Tema 1

Introducción a JavaScript

Índice

7. Sentencias JavaScript
    1. Sentencias JavaScript
    2. Programas JavaScript
    3. Punto y coma;
    4. Espacios en blanco en JavaScript
    5. Longitud de línea JavaScript y salto de línea
    6. Bloques de código JavaScript
    7. JavaScript Keywords
    8. Comentarios de JavaScript
    9. Comentarios de línea única
    10. Comentarios de varias líneas
    11. Uso de comentarios para evitar la ejecución

8. Variables JavaScript
    1. Variables JavaScript
    2. Al igual que el álgebra
    3. Identificadores de JavaScript
    4. El operador de asignación
    5. Tipos de datos JavaScript
    6. Declaración (creación) de variables JavaScript
    7. Una declaración, muchas variables
    8. Valor = undefined
    9. Volver a declarar variables JavaScript
    10. Aritmética JavaScript

9. Operadores JavaScript
    1. Operadores aritméticos JavaScript
    2. Operadores de asignación de JavaScript
    3. Operadores de String JavaScript
    4. Añadir cadenas y números
    5. Operadores de comparación de JavaScript
    6. Operadores lógicos JavaScript
    7. Operadores de tipo JavaScript
    8. Operadores Bitwise de JavaScript

10. Comparación de JavaScript y operadores lógicos
    1. Operadores de comparación
    2. Como puede ser usado
    3. Operadores lógicos
    4. Operador condicional (ternario)
    5. Comparación de diferentes tipos

11. Aritmética JavaScript
    1. Operadores aritméticos JavaScript
    2. Operaciones aritmeticas
    3. Operadores y Operandos
    4. Precedencia del operador
    5. Valores de precedencia del operador JavaScript

12. Asignación de JavaScript
    1. Operadores de asignación de JavaScript

13. Tipos de datos JavaScript
    1. Tipos de datos JavaScript
    2. El concepto de tipos de datos
    3. Los tipos JavaScript son dinámicos.
    4. Cadenas en JavaScript
    5. Números de JavaScript
    6. Booleanos JavaScript
    7. Arrays JavaScript
    8. Objetos JavaScript
    9. El tipo de Operador
    10. Undefinded
    11. Valores vacíos
    12. Null
    13. Diferencia entre undefined y null
    14. Datos primitivos
    15. Datos complejos
    16. 14. Sentencias JavaScript If...Else
    17. Sentencias condicionales
    18. La sentencia if
    19. La sentencia else
    20. La sentencia else if

14. Sentencia Switch de JavaScript
    1. La instrucción Switch de JavaScript
    2. La palabra clave default
    3. Bloques de código comunes

15. El bucle For de JavaScript
    1. Los bucles de JavaScript
    2. Diferentes tipos de bucles
    3. El bucle for
    4. El bucle For / In
    5. El bucle while

16. El bucle While JavaScript
    1. El bucle while
    2. El bucle Do / While
    3. Comparando For y While

Capítulo 1. Introducción

1. ¿Qué es JavaScript?

JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas.

Una página web dinámica es aquella que incorpora efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario.

Técnicamente, JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.

A pesar de su nombre, JavaScript no guarda ninguna relación directa con el lenguaje de programación Java. Legalmente, JavaScript es una marca registrada de la empresa Sun Microsystems, como se puede ver en http://www.sun.com/suntrademarks/.

2. Hola Mundo

La forma más inmediata de empezar a experimentar con JavaScript es escribir secuencias de comandos simples. Es necesario sólo un navegador web y un editor de texto.

En primer lugar deberemos crear una página HTML. Para ello podemos utilizar cualquier editor de texto que tengamos a mano.

```
<html>
      <head><title>Mi primer codigo JavaScript</title></head>
<body>

<h1>Mi primer código JavaScript</h1>

</body>
</html>
```

Para poder utilizar código JavaScript deberemos de ayudarnos del elemento <script>. Dentro de esta etiqueta es donde pondremos nuestro código JavaScript.

Una forma de escibir texto en pantalla con JavaScript es utilizar el objeto document y el método .write("texto")

```
<html>
      <head><title>Mi primer codigo JavaScript</title></head>
<body>

<h1>Mi primer código JavaScript</h1>

<script>
  document.write("Hola Mundo");
</script>

</body>
</html>
```

Otra forma es mostrar un mensaje en pantalla:

```
<!DOCTYPE html>
<html>
	<head>
		<meta http-equiv= "content-type" content="text/html; charset=utf-8">
		<title>Hola Mundo</title>
	</head>
	<body>
		<script>
			alert(“Hola mundo en JavaScript”)
		</script>
	</body>
</html>
```

3. Cómo incluir JavaScript en documentos HTML

La etiqueta <script>

En HTML, el código JavaScript debe insertarse entre las etiquetas <script> y </script>.

Ejemplo

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

Inténtalo tú mismo "

Los antiguos ejemplos de JavaScript pueden usar un atributo de tipo: <script type = "text / javascript">.

El atributo type no es necesario. JavaScript es el lenguaje de script predeterminado en HTML.

JavaScript en <head> o <body>

Puede colocar cualquier número de secuencias de comandos en un documento HTML.

Los scripts se pueden colocar en el <body>, o en la sección <head> de una página HTML, o en ambos.

JavaScript en <head>

En este ejemplo, una función JavaScript se coloca en la sección <head> de una página HTML.

La función se invoca (llamada) cuando se hace clic en un botón:

Ejemplo

```
<!DOCTYPE html>
<html>

<head>
<script>
function myFunction() {
    document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>

<body>

<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```

Inténtalo tú mismo "

JavaScript en <body>

En este ejemplo, una función JavaScript se coloca en la sección <body> de una página HTML.

La función se invoca (es llamada) cuando se hace clic en un botón:

Ejemplo

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

<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</body>
</html>
```

Inténtalo tú mismo "

Colocar secuencias de comandos en la parte inferior del elemento <body> mejora la velocidad de visualización, porque la compilación de secuencias de comandos ralentiza la visualización.

JavaScript externo

Los scripts también se pueden colocar en archivos externos:

Archivo externo: myScript.js

```
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
```

Los scripts externos son prácticos cuando se utiliza el mismo código en muchas páginas web diferentes.

Los archivos JavaScript tienen la extensión de archivo .js .

Para usar un script externo, coloque el nombre del archivo de script en el atributo src (fuente) de una etiqueta <script>:

Ejemplo

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

<script src="myScript.js"></script>

</body>
</html>
```

Inténtalo tú mismo "

Puede colocar una referencia de guión externa en <head> o <body> a su gusto.

El script se comportará como si estuviera ubicado exactamente donde se encuentra la etiqueta <script>.

Las secuencias de comandos externas no pueden contener etiquetas <script>.

Ventajas del JavaScript externo

La colocación de scripts en archivos externos tiene algunas ventajas:

- Separa HTML y código
- Hace que HTML y JavaScript sean más fáciles de leer y mantener
- Los archivos en caché de JavaScript pueden acelerar la carga de la página

Para agregar varios archivos de secuencia de comandos a una página, utilice varias etiquetas de secuencia de comandos:

Ejemplo

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

Referencias externas

Los scripts externos pueden referenciarse con una URL completa o con una ruta relativa a la página web actual.

Este ejemplo utiliza una URL completa para vincular a un script:

Ejemplo

```
<script src="https://www.w3schools.com/js/myScript1.js"></script>
```

Inténtalo tú mismo "

Este ejemplo utiliza un script ubicado en una carpeta especificada en el sitio web actual:

Ejemplo

```
<script src="/js/myScript1.js"></script>
```

Inténtalo tú mismo "

Este ejemplo enlaza con un script ubicado en la misma carpeta que la página actual:

Ejemplo

```
<script src="myScript1.js"></script>
```

Inténtalo tú mismo "

Puede leer más acerca de las rutas de archivo en el capítulo Rutas de archivos HTML .

4. Etiqueta noscript

Algunos navegadores no disponen de soporte completo de JavaScript, otros navegadores permiten bloquearlo parcialmente e incluso algunos usuarios bloquean completamente el uso de JavaScript porque creen que así navegan de forma más segura.

En estos casos, es habitual que si la página web requiere JavaScript para su correcto funcionamiento, se incluya un mensaje de aviso al usuario indicándole que debería activar JavaScript para disfrutar completamente de la página. El siguiente ejemplo muestra una página web basada en JavaScript cuando se accede con JavaScript activado y cuando se accede con JavaScript completamente desactivado.

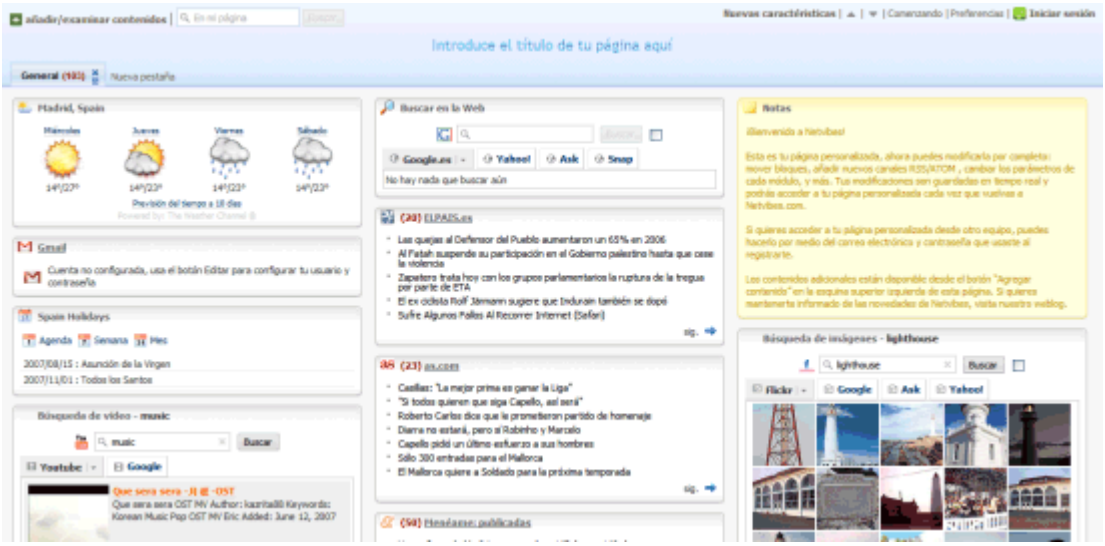


Figura 1.1 Imagen de www.Netvibes.com con JavaScript activado

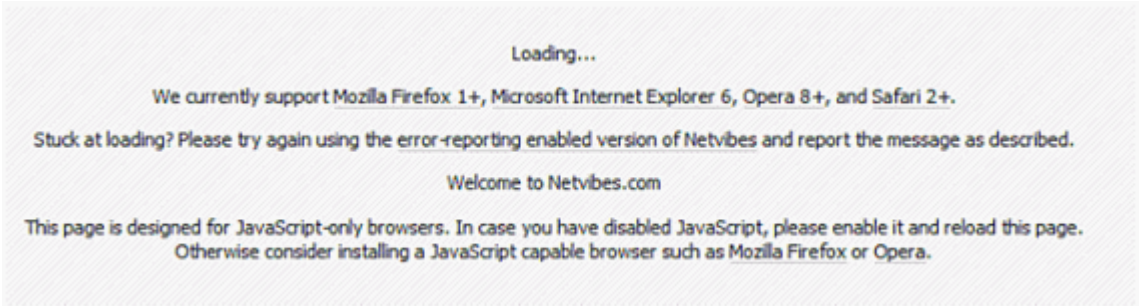


Figura 1.2 Imagen de www.Netvibes.com con JavaScript desactivado

El lenguaje HTML define la etiqueta <noscript> para mostrar un mensaje al usuario cuando su navegador no puede ejecutar JavaScript. El siguiente código muestra un ejemplo del uso de la etiqueta <noscript>:

```
<head> ... </head>
<body>
<noscript>
  <p>Bienvenido a Mi Sitio</p>
  <p>La página que estás viendo requiere para su funcionamiento el uso de JavaScript.
Si lo has deshabilitado intencionadamente, por favor vuelve a activarlo.</p>
</noscript>
</body>
```

La etiqueta <noscript> se debe incluir en el interior de la etiqueta <body> (normalmente se incluye al principio de <body>). El mensaje que muestra <noscript> puede incluir cualquier elemento o etiqueta XHTML.

5. Posibilidades de visualización de JavaScript

JavaScript puede "mostrar" los datos de diferentes maneras:

- Escribir en un elemento HTML, usando innerHTML .
- Escribir en la salida HTML usando document.write () .
- Escribir en un cuadro de alerta, utilizando window.alert () .
- Escribir en la consola del navegador, usando console.log () .

Uso de innerHTML

Para acceder a un elemento HTML, JavaScript puede utilizar el método document.getElementById (id) .

El atributo id define el elemento HTML. La propiedad innerHTML define el contenido HTML:

Ejemplo

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

<h1>My First Web Page</h1>
<p>My First Paragraph</p>

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

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

</body>
</html>
```

Inténtalo tú mismo "

Cambiar la propiedad innerHTML de un elemento HTML es una forma común de mostrar datos en HTML.

Uso de document.write ()

Para hacer pruebas, es conveniente utilizar document.write () :

Ejemplo

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

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

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

</body>
</html>
```

Inténtalo tú mismo "

El uso de document.write () después de que un documento HTML esté completamente cargado, eliminará todo el HTML existente :

Ejemplo

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

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<button onclick="document.write(5 + 6)">Try it</button>

</body>
</html>
```

Inténtalo tú mismo "

El método document.write () sólo debe utilizarse para las pruebas.

Utilizando window.alert ()

Puede utilizar un cuadro de alerta para mostrar datos:

Ejemplo

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

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

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

</body>
</html>
```

Inténtalo tú mismo "

Utilizar console.log ()

Para fines de depuración, puede utilizar el método console.log () para mostrar datos.

Aprenderá más sobre la depuración en un capítulo posterior.

Ejemplo

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

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

</body>
</html>
```

Inténtalo tú mismo "

6. Sintaxis de JavaScript

La sintaxis de JavaScript es el conjunto de reglas, cómo se construyen los programas de JavaScript.

Programas JavaScript

Un programa de computadora es una lista de "instrucciones" que debe ser "ejecutada" por el ordenador.

En un lenguaje de programación, estas instrucciones del programa se llaman sentencias.

JavaScript es un lenguaje de programación .

Las sentencias JavaScript están separadas por punto y coma :

Ejemplo

```
var x, y, z;
x = 5;
y = 6;
z = x + y;
```

Inténtalo tú mismo "

En HTML, los programas JavaScript son ejecutados por el navegador web.

Sentencias JavaScript

Las sentencias JavaScript se componen de:

Valores, operadores, expresiones, palabras clave y comentarios.

Valores JavaScript

La sintaxis de JavaScript define dos tipos de valores: Valores fijos y valores de variables.

Los valores fijos se llaman literales. Los valores variables se llaman variables .

Literales de JavaScript

Las reglas más importantes para escribir valores fijos son:

Los números se escriben con o sin decimales:

```
10.50
```

```
1001
```

Inténtalo tú mismo "

Las cadenas son texto, escrito con comillas dobles o simples:

```
"John Doe"
```

```
'John Doe'
```

Inténtalo tú mismo "

Variables JavaScript

En un lenguaje de programación, las variables se utilizan para almacenar valores de datos.

JavaScript utiliza la palabra clave var para declarar variables.

Un signo igual se utiliza para asignar valores a las variables.

En este ejemplo, x se define como una variable. Entonces, x se asigna (dado) el valor 6:

```
var x;
```

```
x = 6;
```

Inténtalo tú mismo "

Operadores JavaScript

JavaScript utiliza operadores aritméticos (+ - * /) para calcular valores:

```
(5 + 6) * 10
```

Inténtalo tú mismo "

JavaScript utiliza un operador de asignación (=) para asignar valores a variables:

```
var x, y;
x = 5;
y = 6;
```

Inténtalo tú mismo "

JavaScript Expressions

Una expresión es una combinación de valores, variables y operadores, que calcula a un valor.

El cálculo se llama evaluación.

Por ejemplo, 5 * 10 se evalúa a 50:

```
5 * 10
```

Inténtalo tú mismo "

Las expresiones también pueden contener valores de variable:

```
x * 10
```

Inténtalo tú mismo "

Los valores pueden ser de varios tipos, como números y cadenas.

Por ejemplo, "John" + "" + "Doe", se evalúa como "John Doe":

```
"John" + " " + "Doe"
```

Inténtalo tú mismo "

JavaScript Keywords

Las palabras clave JavaScript se usan para identificar las acciones a realizar.

La palabra clave var indica al navegador que cree variables:

```
var x, y;
x = 5 + 6;
y = x * 10;
```

Inténtalo tú mismo "

Comentarios de JavaScript

No todas las sentencias JavaScript son "ejecutadas".

El código después de las barras dobles // o entre / * y * / se trata como un comentario .

Los comentarios se ignoran y no se ejecutarán:

```
var x = 5;   // I will be executed
```

```
// var x = 6;   I will NOT be executed
```

Inténtalo tú mismo "

Aprenderá más sobre los comentarios en un capítulo posterior.

Identificadores de JavaScript

Los identificadores son nombres.

En JavaScript, los identificadores se utilizan para nombrar variables (y palabras clave, funciones y etiquetas).

Las reglas para los nombres legales son muy similares en la mayoría de los lenguajes de programación.

En JavaScript, el primer carácter debe ser una letra, o un guión bajo (_), o un signo de dólar ($).

Los caracteres subsiguientes pueden ser letras, dígitos, subrayados o signos de dólar.

Los números no están permitidos como el primer carácter.
De esta manera, JavaScript puede distinguir fácilmente los identificadores de los números.

JavaScript es sensible a mayúsculas

Todos los identificadores JavaScript distinguen entre mayúsculas y minúsculas .

Las variables lastName y lastname son dos variables diferentes.

```
var lastname, lastName;
lastName = "Doe";
lastname = "Peterson";
```

Inténtalo tú mismo "

JavaScript no interpreta VAR o Var como la palabra clave var .

JavaScript y Camel Case

Históricamente, los programadores han utilizado diferentes maneras de unir múltiples palabras en un nombre de variable:

Guiones

first-name, last-name, master-card, inter-city.

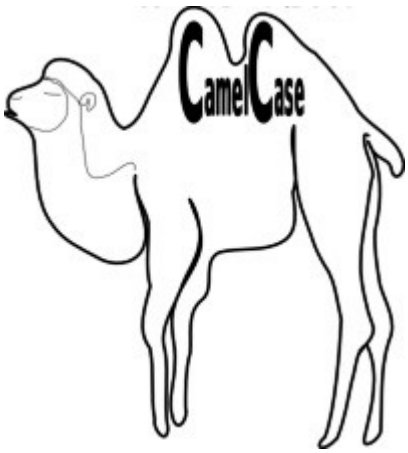
Los guiones no están permitidos en JavaScript. Se reserva para las restas.

Guion bajo:

first_name, last_name, master_card, inter_city.

Upper Camel Case (Pascal Case):

FirstName, LastName, MasterCard, InterCity.



Lower Camel Case:

Los programadores de JavaScript tienden a usar el camel case que comienza con una letra minúscula:

firstName, lastName, masterCard, interCity.

Juego de caracteres JavaScript

JavaScript utiliza el conjunto de caracteres Unicode .

Unicode cubre (casi) todos los caracteres, signos de puntuación y símbolos del mundo.

Para una mirada más cercana, estudie por favor nuestra referencia completa de Unicode .

7. Sentencias JavaScript

En HTML, las sentencias JavaScript son "instrucciones" para ser "ejecutadas" por el navegador web.

Sentencias JavaScript

Esta declaración le dice al navegador que escriba "Hello Dolly" dentro de un elemento HTML con id = "demo":

Ejemplo

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

Inténtalo tú mismo "

Programas JavaScript

La mayoría de los programas JavaScript contienen muchas sentencias JavaScript.

Las declaraciones se ejecutan, una por una, en el mismo orden en que se escriben.

En este ejemplo x, y, y z se dan valores, y finalmente se muestra z:

Ejemplo

```
var x, y, z;
x = 5;
y = 6;
z = x + y;
document.getElementById("demo").innerHTML = z;
```

Inténtalo tú mismo "

Los programas JavaScript (y las sentencias JavaScript) se llaman a menudo código JavaScript.

Punto y coma;

Las comas y comillas separan las declaraciones JavaScript.

Agregue un punto y coma al final de cada sentencia ejecutable:

```
var a, b, c;
a = 5;
b = 6;
c = a + b;
```

Inténtalo tú mismo "

Cuando se separan por punto y coma, se permiten varias sentencias en una línea:

```
a = 5; b = 6; c = a + b;
```

Inténtalo tú mismo "

En la web, puede ver ejemplos sin punto y coma.
Terminar las declaraciones con punto y coma no es necesario, pero muy recomendable.

Espacios en blanco en JavaScript

JavaScript ignora los espacios en blanco. Puede agregar espacios en blanco a su script para hacerlo más legible.

Las líneas siguientes son equivalentes:

```
var person = "Hege";
var person="Hege";
```

Una buena práctica es poner espacios alrededor de los operadores (= + - * /):

```
var x = y + z;
```

Longitud de línea JavaScript y salto de línea

Para una mejor legibilidad, a los programadores a menudo les gusta evitar líneas de código de más de 80 caracteres.

Si una declaración de JavaScript no encaja en una línea, el mejor lugar para romperla, es después de un operador:

Ejemplo

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

Inténtalo tú mismo "

Bloques de código JavaScript

Las sentencias de JavaScript se pueden agrupar en bloques de código, dentro de llaves {...}.

El propósito de los bloques de código es definir las sentencias que se deben ejecutar conjuntamente.

Un lugar en el que encontrará las declaraciones agrupadas en bloques, está en las funciones JavaScript:

Ejemplo

```
function myFunction() {
    document.getElementById("demo1").innerHTML = "Hello Dolly!";
    document.getElementById("demo2").innerHTML = "How are you?";
}
```

Inténtalo tú mismo "

En este tutorial utilizamos 4 espacios de sangría para bloques de código.
Aprenderá más sobre las funciones más adelante en este tutorial.

JavaScript Keywords

Las sentencias JavaScript suelen comenzar con una palabra clave para identificar la acción JavaScript que se va a realizar.

Aquí hay una lista de algunas de las palabras clave que aprenderá en este tutorial:

Palabra clave	Descripción
break	Termina un conmutador o un bucle
continue	Salta de un bucle y comienza en la parte superior
debugger	Detiene la ejecución de JavaScript y llama (si está disponible) la función de depuración
do ... while	Ejecuta un bloque de sentencias, y repite el bloque, mientras que una condición es verdadera
for	Marca un bloque de sentencias a ejecutar, siempre y cuando una condición sea verdadera
function	Declara una función
if ... else	Marca un bloque de sentencias a ejecutar, dependiendo de una condición
return	Sale de una función
switch	Marca un bloque de estados a ejecutar, dependiendo de los diferentes casos
try ... catch	Implementa el manejo de errores en un bloque de sentencias
var	Declara una variable

> Las palabras clave JavaScript son palabras reservadas. Las palabras reservadas no se pueden utilizar como nombres para las variables.

Comentarios de JavaScript

Los comentarios JavaScript se pueden utilizar para explicar el código JavaScript y para hacerlo más legible.

Los comentarios de JavaScript también se pueden utilizar para evitar la ejecución, al probar el código alternativo.

Comentarios de línea única

Los comentarios de una sola línea comienzan con //.

Cualquier texto entre // y el final de la línea será ignorado por JavaScript (no se ejecutará).

Este ejemplo utiliza un comentario de una sola línea antes de cada línea de código:

Ejemplo

```
// Change heading:
document.getElementById("myH").innerHTML = "My First Page";
// Change paragraph:
document.getElementById("myP").innerHTML = "My first paragraph.";
```

Inténtalo tú mismo "

Este ejemplo utiliza un comentario de una sola línea al final de cada línea para explicar el código:

Ejemplo

```
var x = 5;      // Declare x, give it the value of 5
var y = x + 2;  // Declare y, give it the value of x + 2
```

Inténtalo tú mismo "

Comentarios de varias líneas

Los comentarios de varias líneas comienzan con / * y terminan con * /.

Cualquier texto entre / * y * / será ignorado por JavaScript.

Este ejemplo utiliza un comentario de varias líneas (un bloque de comentario) para explicar el código:

Ejemplo

```
/*
The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
in my web page:
*/
document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

Inténtalo tú mismo "

Es más común usar comentarios de una sola línea.
Los comentarios de bloque se usan a menudo para la documentación formal.

Uso de comentarios para evitar la ejecución

El uso de comentarios para evitar la ejecución de código es adecuado para las pruebas de código.

Agregar // delante de una línea de código cambia las líneas de código de una línea ejecutable a un comentario.

Este ejemplo utiliza // para evitar la ejecución de una de las líneas de código:

Ejemplo

```
//document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

Inténtalo tú mismo "

Este ejemplo utiliza un bloque de comentario para evitar la ejecución de varias líneas:

Ejemplo

```
/*
document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
*/
```

Inténtalo tú mismo "

8. Variables JavaScript

Variables JavaScript

Las variables JavaScript son contenedores para almacenar valores de datos.

En este ejemplo, x, y, y z, son variables:

Ejemplo

```
var x = 5;
var y = 6;
var z = x + y;
```

Inténtalo tú mismo "

En el ejemplo anterior, puede esperar:

- x almacena el valor 5
- y almacena el valor 6
- z almacena el valor 11

Al igual que el álgebra

En este ejemplo, price1, price2 y total, son variables:

Ejemplo

```
var price1 = 5;
var price2 = 6;
var total = price1 + price2;
```

Inténtalo tú mismo "

En la programación, al igual que en álgebra, usamos variables (como price1) para contener valores.

En la programación, al igual que en álgebra, usamos variables en expresiones (total = price1 + price2).

En el ejemplo anterior, puede calcular el total a 11.

Las variables JavaScript son contenedores para almacenar valores de datos.

Identificadores de JavaScript

Todas las variables JavaScript deben identificarse con nombres únicos .

Estos nombres únicos se llaman identificadores .

Los identificadores pueden ser nombres cortos (como xey) o más nombres descriptivos (edad, suma, volumen total).

Las reglas generales para construir nombres para variables (identificadores únicos) son:

- Los nombres pueden contener letras, dígitos, subrayados y signos de dólar.
- Los nombres deben comenzar con una letra
- Los nombres también pueden comenzar con $ y _ (pero no lo usaremos en este tutorial)
- Los nombres son sensibles a mayúsculas y minúsculas (y y Y son diferentes variables)
- Las palabras reservadas (como las palabras clave JavaScript) no se pueden utilizar como nombres

Los identificadores de JavaScript distinguen entre mayúsculas y minúsculas.

El operador de asignación

En JavaScript, el signo de igualdad (=) es un operador de "asignación", no un operador "igual a".

Esto es diferente del álgebra. Lo siguiente no tiene sentido en álgebra:

```
x = x + 5
```

En JavaScript, sin embargo, tiene mucho sentido: asigna el valor de x + 5 a x.

(Calcula el valor de x + 5 y pone el resultado en x. El valor de x se incrementa en 5.)

El operador "igual a" se escribe como == en JavaScript.

Tipos de datos JavaScript

Las variables JavaScript pueden contener números como 100 y valores de texto como "John Doe".

En la programación, los valores de texto se llaman cadenas de texto.

JavaScript puede manejar muchos tipos de datos, pero por ahora, solo piensa en números y cadenas.

Las cadenas se escriben dentro de comillas dobles o simples. Los números se escriben sin comillas.

Si coloca un número entre comillas, se tratará como una cadena de texto.

Ejemplo

```
var pi = 3.14;
var person = "John Doe";
var answer = 'Yes I am!';
```

Inténtalo tú mismo "

Declaración (creación) de variables JavaScript

La creación de una variable en JavaScript se denomina "declarar" una variable.

Usted declara una variable JavaScript con la palabra clave **var** :

```
var carName;
```

Después de la declaración, la variable no tiene valor. (Técnicamente tiene el valor de undefined)

Para asignar un valor a la variable, utilice el signo igual:

```
carName = "Volvo";
```

También puede asignar un valor a la variable cuando lo declara:

```
var carName = "Volvo";
```

En el ejemplo siguiente, creamos una variable llamada carName y le asignamos el valor "Volvo".

Entonces "salida" el valor dentro de un párrafo HTML con id = "demo":

Ejemplo

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

<script>
var carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
</script>
```

Inténtalo tú mismo "

Es una buena práctica de programación declarar todas las variables al principio de un guión (script).

Una declaración, muchas variables

Puede declarar muchas variables en una sentencia.

Inicie la instrucción con **var** y separe las variables por coma :

```
var person = "John Doe", carName = "Volvo", price = 200;
```

Inténtalo tú mismo "

Una declaración puede abarcar varias líneas:

```
var person = "John Doe",
carName = "Volvo",
price = 200;
```

Inténtalo tú mismo "

Valor = undefined

En los programas informáticos, las variables suelen declararse sin valor. El valor puede ser algo que tiene que ser calculado, o algo que se proporcionará más adelante, como la entrada del usuario.

Una variable declarada sin valor tendrá el valor **undefined** .

La variable carName tendrá el valor undefined después de la ejecución de esta sentencia:

Ejemplo

```
var carName;
```

Inténtalo tú mismo "

Volver a declarar variables JavaScript

Si vuelve a declarar una variable de JavaScript, no perderá su valor.

La variable carName seguirá teniendo el valor "Volvo" después de la ejecución de estas declaraciones:

Ejemplo

```
var carName = "Volvo";
var carName;
```

Inténtalo tú mismo "

Aritmética JavaScript

Al igual que con álgebra, puede hacer aritmética con variables JavaScript, utilizando operadores como = y +:

Ejemplo

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

Inténtalo tú mismo "

También puede agregar cadenas, pero las cadenas se concatenarán:

Ejemplo

```
var x = "John" + " " + "Doe";
```

Inténtalo tú mismo "

También prueba esto:

Ejemplo

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

Inténtalo tú mismo "

Si coloca un número entre comillas, el resto de los números serán tratados como cadenas y concatenados.

Ahora prueba esto:

Ejemplo

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

Inténtalo tú mismo "

Pruebe usted mismo con ejercicios!

Ejercicio 1 » Ejercicio 2» Ejercicio 3 » Ejercicio 4» Ejercicio 5 » Ejercicio 6»

9. Operadores JavaScript

Ejemplo

Asigne valores a variables y agréguelos:

```
var x = 5;        // assign the value 5 to x
var y = 2;        // assign the value 2 to y
var z = x + y;    // assign the value 7 to z (x + y)
```

Inténtalo tú mismo "

El operador de asignación (=) asigna un valor a una variable.

Asignación

```
var x = 10;
```

Inténtalo tú mismo "

El operador de adición (+) añade números:

Añadiendo

```
var x = 5;
var y = 2;
var z = x + y;
```

Inténtalo tú mismo "

El operador de multiplicación (*) multiplica los números.

Multiplicando

```
var x = 5;
var y = 2;
var z = x * y;
```

Inténtalo tú mismo "

Operadores aritméticos JavaScript

Los operadores aritméticos se utilizan para realizar la aritmética en los números:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

Los operadores aritméticos se describen completamente en el capítulo JS Arithmetic .

Operadores de asignación de JavaScript

Los operadores de asignación asignan valores a variables JavaScript.

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

El operador de asignación de adición (+ =) añade un valor a una variable.

Asignación

```
var x = 10;
x += 5;
```

Inténtalo tú mismo "

Los operadores de asignación se describen completamente en el capítulo Asignación JS .

Operadores de String JavaScript

El operador + también se puede utilizar para agregar cadenas (concatenar).

Ejemplo

```
txt1 = "John";
txt2 = "Doe";
txt3 = txt1 + " " + txt2;
```

El resultado de txt3 será:

```
John Doe
```

Inténtalo tú mismo "

El operador de asignación + = también se puede utilizar para agregar cadenas (concatenar):

Ejemplo

```
txt1 = "What a very ";
txt1 += "nice day";
```

El resultado de txt1 será:

```
What a very nice day
```

Inténtalo tú mismo "

Cuando se utiliza en cadenas, el operador + se llama el operador de concatenación.

Añadir cadenas y números

Añadiendo dos números, devolverá la suma, pero agregando un número y una cadena devolverá una cadena:

Ejemplo

```
x = 5 + 5;
y = "5" + 5;
z = "Hello" + 5;
```

El resultado de x , y , yz será:

```
10
55
Hello5
```

Inténtalo tú mismo "

Si agrega un número y una cadena, el resultado será una cadena!

Operadores de comparación de JavaScript

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Los operadores de comparación se describen completamente en el capítulo Comparaciones de JS .

Operadores lógicos JavaScript

Operator	Description
&&	logical and
||	logical or
!	logical not

Los operadores lógicos se describen completamente en el capítulo Comparaciones de JS .

Operadores de tipo JavaScript

Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

Los operadores de tipo se describen completamente en el capítulo Conversión de tipo JS .

Operadores Bitwise de JavaScript

Los operadores de bit trabajan en números de 32 bits.

Cualquier operando numérico en la operación se convierte en un número de 32 bits. El resultado se convierte de nuevo a un número JavaScript.

Operator	Description	Example	Same as	Result	Decimal
&	AND	5 & 1	0101 & 0001	0001	1
|	OR	5 | 1	0101 | 0001	0101	5
~	NOT	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	0100	4
<<	Zero fill left shift	5 << 1	0101 << 1	1010	10
>>	Signed right shift	5 >> 1	0101 >> 1	0010	2
>>>	Zero fill right shift	5 >>> 1	0101 >>> 1	0010	2

Los ejemplos anteriores usan ejemplos sin signo de 4 bits. Pero JavaScript utiliza números firmados de 32 bits.

Debido a esto, en JavaScript, ~ 5 no volverá 10. Volverá -6.

~ 00000000000000000000000000000101 devolverá 11111111111111111111111111111010

Los operadores de bits se describen completamente en el capítulo JS Bitwise .

10. Comparación de JavaScript y operadores lógicos

Comparación y Operadores lógicos se utilizan para probar para verdadero o falso .

Operadores de comparación

Los operadores de comparación se utilizan en sentencias lógicas para determinar la igualdad o diferencia entre variables o valores.

Dado que **x = 5** , la tabla siguiente explica los operadores de comparación:

Operator	Description	Comparing	Returns	Try it
==	equal to	x == 8	false	Try it »
		x == 5	true	Try it »
		x == "5"	true	Try it »
===	equal value and equal type	x === 5	true	Try it »
		x === "5"	false	Try it »
!=	not equal	x != 8	true	Try it »
!==	not equal value or not equal type	x !== 5	false	Try it »
		x !== "5"	true	Try it »
		x !== 8	true	Try it »
>	greater than	x > 8	false	Try it »
<	less than	x < 8	true	Try it »
>=	greater than or equal to	x >= 8	false	Try it »
<=	less than or equal to	x <= 8	true	Try it »

Como puede ser usado

Los operadores de comparación se pueden utilizar en declaraciones condicionales para comparar valores y tomar medidas dependiendo del resultado:

```
if (age < 18) text = "Too young";
```

Operadores logicos

Los operadores lógicos se utilizan para determinar la lógica entre variables o valores.

Dado que **x = 6** y **y = 3** , la tabla de abajo se explican los operadores lógicos:

Operator	Description	Example	Try it
&&	and	(x < 10 && y > 1) is true	Try it »
||	or	(x == 5 || y == 5) is false	Try it »
!	not	!(x == y) is true	Try it »

Operador condicional (ternario)

JavaScript también contiene un operador condicional que asigna un valor a una variable basada en alguna condición.

Sintaxis

```
variablename = (condition) ? value1:value2
```

Ejemplo

```
var voteable = (age < 18) ? "Too young":"Old enough";
```

Inténtalo tú mismo "

Si la edad variable es un valor por debajo de 18, el valor de la variable que se vota será "Demasiado joven", de lo contrario el valor de votante será "Antiguo".

Comparación de diferentes tipos

La comparación de datos de diferentes tipos puede dar resultados inesperados.

Al comparar una cadena con un número, JavaScript convertirá la cadena en un número al hacer la comparación. Una cadena vacía se convierte en 0. Una cadena no numérica se convierte en NaN que siempre es falsa.

Case	Value	Try
2 < 12	true	Try it »
2 < "12"	true	Try it »
2 < "John"	false	Try it »
2 > "John"	false	Try it »
2 == "John"	false	Try it »
"2" < "12"	false	Try it »
"2" > "12"	true	Try it »
"2" == "12"	false	Try it »

Cuando se comparan dos cadenas, "2" será mayor que "12", porque (alfabéticamente) 1 es menor que 2.

Para asegurar un resultado correcto, las variables deben ser convertidas al tipo apropiado antes de la comparación:

```
age = Number(age);
if (isNaN(age)) {
    voteable = "Input is not a number";
} else {
    voteable = (age < 18) ? "Too young" : "Old enough";
}
```

Inténtalo tú mismo "

Pruebe usted mismo con ejercicios!

Ejercicio 1 » Ejercicio 2» Ejercicio 3 » Ejercicio 4» Ejercicio 5 » Ejercicio 6»

11. Aritmética JavaScript

Una cosa típica de los números es la aritmética.

Operadores aritméticos JavaScript

Los operadores aritméticos realizan la aritmética en números (literales o variables).

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

Operaciones aritmeticas

Una operación aritmética típica opera en dos números.

Los dos números pueden ser literales:

Ejemplo

```
var x = 100 + 50;
```

Inténtalo tú mismo "

o variables:

Ejemplo

```
var x = a + b;
```

Inténtalo tú mismo "

o expresiones:

Ejemplo

```
var x = (100 + 50) * a;
```

Inténtalo tú mismo "

Operadores y Operandos

Los números (en una operación aritmética) se llaman operandos .

La operación (a realizar entre los dos operandos) es definida por un operador .

Operando	Operador	Operando
100	+	50

El operador de adición (+) añade números:

Añadiendo

```
var x = 5;
var y = 2;
var z = x + y;
```

Inténtalo tú mismo "

El operador de substracción (-) resta números.

Restando

```
var x = 5;
var y = 2;
var z = x - y;
```

Inténtalo tú mismo "

El operador de multiplicación (*) multiplica los números.

Multiplicando

```
var x = 5;
var y = 2;
var z = x * y;
```

Inténtalo tú mismo "

El operador de división (/) divide números.

Divisor

```
var x = 5;
var y = 2;
var z = x / y;
```

Inténtalo tú mismo "

El operador modular (%) devuelve el resto de división.

Módulo

```
var x = 5;
var y = 2;
var z = x % y;
```

Inténtalo tú mismo "

El operador de incremento (++) incrementa los números.

Incrementando

```
var x = 5;
x++;
var z = x;
```

Inténtalo tú mismo "

El operador decremento (-) decrementa números.

Descenso

```
var x = 5;
x--;
var z = x;
```

Inténtalo tú mismo "

Precedencia del operador

La precedencia del operador describe el orden en el que se realizan las operaciones en una expresión aritmética.

Ejemplo

```
var x = 100 + 50 * 3;
```

Inténtalo tú mismo "

¿Es el resultado del ejemplo anterior el mismo que 150 * 3, o es el mismo que 100 + 150?

¿Es la adición o la multiplicación hecha primero?

Como en las matemáticas tradicionales de la escuela, la multiplicación se hace primero.

La multiplicación (*) y la división (/) tienen mayor precedencia que la adición (+) y la substracción (-).

Y (como en las matemáticas de la escuela) la precedencia se puede cambiar usando paréntesis:

Ejemplo

```
var x = (100 + 50) * 3;
```

Inténtalo tú mismo "

Cuando se utilizan paréntesis, las operaciones dentro de los paréntesis se calculan primero.

Cuando muchas operaciones tienen la misma precedencia (como suma y resta), se calculan de izquierda a derecha:

Ejemplo

```
var x = 100 + 50 - 3;
```

Inténtalo tú mismo "

Valores de precedencia del operador JavaScript

Valor	Operador	Descripción	Ejemplo
19	()	Agrupación de expresiones	(3 + 4)
18	.	Miembro	nombre de persona
18	[]	Miembro	persona ["nombre"]
17	()	Llamada de función	myFunction ()
17	nuevo	Crear	nueva fecha
dieciséis	++	Incremento de Postfix	i ++
dieciséis	-	Decremento de Postfix	yo--
15	++	Incremento de prefijo	yo os
15	-	Decremento del prefijo	--yo
15	!	Lógico no	x
15	tipo de	Tipo	tipo de x
14	*	Multiplicación	10 * 5
14	/	División	10/5
14	%	División Modulo	10% 5
14	**	Exponenciación	10 ** 2
13	+	Adición	10 + 5
13	-	Sustracción	10 - 5
12	<<	Desplazar hacia la izquierda	x << 2
12	>>	Cambiar a la derecha	x >> 2
12	>>>	Desplazar a la derecha (sin firmar)	x >>> 2
11	<	Menos que	x <y
11	<=	Menor o igual	x <= y
11	>	Mas grande que	xie
11	> =	Mayor que o igual	x> = y
10	==	Igual	x == y
10	===	Estricta igualdad	x
10	¡Bienvenido!	Desigual	xi
10	Unesdoc.unesco.org	Estricto desigual	xi
6	&&	Lógico y	xie
5	||	Lógico o	x || y
3	=	Asignación	x = y
3	+ =	Asignación	x + = y
3	-benzóico.	Asignación	x = y
3	* =	Asignación	x * = y
3	% =	Asignación	x% = y

3	<< =	Asignación	x << = y
3	>> =	Asignación	xi
3	>>> =	Asignación	xie
3	& =	Asignación	xy = y
3	^ =	Asignación	x ^ = y
3	| =	Asignación	x | = y

Las entradas de color rojo pálido indican tecnología experimental o propuesta (ECMAScript 2016 o ES7)

Las expresiones entre paréntesis se calculan completamente antes de utilizar el valor en el resto de la expresión.

Pruebe usted mismo con ejercicios!

Ejercicio 1 » Ejercicio 2» Ejercicio 3 » Ejercicio 4» Ejercicio 5 »

12. Asignación de JavaScript

Operadores de asignación de JavaScript

Los operadores de asignación asignan valores a variables JavaScript.

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
<<=	x <<= y	x = x << y
>>=	x >>= y	x = x >> y
>>>=	x >>>= y	x = x >>> y
&=	x &= y	x = x & y
^=	x ^= y	x = x ^ y
|=	x |= y	x = x | y
**=	x **= y	x = x ** y

> El operador ** = es una parte experimental de la propuesta ECMAScript 2016 (ES7). No es estable en todos los navegadores. No lo uses.

Ejemplos de asignación

El operador = assign asigna un valor a una variable.

Asignación

```
var x = 10;
```

Inténtalo tú mismo "

El operador de asignación + = agrega un valor a una variable.

Asignación

```
var x = 10;
x += 5;
```

Inténtalo tú mismo "

El operador - = asignación resta un valor de una variable.

Asignación

```
var x = 10;
x -= 5;
```

Inténtalo tú mismo "

El operador * = assignment multiplica una variable.

Asignación

```
var x = 10;
x *= 5;
```

Inténtalo tú mismo "

La asignación / = divide una variable.

Asignación

```
var x = 10;
x /= 5;
```

Inténtalo tú mismo "

El operador% = asignación asigna un resto a una variable.

Asignación

```
var x = 10;
x %= 5;
```

Inténtalo tú mismo "

Pruebe usted mismo con ejercicios!

Ejercicio 1 » Ejercicio 2» Ejercicio 3 » Ejercicio 4» Ejercicio 5 »

13. Tipos de datos JavaScript

Tipos de datos JavaScript

Las variables JavaScript pueden contener muchos tipos de datos : números, cadenas, objetos y más:

```
var length = 16;                               // Number
var lastName = "Johnson";                      // String
var x = {firstName:"John", lastName:"Doe"};    // Object
```

El concepto de tipos de datos

En la programación, los tipos de datos son un concepto importante.

Para poder operar sobre variables, es importante saber algo sobre el tipo.

Sin tipos de datos, una computadora no puede solucionar esto de forma segura:

```
var x = 16 + "Volvo";
```

¿Tiene sentido añadir "Volvo" a dieciséis? ¿Producirá un error o producirá un resultado?

JavaScript tratará el ejemplo anterior como:

```
var x = "16" + "Volvo";
```

Al agregar un número y una cadena, JavaScript tratará el número como una cadena.

Ejemplo

```
var x = 16 + "Volvo";
```

Inténtalo tú mismo "

Ejemplo

```
var x = "Volvo" + 16;
```

Inténtalo tú mismo "

JavaScript evalúa las expresiones de izquierda a derecha. Diferentes secuencias pueden producir resultados diferentes:

JavaScript:

```
var x = 16 + 4 + "Volvo";
```

Resultado:

```
20Volvo
```

Inténtalo tú mismo "

JavaScript:

```
var x = "Volvo" + 16 + 4;
```

Resultado:

```
Volvo164
```

Inténtalo tú mismo "

En el primer ejemplo, JavaScript trata 16 y 4 como números, hasta que llegue a "Volvo".

En el segundo ejemplo, dado que el primer operando es una cadena, todos los operandos son tratados como cadenas.

Los tipos JavaScript son dinámicos.

JavaScript tiene tipos dinámicos. Esto significa que la misma variable se puede utilizar para mantener diferentes tipos de datos:

Ejemplo

```
var x;               // Now x is undefined
var x = 5;           // Now x is a Number
var x = "John";      // Now x is a String
```

Inténtalo tú mismo "

Cadenas en JavaScript

Una cadena (o una cadena de texto) es una serie de caracteres como "John Doe".

Las cadenas están escritas con comillas. Puede usar comillas simples o dobles:

Ejemplo

```
var carName = "Volvo XC60";   // Using double quotes
var carName = 'Volvo XC60';   // Using single quotes
```

Inténtalo tú mismo "

Puede utilizar comillas dentro de una cadena, siempre y cuando no coincidan con las comillas que rodean la cadena:

Ejemplo

```
var answer = "It's alright";            // Single quote inside double quotes
var answer = "He is called 'Johnny'";   // Single quotes inside double quotes
var answer = 'He is called "Johnny"';   // Double quotes inside single quotes
```

Inténtalo tú mismo "

Aprenderá más sobre las cadenas más adelante en este tutorial.

Números de JavaScript

JavaScript tiene sólo un tipo de números.

Los números se pueden escribir con, o sin decimales:

Ejemplo

```
var x1 = 34.00;     // Written with decimals
var x2 = 34;        // Written without decimals
```

Inténtalo tú mismo "

Los números extra grandes o extra pequeños se pueden escribir con la notación científica (exponencial):

Ejemplo

```
var y = 123e5;      // 12300000
var z = 123e-5;     // 0.00123
```

Inténtalo tú mismo "

Aprenderá más sobre los números más adelante en este tutorial.

Booleanos JavaScript

Los booleanos sólo pueden tener dos valores: verdadero o falso.

Ejemplo

```
var x = 5;
var y = 5;
var z = 6;
(x == y)       // Returns true
(x == z)       // Returns false
```

Inténtalo tú mismo "

Los booleanos se usan a menudo en las pruebas condicionales.

Más adelante aprenderá más sobre las pruebas condicionales en este tutorial.

Arrays JavaScript

Los arrays de JavaScript se escriben con corchetes.

Los elementos de la matriz están separados por comas.

El siguiente código declara (crea) una matriz llamada cars, que contiene tres elementos (nombres de automóviles):

Ejemplo

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

Inténtalo tú mismo "

Los índices de matriz son basados en cero, lo que significa que el primer elemento es [0], el segundo es [1], y así sucesivamente.

Objetos JavaScript

Los objetos JavaScript se escriben con llaves.

Las propiedades del objeto se escriben como pares nombre: valor, separados por comas.

Ejemplo

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Inténtalo tú mismo "

El objeto (persona) en el ejemplo anterior tiene 4 propiedades: firstName, lastName, age y eyeColor.

Aprenderá más sobre los objetos más adelante en este tutorial.

El operador typeof

Puede utilizar el operador JavaScript typeof para encontrar el tipo de una variable de JavaScript.

El operador typeof devuelve el tipo de una variable o una expresión:

Ejemplo

```
typeof ""              // Returns "string"
typeof "John"            // Returns "string"
typeof "John Doe"          // Returns "string"
```

Inténtalo tú mismo "

Ejemplo

```
typeof 0               // Returns "number"
typeof 314               // Returns "number"
typeof 3.14              // Returns "number"
typeof (3)             // Returns "number"
typeof (3 + 4)           // Returns "number"
```

Inténtalo tú mismo "

Undefinded

En JavaScript, una variable sin un valor, tiene el valor undefined . El typeof es también indefinido .

Ejemplo

```
var car;            // Value is undefined, type is undefined
```

Inténtalo tú mismo "

Cualquier variable se puede vaciar, estableciendo el valor en indefinido . El tipo también será indefinido .

Ejemplo

```
car = undefined;       // Value is undefined, type is undefined
```

Inténtalo tú mismo "

Valores vacíos

Un valor vacío no tiene nada que ver con indefinido.

Una cadena vacía tiene un valor legal y un tipo.

Ejemplo

```
var car = "";           // The value is "", the typeof is "string"
```

Inténtalo tú mismo "

Null

En JavaScript, null es "nada". Se supone que es algo que no existe.

Desafortunadamente, en JavaScript, el tipo de datos de null es un objeto.

> Puedes considerarlo un error en JavaScript que typeof null es un objeto. Debe ser nulo.

Puede vaciar un objeto estableciéndolo en null:

Ejemplo

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
person = null;       // Now value is null, but type is still an object
```

Inténtalo tú mismo "

También puede vaciar un objeto estableciéndolo en undefined:

Ejemplo

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
person = undefined;   // Now both value and type is undefined
```

Inténtalo tú mismo "

Diferencia entre undefined y null

Undefined y null son iguales en valor pero diferentes en tipo:

```
typeof undefined          // undefined
typeof null               // object

null === undefined        // false
null == undefined         // true
```

Inténtalo tú mismo "

Datos primitivos

Un valor de datos primitivo es un único valor de datos simple sin propiedades y métodos adicionales.

El operador typeof puede devolver uno de estos tipos primitivos:

- cuerda
- número
- booleano
- indefinido

Ejemplo

```
typeof "John"              // Returns "string"
typeof 3.14                // Returns "number"
typeof true                // Returns "boolean"
typeof false               // Returns "boolean"
typeof x                   // Returns "undefined" (if x has no value)
```

Inténtalo tú mismo "

Datos complejos

El operador typeof puede devolver uno de dos tipos complejos:

- función
- objeto

El operador typeof devuelve el objeto para ambos objetos, arrays y null.

El operador typeof no devuelve objeto para funciones.

Ejemplo

```
typeof {name:'John', age:34} // Returns "object"
typeof [1,2,3,4]             // Returns "object" (not "array", see note below)
typeof null                  // Returns "object"
typeof function myFunc(){}   // Returns "function"
```

Inténtalo tú mismo "

El operador typeof devuelve "objeto" para matrices porque en arrays JavaScript son objetos.

14. Sentencias JavaScript If...Else

Las sentencias condicionales se utilizan para realizar diferentes acciones basadas en diferentes condiciones.

Sentencias condicionales

Muy a menudo cuando se escribe código, se desea realizar diferentes acciones para diferentes decisiones.

Puede utilizar declaraciones condicionales en su código para hacer esto.

En JavaScript tenemos las siguientes declaraciones condicionales:

- Utilice **si** para especificar un bloque de código que se ejecutará, si una condición especificada es verdadera
- Use **else** para especificar un bloque de código que se ejecutará, si la misma condición es falsa
- Use **else si** para especificar una nueva condición para probar, si la primera condición es falsa
- Utilice el **interruptor** para especificar muchos bloques alternativos de código que se ejecutarán

---

La sentencia if

Utilice la instrucción if para especificar un bloque de código JavaScript que se ejecutará si una condición es verdadera.

Sintaxis

```
if (condition) {
    block of code to be executed if the condition is true
}
```

Tenga en cuenta que si está en minúsculas. Las letras mayúsculas (If o IF) generarán un error de JavaScript.

Ejemplo

Hacer un saludo de "Buen día" si la hora es menor que las 18:00:

```
if (hour < 18) {
    greeting = "Good day";
}
```

El resultado del saludo será:

```
Good day
```

Inténtalo tú mismo "

La sentencia else

Utilice la instrucción else para especificar un bloque de código que se ejecutará si la condición es falsa.

```
if (condition) {
    block of code to be executed if the condition is true
} else {
    block of code to be executed if the condition is false
}
```

Ejemplo

Si la hora es menor de 18, crear un saludo "Buen día", de lo contrario "Buenas noches":

```
if (hour < 18) {
    greeting = "Good day";
} else {
    greeting = "Good evening";
}
```

El resultado del saludo será:

```
Good day
```

Inténtalo tú mismo "

La sentencia else if

Utilice la instrucción else if para especificar una nueva condición si la primera condición es falsa.

Sintaxis

```
if (condition1) {
    block of code to be executed if condition1 is true
} else if (condition2) {
    block of code to be executed if the condition1 is false and condition2 is
true
} else {
    block of code to be executed if the condition1 is false and condition2 is
false
}
```

Ejemplo

Si el tiempo es menos de 10:00, crear un saludo "Buenos días", si no, pero el tiempo es menos de 20:00, crear un saludo "Buen día", de lo contrario una "Buenas noches":

```
if (time < 10) {
    greeting = "Good morning";
} else if (time < 20) {
```

```
    greeting = "Good day";
} else {
    greeting = "Good evening";
}
```

El resultado del saludo será:

```
Good day
```

Inténtalo tú mismo "

Más ejemplos

Enlace aleatorio
Este ejemplo escribirá un enlace a W3Schools oa World Wildlife Foundation (WWF). Mediante el uso de un número aleatorio, hay un 50% de posibilidades para cada uno de los enlaces.

Pruebe usted mismo con ejercicios!

Ejercicio 1 » Ejercicio 2» Ejercicio 3 » Ejercicio 4» Ejercicio 5 » Ejercicio 6»

15. Sentencia Switch de JavaScript

La instrucción switch se utiliza para realizar diferentes acciones basadas en diferentes condiciones.

La instrucción Switch de JavaScript

Utilice la instrucción switch para seleccionar uno de los muchos bloques de código que se van a ejecutar.

Sintaxis

```
switch(expression) {
  case n:
    code block
    break;
  case n:
    code block
    break;
  default:
    code block
}
```

Así es como funciona:

- La expresión del conmutador se evalúa una vez.
- El valor de la expresión se compara con los valores de cada caso.
- Si hay una coincidencia, se ejecuta el bloque de código asociado.

Ejemplo

El método getDay () devuelve el día de la semana como un número entre 0 y 6.

(Domingo = 0, Lunes = 1, Martes = 2 ..)

En este ejemplo se utiliza el número del día de la semana para calcular el nombre del día de la semana:

```
switch (new Date().getDay()) {
   case 0:
      day = "Sunday";
      break;
   case 1:
      day = "Monday";
      break;
   case 2:
      day = "Tuesday";
      break;
   case 3:
      day = "Wednesday";
      break;
   case 4:
      day = "Thursday";
      break;
   case 5:
      day = "Friday";
      break;
   case 6:
      day = "Saturday";
}
```

El resultado del día será:

```
Sunday
```

Inténtalo tú mismo "

La palabra clave break

Cuando JavaScript alcanza una palabra clave break , se rompe el bloque de interruptores.

Esto detendrá la ejecución de más pruebas de código y caso dentro del bloque.

Cuando se encuentra una coincidencia, y el trabajo está hecho, es hora de un descanso. No hay necesidad de más pruebas.

> Una ruptura puede ahorrar mucho tiempo de ejecución porque "ignora" la ejecución de todo el resto del código en el bloque de interruptores.

No es necesario romper el último caso en un bloque de interruptores. El bloque se rompe (termina) allí de todos modos.

La palabra clave default

La palabra clave default especifica el código que se ejecutará si no hay coincidencia con ningún caso:

Ejemplo

El método getDay () devuelve el día de la semana como un número entre 0 y 6.

Si hoy no es ni el sábado (6) ni el domingo (0), escriba un mensaje predeterminado:

```
switch (new Date().getDay()) {
  case 6:
    text = "Today is Saturday";
    break;
  case 0:
    text = "Today is Sunday";
    break;
  default:
    text = "Looking forward to the Weekend";
}
```

El resultado del texto será:

```
Today is Sunday
```

Inténtalo tú mismo "

El caso por defecto no tiene que ser el último caso en un bloque de conmutación:

Ejemplo

```
switch (new Date().getDay()) {
  default:
    text = "Looking forward to the Weekend";
    break;
  case 6:
    text = "Today is Saturday";
    break;
  case 0:
    text = "Today is Sunday";
}
```

Inténtalo tú mismo "

Si default no es el último caso en el bloque switch, recuerde terminar el caso por defecto con un break.

Bloques de código comunes

A veces querrás que diferentes casos switch utilicen el mismo código.

En este caso de ejemplo 4 y 5 comparten el mismo bloque de código, y 0 y 6 comparten otro bloque de código:

Ejemplo

```
switch (new Date().getDay()) {
  case 4:
  case 5:
    text = "Soon it is Weekend";
    break;
  case 0:
  case 6:
    text = "It is Weekend";
    break;
  default:
    text = "Looking forward to the Weekend";
}
```

Inténtalo tú mismo "

Pruebe usted mismo con ejercicios!

Ejercicio 1 » Ejercicio 2» Ejercicio 3 » Ejercicio 4»

16. El bucle For de JavaScript

Los bucles pueden ejecutar un bloque de código un número de veces.

Los bucles de JavaScript

Loops son útiles, si desea ejecutar el mismo código una y otra vez, cada vez con un valor diferente.

A menudo este es el caso cuando se trabaja con matrices:

En lugar de escribir:

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
text += cars[4] + "<br>";
text += cars[5] + "<br>";
```

Puedes escribir:

```
for (i = 0; i < cars.length; i++) {
    text += cars[i] + "<br>";
}
```

Inténtalo tú mismo "

Diferentes tipos de bucles

JavaScript soporta diferentes tipos de bucles:

- for - bucles a través de un bloque de código un número de veces
- **for / in** - bucles a través de las propiedades de un objeto
- **while** - bucles a través de un bloque de código mientras una condición especificada es verdadera
- **do / while** - también bucles a través de un bloque de código mientras una condición especificada es verdadera

El bucle for

El bucle for es a menudo la herramienta que utilizará cuando desee crear un bucle.

El bucle for tiene la siguiente sintaxis:

```
for (statement 1; statement 2; statement 3) {
    code block to be executed
}
```

La sentencia 1 se ejecuta antes de que se inicie el bucle (el bloque de código).

La sentencia 2 define la condición para ejecutar el bucle (el bloque de código).

La sentencia 3 se ejecuta cada vez que se ha ejecutado el bucle (el bloque de código).

Ejemplo

```
for (i = 0; i < 5; i++) {
    text += "The number is " + i + "<br>";
}
```

Inténtalo tú mismo "

En el ejemplo anterior, puede leer:

La sentencia 1 establece una variable antes de que comience el ciclo (var i = 0).

La sentencia 2 define la condición para que se ejecute el bucle (i debe ser menor que 5).

La sentencia 3 aumenta un valor (i ++) cada vez que se ejecuta el bloque de código en el bucle.

Normalmente se utilizará la instrucción 1 para inicializar la variable utilizada en el bucle (i = 0).

Esto no es siempre el caso, JavaScript no importa. La instrucción 1 es opcional.

Puede iniciar muchos valores en la instrucción 1 (separados por coma):

Ejemplo

```
for (i = 0, len = cars.length, text = ""; i < len; i++) {
    text += cars[i] + "<br>";
}
```

Inténtalo tú mismo "

Y puede omitir la sentencia 1 (como cuando los valores se establecen antes de que se inicie el bucle):

Ejemplo

```
var i = 2;
var len = cars.length;
var text = "";
for (; i < len; i++) {
    text += cars[i] + "<br>";
}
```

Inténtalo tú mismo "

Declaración 2

A menudo, la sentencia 2 se utiliza para evaluar la condición de la variable inicial.

Esto no es siempre el caso, JavaScript no importa. La sentencia 2 también es opcional.

Si la sentencia 2 devuelve true, el bucle comenzará de nuevo, si devuelve false, el bucle terminará.

Si omite la sentencia 2, debe proporcionar una interrupción dentro del bucle. De lo contrario, el bucle nunca terminará. Esto bloqueará su navegador. Lea acerca de las pausas en un capítulo posterior de este tutorial.

Declaración 3

A menudo, la sentencia 3 incrementa el valor de la variable inicial.

Esto no siempre es el caso, JavaScript no le importa, y la declaración 3 es opcional.

La sentencia 3 puede hacer algo como incremento negativo (i--), incremento positivo (i = i + 15), o cualquier otra cosa.

La sentencia 3 también se puede omitir (como cuando se incrementan los valores dentro del bucle):

Ejemplo

```
var i = 0;
var len = cars.length;
for (; i < len; ) {
    text += cars[i] + "<br>";
    i++;
}
```

Inténtalo tú mismo "

El bucle For / In

La instrucción JavaScript for / in hace un bucle a través de las propiedades de un objeto:

Ejemplo

```
var person = {fname:"John", lname:"Doe", age:25};

var text = "";
var x;
for (x in person) {
    text += person[x];
}
```

Inténtalo tú mismo "

El bucle while

El bucle while y el bucle do / while se explicarán en el siguiente capítulo.

Pruebe usted mismo con ejercicios!

Ejercicio 1 » Ejercicio 2» Ejercicio 3 » Ejercicio 4» Ejercicio 5 » Ejercicio 6»

17. El bucle While JavaScript

Los bucles pueden ejecutar un bloque de código siempre y cuando una condición especificada sea verdadera.

El bucle while

El bucle while realiza un bucle a través de un bloque de código siempre y cuando una condición especificada sea verdadera.

Sintaxis

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

Ejemplo

En el ejemplo siguiente, el código en el bucle se ejecutará, una y otra vez, siempre y cuando una variable (i) sea inferior a 10:

Ejemplo

```
while (i < 10) {
    text += "The number is " + i;
    i++;
}
```

Inténtalo tú mismo "

Si se olvida de aumentar la variable utilizada en la condición, el bucle nunca finalizará. Esto bloqueará su navegador.

El bucle Do / While

El bucle do / while es una variante del bucle while. Este bucle ejecuta el bloque de código una vez, antes de comprobar si la condición es verdadera, entonces repetirá el bucle siempre y cuando la condición sea verdadera.

Sintaxis

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

Ejemplo

El siguiente ejemplo usa un bucle do / while. El bucle siempre se ejecutará al menos una vez, incluso si la condición es falsa, porque el bloque de código se ejecuta antes de probar la condición:

Ejemplo

```
do {
  text += "The number is " + i;
  i++;
}
while (i < 10);
```

Inténtalo tú mismo "

No se olvide de aumentar la variable utilizada en la condición, de lo contrario el bucle nunca terminará!

Comparando For y While

Si ha leído el capítulo anterior, sobre el bucle for, descubrirá que un bucle while es lo mismo que un bucle for, con la sentencia 1 y la sentencia 3 omitidas.

El bucle de este ejemplo utiliza un bucle for para recopilar los nombres de coches de la matriz de coches:

Ejemplo

```
var cars = ["BMW", "Volvo", "Saab", "Ford"];
var i = 0;
var text = "";

for (;cars[i];) {
    text += cars[i] + "<br>";
    i++;
}
```

Inténtalo tú mismo "

El bucle de este ejemplo utiliza un bucle while para recoger los nombres de coches de la matriz de coches:

Ejemplo

```
var cars = ["BMW", "Volvo", "Saab", "Ford"];
var i = 0;
var text = "";

while (cars[i]) {
    text += cars[i] + "<br>";
    i++;
}
```

Inténtalo tú mismo "

Pruebe usted mismo con ejercicios!

Ejercicio 1 » Ejercicio 2» Ejercicio 3 » Ejercicio 4» Ejercicio 5 »