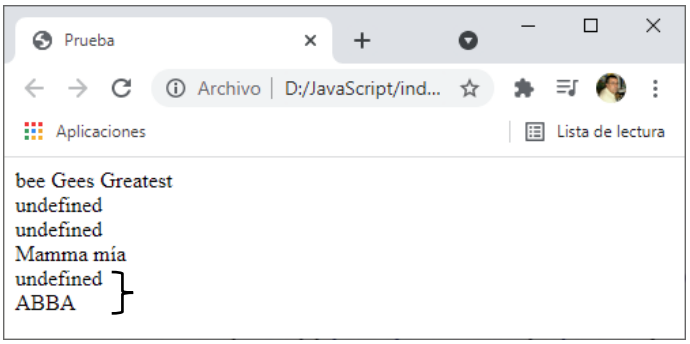
Este será el resultado:



Vamos a asignar el artista al álbum 5439.

```
33    document.write("<br>");
34    document.write(coleccionDeDiscos[5439].artista);
35    actualizarDiscos(coleccionDeDiscos,5439, "artista", "ABBA");
36    document.write("<br>");
37    document.write(coleccionDeDiscos[5439].artista);
```

Este será el resultado:



Ciclo “while”

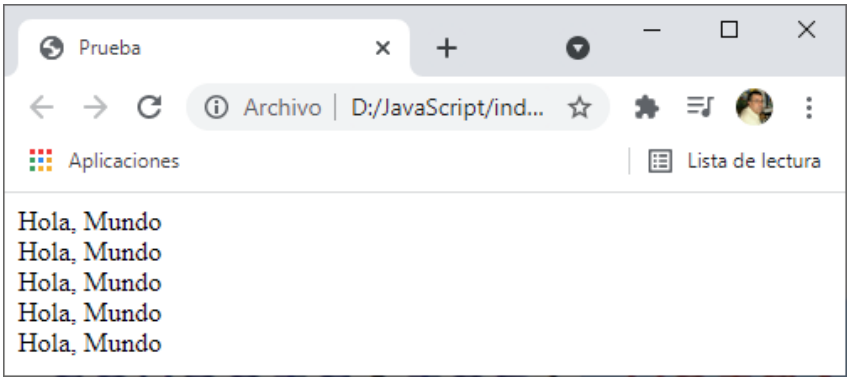
Ciclos o bucles nos periten repetir una secuencia de instrucciones un número especifico de veces.

Se utiliza cuando no sabemos el número de iteraciones que tiene que realizar.

Supón que queremos imprimir “Hola, Mundo” 5 veces, con lo que sabemos hasta hoy haríamos los siguiente:

```
1    document.write("Hola, Mundo <br>");
2    document.write("Hola, Mundo <br>");
3    document.write("Hola, Mundo <br>");
4    document.write("Hola, Mundo <br>");
5    document.write("Hola, Mundo <br>");
```

Este será el resultado:




```
1    var a = 0;
2    while(a<5){
3        document.write("Hola, Mundo <br>");
4        a++
5    }
```

Línea 1 definimos la variable A y le asignamos el valor 0.

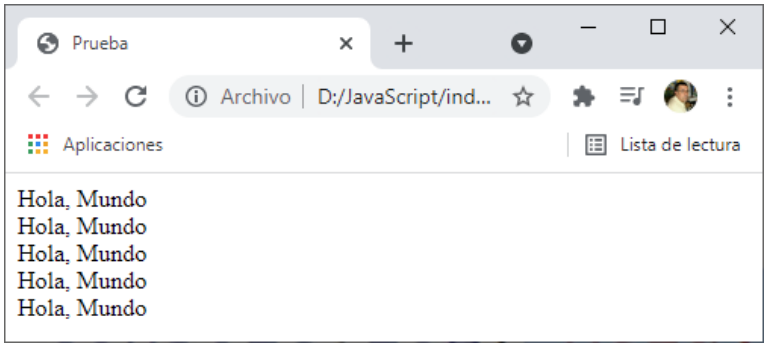
Línea 2 mientras la variable a sea menor de 5 (se repite el bucle).

Línea 3 muestra el texto “Hola, Mundo” más salto de línea.

Línea 4 a la variable a se le incrementa en 1.

Cuando la variable a tenga a tenga un valor de 5 se termina el bucle.

Este será el resultado:



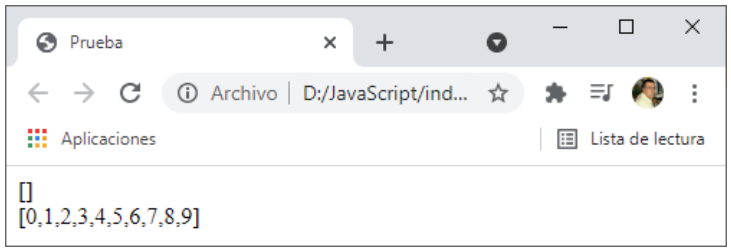
Otro ejemplo:

Queremos agregar en un array los valores desde el 0 hasta el 9.

D: > JavaScript > JS operadores.js > ...

```
1   var miArreglo = [];
2   var i = 0;
3   document.write(JSON.stringify(miArreglo));
4   while (i < 10){
5       miArreglo.push(i);
6       i++
7   }
8   document.write("<br>");
9   document.write(JSON.stringify(miArreglo));
```

Este será el resultado:



Otro ejemplo:

D: > JavaScript > JS operadores.js > ...

```
1   var numeros = [2, 3, 4, 5, 6, 8, 9, 34];
2   var x = 0;
3   while(x < numeros.length){
4       document.write(numeros[x] + "<br>");
5       x++
6   }
```

Queremos imprimir todos los elementos de la array uno por uno.

Línea 1 definimos una array llamada números con sus correspondientes valores.

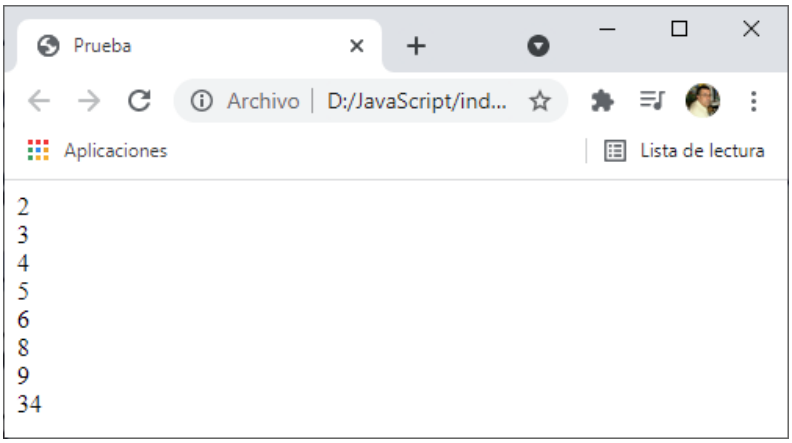
Línea 2 Asignamos la variable x con el valor 0.

Línea 3 mientras x tenga un valor menor al números de los elementos que contiene el array.

Línea 4 muestra dicho elemento y realiza un salto de línea.

Línea 5 la variable x se incrementa en 1.

Este será el resultado:



Otro ejemplo:

Queremos que se los valores del array se eliminen un determinado elementos empezando por la derecha en este caso va a eliminar 4 elementos.

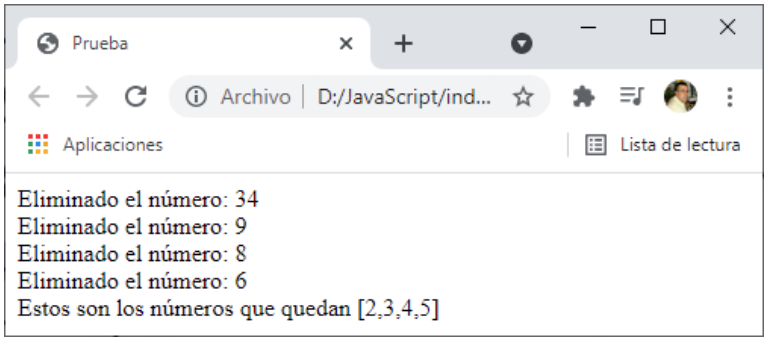
D: > JavaScript > JS operadores.js > ...

```
1   var numeros = [2, 3, 4, 5, 6, 8, 9, 34];
2
3   while(numeros.length > 4){
4       document.write("Eliminado el número: " +numeros.pop() + "<br>");
5   }
6   document.write("Estos son los números que quedan " + JSON.stringify(numeros));
```

Línea 3 Mientras la array tenga más de 4 elementos.

Línea 4 Muestra el mensaje (“Eliminando …. Con el valor eliminado más salto de línea.

Línea 6 Muestra los elementos que quedan, este será el resultado:



Ciclo “for”

Usamos el ciclo for cuando sabemos cuantas iteraciones tiene que hacerse.

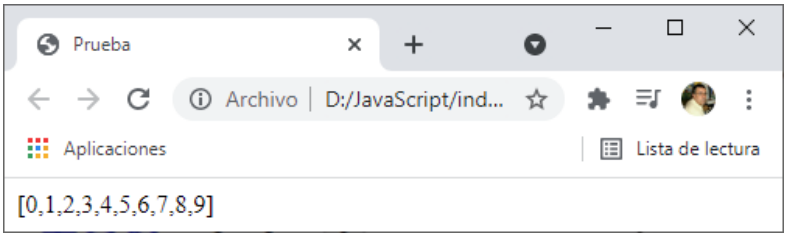
D: > JavaScript > JS operadores.js > ...

```
1   var miArreglo = [];
```

```
2    for(var i = 0; i < 10; i++){
3        miArreglo.push(i);
4    }
5    document.write(JSON.stringify(miArreglo));
```

El ciclo for tenemos que inicializar una variable al valor inicial hasta el valor final y con un incremento, todo ello separado por punto y coma ';'.

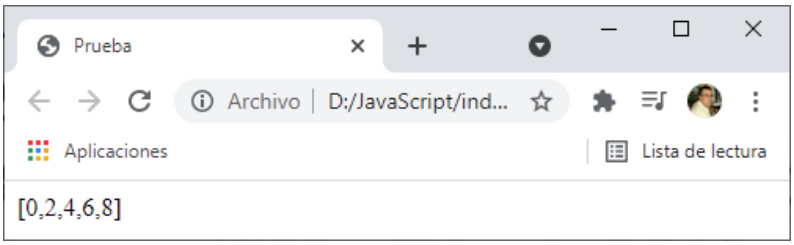
En este ejemplo queremos agregar los valores desde el 0 hasta el 9 en un array llamado miArreglo, este será el resultado:



Vamos a modificar el incremento.

```
2    for(var i = 0; i < 10; i+=2){
```

Vamos a cambiar el incremento, que en lugar de 1 sea 2, este será el resultado.



Ciclos "for": Números impares

Vamos a obtener los números impares entre 1 y 19.

D: > JavaScript > JS operadores.js > ...

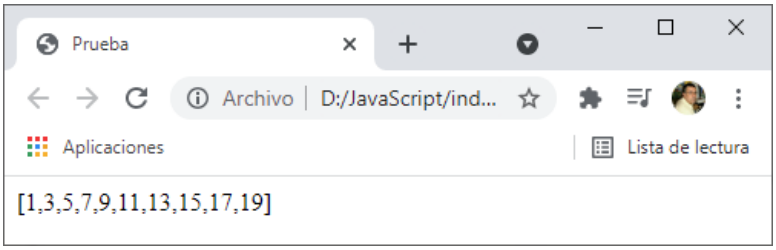
```
1    var miArreglo = [];
2    for(var i = 0; i <= 20; i++){
3        if(i % 2 != 0){
4            miArreglo.push(i)
5        }
6    }
7    document.write(JSON.stringify(miArreglo));
```

Hacemos un ciclo for desde 1 hasta 20 con un incremento en 1.

Si el valor de i dividido entre 2 con respecto al resto es distinto a 0, significa que el número es impar.

Lo añadimos al array miArreglo.

Una vez fuera del bucle mostramos el contenido del array, este será el resultado:



Otra forma de hacerlo evitando el condicionante.

D: > JavaScript > JS operadores.js > ...

```
1   var miArreglo = [];
2   for(var i = 1; i < 20; i+=2){
3       miArreglo.push(i)
4   }
5   document.write(JSON.stringify(miArreglo));
```

El resultado será el mismo.

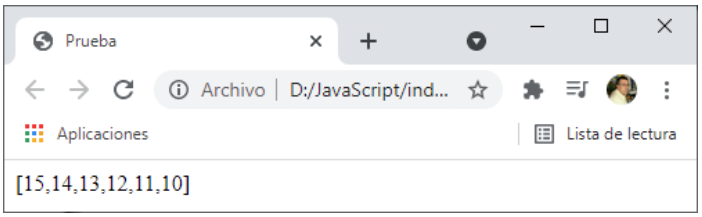
Te propongo que realices un ejercicio donde tienen que agregar en una array los números pares desde el 2 hasta el número 26.

Ciclos “for”: Contar hacia atrás

D: > JavaScript > JS operadores.js > ...

```
1   var miArreglo = [];
2   for(var i = 15; i >= 10; i--){
3       miArreglo.push(i)
4   }
5   document.write(JSON.stringify(miArreglo));
```

En este ejemplo empezamos con valor de i mayor al principio y llegar a un valor menor, para que eso ocurra realizaremos un decremento de 1, ira restando 1. Este será el resultado:

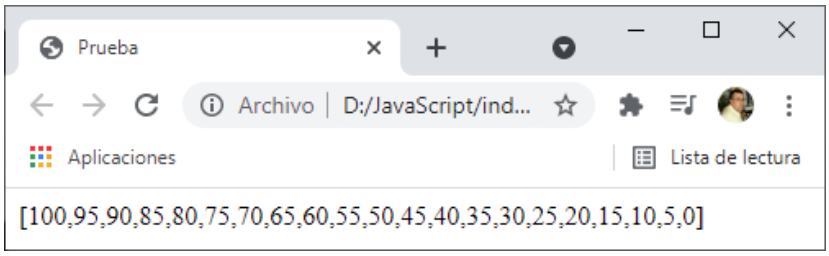


Vamos a iniciar a 100 hasta 0 con un decremento de 5.

D: > JavaScript > JS operadores.js > ...

```
1   var miArreglo = [];
2   for(var i = 100; i >= 0; i-=5){
3       miArreglo.push(i)
4   }
5   document.write(JSON.stringify(miArreglo));
```

Este será el resultado:

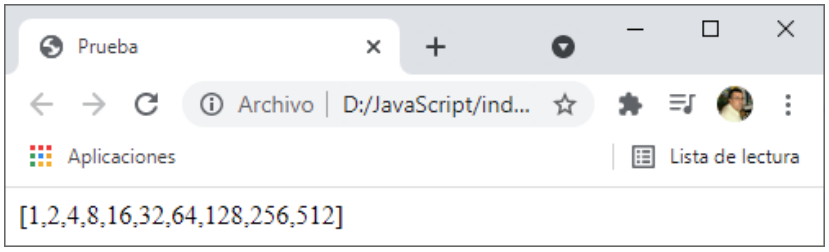


También podemos multiplicar entre 2, durante en ciclo donde i será igual a 1 hasta menor igual a 1000.

D: > JavaScript > JS operadores.js > ...

```
1   var miArreglo = [];
2   for(var i = 1; i <= 1000; i *= 2){
3       miArreglo.push(i)
4   }
5   document.write(JSON.stringify(miArreglo));
```

Este será el resultado:



Iterar sobre un arreglo con un ciclo “for”

D: > JavaScript > JS operadores.js > ...

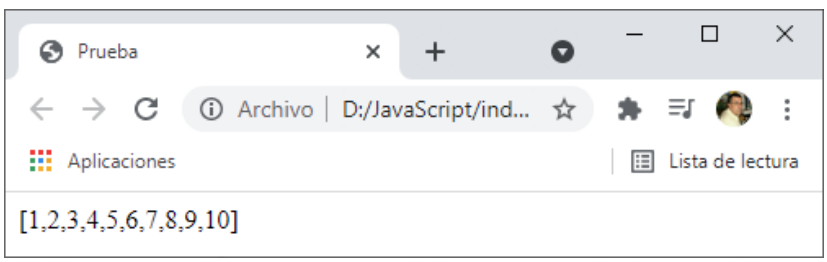
```
1   var miArreglo1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
2   var miArreglo2 = [];
3   for(var i = 0; i < miArreglo1.length; i++){
4       miArreglo2.push(miArreglo1[i]);
5   }
6   document.write(JSON.stringify(miArreglo2));
```

Creamos dos array una con valores y otra que está vacía.

Hacemos un ciclo for que recorrerá desde 0 hasta el número de elementos de miArreglo1.

En el ciclo iremos agregando cada uno de los elementos de miArreglo1 al array miArreglo2.

Una vez salido del bucle, mostramos el contenido de miArreglo2, este será el resultado:



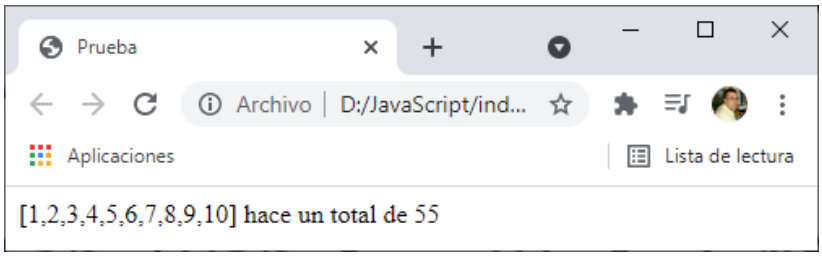
Otro ejemplo:

Queremos calcular la suma de todos los valores de un arreglo.

D: > JavaScript > JS operadores.js > ...

```
1   var miArreglo1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
2   var miArreglo2 = [];
3   var total = 0;
4   for(var i = 0; i < miArreglo1.length; i++){
5       total  += miArreglo1[i];
6       miArreglo2.push(miArreglo1[i]);
7   }
8   document.write(JSON.stringify(miArreglo2));
9   document.write("  hace un total de " + total);
```

Este será el resultado:



Para ver con más detalla las iteraciones:

D: > JavaScript > JS operadores.js > ...

```
1   var miArreglo1 = [3, 7, 12, 9, 25, 32, 17, 6, 14, 33];
2   var miArreglo2 = [];
3   var total = 0;
4   for(var i = 0; i < miArreglo1.length; i++){
5       document.write("Iteración: "+ i + ": ");
6       document.write("valor del array : " + miArreglo1[i] + "<br>");
7       total  += miArreglo1[i];
8       miArreglo2.push(miArreglo1[i]);
9   }
10  document.write(JSON.stringify(miArreglo2));
11  document.write("  hace un total de " + total);
```

Este será el resultado:

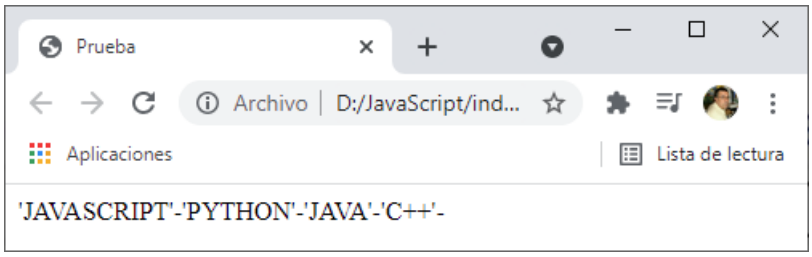
Otro ejemplo:

Queremos mostrar los elementos en mayúsculas.

D: > JavaScript > JS operadores.js > ...

```
1   var lenguajes = ["JavaScript","Python", "Java", "C++"];
2
3   for (var i = 0;lenguajes.length; i++ ){
4       document.write("\'" + lenguajes[i].toUpperCase() + "\'-");
5   }
```

Este será el resultado:



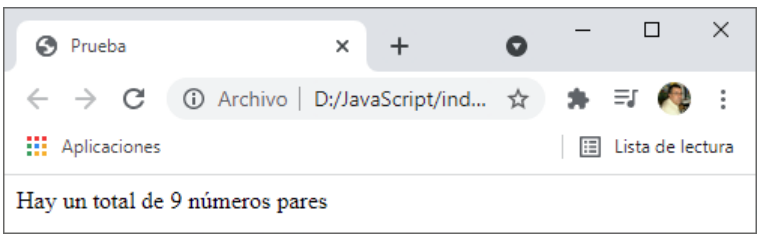
Otro ejemplo:

Vamos a crear una función que va a contar lo números pares de una array.

D: > JavaScript > JS operadores.js > ...

```
1   function contarNumerosPares(arreglo){
2       var total = 0;
3       for(var i = 0; i < arreglo.length; i++){
4           if(arreglo[i] % 2 == 0){
5               total++;
6           }
7       }
8       return total;
9   }
10  misNumeros = [22, 17, 14, 31, 46, 85, 38, 42, 66, 44, 11, 8, 1, 4];
11
12  document.write("Hay un total de " + contarNumerosPares(misNumeros) + " números pares");
```

Línea 1 Creamos una función contarNumerosPares y como parámetros irá un array.
Línea 2 la variable total la declaramos y la inicializamos a 0.
Línea 3 hacemos un ciclo for que recorrerá por todos los elementos del array.
Línea 4 Si arreglo[i] % 2 (el resto de una división entre 2) es igual a 0, significa que el número es par.
Línea 5 la variable total se incrementa en 1.
Línea 8 retorna el valor de total
Línea 10 declaramos e inicializamos un array.
Línea 12 mostramos el total de números pares.
Este será el resultado:



Ciclos “for” anidados

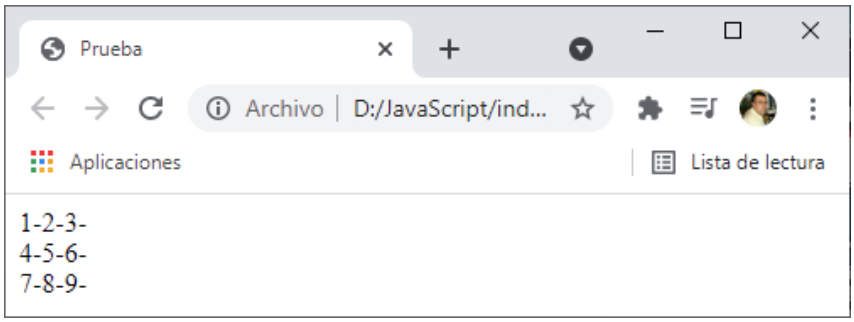
D: > JavaScript > JS operadores.js > ...

```
1   var miArreglo = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];
2   for (var i = 0; i < miArreglo.length; i++){
3       var arregloAnidado = miArreglo[i];
4       for (var j = 0; j < arregloAnidado.length; j++){
5           document.write( arregloAnidado[j] + "-");
6       }
7       document.write("<br>");
8   }
```

Línea 1 Hacemos un array de dos dimensiones.

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

Línea 2 Hacemos un ciclo for desde i igual a 0 hasta la longitud el número de elementos de la array, en este caso son tres elementos y cada uno de ellos son arrays, con incremento de 1.
Línea 3 La variable arregloAnidado en cada bucle asume los siguientes valores:
[1, 2, 3] - [4, 5, 6] - [7, 8, 9] Estos grupos de array también son de 3 elementos.
Línea 4 Hacemos un for anidado desde j igual a 0 hasta el número de elementos de la array arregloAnidado con incremento de 1.
Línea 5 muestra los elementos del arregloAnidado.
Línea 7 fuera del bucle j realizamos un salto de línea.
Este será el resultado:



Ciclos “do…while”

Este ciclo do while se diferencia del ciclo while que este ciclo se ejecuta como mínimo una vez, en cambio el ciclo while si ya no cumple la condición en el primer momento este ya no se ejecuta.

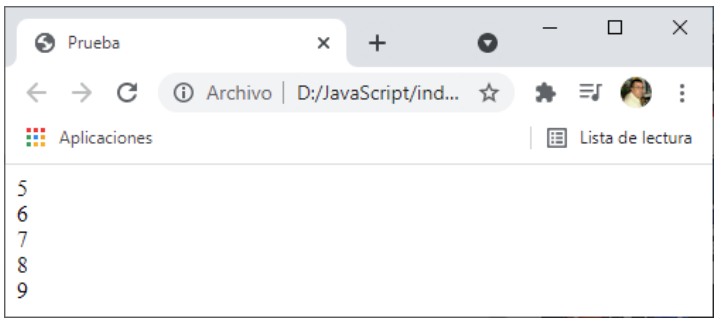
Vamos a realizar una comparación entre los dos ciclos.

Ciclo while:

D: > JavaScript > JS operadores.js > ...

```
1   var x;
2   x = 5;
3   while (x < 10) {
4       document.write(x + "<br>");
5       x++;
6   }
```

Este será el resultado:

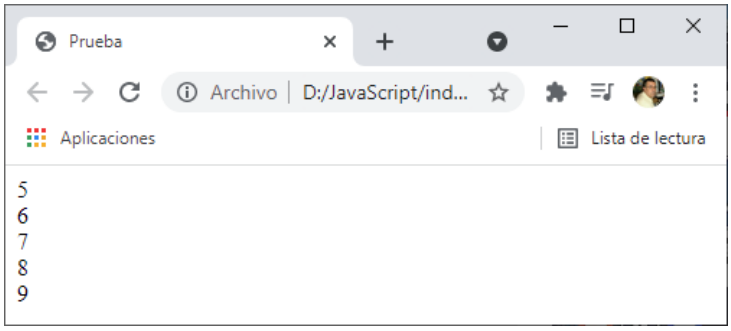


Ciclo do while

D: > JavaScript > JS operadores.js > ...

```
1   var x;
2   x = 5;
3   do {
4       document.write(x + "<br>");
5       x++;
6   } while (x < 10);
```

Este será el resultado:



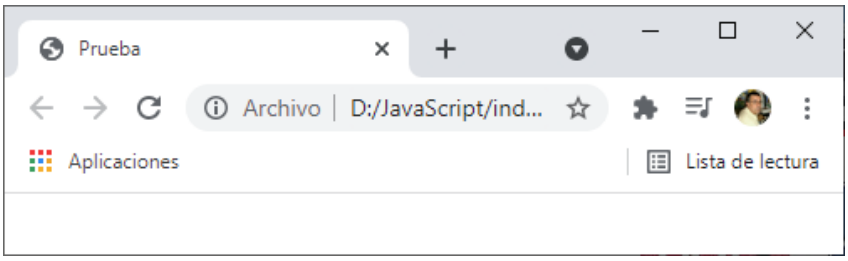
Podrás observar que el resultado a sido el mismo ya que los dos ciclos se han ejecutado como mínimo una vez.

Otro ejemplo while

```
1   var x;
2   x = 16;
3   while (x < 10){
```

```
4       document.write(x + "<br>");
5       x++;
6   }
```

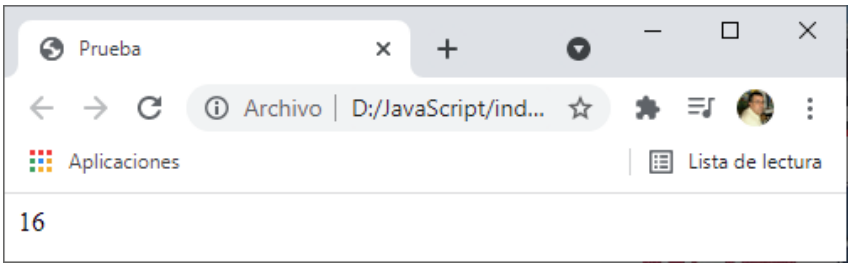
En este ejemplo como la condición ya no se cumple desde el principio el ciclo no se ejecuta ninguna vez, este será el resultado:



Ahora con do while.

```
1   var x;
2   x = 16;
3   do {
4       document.write(x + "<br>");
5       x++;
6   } while (x < 10);
```

En este caso empieza a ejecutarse una vez y luego compara si tiene que continuar con el bucle, este será el resultado:



Búsqueda de perfil

```
1   var contactos = [
2       {
3           "nombre": "Nora",
4           "apellido": "Nav",
5           "numero": "0543236543",
6           "gustos": ["Pizza", "Programación"]
7       },
8       {
9           "nombre": "Harry",
```

```
        "apellido": "Potter",
        "numero": "0994372684",
        "gustos": ["Hogwarts", "Magia"]
    },
    {
        "nombre": "Sherlock",
        "apellido": "Holmes",
        "numero": "0487345643",
        "gustos": ["Casos interesantes", "Violín"]
    }
]
```

Tenemos esta array anidada que contiene las propiedades de tres personas, nombre, apellido, número y gustos.

Gustos es otra array que contiene cada elemento con los gustos que tiene cada persona.

Tenemos que crear una función con los parámetros nombre y propiedades, en esta función introduciremos el nombre de la persona y como propiedades tenemos que escribir una de estas opciones (apellido, numero o gustos).

Nos tiene que retornar de dicha persona la propiedad que se le solicita.

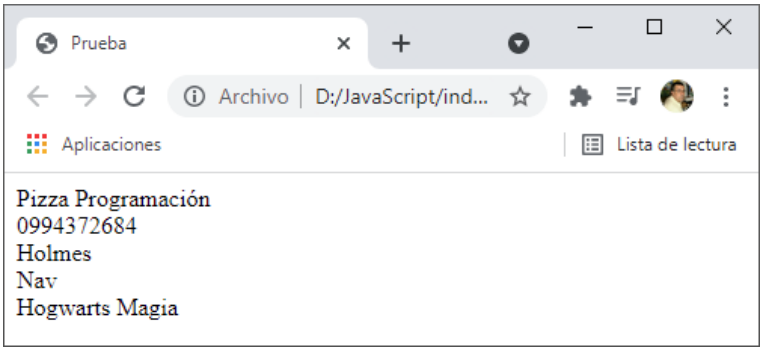
```
function buscar(nom, propiedad){
    for(var i = 0; i < contactos.length; i++ ){
        if(contactos[i].nombre == nom){
            var posicion = i;
        }
    }
    if (propiedad == "apellido" ){
        return contactos[posicion].apellido;
    }
    if (propiedad == "numero" ){
        return contactos[posicion].numero;
    }
    if (propiedad == "gustos"){
        var misGustos = ""
        var arrayGustos = contactos[posicion].gustos;
        for (var j = 0; j < arrayGustos.length; j++){
            misGustos = misGustos + (arrayGustos[j] + " ")
        }
    return misGustos;
    }
}
```

Línea 21 Creamos una función llamada buscar con los parámetros nom y propiedad.
Línea 22 Creamos un bucle para recorrer todos los elementos de la array contactos.
Línea 23 Si contactos[i].nombre es igual a nom (valor que se le pasa por parámetro).
Línea 24 La variable posición se le asigna el valor de i (valor índice de la array).
Línea 27 Si propiedad es igual a apellido.
Línea 28 retorna contactos[posicion].apellidos. Es el apellido de la persona que luego llamaras con el nombre en el argumento de la función.
Línea 30 Si propiedad es igual a número.
Línea 31 retorna contactos[posición].numero. Es el numero de identificación de la persona que luego llamarás con el nombre en el argumento de la función.
Línea 33 Si propiedad es igual a gustos.
Línea 34 definimos una variable llamada misGustos sin asignarle ningún valor.
Línea 35 la variable arrayGustos se le asigna los valores de la array contacto[posición].gustos.
Línea 36 con un bucle for para recorrer todos los elementos de arrayGustos.
Línea 37 con la variable misGustos acumulamos todos los elementos de la array arrayGustos.
Línea 39 retorna los valores de la variable misGustos.

El siguiente paso será llamar a la función con sus respectivos argumentos.

```
43    document.write(buscar("Nora", "gustos" )+ "<br>");
44    document.write(buscar("Harry", "numero")+ "<br>");
45    document.write(buscar("Sherlock", "apellido")+ "<br>");
46    document.write(buscar("Nora", "apellido")+ "<br>");
47    document.write(buscar("Harry", "gustos"));
```

Este será el resultado:



Después de las explicaciones en el tutorial, el proyecto es mucho más reducido y mejor planteado por la profesora:

```
21  function buscar(nom, propiedad){
22      for(var i = 0; i < contactos.length; i++ ){
23          if(contactos[i].nombre === nom){
24              return contactos[i][propiedad] || "La propiedad no existe.";
25          }
26      }
27      return "El contacto no está en la lista.";
28  }
```

Línea 21 creamos la función buscar con los parámetros nom y propiedad.

Línea 22 un ciclo for que recorre por todos los elementos del array.

Línea 23 si contactos[i].nombre es igual nom (nombre que pasamos por argumentos en la función)

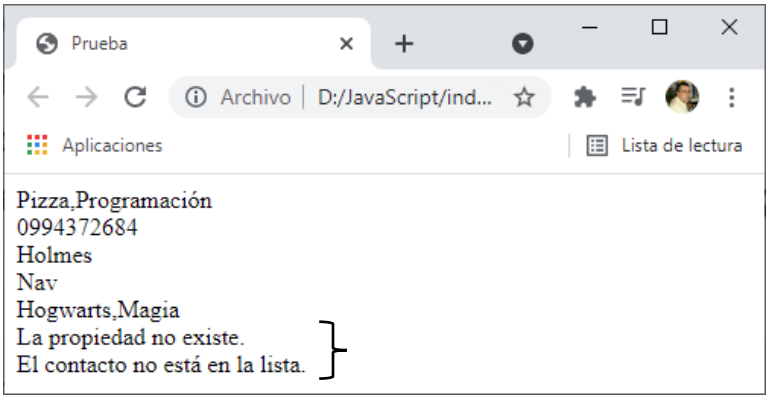
Línea 24 retorna contactos[i][propiedad (pasado por argumentos) si esta parte genera error porque no ha encontrado este nombre en la lista o tampoco la propiedad || o muestre el mensaje “La propiedad no existe.”.

Línea 27 si ha realizado todo el bucle y no ha encontrado nada fuera del bucle retornaremos el siguiente mensaje “El contacto no está en la lista.”.

Ahora para verificar este programa hemos agregados dos llamadas de función.

```
35    document.write(buscar("Nora", "telefono")+ "<br>");
36    document.write(buscar("Carlos", "gustos"));
```

En la línea 35 la propiedad teléfono no existe y en la línea 26 en la lista no existe ningún nombre con Carlos, este será el resultado:

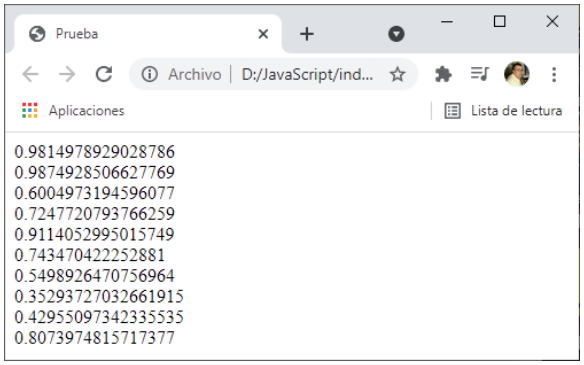


Números aleatorios

D: > JavaScript > JS operadores.js > ...

```
1    function generarFraccionAleatoria(){
2        return Math.random(); // Valor aleatoiro entre 0 y 1.
3    }
4    for(var i = 0; i < 10; i++){
5        document.write(generarFraccionAleatoria() + "<br>");
6    }
```

Con la función Math tenemos el método random(), los valores aleatorios estarán desde 0 hasta 1 sin llegar a tener el valor 1. En este ejemplo hacemos que se muestren 10 valores aleatorios, este será el resultado:



Podemos asignar un valor aleatorio a una variable:

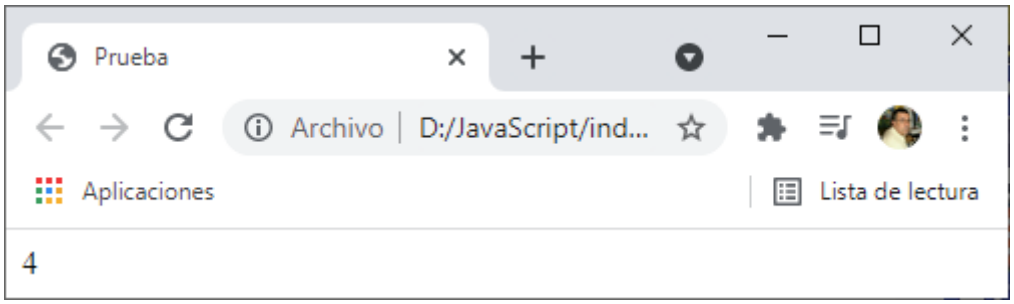
var numAleatorio = Math.random();

Números enteros aleatorios

D: > JavaScript > JS operadores.js > ...

```
1   // floor trunca la parte decimal
2   var numEntre0y19 = Math.floor(Math.random() * 20);
3   document.write(numEntre0y19);
```

Este será el resultado:

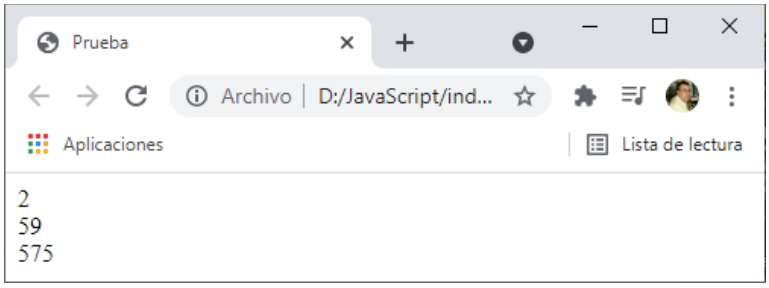


Cada vez que ejecutemos el valor se actualizará automáticamente.

D: > JavaScript > JS operadores.js > ...

```
1   //Generar un entero aleatorio entre 0 y el límite
2   //superior (sin incluirlo).
3   function generarEnteroAleatorio(limiteSuperior) {
4       return Math.floor(Math.random() * limiteSuperior);
5   }
6
7   document.write(generarEnteroAleatorio(5) + "<br>");
8   document.write(generarEnteroAleatorio(100) + "<br>");
9   document.write(generarEnteroAleatorio(1000) + "<br>");
```

Este será el resultado:



Números enteros aleatorios en un rango

D: > JavaScript > JS operadores.js > ...

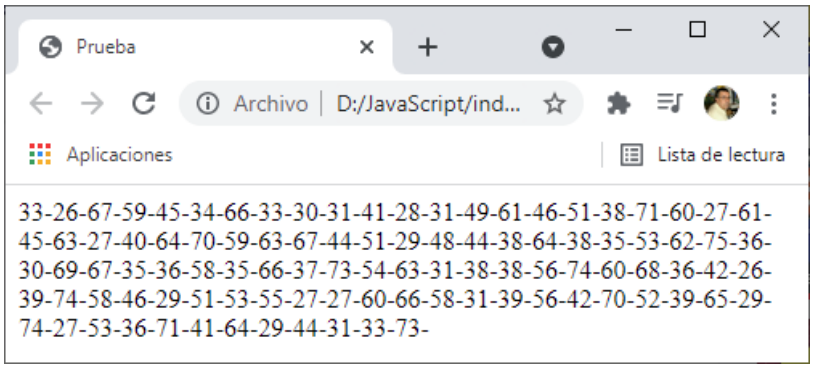
```
1   function rangoAleatorio(limiteInferior, limiteSuperior){
2       return Math.floor(Math.random() * (limiteSuperior-limiteInferior + 1) + limiteInferior);
3   }
```

```
4
5   for (var i = 0; i < 100; i++){
6       document.write(rangoAleatorio(25, 75) + "-");
7   }
```

La formula para controlar tanto el limite inferior como el límite superior es:

(limitieSuperior – limiteInferior +1) + limiteInferior.

Este será el resultado:



Función parseInt()

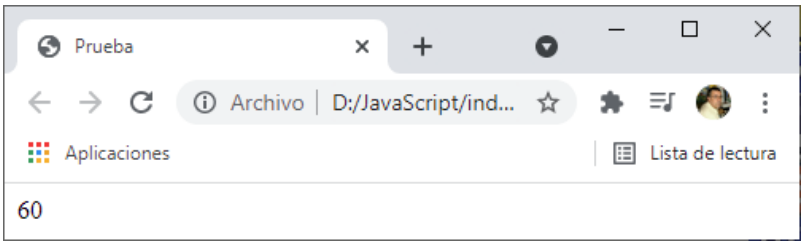
Este elemento “5” no es igual a 5, el primero está como cadena de caracteres en cambio el segundo si es un número.

¿Cómo podemos convertir una cadena de caracteres que representan un número a un numero y así poder operar con él.

D: > JavaScript > JS operadores.js

```
1   valorCadena = "5";
2   valorNumero = (parseInt(valorCadena));
3   document.write(valorNumero * 12);
```

Este será el resultado:

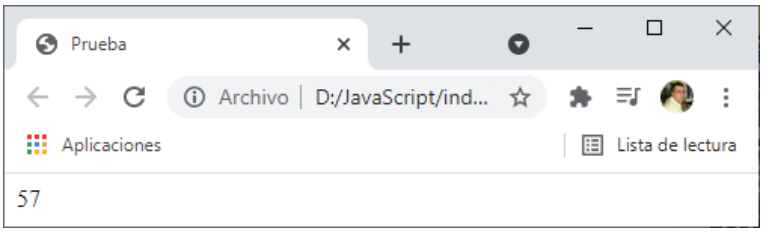


Si pasamos el siguiente valor:
ValorFinal = parseInt(“6.7);
El resultado será 6 ya que la parte decimal se elimina no redondea.
Vamos a ver el siguiente ejemplo:

D: > JavaScript > JS operadores.js > ...

```
1   var a = "5";
2   var b = "7";
3   document.write(a + b);
```

Este será el resultado:



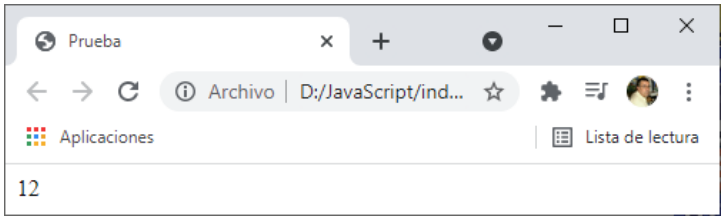
Muestra el valor 57 porque ha concatenado las cadenas y no por su valor numérico.

Vamos a resolverlo.

D: > JavaScript > JS operadores.js > ...

```
1   var a = parseInt("5");
2   var b = parseInt("7");
3   document.write(a + b);
```

Hemos pasado su valor numérico de tipo entero, este será el resultado:



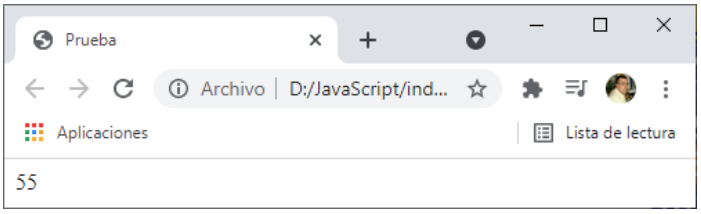
Función parseInt() con una base

D: > JavaScript > JS operadores.js

```
1   document.write(parseInt("110111", 2));
```

En este ejemplo tenemos un valor binario entre comillas y para decirle que es binario adjuntamos después de la coma el número 2.

Este será el resultado:



$$(1 * 2^5) + (1 * 2^4) + (0 * 2^3) + (1 * 2^2) + (1 * 2^1) + (1 * 2^0)$$

$$32 + 16 + 0 + 4 + 2 + 1 = 55$$

```
1   document.write(parseInt(110111, 2));
```

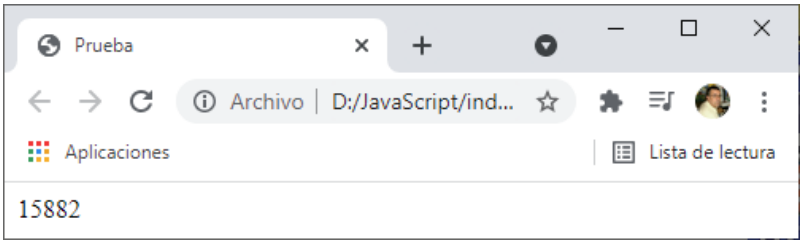
También lo podemos pasar como número binario, el resultado será el mismo.

Ahora como sistema hexadecimal.

D: > JavaScript > JS operadores.js

```
1   document.write(parseInt("3E0A", 16));
```

Este será el resultado:



En este caso no se pueden elimina las comillas del número hexadecimal ya que tiene combinación de números y letras.

Operador condicional (Ternario)

Nos permite compactar un condicional en una sola línea.

D: > JavaScript > JS operadores.js > ...

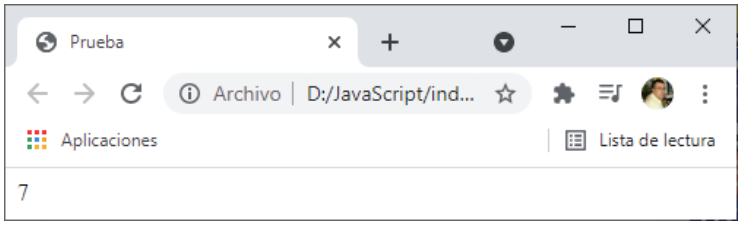
```
1   function retornarMinimo(x , y){
2       if(x < y) {
3           return x;
4       } else {
5        return y;
6       }
7   }
```

Se compacta en una sola línea

Este es un ejemplo de como ahora lo haríamos, esta función que admite dos parámetros retorna el valor mínimo de ambos.

```
9   document.write(retornarMinimo(7, 12));
```

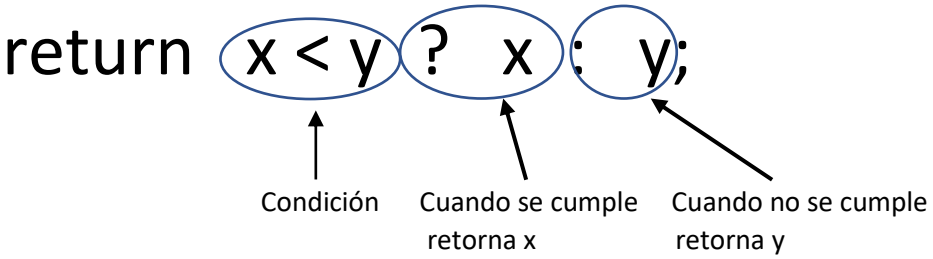
Llamamos a la función con dos argumentos el 7 y el 12, este será el resultado:



D: > JavaScript > JS operadores.js > ...

```
1   function retornarMinimo(x , y){
2       return x < y ? x : y;
3   }
4
5   document.write(retornarMinimo(7, 12));
```

El resultado será el mismo.

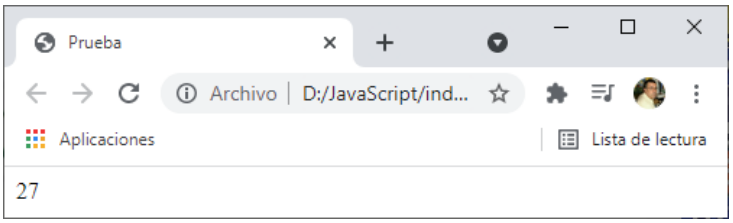


Otro ejemplo:

D: > JavaScript > JS operadores.js > ...

```
1   var a = 5;
2   var b = 9;
3   document.write(a > b ? a + 2 : b * 3);
```

Si a es mayor que b se retorna el valor de la variable a + 2 y sino se retorna el valor de la variable b multiplicado por 3, este será el resultado:

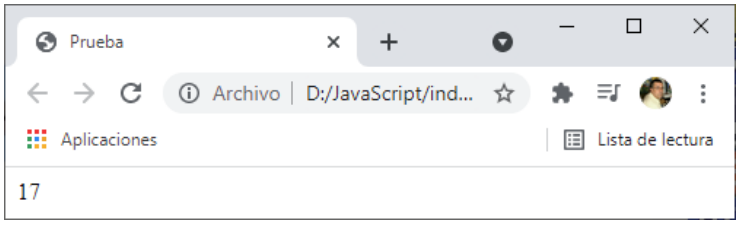


Vamos a cambiar el valor de a.

D: > JavaScript > JS operadores.js > ...

```
1   var a = 15;
2   var b = 9;
3   document.write(a > b ? a + 2 : b * 3);
```

Entonces este será el resultado.



Múltiples operadores condicionales

D: > JavaScript > JS operadores.js > ...

```
1   function compararNumeros(a, b) {
2       if (a == b) {
3           return "a y b son iguales";
4       } else if (a > b){
5           return "a es mayor que b";
```

```
    } else {
        return "b es mayor que a";
    }
}
```

En este ejemplo comparamos dos valores y nos tiene que decir si son iguales o que número es mayor.

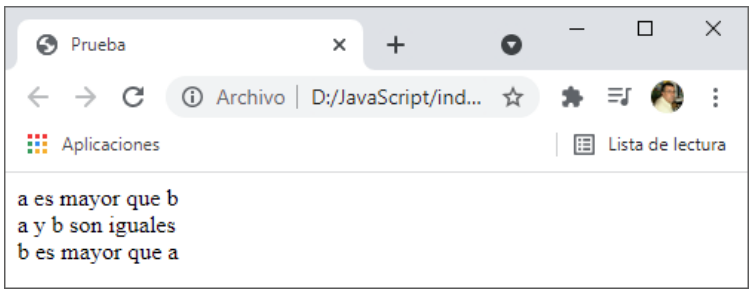
Ahora utilizaremos el múltiples operadores condicionales.

D: > JavaScript > JS operadores.js > ...

```
function compararNumeros(a, b) {
    return a == b ? "a y b son iguales "
        : a > b ? "a es mayor que b"
        : "b es mayor que a";
}

document.write(compararNumeros(7, 3) + "<br>");
document.write(compararNumeros(7, 7) + "<br>");
document.write(compararNumeros(3, 7) );
```

Lo que está enmarcado puede estar en una sola línea y lo que está diciendo es si a es mayor que b que salga el mensaje “a es mayor que b” al poner : a > b ? es como un else if comapara si a es mayor que b entonces muestra el mensaje “a es mayor que b” luego los “:” significa else muestra el mensaje “b es mayor que a”, este será el resultado después de llamar la función tres veces con distintos argumentos.



Estas condiciones se pueden alinear para su mejor comprensión ya que si no quedaría de la siguiente forma que también es correcto.

D: > JavaScript > JS operadores.js > ...

```
function compararNumeros(a, b) {
    return a == b ? "a y b son iguales " : a > b ? "a es mayor que b" : "b es mayor que a";
}
document.write(compararNumeros(7, 3) + "<br>");
document.write(compararNumeros(7, 7) + "<br>");
document.write(compararNumeros(3, 7) );
```

El resultado será el mismo.

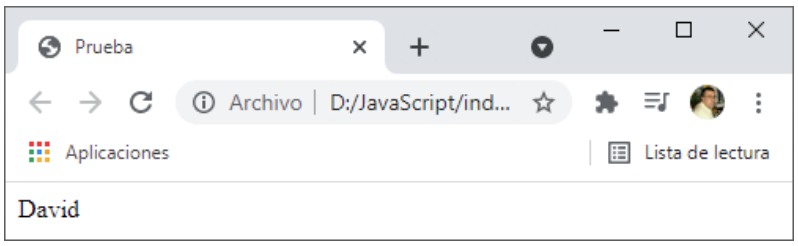
var vs. let

En este apartado vamos a ver las diferencias en declarar una variable con var y con let.

D: > JavaScript > JS operadores.js > ...

```
var campista = "James";
var campista = "David";
document.write(campista);
```

Este será el resultado:



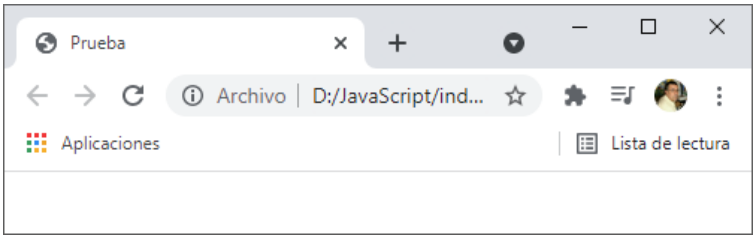
Podemos ir actualizando el valor de la variable campista.

D: > JavaScript > JS operadores.js > ...

```
let campista = "James";
let campista = "David";
document.write(campista);
```

Con let solo se puede declarar una vez y no puede ser modificada.

Si ejecutamos nos dará error (el identificador campista ya ha sido declarado) y veremos este resultado:

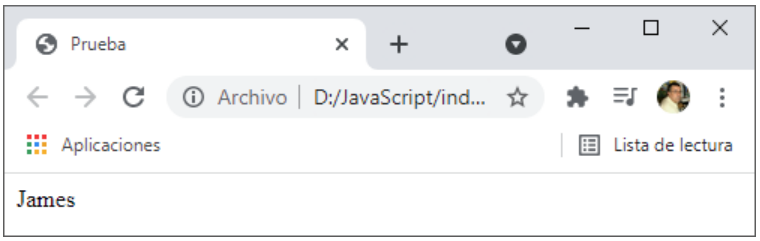


Vamos a comentar la segunda línea.

D: > JavaScript > JS operadores.js > ...

```
let campista = "James";
// let campista = "David";
document.write(campista);
```

Ya que es la línea quería cambiar el valor de la variable campista, este será el resultado:

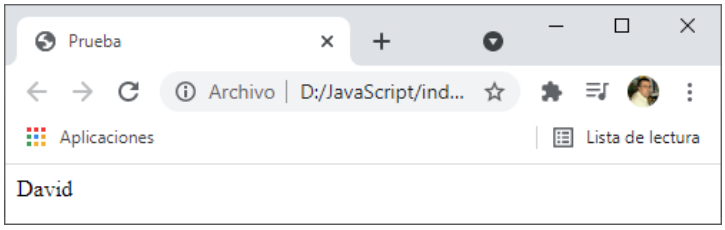


Si en el siguiente ejemplo en la segunda línea eliminamos la palabra let, esto es lo que pasará:

D: > JavaScript > JS operadores.js > ...

```
1   let campista = "James";
2   campista = "David";
3   document.write(campista);
```

Este será el resultado:



Que podremos modificar el valor de la variable.

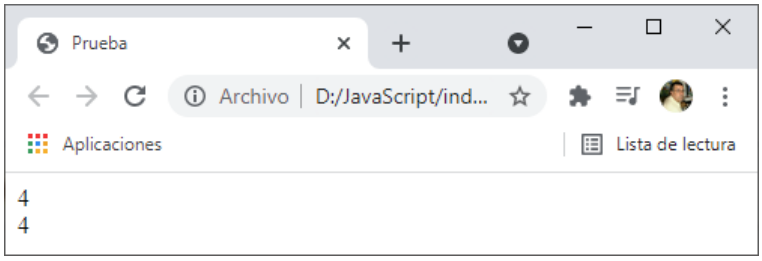
Ámbito de var vs. let

Cuando creamos una variable con var, estamos creando una variable global si esta está definida en el programa principal o una variable local si la hemos creando en el ámbito de una función.

D: > JavaScript > JS operadores.js > ...

```
1   var miVariableGlobal = 4;
2   document.write(miVariableGlobal + "<br>");
3
4   function miFuncion(){
5       document.write(miVariableGlobal);
6   }
7
8   miFuncion();
```

Hemos declarado una variable global llamada miVariableGlobal y le hemos asignado el valor de 4.
Desde el programa principal la mostramos.
Creamos una función llamada miFuncion() para mostrar la misma variable.
Llamamos a la función, y este será el resultado:



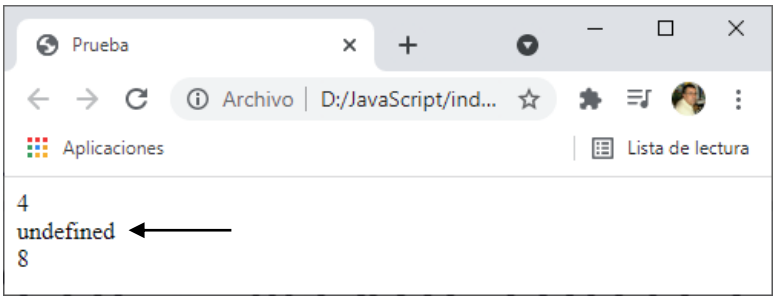
Tenemos acceso del valor desde el programa principal y desde la función por eso es una variable global.

D: > JavaScript > JS operadores.js > ...

```
1   var miVariableGlobal = 4;
2   document.write(miVariableGlobal + "<br>");
3
```

```
4   function miFuncion(){
5       document.write(miVariableGlobal + "<br>");
6       var miVariableGlobal = 8;
7       document.write(miVariableGlobal)
8   }
9
10  miFuncion();
```

En el momento que definimos una variable local con el mismo nombre de la variable global que hemos creado en el programa principal , la variable local tiene prioridad sobre la variable global cuando esta está creada en una función, este será el resultado:

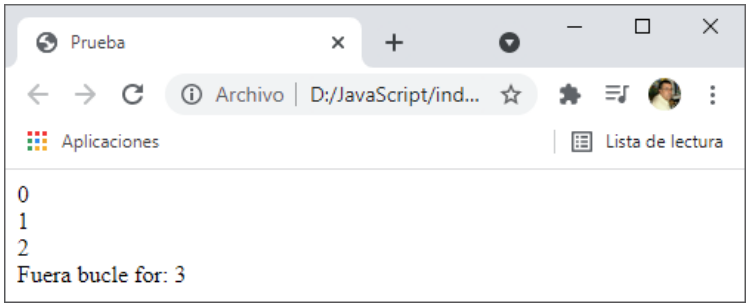


Cuando estamos en la función ya no la reconoce, porque con el mismo nombre hemos declarado otra variable en el ámbito local.

D: > JavaScript > JS operadores.js > ...

```
1   for (var i = 0; i < 3; i++) {
2       document.write(i + "<br>");
3   }
4   document.write("Fuera bucle for: " + i);
```

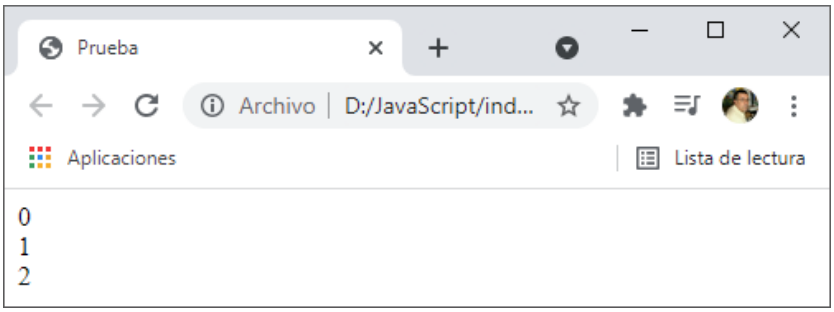
La variable 'i' fuera del ciclo for sigue manteniendo su valor, este será el resultado.



D: > JavaScript > JS operadores.js > ...

```
1   for (let i = 0; i < 3; i++) {
2       document.write(i + "<br>");
3   }
4   document.write("Fuera bucle for: " + i);
```

Si la definimos con let ya no tendremos acceso del valor fuera del bucle for, este será el resultado:



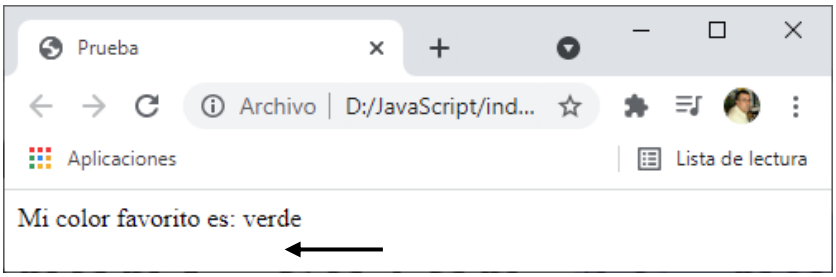
Y finalmente tendremos este menaje de error:

ReferenceError: i is not defined (i no está definida).

D: > JavaScript > JS operadores.js > ...

```
1   var mostrarColor = true;
2   if (mostrarColor) {
3       let color = "verde";
4       document.write("Mi color favorito es: " + color + "<br>");
5   }
6
7   document.write("Mi color favorito es después del condicional: " + color);
```

Si defines una variable con let dentro de un condicional, cuando queramos acceder a ella fuera del condición, no tendremos acceso a ella, este será el resultado:



Si la cambiamos por var verás que podrás acceder a su valor fuera del condicional if.

Resumiendo si la definimos con let es acceso de su valor será solo en bloque mucho más restringido.

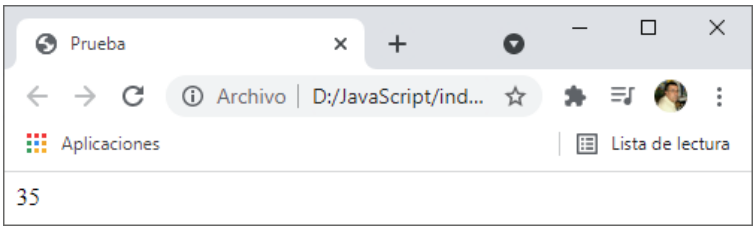
const

D: > JavaScript > JS operadores.js > ...

```
1   const miConstante = 35;
2   document.write(miConstante + "<br>");
3   miConstante = 15;
4   document.write(miConstante);
```

Al crear una variable de constante a un valor, esta lo podemos mostrar, pero en el momento que queremos cambiar su valor aparece el siguiente mensaje de error:

TypeError: “miConstante” is read-only (miConstante es solo de lectura, este será el resultado:



La muestra la primera vez, pero como se genera un error ya no se muestra en la segunda vez.

Si declaramos una variable de tipo const y no inicializamos su valor observaremos el siguiente mensaje de error.

SyntaxError: unKnown: 'Const dclarations' require an initialization valure. Las variables de tipo const se les tiene que asignar un valor, esto no es correcto.

D: > JavaScript > JS operadores.js > ...

```
1    const miConstante ;
2    miConstante = 35;
3    document.write(miConstante);
```

Como se genera un error no muestra el resultado de la línea 3.

Es aconsejable que las variables de tipo const escribirlas en mayúscula de este modo cuando estemos en otra parte del código sabremos que tipo de variable es:

D: > JavaScript > JS operadores.js > ...

```
1    const MI_CONSTANTE = 35;
2    document.write(MI_CONSTANTE);
```

Vamos a ver un ejemplo:

D: > JavaScript > JS operadores.js > ...

```
1    function calcularAreaCirculo(radio){
2        const PI = 3.14;
3        if (radio < 0) {
4            return undefined;
5        }
6        return PI * (radio ** 2);
7    }
```

Si radio menor a 0, es decir un numero negativo que retorne el valor undefined, si no es así que retorne el valor de PI multiplicado por el valor del radio que pasaremos como argumento cuando llamemos a la función elevado a 2.

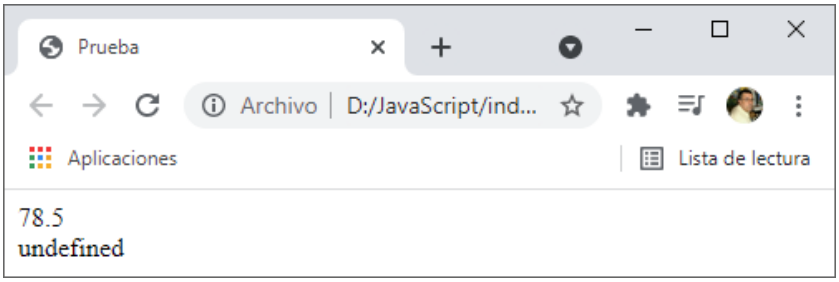
Vamos a crear dos llamadas a la función.

D: > JavaScript > JS operadores.js > ...

```
1    function calcularAreaCirculo(radio){
2        const PI = 3.14;
```

```
    if (radio < 0) {
        return undefined;
    }
    return PI * (radio ** 2);
}
document.write(calcularAreaCirculo(5) + "<br>");
document.write(calcularAreaCirculo(-3));
```

Este será el resultado:



La const PI no podrá cambiar su valor durante la ejecución del programa, esto nos garantiza que esta variable no le cambiaremos su valor por error.

Mutar arreglo declarado con const

D: > JavaScript > JS operadores.js > ...

```
const MI_ARREGLO = [1, 2, 3, 4];
MI_ARREGLO = [5, 6, 7, 8]; // Error.
```

TypeError: “MI_ARREGLO” is read-only (es solo de lectura).

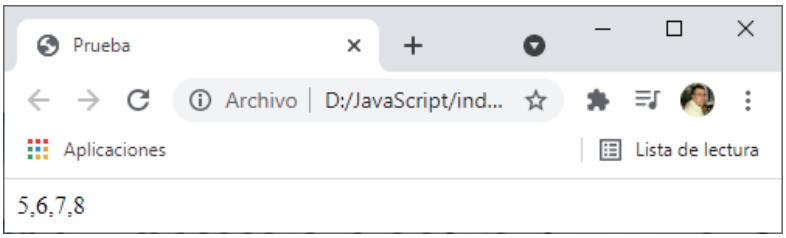
D: > JavaScript > JS operadores.js > ...

```
const MI_ARREGLO = [1, 2, 3, 4];

MI_ARREGLO[0] = 5;
MI_ARREGLO[1] = 6;
MI_ARREGLO[2] = 7;
MI_ARREGLO[3] = 8;

document.write(MI_ARREGLO);
```

Pero si que podemos cambiar elemento por elemento del arreglo.



Crear un objeto inmutable

D: > JavaScript > JS operadores.js > ...

```
1   let colores = {
2       "verde": "#10e04b",
3       "azul": "#1b50e0",
4       "negro": "#00000",
5       "blanco": "#fffff"
6   };
7
8   Object.freeze(colores);
9   document.write(JSON.stringify(colores) + "<br>");
10  colores.amrillo = "#fff200";
11  document.write(JSON.stringify(colores));
```

congelar

Object.freeze(): impide agregar en dicha variable más elementos o modificar un determinado elemento, ya que genera el siguiente error:

TypeError: Cannot add property amarillo, object in not estensible, No permite agregar la propiedad Amarillo, este objeto no es extensible.

Este será el resultado:

Si queremos cambiar algún elemento.

D: > JavaScript > JS operadores.js > ...

```
1   let colores = {
2       "verde": "#10e04b",
3       "azul": "#1b50e0",
4       "negro": "#00000",
5       "blanco": "#fffff"
6   };
7
8   Object.freeze(colores);
9   document.write(JSON.stringify(colores) + "<br>");
10  colores.verde = "#345sg5";
11  document.write(JSON.stringify(colores));
```

Generará el siguiente error:

TypeError: Cannot assign to read only property 'verde' of object '#<Object>'

No se puede asignar a una propiedad que únicamente se puede leer la propiedad 'verde' del objeto.

D: > JavaScript > JS operadores.js > ...

```
let colores = {
    "verde": "#10e04b",
    "azul": "#1b50e0",
    "negro": "#00000",
    "blanco": "#fffff"
};

Object.freeze(colores);
document.write(JSON.stringify(colores) + "<br>");
delete colores.verde;
document.write(JSON.stringify(colores));
```

Si queremos borrar la propiedad 'verde' observaremos el siguiente mensaje de error:

TypeError: Cannot delete property 'verde' of #<Object>, No se puede borrar la propiedad verde de este objeto.

Funciones Flecha

Son un tipo más compacto de funciones que podemos escribir en JavaScript. Se suelen utilizar para definir funciones anónimas es decir que no tienen un nombre especifico.

D: > JavaScript > JS operadores.js > ...

```
const fecha = function(){
    return new Date();
};
```

Para convertirla en una función flecha.

D: > JavaScript > JS operadores.js > ...

```
const fecha = () => new Date();
```

Son muy útiles cuando queremos pasar a una función como argumento de otra función.

Funciones Flecha con parámetros

D: > JavaScript > JS operadores.js > ...

```
const sumarTres = function(x) {
    return x + 3;
```

Vamos a convertirla en una función flecha.

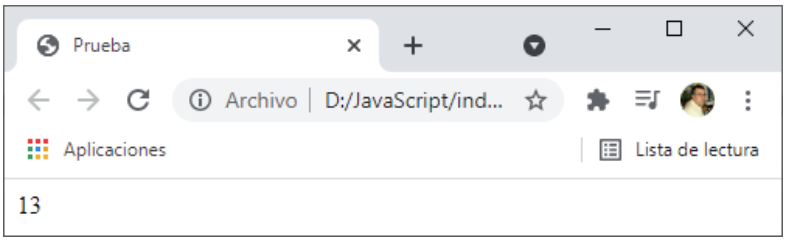
D: > JavaScript > JS operadores.js > ...

```
const sumarTres = (x) => x + 3;
```

Podemos llamar a la función.

```
document.write(sumarTres(10));
```

Este será el resultado:

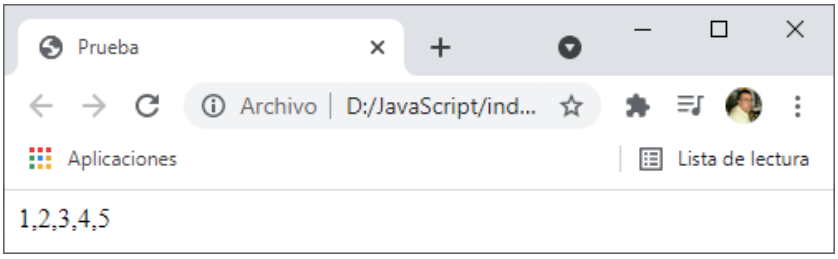


Otro ejemplo:

D: > JavaScript > JS operadores.js > ...

```
1  const concatenarArrglos = function(arr1, arr2){
2      return arr1.concat(arr2);
3  };
4  document.write(concatenarArrglos([1, 2],[3, 4, 5]));
```

Este será el resultado:



¿Como será función flecha?

D: > JavaScript > JS operadores.js > ...

```
1  const concatenarArrglos = (arr1, arr2) => arr1.concat(arr2);
2
3  document.write(concatenarArrglos([1, 2],[3, 4, 5]));
```

El resultado será el mismo.

Si la función tiene más de una línea tendremos que mantener las llaves.

D: > JavaScript > JS operadores.js > ...

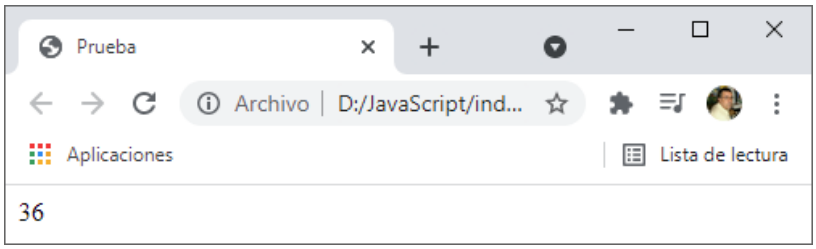
```
1  const sumar = function(a, b) {
2      let num = 6;
3      return a + b + num;
4  };
```

Para convertirla en función flecha.

D: > JavaScript > JS operadores.js > ...

```
1  const sumar = (a, b) => {
2      let num = 6;
3      return a + b + num;
4  };
5
6  document.write(sumar(12, 18));
```

Este será el resultado:

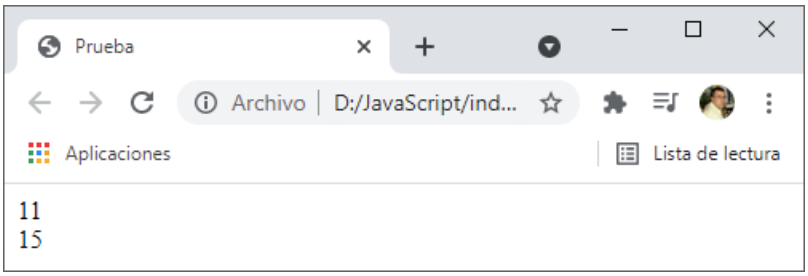


Valores por defecto para parámetros

D: > JavaScript > JS operadores.js > ...

```
1   const incrementar = (num, valor = 1) => num + valor;
2   document.write(incrementar(10) + "<br>");
3   document.write(incrementar(10, 5));
```

En este caso si omitimos el argumento valor al llamar a la función el incremento será 1, si lo ponemos será el valor que definamos, este será el resultado:



Este principio también serve para las funciones anónimas y las funciones normales.

Operador rest

D: > JavaScript > JS operadores.js > ...

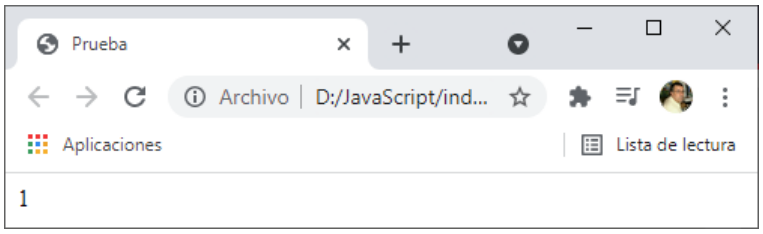
```
1   function miFuncion(...args) {
2       document.write(args);
3   }
4
5   miFuncion(1);
6
```

...args -> operador rest

Variable

Esta función de acepta diferente cantidad de argumentos, para ello después del nombre de la función dentro de los paréntesis pondremos … tres puntos (hace que todos los elementos que aportemos se conviertan en argumentos), seguido del nombre de la variable, he utilizado args para referirme a argumentos pero puede ser cualquier nombre.

En ese primer caso le pasamos solo un argumento, este será el resultado:

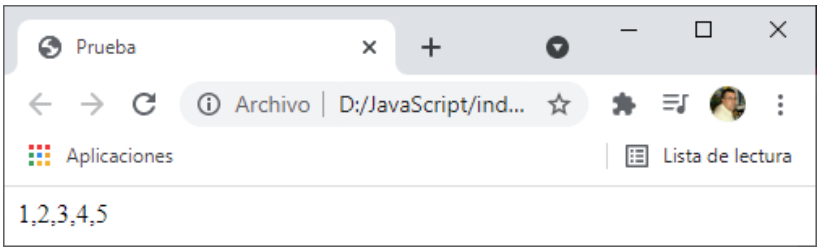


Ahora le vamos a pasar más argumentos.

D: > JavaScript > JS operadores.js > ...

```
1   function miFuncion(...args) {
2       document.write(args);
3   }
4
5   miFuncion(1, 2, 3, 4, 5);
```

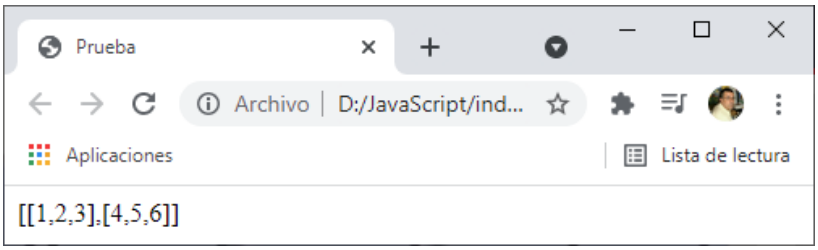
Ahora este será el resultado:



D: > JavaScript > JS operadores.js > ...

```
1   function miFuncion(...args) {
2       document.write(JSON.stringify(args));
3   }
4
5   miFuncion([1, 2, 3], [4, 5, 6]);
```

Esta ver llamamos a la función con dos grupos de arreglos, este será el resultado:



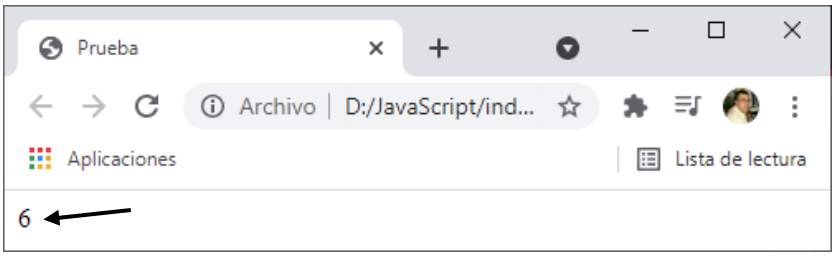
También nos podemos encontrar con la necesidad de saber el número de argumentos que estamos pasando ya que según el valor de este realizaremos un ciclo 'for'.

D: > JavaScript > JS operadores.js > ...

```
1   function miFuncion(...args) {
2       document.write(args.length);
3   }
```

```
4
5    miFuncion(1, 2, 3, 4, 5, 6);
```

Con la función length sabremos el número de argumentos estamos pasando a la función, este será el resultado:



D: > JavaScript > JS operadores.js > ...

```
1    const sumar = (x, y, z) => {
2        const args = [x, y, x];
3        return args.reduce((a, b) => a + b, 0);
4    };
```

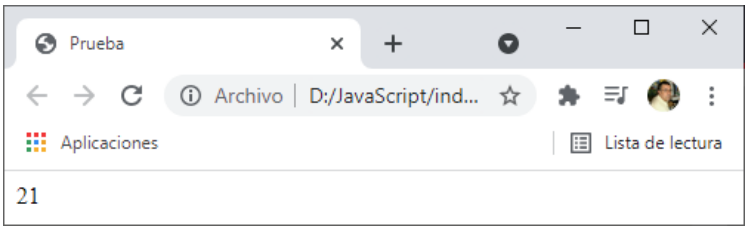
El método .reduce() con estos argumentos suma los elementos del arreglo y retorna el resultado.

Lo vamos a simplificar.

D: > JavaScript > JS operadores.js > ...

```
1    const sumar = (...args) => {
2        return args.reduce((a, b) => a + b, 0);
3    };
4
5    document.write(sumar(1, 2, 3, 4, 5, 6));
```

Este será el resultado:



Operador spread

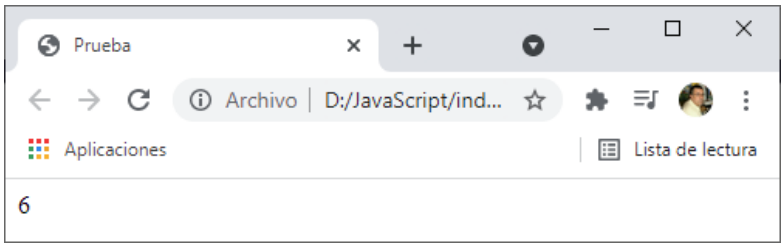
Este operador coge un arreglo y descompone sus elementos para que la función pueda recibirlos y asignarlos a sus correspondientes parámetros.

D: > JavaScript > JS operadores.js > ...

```
1    const numeros = [1, 2, 3];
2
3    function sumar(x, y, z) {
4        return x + y + z;
5    }
```

```
6
7   document.write(sumar(...numeros));
```

Este operador descompone la array números, para que cada uno de los elementos seleccione el argumento de la función, este será el resultado:

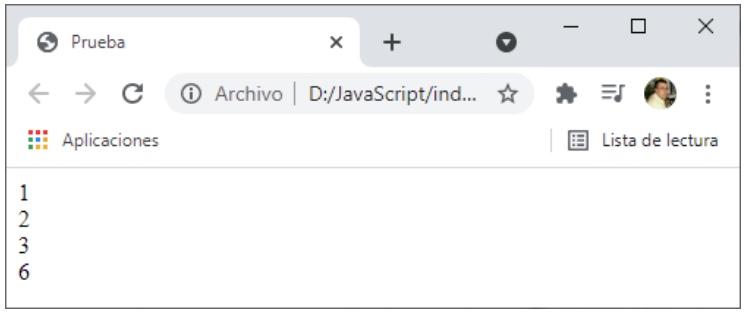


Con el siguiente ejemplo comprobamos como cada elemento de la array es asignado a cada uno de los parámetros.

D: > JavaScript > JS operadores.js > ...

```
1   const numeros = [1, 2, 3];
2
3   function sumar(x, y, z) {
4       document.write(x +"<br>");
5       document.write(y +"<br>");
6       document.write(z +"<br>");
7       return x + y + z;
8   }
9
10  document.write(sumar(...numeros));
```

Este será el resultado:



Sintaxis de desestructuración

Esta sintaxis nos permite asignar a las propiedades de un objeto a variables.

D: > JavaScript > JS operadores.js > ...

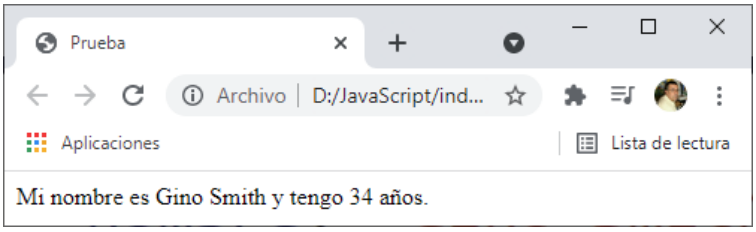
```
1   const usuario = {
2       nombre: "Gino Smith",
3       edad: 34
4   }
5
6   const nombre = usuario.nombre;
7   const edad = usuario.edad;
```

Las variables nombre y edad han adquirido el valor de las propiedades nombre y edad del objeto usuario.

Vamos a agregar la siguiente línea:

```
11    document.write("Mi nombre es "+ nombre + " y tengo " + edad + " años.")
```

Este será el resultado:



Ahora vamos a realizar el mismo proyecto pero más simplificado.

D: > JavaScript > JS operadores.js > ...

```
1    const usuario = {
2        nombre: "Gino Smith",
3        edad: 34
4    }
5
6    const {nombre, edad} = usuario;
7
8    document.write("Mi nombre es "+ nombre + " y tengo " + edad + " años.")
```

En la línea 6 ya definimos todas las variables para asignarles los valores de las propiedades, si ejecutamos el resultado será el mismo.

Otro ejemplo:

D: > JavaScript > JS operadores.js > ...

```
1    var coordenadas = {
2        x: 4,
3        y: 6,
4        z: 12
5    };
6
7    var x = coordenadas.x;
8    var y = coordenadas.y;
9    var z = coordenadas.z;
```

D: > JavaScript > JS operadores.js > ...

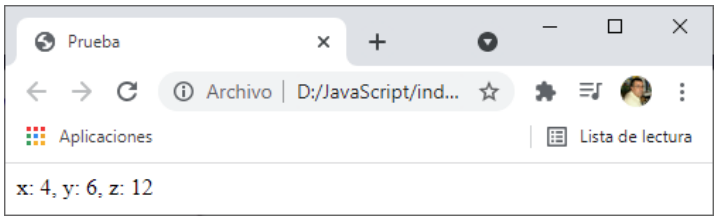
```
1    var coordenadas = {
2        x: 4,
3        y: 6,
4        z: 12
5    };
6
7    const {x, y, z} = coordenadas;
```

Las tres líneas se simplifican a una.

Vamos a agregar la siguiente línea.

```
9    document.write("x: " + x +", " + "y: " + y +", " + "z: " + z);
```

Este será el resultado:

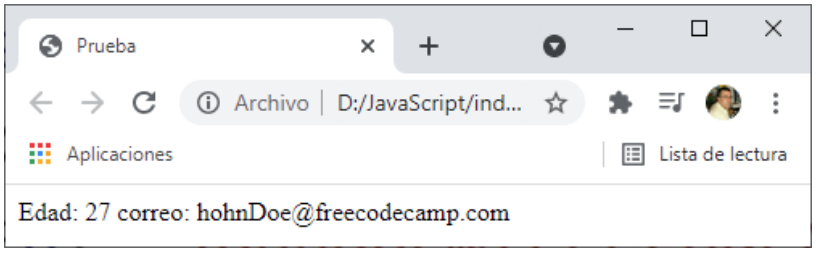


Sintaxis de desestructuración: Objetos Anidados

D: > JavaScript > JS operadores.js > ...

```
const usuario = {
    johnDoe: {
        edad: 27,
        correo: "hohnDoe@freecodecamp.com"
    }
};
const {johnDoe:{edad, correo}} = usuario;
document.write("Edad: " + edad + " correo: " + correo);
```

Este será el resultado:



También se les puede asignar nombres distintos:

D: > JavaScript > JS operadores.js > ...

```
const usuario = {
    johnDoe: {
        edad: 27,
        correo: "hohnDoe@freecodecamp.com"
    }
};
const {johnDoe:{edad:edadUsuario, correo: correoUsuario}} = usuario;
document.write("Edad: " + edadUsuario + " correo: " + correoUsuario);
```

El resultado será el mismo.

D: > JavaScript > JS operadores.js > ...

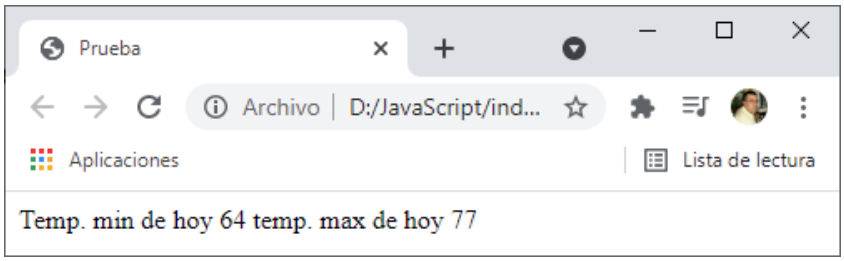
```
const PRONOSTICO_LOCAL = {
    "ayer": {
        minima: 61,
        maxima: 75
    },
    "hoy": {
        minima: 64,
        maxima: 77
    },
```

```
    "mañana":{
        minima: 68,
        maxima: 80
    }
};

const minimoHoy = PRONOSTICO_LOCAL.hoy.minima;
const maximoHoy = PRONOSTICO_LOCAL.hoy.maxima;

document.write("Temp. min de hoy " + minimoHoy + " temp. max de hoy " + maximoHoy);
```

Este será el resultado:



La otra forma de hacerlo:

D: > JavaScript > JS operadores.js > ...

```
const PRONOSTICO_LOCAL = {
    "ayer": {
        minima: 61,
        maxima: 75
    },
    "hoy": {
        minima: 64,
        maxima: 77
    },
    "mañana":{
        minima: 68,
        maxima: 80
    }
};

const {hoy:{minima: minimoHoy, maxima: maximoHoy}} = PRONOSTICO_LOCAL;

document.write("Temp. min de hoy " + minimoHoy + " temp. max de hoy " + maximoHoy);
```

El resultado será el mismo.

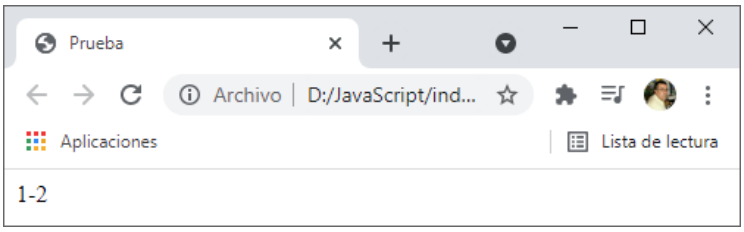
Sintaxis de desestructuración: Arreglos

D: > JavaScript > JS operadores.js > ...

```
var a;
var b;
[a, b] = [1, 2];

document.write(a + "-" + b);
```

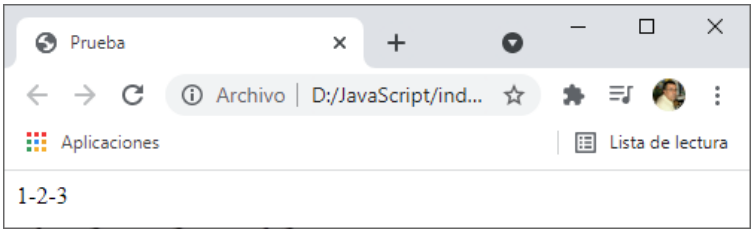
Este será el resultado:



D: > JavaScript > JS operadores.js > ...

```
1   var a;
2   var b;
3   var c;
4   [a, b, c] = [1, 2, 3];
5
6   document.write(a + "-" + b + "-" + c);
```

Y así podemos agregar más variables, este sería el resultado:



D: > JavaScript > JS operadores.js > ...

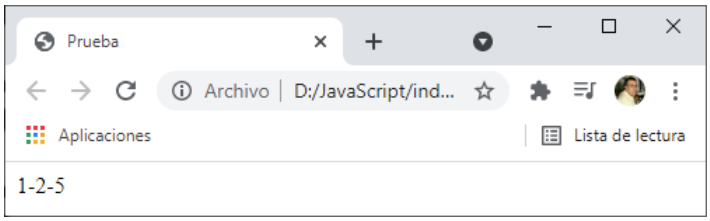
```
1   var a;
2   var b;
3
4   [a, b] = [1, 2, 3, 4, 5, 6];
5
6   document.write(a + "-" + b);
```

En este ejemplo la variable a se le asigna el valor del primer elemento y la variable b se le asigna el valor del segundo elemento.

D: > JavaScript > JS operadores.js > ...

```
1   var a;
2   var b;
3   var c;
4   [a, b,,, c] = [1, 2, 3, 4, 5, 6];
5
6   document.write(a + "-" + b + "-" + c);
```

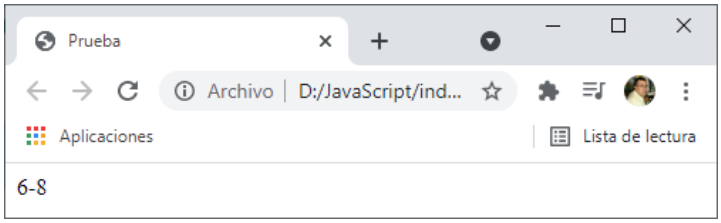
Con cada coma saltamos una posición, de este modo la variable c asume el valor 5, este será el resultado:



D: > JavaScript > JS operadores.js > ...

```
1   var a = 8;
2   var b = 6;
3
4   [b, a] = [a, b];
5
6   document.write (a + "-" + b);
```

Para intercambiar valores entre variables, este será el resultado:



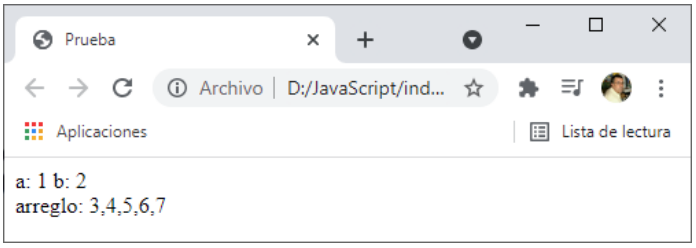
Sintaxis de desestructuración con el operador rest

Recuerda que el operador rest agrupaba los elementos y formaba un arreglo cuando lo usábamos en conjunciones . Vamos a utilizar este operador para reasignar a un arreglo en varias variables.

D: > JavaScript > JS operadores.js > ...

```
1   var a;
2   var b;
3   var arr;
4   [a, b,...arr ] = [1, 2, 3, 4, 5, 6, 7];
5   document.write("a: " + a + " b: " + b + "<br>");
6   document.write("arreglo: " + arr);
```

Este será el resultado:



D: > JavaScript > JS operadores.js > ...

```
1   const arregloInicial = [1, 2, 3, 4, 5, 6, 7, 8];
2   function removerTresPrimerosElementos(arreglo){
3       var nueveArrglo;
4       const[ , , ,...nuevoArreglo ] = arreglo;
5       return nuevoArreglo;
6   }
7
8   const arregloFinal = removerTresPrimerosElementos(arregloInicial);
9   document.write(arregloFinal);
```

Queremos eliminar los tres primeros elementos del arregloInicial.

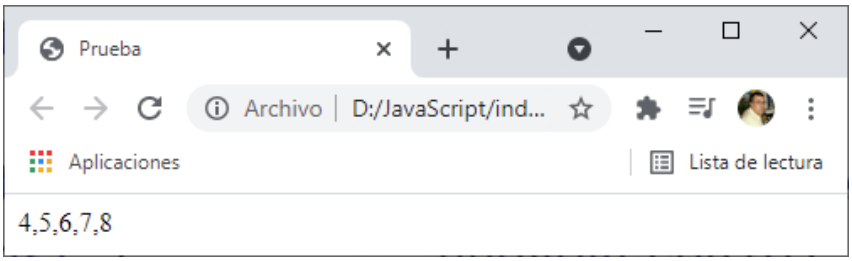
Creamos una función llamada removerTresPrimerosElemntos con un parámetro.

En la línea 4 con la opción de comas le decimos que nuevoArreglo empieze a coger valores a partir de la posición 4 por este motivo hay 3 comas.

Línea 5 retornamos el nuevoArrglo.

Línea 8 llamamos a la función removerTresPrimerosElementos con el argumento arregloInicial.

Línea 9 mostramos el resultado:



Sintaxis de desestructuración: Pasar Objeto como Argumento

Sirve para captar un objeto como argumento.

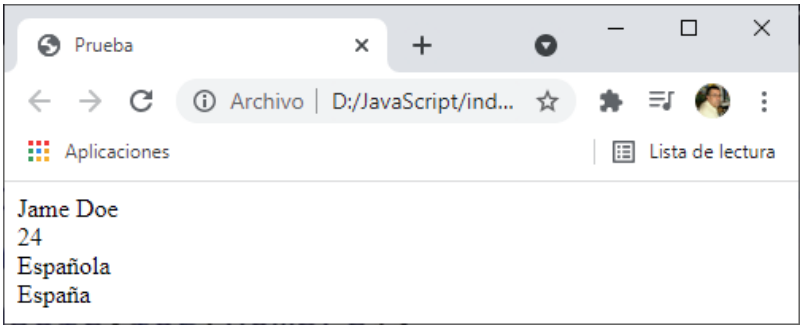
D: > JavaScript > JS operadores.js > ...

```
1   var nuevoPerfilCliente = {
2       nombre: "Jame Doe",
3       edad: 24,
4       nacionalidad: "Española",
5       ubicacion: "España"
6   };
7
8   const actualizarPerfil = (informacionDePerfil) => {
9       const{nombre, edad, nacionalidad, ubicacion} = informacionDePerfil;
10      document.write(nombre +"<br>");
11      document.write(edad +"<br>");
12      document.write(nacionalidad +"<br>");
13      document.write( ubicacion);
14  };
15
16  actualizarPerfil(nuevoPerfilCliente);
```

Hemos creado un nuevo objeto llamado nuevoPerfilCliente.

Hemos creado una función de tipo flecha con un parámetro de tipo objeto llamado informacioDePerfil con las mismas propiedades que nuevoPerfilCliente.

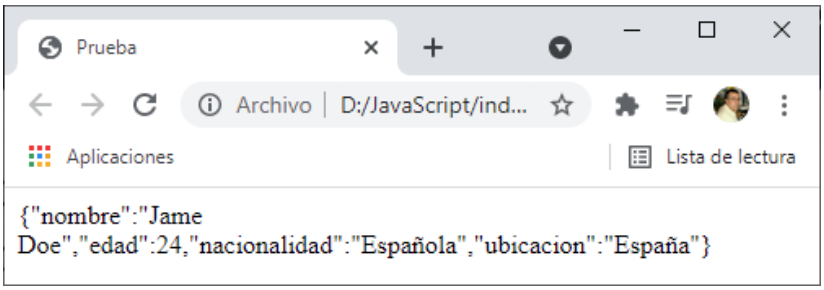
Si llamamos a la función y como agrumento agregamos el objeto nuevoPerfilCliete, este será el resultado:



D: > JavaScript > JS operadores.js > ...

```
1   var nuevoPerfilCliente = {
2       nombre: "Jame Doe",
3       edad: 24,
4       nacionalidad: "Española",
5       ubicacion: "España"
6   };
7
8   const actualizarPerfil = (informacionDePerfil) => {
9       const{nombre, edad, nacionalidad, ubicacion} = informacionDePerfil;
10      document.write(JSON.stringify(informacionDePerfil));
11      /*document.write(nombre +"<br>");
12      document.write(edad +"<br>");
13      document.write(nacionalidad +"<br>");
14      document.write( ubicacion);*/
15  };
16
17  actualizarPerfil(nuevoPerfilCliente);
```

Hemos comentado las líneas del 11 hasta el 14 y hemos agregado la línea 10, para que nos muestre la información que tiene informacionDePerfil.



D: > JavaScript > JS operadores.js > ...

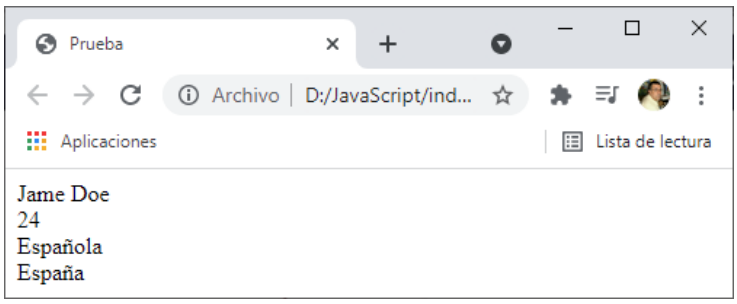
```
1   var nuevoPerfilCliente = {
2       nombre: "Jame Doe",
3       edad: 24,
4       nacionalidad: "Española",
```

```
    ubicacion: "España"
};

const actualizarPerfil = ({nombre, edad, nacionalidad, ubicacion}) => {
    document.write(nombre + "<br>");
    document.write(edad + "<br>");
    document.write(nacionalidad + "<br>");
    document.write(ubicacion);
};

actualizarPerfil(nuevoPerfilCliente);
```

Este será el resultado:



No hay por que seleccionar todos los elementos.

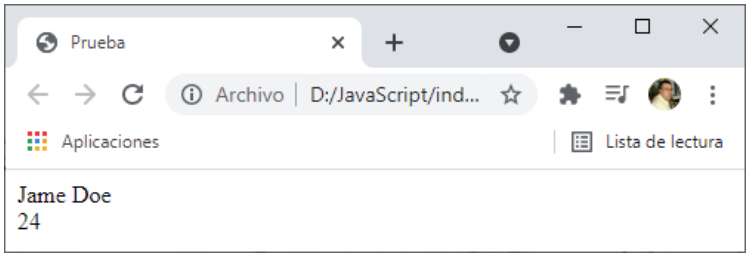
D: > JavaScript > JS operadores.js > ...

```
var nuevoPerfilCliente = {
    nombre: "Jame Doe",
    edad: 24,
    nacionalidad: "Española",
    ubicacion: "España"
};

const actualizarPerfil = ({nombre, edad}) => {
    document.write(nombre + "<br>");
    document.write(edad );
};

actualizarPerfil(nuevoPerfilCliente);
```

Este será el resultado:



Otro ejemplo:

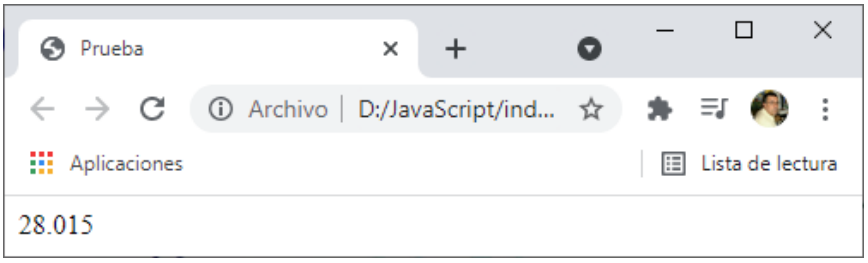
Como puedes pasar un objeto usando la sintaxis de desestructuración.

D: > JavaScript > JS operadores.js > ...

```
const estaisticas = {
    max: 56.78,
    desviacionEstandar: 4.34,
    mediana: 34.54,
    moda: 23.87,
    min: -0.75,
    propmedio: 35.85
};
const mitad = ({max, min}) => (max + min) / 2.0;

document.write(mitad(estaisticas));
```

Este será el resultado:



Plantillas literales

Plantillas literales o plantillas de cadena.

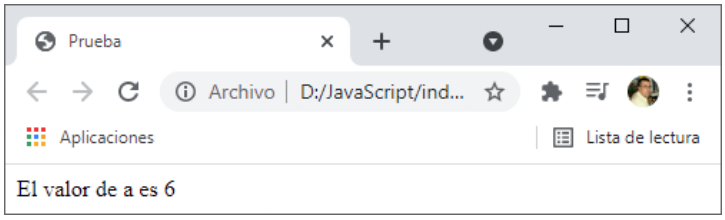
Características:

- Se usa el acento invertido (backtick) ` en lugar de comillas.
- Puede contener comillas simples y dobles.
- Las líneas se preservan como se escriben en el código.
- Para reemplazar una variable se escribe ${variable}.
- Dentro de ${} también se pueden escribir expresiones.

D: > JavaScript > JS operadores.js > ...

```
var a = 6;
document.write(`El valor de a es ${a}`);
```

Este será el resultado:

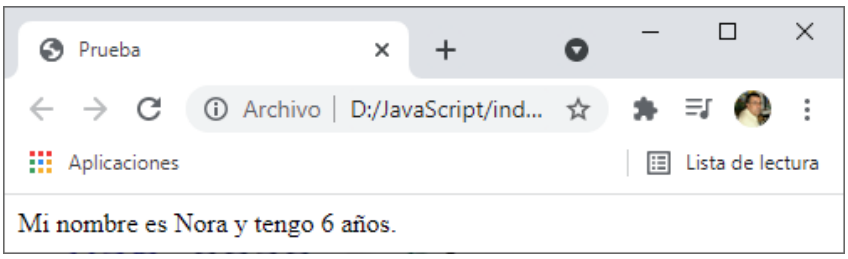


Otro ejemplo con una cadena de caracteres:

D: > JavaScript > JS operadores.js > ...

```
1   var nombre = "Nora"
2   var edad = 6;
3
4   document.write(`Mi nombre es ${nombre} y tengo ${edad} años.`);
```

Este será el resultado:

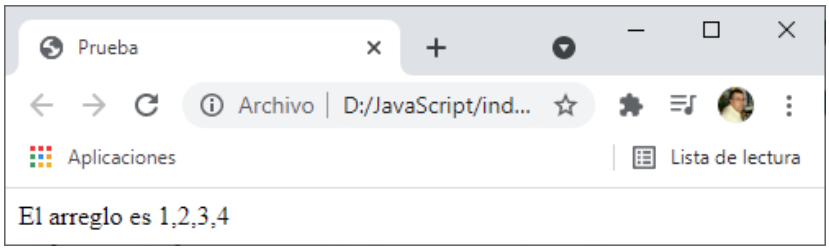


También podemos reemplazar arreglos.

D: > JavaScript > JS operadores.js > ...

```
1   var miArreglo = [1, 2, 3, 4];
2   document.write(`El arreglo es ${miArreglo}`);
```

Este será el resultado:

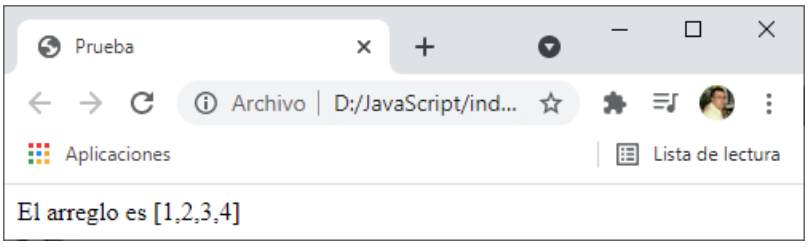


Si queremos ver el arreglo con corchetes.

D: > JavaScript > JS operadores.js > ...

```
1   var miArreglo = [1, 2, 3, 4];
2   document.write(`El arreglo es ${JSON.stringify(miArreglo)}`);
```

Este será el resultado:



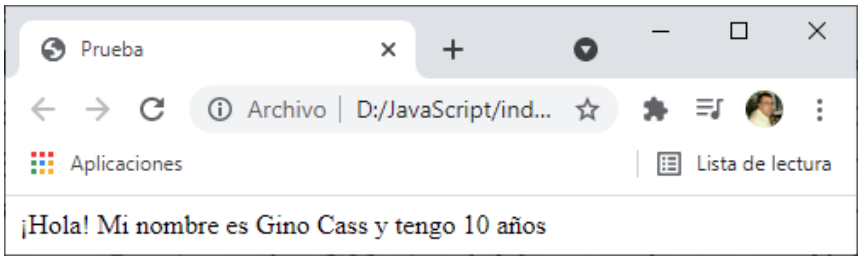
También podemos reemplazar el valor de la propiedad de un objeto.

D: > JavaScript > JS operadores.js > ...

```
1   var persona = {
2       nombre: "Gino Cass",
3       edad: 10
4   };
5
```

```
6   const saludo = `¡Hola! Mi nombre es ${persona.nombre} y tengo ${persona.edad} años`;
7   document.write(saludo);
```

Este será el resultado:

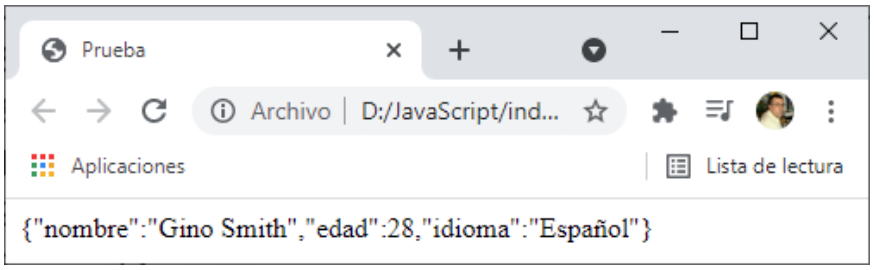


Crear Objetos de Forma concisa

D: > JavaScript > JS operadores.js > ...

```
1   const crearPersona = (nombre, edad, idioma) => {
2       return {
3           nombre: nombre,
4           edad: edad,
5           idioma: idioma
6       };
7   };
8
9   document.write(JSON.stringify(crearPersona("Gino Smith", 28, "Español" )));
```

Este será el resultado:



Ya tenemos el objeto.

El ejemplo anterior lo podemos hacer un una sola línea.

```
1   const crearPersona = (nombre, edad, idioma) => ({nombre, edad, idioma});
2
3   document.write(JSON.stringify(crearPersona("Gino Smith", 28, "Español" )));
```

El resultado es exactamente el mismo.

Métodos

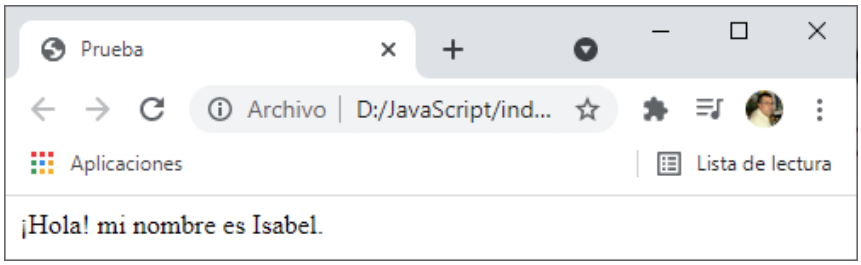
La función 'this.', hace referencia al objeto que estamos trabajando.

D: > JavaScript > JS operadores.js > ...

```
1   const persona = {
2       nombre: "Isabel",
3       presentarse: function(){
```

```
        return `¡Hola! mi nombre es ${this.nombre}.`
    }
};
// Si el valor de una propiedad es una función,
// se denomina "método".
document.write(persona.presentarse());
```

Este será el resultado:



Aun lo podemos reducir más.

D: > JavaScript > JS operadores.js > ...

```
const persona = {
    nombre: "Isabel",
    presentarse(){
        return `¡Hola! mi nombre es ${this.nombre}.`
    }
};
// Si el valor de una propiedad es una función,
// se denomina "método".
document.write(persona.presentarse());
```

Definir una Clase

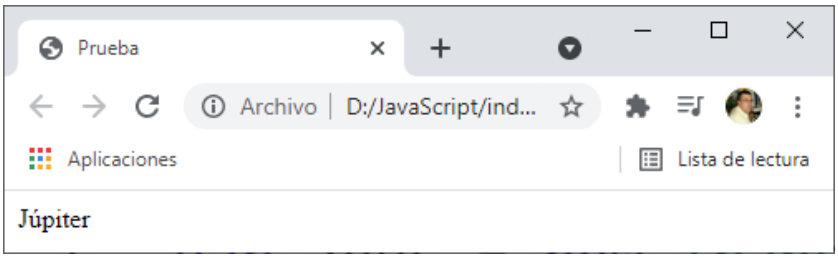
Una clase es algo parecido a un plano de un edificio de un objeto que nos permite crear múltiples objetos con la misma estructura, las mismas propiedades y la misma funcionalidad.

Los nombres de las clases empiezan con mayúsculas según los estándares de JavaScript.

D: > JavaScript > JS operadores.js > ...

```
class TrasbordadorEspacial {
    constructor(planeta) {
        this.planeta = planeta;
    }
}
var zeus = new TrasbordadorEspacial("Júpiter");
document.write(zeus.planeta);
```

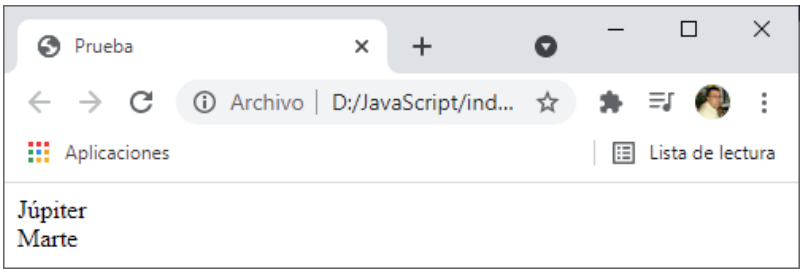
Este será el resultado:



Ahora vamos a crear un segundo objeto.

```
 9   var apolo = new TrasbordadorEspacial("Marte");
10   document.write("<br>" + apolo.planeta);
```

Este será el resultado:

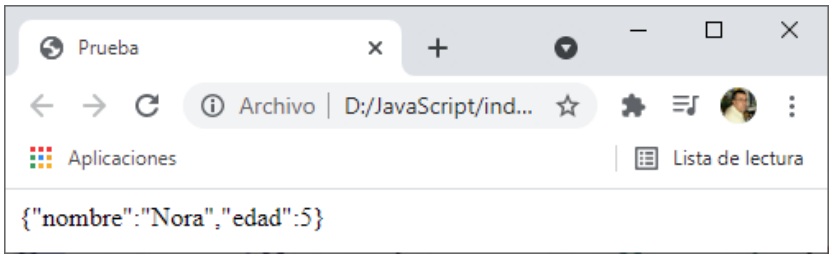


Otro ejemplo:

D: > JavaScript > JS operadores.js > ...

```
1  class Mascotas {
2      constructor(nombre, edad) {
3          this.nombre = nombre;
4          this.edad = edad;
5      }
6  }
7
8  var miMascota = new Mascotas("Nora", 5);
9
10 document.write(JSON.stringify(miMascota));
```

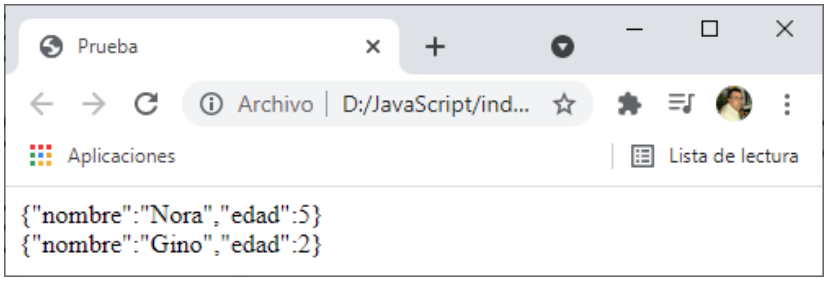
Este será el resultado:



Vamos a crear un segundo objeto:

```
12    var tuMascota = new Mascotas("Gino", 2);
13    document.write("<br>" + JSON.stringify(tuMascota));
```

Este será el resultado:



Getters y Setters

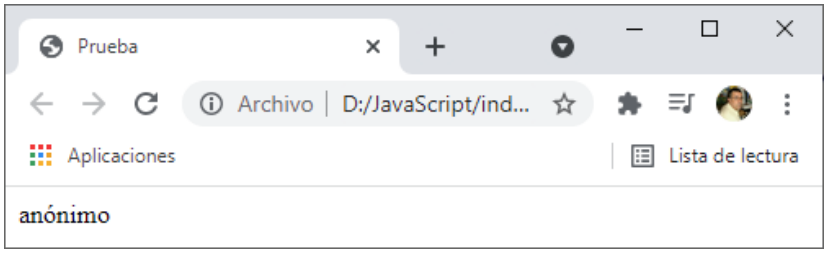
Estas funciones nos permitirán proteger el objeto.

D: > JavaScript > JS operadores.js > ...

```
1  class Libro {
2      constructor(autor){
3          this._autor = autor;
4      }
5      get autor(){
6          return this._autor;
7      }
8
9      set autor(nuevoAutor){
10         this._autor = nuevoAutor;
11     }
12 }
13
14 const libro = new Libro("anónimo");
15 document.write(libro.autor);
```

Get significa obtener y set colocar.

Este será el resultado:

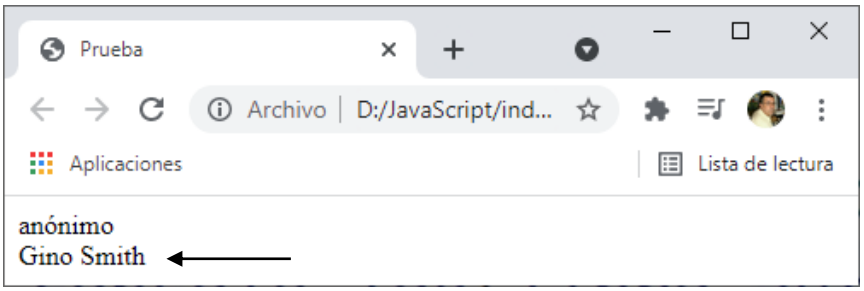


En la línea 5 declaramos el nombre de la función get con autor(), esto nos permite mostrar en la línea 15 con el nombre del objete, seguido de un punto y el método get mostrar la información de esta propiedad del objeto libro.

Ahora vamos a actualizar el valor de la propiedad.

```
17    libro.autor = "Gino Smith";
18    document.write("<br>" + libro.autor);
```

Este será el nuevo resultado:



Contenido

¿Cómo ejecutar el JavaScript? 1
Comentarios 4
Variables 4
Operadores de asignación 6
Inicializar variables 7
Variable no inicializada 7
Asignar el valor de una variable a otra variable 8
Mayúsculas y minúsculas 8
Suma 9
Resta 9
Multiplicación 9
División 10
Números decimales 10
Multiplicar números decimales 11
Dividir números decimales 11
Resto de la división 12
Incrementar el valor de una variable 12
Reducir el valor de una variable 13
Asignación de suma 13
Asignación de Resta 14
Asignación de multiplicación 15
Asignación de la división 15
Variables con cadena de caracteres 16
Escapar comillas en cadena de caracteres 17
Cadena de caracteres con comillas simples 17
Secuencias de escape 18
Concatenar cadena de caracteres 19
Construir cadenas con variables 19
Agregar variables a Cadena de caracteres 20
Longitud de una cadena de caracteres 20
Notación de corchetes: Primer carácter 21
Inmutabilidad de cadena de caracteres 21
Notación de corchetes: Enésimo Carácter 22
Notación de corchetes: Último carácter 23
Notación de corchetes: De derecha a izquierda 24

Palabras en blanco 25
Arreglos (Arrays) 25
Arreglos anidados 26
Acceder a los elementos de un arreglo 27
Modificar los datos de un arreglo 28
Acceder a Arreglos multidimensionales 29
Método .push() 30
Método .pop() 30
Método .shift() 31
Método .unshift() 31
Lista de compras 31
Funciones 32
Argumentos 33
Ámbito global 34
Ámbito local 34
Ámbito Global vs. Ámbito Local 35
Retornar un valor 36
undefined 36
Asignar un valor retorno 37
Permanece en fila 38
Valores Booleanos 39
Operadores de igualdad 39
Operadores de igualdad estricta 40
Practica cómo comparar valores 41
Operador de desigualdad 42
Operadores de desigualdad estricta 42
Operador mayor que 43
Operador mayor o igual que 44
Operador menor que 45
Operador menor o igual que 46
Operador lógico and 46
Operador lógico or 47
Operador lógico not 48
Sentencias condicionales 49
Cláusula “else” 52
Cláusula “else if” 53

Condicionales: Orden lógico 54
Encadenar sentencias “if…else” 54
Código de Golf 55
Sentencias Switch 56
Sentencia Switch: Opción predeterminada 58
Sentencias Switch: Múltiples casos 59
Reemplazar “if…else” por “switch” 60
Retornar valores Booleanos 61
Patrón de retorno anticipado 62
Conteo de cartas 63
Crear Objetos 65
Acceder a Propiedades: Notación de Punto 66
Acceder a Propiedades: Notación de corchetes 66
Acceder a propiedades: Variables 67
Actualizar propiedades 67
Agregar propiedades 69
Eliminar propiedades 69
Objetos para búsqueda 70
Verificar propiedades 71
Objetos complejos 72
Objetos anidados 74
Arreglos anidados 76
Colección de discos 77
Ciclo “while” 80
Ciclo “for” 82
Ciclos “for”: Números impares 83
Ciclos “for”: Contar hacia atrás 84
Iterar sobre un arreglo con un ciclo “for” 85
Ciclos “for” anidados 88
Ciclos “do…while” 88
Búsqueda de perfil 90
Números aleatorios 93
Números enteros aleatorios 94
Números enteros aleatorios en un rango 94
Función parseInt() 95
Función parseInt() con una base 96

Operador condicional (Ternario) 97
Múltiples operadores condicionales 98
var vs. let 99
Ámbito de var vs. let 101
const 103
Mutar arreglo declarado con const 105
Crear un objeto inmutable 106
Funciones Flecha 107
Funciones Flecha con parámetros 107
Valores por defecto para parámetros 109
Operador rest 109
Operador spread 111
Sintaxis de desestructuración 112
Sintaxis de desestructuración: Objetos Anidados 114
Sintaxis de desestructuración: Arreglos 115
Sintaxis de desestructuración con el operador rest 117
Sintaxis de desestructuración: Pasar Objeto como Argumento 118
Plantillas literales 121
Crear Objetos de Forma concisa 123
Métodos 123
Definir una Clase 124
Getters y Setters 126