

JavaScript

Las Bases del Lenguaje

JS



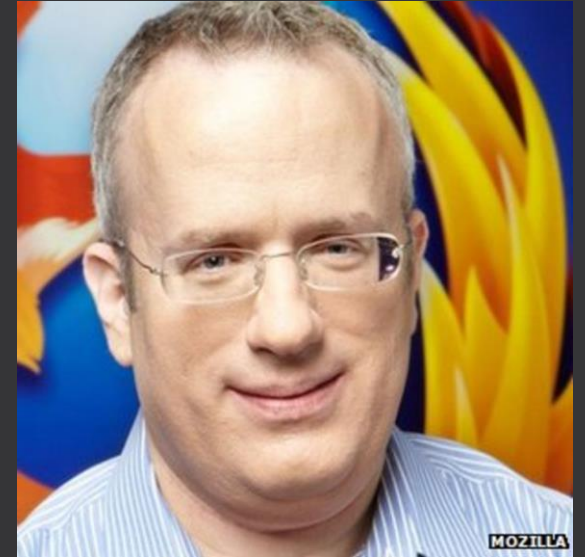
DEFINICION

¿Qué es JavaScript?

- JavaScript (abreviado comúnmente JS) es un lenguaje de programación interpretado, dialecto del estándar ECMAScript.
- Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.
- Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web.
- JavaScript se diseñó con una sintaxis similar a C, aunque adopta nombres y convenciones del lenguaje de programación Java.
- Sin embargo, Java y JavaScript tienen semánticas y propósitos diferentes.
- Se ejecuta fundamentalmente en navegador aunque puede ser ejecutado en servidor (NodeJS).
- En este curso vamos a ver JavaScript en navegador

Historia de JavaScript

- JavaScript fue desarrollado originalmente por **Brendan Eich** de Netscape con el nombre de Mocha, el cual fue renombrado posteriormente a LiveScript, para finalmente quedar como JavaScript.
- En 1997 los autores propusieron JavaScript para que fuera adoptado como estándar de la European Computer Manufacturers Association ECMA, que a pesar de su nombre no es europeo sino internacional, con sede en Ginebra.
- En junio de 1997 fue adoptado como un estándar ECMA, con el nombre de ECMAScript. Poco después también como un estándar ISO.



JavaScript en Febrero de 2020

- JavaScript junto a HTML y CSS es una de las tecnologías más importantes que da vida a lo que conocemos como World Wide Web o Internet

INDICE TIOBE	Lenguaje
1	Java
2	C
3	Python
4	C++
5	C#
6	Visual Basic .NET
7	JavaScript
8	PHP
9	SQL
10	Swift

Popularidad de
los lenguajes
de
programación



INDICE PYPL	Lenguaje
1	Python
2	Java
3	JavaScript
4	C#
5	PHP
6	C/C++
7	R
8	Objective-C
9	Swift
10	TypeScript

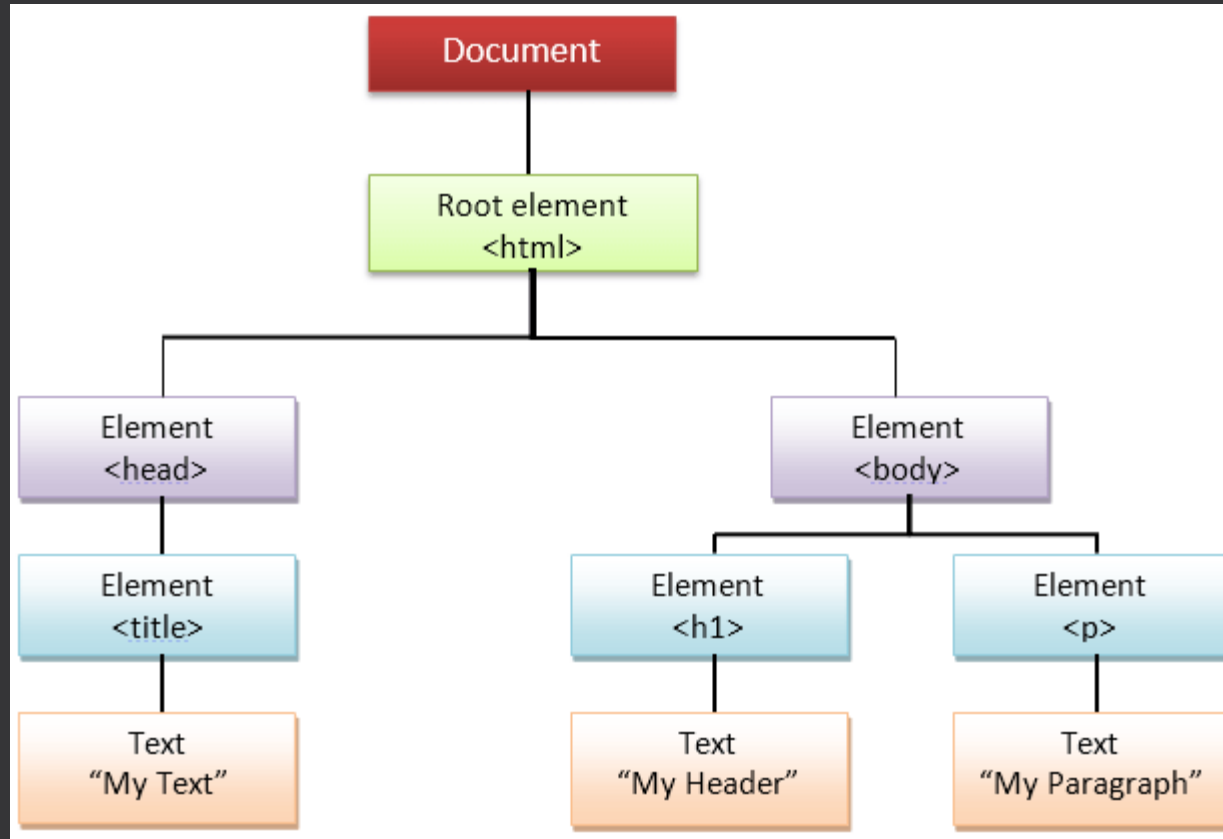
JAVASCRIPT EN NAVEGADOR

¿Para que sirve JavaScript?

HTML	CSS	JAVASCRIPT
Aporta la estructura (secciones, artículos, capas, párrafos) y el contenido de la página web (textos, imágenes)	Aporta cómo se ve el contenido de la página (colores, estilos, tipos de letras, efectos de sombra, degradados) y un cierto nivel de interactividad	Aporta un alto nivel de interactividad con el usuario usando lógica (cómo cambia la página o que información mostrar cuando el usuario interacciona, por ejemplo presionando un botón o cambiando un texto)

HTML**CSS**

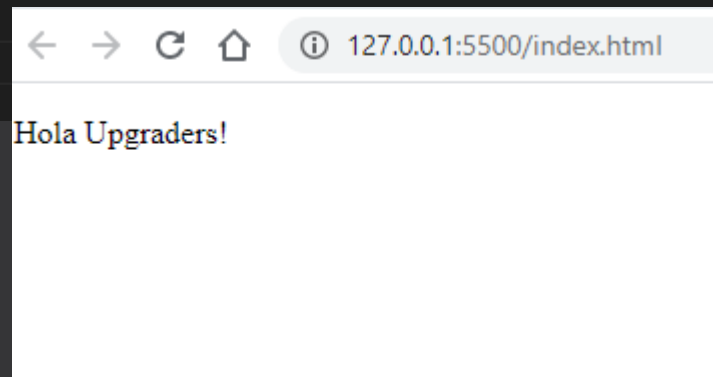
¿Qué es el DOM?



- El modelo de objeto de documento (**DOM**) es una interfaz de programación para los documentos HTML.
- Facilita una representación estructurada del documento y define de qué manera los programas pueden acceder, a fin de modificar, tanto su estructura, estilo y contenido.
- El **DOM** da una representación del documento como un grupo de nodos y objetos estructurados que tienen propiedades y métodos.
- Esencialmente, conecta las páginas web a lenguajes de programación como JavaScript.

El DOM y JavaScript

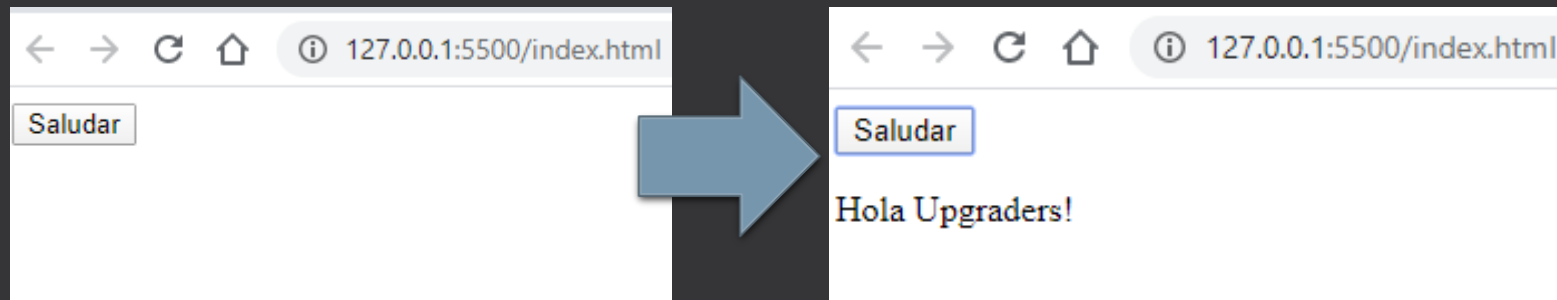
```
index.html > ...
1 <!DOCTYPE html>
2 <html lang="es">
3   <head>
4     <meta charset="UTF-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6     <title>Probando JavaScript</title>
7   </head>
8   <body>
9     <p id="mi-parrafo"></p>
10    <script>
11      document.getElementById("mi-parrafo").innerHTML += "Hola Upgraders!";
12    </script>
13  </body>
14 </html>
15
```



- Tenemos un párrafo `<p>` con un `id="mi-párrafo"`
- Desde el código en JavaScript encerrado en la etiqueta `<script>`, usamos el **DOM** para modificar el contenido de `<p>`
- En el documento, buscamos un elemento con `id "mi-párrafo"` y asignamos "Hola Upgraders!" al atributo `innerHTML` que representa el contenido entre una etiqueta de apertura y una de cierre en HTML, en este caso entre `<p>` y `</p>`
- `=` es el operador de asignación

Interactuando con el usuario

```
1 <!DOCTYPE html>
2 <html lang="es">
3   <head>
4     <meta charset="UTF-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6     <title>Probando JavaScript</title>
7   </head>
8   <body>
9     <button onclick="saluda()">Saludar</button>
10    <p id="mi-parrafo"></p>
11    <script>
12      function saluda(){
13        document.getElementById("mi-parrafo").innerHTML = "Hola Upgraders!";
14      }
15    </script>
16  </body>
17 </html>
```

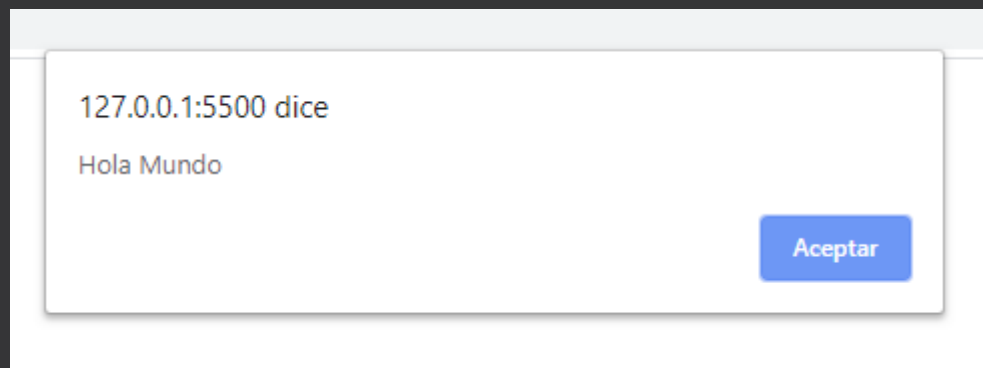


- Tenemos un párrafo `<p>` con un `id="mi-parrafo"` y un botón
- El botón tiene una función en JavaScript asociada a su evento click
- El evento es algo que ocurre en el **DOM** como resultado de la interacción del usuario (en este caso hacer click con el ratón sobre el botón)
- Cuando se pulsa el botón se invoca a la función `saluda()`, la cuál busca el elemento con `id "mi-parrafo"` y asigna "Hola Upgraders!" como contenido del mismo

Cómo añadir JavaScript a mi página (1)

```
1 <!DOCTYPE html>
2 <html lang="es">
3   <head>
4     <meta charset="UTF-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6     <title>Probando JavaScript</title>
7   </head>
8   <body>
9     <button onclick="alert('Hola Mundo')">Hola Mundo!</button>
10  </body>
11 </html>
12
```

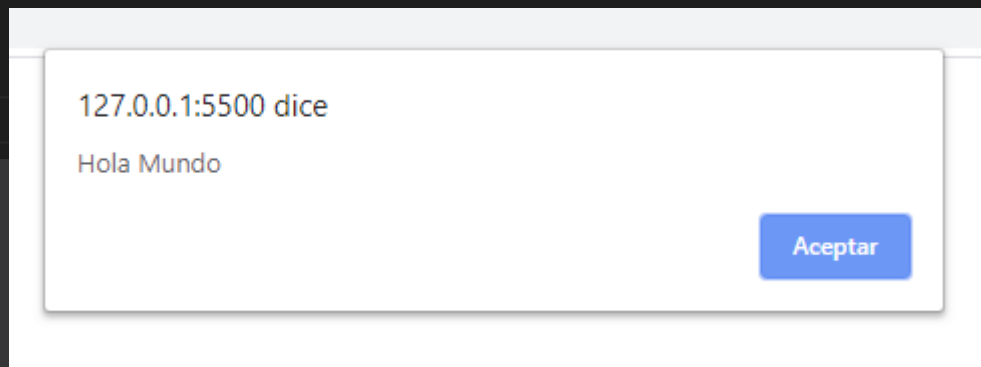
- Podemos añadir JavaScript directamente en el código HTML, pero solo para interacciones muy pequeñas como un `alert()` para ver un mensaje



Cómo añadir JavaScript a mi página (2)

```
1 <!DOCTYPE html>
2 <html lang="es">
3   <head>
4     <meta charset="UTF-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6     <title>Probando JavaScript</title>
7   </head>
8   <body>
9     <button onclick="holaMundo()">Hola Mundo!</button>
10    <script>
11      function holaMundo() {
12        alert("Hola Mundo!");
13      }
14    </script>
15  </body>
16 </html>
17
```

- Podemos añadir JavaScript en una sección de código delimitada por las etiquetas de HTML `<script>` y `</script>`
- Una página puede incluir scripts
- Los scripts pueden ir en el head o en el body de la página



Cómo añadir JavaScript a mi página (3)

```
1 <!DOCTYPE html>
2 <html lang="es">
3   <head>
4     <meta charset="UTF-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6     <title>Probando JavaScript</title>
7   </head>
8   <body>
9     <button onclick="holaMundo()">Hola Mundo!</button>
10    <script src="miscript.js"></script>
11  </body>
12 </html>
```

index.html JS miscript.js ×

JS miscript.js > ...

```
1 function holaMundo() {
2   alert("Hola Mundo");
3 }
4
```

127.0.0.1:5500 dice

Hola Mundo

Aceptar

- Podemos incluir el código JavaScript desde un archivo local que no debe contener las etiquetas `<script>`
- Todo lo que esté dentro de ese archivo será nuestro código JavaScript
- Los scripts pueden incluirse en el head o en el body de la página

Cómo añadir JavaScript a mi página (4)

```
1 <!DOCTYPE html>
2 <html lang="es">
3   <head>
4     <meta charset="UTF-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6     <title>Probando JavaScript</title>
7   </head>
8   <body>
9     .....
10    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.4.1/jquery.min.js">
11    </script>
12    <script src="https://cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.4.1/semantic.min.js">
13    </script>
14  </body>
15 </html>
```

- Podemos incluir el código JavaScript desde un archivo remoto en un CDN (Content Delivery Network)
- Muchas veces se utiliza esta solución para incluir librerías JS de terceros
- Igualmente se pueden incluir múltiples scripts remotos en el head o el body de la página

JavaScript mostrando datos

Forma	Ejemplo
Escribiendo dentro de un elemento HTML en la propiedad innerHTML	<code>document.getElementById("p-1").innerHTML = "Hola chavales!"</code> ;
Escribiendo en la página usando <code>document.write()</code>	<code>document.write("Hola Central!")</code> ;
Mostrando datos con <code>window.alert()</code>	<code>alert("Hola amigos!")</code> ;
Escribiendo datos con usando <code>console.log()</code>	<code>Console.log("Este es el valor de x = ",x)</code> ;

EL LENGUAJE JAVASCRIPT

Los comentarios

```
1  /*
2  |  ··Esta es una funcion en JavaScript
3  |  ··escrita en Upgrade
4  |  */
5  function holaMundo() {
6  |  ···// Modificando el DOM
7  |  ··document.getElementById("mi-p").innerHTML = "Hola Mundo!";
8  |  }
9  |
```

Los comentarios son ignorados por el lenguaje JavaScript

- Los comentarios son formas de documentar nuestro código y hacerlo mantenible
- El uso de comentarios se considera una buena práctica de programación
- Comentarios de bloque son los que ocupan múltiples líneas entre `/*` y `*/`
- Comentarios de línea ocupan solo una línea que comience por `//`

Los literales

```
1 <!DOCTYPE html>
2 <html lang="es">
3   <head>
4     <meta charset="UTF-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6     <title>Probando JavaScript</title>
7   </head>
8   <body>
9     <button onclick="holaMundo()">Hola Mundo!</button>
10    <script>
11      function holaMundo() {
12        alert("Hola Mundo!");
13      }
14    </script>
15  </body>
16 </html>
17
```

- Los literales son valores constantes que escribimos dentro de nuestro programa JavaScript
- Ejemplos de literales son:
 - “**Hola Mundo**”, o el nombre de la etiqueta párrafo “**mi-p**”
 - o un valor numérico como **123** o simbólico como **false** o **true**

Los identificadores

```
1 <!DOCTYPE html>
2 <html lang="es">
3   <head>
4     <meta charset="UTF-8" />
5     <meta
6       name="viewport"
7       content="width=device-width,
8         initial-scale=1.0"
9     />
10    <title>Probando JavaScript</title>
11  </head>
12  <body>
13    <button onclick="holaMundo()">Hola Mundo!</button>
14    <script>
15      var miVariable = "primer valor";
16      function holaMundo() {
17        alert("Hola Mundo!");
18      }
19    </script>
20  </body>
21 </html>
```

- Los identificadores son **nombres únicos** que identifican elementos del programa creados por nosotros como las variables o las funciones
- Pueden ser cortos como **x**, **y**, **z** o descriptivos como **cantidadClases** o **totalAlumnos**
- Los identificadores son sensibles a mayúsculas y minúsculas (case-sensitive). Así **aaa** es distinto de **AAA** y de **aaA**
- En este ejemplo, **miVariable** es el nombre de una variable de cadena de caracteres mientras que **holaMundo** es el nombre de una función
- Los identificadores pueden comenzar por una letra, el guión bajo (**_**) y el símbolo de dolar (**\$**); el resto de los caracteres puede ser cualquier combinación de los anteriores, además de cifras. Así, **registro_01**, **\$vínculos**, **listaNodos** o **_cuñao1** son nombres correctos; **1_lista_de_cosas**, **elementos-lista** o **nuevo%elemento** no.

Las palabras reservadas

abstract	arguments	await*	boolean
break	byte	case	catch
char	class*	const	continue
debugger	default	delete	do
double	else	enum*	eval
export*	extends*	false	final
finally	float	for	function
goto	if	implements	import*
in	instanceof	int	interface
let*	long	native	new
null	package	private	protected
public	return	short	static
super*	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

- No pueden utilizarse como identificadores las palabras reservadas con las que se expresan las instrucciones del propio lenguaje JavaScript



Notación CamelCase

```
1 <!DOCTYPE html>
2 <html lang="es">
3   <head>
4     <meta charset="UTF-8" />
5     <meta
6       name="viewport"
7       content="width=device-width,
8         initial-scale=1.0"
9     />
10    <title>Probando JavaScript</title>
11  </head>
12  <body>
13    <button onclick="holaMundo()">Hola Mundo!</button>
14    <script>
15      var miVariable = "primer valor";
16      function holaMundo(){
17        alert("Hola Mundo!");
18      }
19      function adiosMundoCruel(){
20        alert("Adios Mundo Cruel!");
21      }
22    </script>
23  </body>
24 </html>
```

- Notación CamelCase: Se escribe la primera palabra del nombre en minúscula, y si el nombre consiste en varias, las siguientes se empiezan por mayúsculas
- Por ejemplo: `listaCamposFormulario`, `miVariable`, `holaMundo`, `adiosMundoCruel`



Las instrucciones

```
<script>
  var miVariable = "primer valor";
  function holaMundo(nombre) {
    alert("Hola "+nombre);
  }
  function adiosMundoCruel(gradoCruel) {
    if (gradoCruel == 0) {
      console.log("no muy cruel");
      alert("Adios Mundo Cruel!");
    }
    else {
      console.log("en verdad muy cruel");
      alert("Adios Mundo MUY Cruel!");
    }
  }
</script>
```

- Las instrucciones son fragmentos o piezas de código que le dicen al ordenador que haga algo
- **Declarativas:** Creación de variables, clases o funciones
- **Imperativas:** las que realizan alguna acción como cuando invocamos funciones, realizamos cálculos, tomamos decisiones o modificamos el **DOM**
- **Simples:** ejecutan una única acción y por lo general ocupan una línea y terminan en punto y coma ;
- **Bloques:** agrupan a un conjunto de instrucciones simples entre llaves **}** que se ejecutan como si fuera una única acción

Las variables

```
• var miVariable = "primer valor"; // Cadena de caracteres
• var miEntero = 100; // Numérica
• var soyAstronauta = false; // Booleana
•
• var soyAstronauta = 0; // soyAstronauta es ahora numérica
• var soyAstronauta = "No"; // soyAstronauta es ahora de cadena
```

- Las variables son los contenedores de datos de nuestro programa
- Sus nombre tienen que ser identificadores únicos
- Las variables pueden inicializarse con un valor que automáticamente define su tipo
- JavaScript es un lenguaje de *tipado débil* o *dinámico*. Esto significa que no es necesario declarar el tipo de variable antes de usarla.
- El tipo será determinado automáticamente cuando el programa comience a ser procesado.
- Esto también significa que puedes tener la misma variable con diferentes tipos

Tipos de datos primitivos

Tipo de Dato	Significado
Boolean	Representa a una entidad lógica y puede tener dos valores: true (verdadero) o false (falso)
Null	Representa un literal de JavaScript que representa un valor nulo o vacío. Su único valor es null
Undefined	Representa el valor por defecto de una variable a la cuál aun no le hemos asignado un valor
Number	De acuerdo al estándar ECMAScript, solo existe el valor numérico de 64 bits de doble precisión, un número entre $-(2^{53} - 1)$ y $2^{53} - 1$. No hay un tipo específico para números enteros. Además de los valores numéricos, el tipo Number tiene los siguientes valores simbólicos: +infinity , -infinity y NaN (Not A Number o No es un número)
String	Representa cadenas de caracteres. Es un conjunto de "elementos", de valores enteros sin signo de 16 bits. Cada elemento ocupa una posición en el String. El primer elemento está en el índice 0, el próximo está en el índice 1, y así sucesivamente. La longitud de un String es el número de elementos en ella. Los strings en JavaScript son inmutables. Esto significa que una vez que una cadena de caracteres es creada no es posible modificarla.
Symbol	Es un valor primitivo único e inmutable y puede ser usado como la clave de una propiedad de un Object

Operadores Aritméticos

Operador	Descripción
+	Adición
-	Substracción
*	Multiplicación
**	Exponenciación
/	División
%	Resto de división
++	Incremento
--	Decremento



Operadores de Asignación

Operador	Ejemplo	Equivalente a
=	$x = y$	$x = y$
+=	$x += y$	$x = x + y$
-=	$x -= y$	$x = x - y$
*=	$x *= y$	$x = x * y$
**=	$x **= y$	$x = x ** y$
/=	$x /= y$	$x = x / y$
%	$x \% = y$	$x = x \% y$



Operadores de Cadenas

Operador	Ejemplo	Resultado
+	x = "hola " + "upgrader"	x = "Hola upgrader"
+=	x += "!"	x = "Hola upgrader!"
+	x = "Cumple " + 20 + "días"	x = "Cumple 20 días"

- El operador + en cadenas de caracteres funciona como un operador de concatenación
- El operador += significa añadir contenido al contenido existente en una variable
- Si sumamos cadenas de caracteres y números, los números serán convertidos en cadenas de caracteres y el resultado final siempre será una cadena de caracteres



Operadores de Comparación (X = 5)

Operador	Significado	Ejemplos	Resultado
==	Igual a	X == 8 X == 5 X == "5"	false true true
===	Igual a e igual tipo	X === 5 X === "5"	true false
!=	Desigual valor	X != 8	true
!==	Desigual valor o distinto tipo	X !== 5 X !== "5"	false true
>	Mayor que	X > 3 X > 10	true false
<	Menor que	X < 3 X < 10	false true
>=	Mayor o igual	X >= 5 X >= 10	true false
<=	Menor o igual	X <= 5 X <= 3	true false
?	Operador condicional	var votable = (x < 18) ? "muy joven": "tiene la edad";	votable = "muy joven"

Operadores Lógicos

X	Y	X && Y
true	true	true
true	false	false
false	true	false
false	false	false

X	Y	X Y
true	true	true
true	false	true
false	true	true
false	false	false

X	!X
true	false
false	true

- El operador lógico AND (&&) devuelve verdadero solo si las dos condiciones de entrada tienen el valor verdadero. En el resto de los casos devuelve siempre falso

- El operador lógico OR (||) devuelve falso si todas las condiciones de entrada son falsas. En cualquier otro caso devuelve verdadero

- El operador lógico NOT (!) devuelve falso si la condición es verdadera o verdadero si la condición falsa (invierte o niega el resultado)



Operadores de Tipo

Operador	Ejemplos	Resultados
typeof	<pre>typeof "John" typeof 3.14 typeof NaN typeof false typeof [1,2,3,4] typeof {name:'John', age:34} typeof new Date() typeof function(){} typeof myCar typeof null</pre>	<pre>"string" "number" "number" "boolean" "object" "object" "object" "function" "undefined" * "object"</pre>
instanceof	<pre>class AAA { constructor() { this.x = "AAA"; } } var a = new AAA(); alert(a instanceof AAA); alert(a instanceof BBB);</pre>	<pre>true false</pre>

Precedencia de Operadores

Orden	Tipo de operador	Ejemplos
1	Agrupación	()
2	Acceso a miembro Llamada a una función	x.color Math.round(x)
3	Crear objeto	new ClaseXY()
4	Incremento y decremento postfijo	x++; y--;
5	Operador lógico NOT (!) Negación unitaria Incremento y decremento prefijo Operador de tipo	!(x == 5) -x ++x; --y; typeof x
6	Exponenciación	**
7	Multiplicación, división y resto de división	x * y / (z % 2)
8	Suma y resta	x + y + z
9	instanceof y operadores de comparación	X instanceof AAA; X > Y; X < Y
10	Operadores de igualdad	X !== Y; X == Y
11	Operador lógico AND (&&)	(X > Y) && (Z == 5)
12	Operador lógico OR ()	(X > Y) (Z == 5)
13	Operador de asignación =	X = X * 2 + Y

ESTRUCTURAS DE CONTROL

Estructuras de control

Estructura	Descripción	Ejemplos
Secuencial	Es la estructura básica de programación que implica que las instrucciones se ejecutan una detrás de otra y están escritas en un orden lógico. Todo programa tiene al menos una estructura secuencial	<pre>x = x + 100; y = x * 50; console.log("x=",x,"y=",y);</pre>
Decisión/Selectiva/Alternativa	El flujo de programa sigue diferentes caminos si se cumplen determinadas condiciones	<pre>if (x == 5){ console.log("OK"); } else{ console.log("ERROR"); }</pre>
Repetición/Bucle /Iterativa	El flujo de programa se repite si se cumplen determinadas condiciones	<pre>for(let i=1,i<10;i++){ console.log("i=",i); }</pre>
Paralela	Parte del flujo de programa se ejecuta en paralelo con el programa principal	<pre>let promise1 = new Promise((resolve) => resolve(executeQuery(...)); let responses = await Promise.all([promise1, promise2, ...]); for(let response of responses) { // Tratamiento de las respuestas }</pre>

ESTRUCTURAS DE DECISION

Estructura de decisión “if-else” (1)

```
if (cantidadAlumnos == 0) {  
    console.log("no tenemos alumnos");  
    alert("No podemos empezar sin alumnos!");  
} else {  
    console.log("ya tenemos alumnos");  
    alert("Ya podemos comenzar con " + cantidadAlumnos + "!");  
}
```

- En esta estructura si la condición entre paréntesis que sigue a la palabra reservada “if” es verdadera (la cantidad de alumnos es cero), se ejecutan las dos primeras instrucciones incluidas en el bloque delimitado por las llaves { }
- Si cantidadAlumnos tiene cualquier otro valor que no sea cero, se ejecutan las instrucciones del bloque precedido por la palabra reservada “else”
- **Buenas prácticas:** utilizar siempre bloques aunque solo tengamos una única instrucción (facilita incluir nuevas instrucciones en el futuro)

Estructura de decisión “if-else” (2)

```
if (cantidadAlumnos == 0) {  
    console.log("no tenemos alumnos");  
    alert("No podemos empezar sin alumnos!");  
}
```

- La cláusula “else” es opcional en una estructura de control con “if-else”

```
if (nombre == "")  
    alert("nada");  
else  
    alert("hay");
```

- El bloque (conjunto de instrucciones delimitadas por llaves { }) puede omitirse si solo hay una única instrucción a ejecutar en la cláusula “if” o en la “else”



Estructura de decisión “if-else” (3)

```
if (cantidadAlumnos == 0) {  
    claseNueva = false;  
}  
else {  
    claseNueva = true;  
}
```

```
claseNueva = (cantidadAlumnos == 0) ? false : true;
```

- Si vamos a utilizar la estructura de control “if-else” para asignar dos posibles valores a una misma variable en dependencia de si se cumple o no una condición, podemos utilizar el operador condicional
- Este operador evalúa la condición entre paréntesis.
 - Si el resultado es verdadero, asigna a la variable el valor que sigue al ?
 - Si el resultado es falso, asigna a la variable el valor que sigue a :

Estructura de decisión “switch” (1)

```
switch (cantidadAlumnos) {  
    case 5:  
        tipoClase = "pequeña";  
        break;  
    case 10:  
        tipoClase = "mediana";  
        break;  
    case 15:  
        tipoClase = "grande";  
        break;  
    default:  
        tipoClase = "cualquier otra clase";  
}
```

- Permite ejecutar instrucciones en dependencia del valor que tome una variable
- Las instrucciones a ejecutar se comenzarían debajo de la cláusula “case” cuyo valor coincide con el valor de la variable y terminarían al encontrar una instrucción break;
- Si ninguno de los valores está representado en las cláusulas “case”, es posible añadir una cláusula “default” similar al “else” de la estructura de control “if-else”
- Al igual que “else”, “default” es opcional
- La estructura de control “switch” usa comparación estricta (===), lo que significa que los literales en “case” tienen que coincidir en valor y tipo con el valor de la variable

Estructura de decisión “switch” (2)

```
var Animal = "Oso";
switch (Animal) {
  case "Tigre":
  case "Leon":
  case "Jabali":
  case "Oso":
    console.log("Este animal va al Zoo!");
    break;
  case "Velociraptor":
  case "T-REX":
  default:
    console.log("Este animal va Jurassic Park!");
}
```

- Se pueden agrupar valores de diferentes cláusulas “case” para ejecutar las mismas instrucciones
- Incluso se pueden agrupar estos valores con la cláusula “default”



Estructura de decisión “switch” (3)

```
var Animal = "Oso";  
switch (Animal) {  
  case "Velociraptor":  
  case "T-REX":  
  default:  
    console.log("Este animal va Jurassic Park!");  
    break;  
  case "Tigre":  
  case "Leon":  
  case "Jabali":  
  case "Oso":  
    console.log("Este animal va al Zoo!");  
    break;  
}
```

- Si “default” no es la última cláusula entonces tenemos que añadir una instrucción break; para delimitar hasta donde llegan las instrucciones a ejecutar

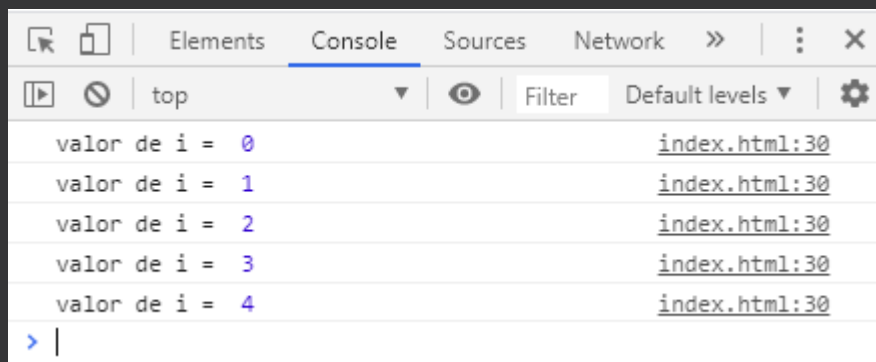


ESTRUCTURAS DE BUCLE

Estructura de bucle “for” (1)

```
console.log("valor de i = ", 0);  
console.log("valor de i = ", 1);  
console.log("valor de i = ", 2);  
console.log("valor de i = ", 3);  
console.log("valor de i = ", 4);  
  
for (var i = 0; i < 5; i++) {  
  console.log("valor de i = ", i);  
}
```

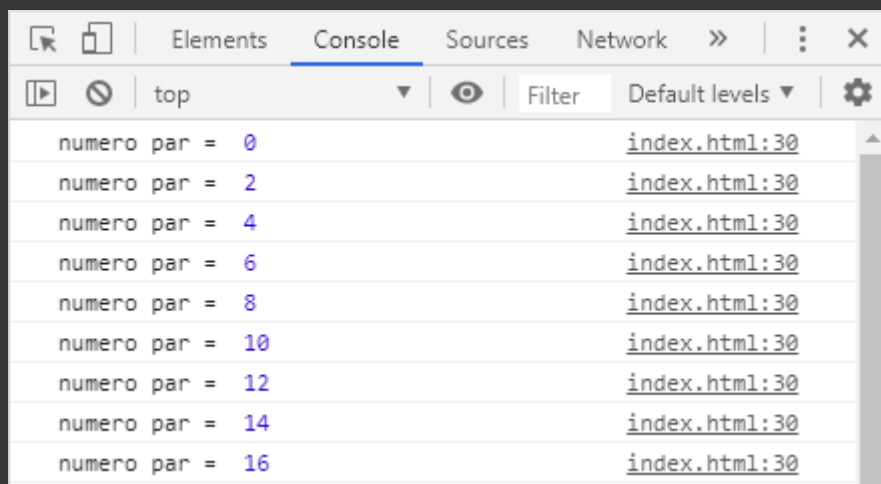
- En vez de repetir escribir los valores de 0 a 4 en la consola utilizando instrucciones individuales, podríamos utilizar un bucle “for”
- El bucle se compone de tres partes y un bloque de instrucciones
- La primera parte crea y asigna un valor a la variable de control que en este caso es i
- La segunda prueba si la condición se cumple para continuar repitiendo el bucle
- La tercera, incrementa la variable de control en uno o el valor deseado



Estructura de bucle “for” (2)

```
for (var i = 0; i < 100; i += 2) {  
  console.log("numero par = ", i);  
}  
  
for (var i = 1; i < 100; i += 2) {  
  console.log("numero impar = ", i);  
}
```

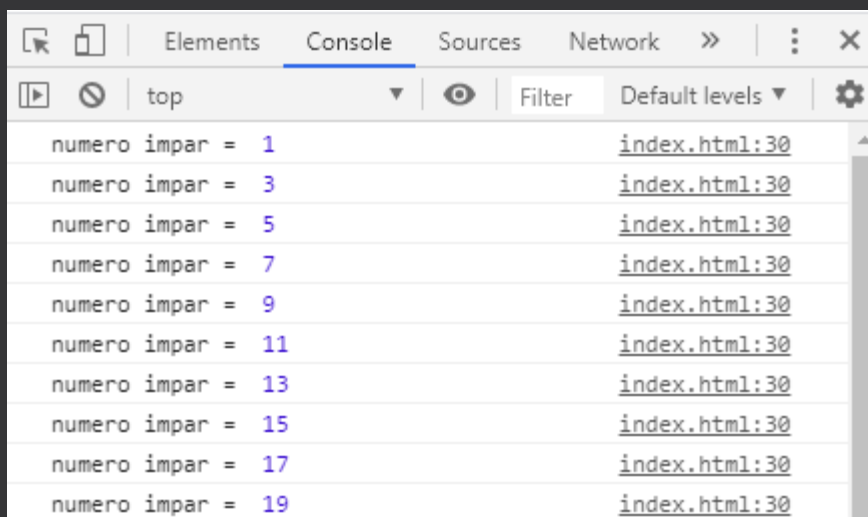
- En este ejemplo tenemos dos bucles, uno para escribir por consola todos los números pares menores que 100 y otro todos los números impares menores que 100
- Conociendo que los pares empiezan en 0 y los impares en 1, solo tenemos que sumar dos a la variable de control para obtener 0,2,4,6,... en el primer caso y 1,3,5,7,... en el segundo caso



Elements Console Sources Network » X

top Filter Default levels

numero par = 0	index.html:30
numero par = 2	index.html:30
numero par = 4	index.html:30
numero par = 6	index.html:30
numero par = 8	index.html:30
numero par = 10	index.html:30
numero par = 12	index.html:30
numero par = 14	index.html:30
numero par = 16	index.html:30



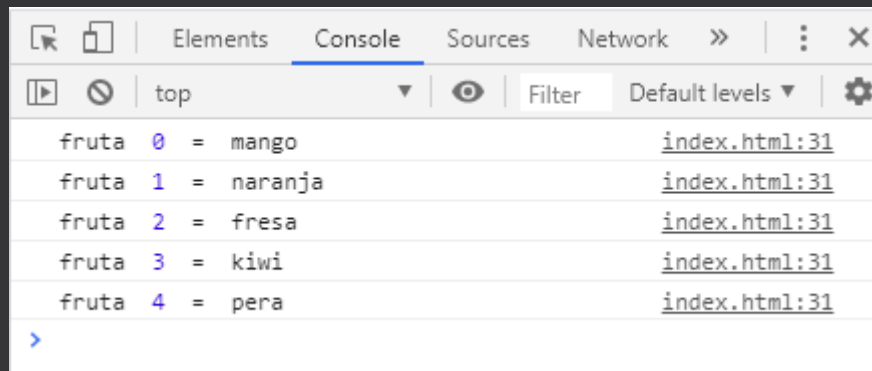
Elements Console Sources Network » X

top Filter Default levels

numero impar = 1	index.html:30
numero impar = 3	index.html:30
numero impar = 5	index.html:30
numero impar = 7	index.html:30
numero impar = 9	index.html:30
numero impar = 11	index.html:30
numero impar = 13	index.html:30
numero impar = 15	index.html:30
numero impar = 17	index.html:30
numero impar = 19	index.html:30

Estructura de bucle “for” (3)

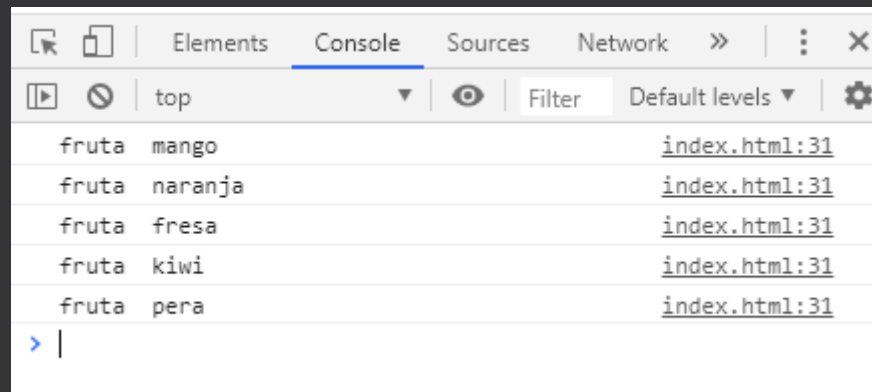
```
var frutas = ["mango", "naranja", "fresa", "kiwi", "pera"];  
for (var i = 0; i < frutas.length; i++) {  
  console.log("fruta ", i, " = ", frutas[i]);  
}
```



- Los bucles “for” son muy utilizados para recorrer los elementos de un arreglo, cuyos índices suelen comenzar en 0 e ir hasta la cantidad de elementos - 1

Estructura de bucle “for” (4)

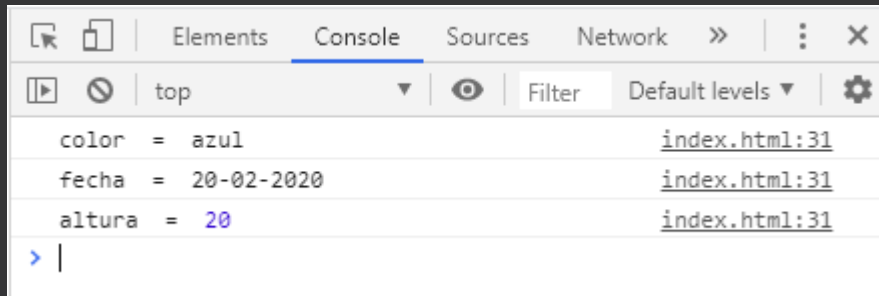
```
var frutas = ["mango", "naranja", "fresa", "kiwi", "pera"];  
for (var fruta of frutas) {  
  console.log("fruta ", fruta);  
}
```



- Esta es otra manera de escribir un bucle “for” para recorrer los elementos de un arreglo si no necesitamos sus índices

Estructura de bucle “for” (5)

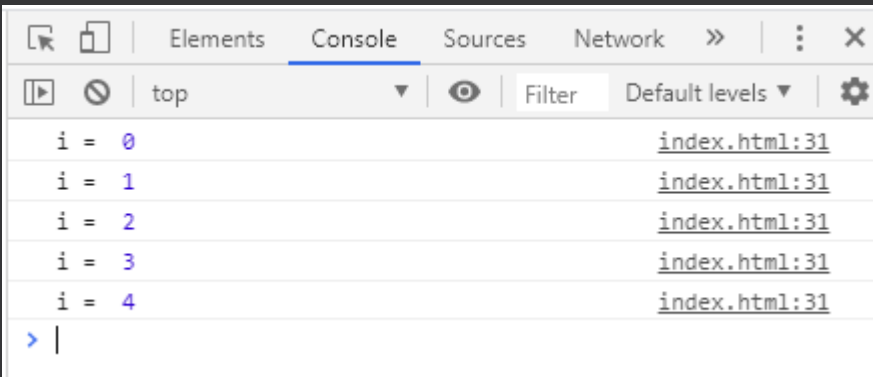
```
var objeto = { color: "azul", fecha: "20-02-2020", altura: 20 };  
for (var propiedad in objeto) {  
  console.log(propiedad, " = ", objeto[propiedad]);  
}
```



- Con un bucle “for” también podemos recorrer las propiedades de un objeto y obtener su valor

Estructura de bucle “while” (1)

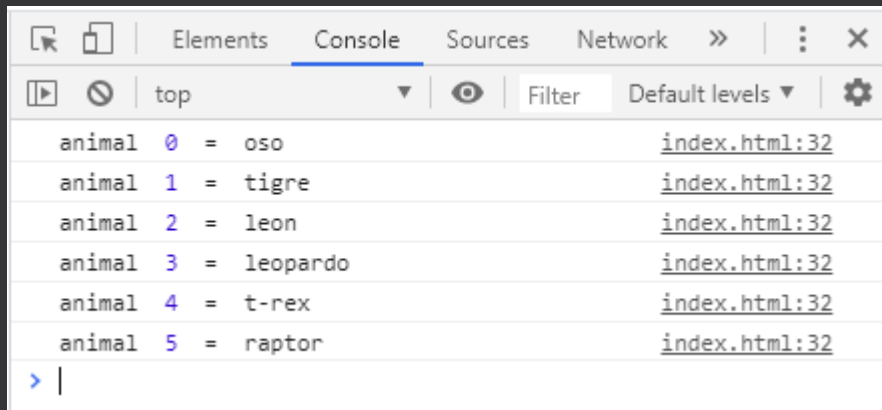
```
var i = 0;
while (i < 5) {
  console.log("i = ", i);
  i++;
}
```



- El bucle “while” repite las instrucciones en su bloque mientras se cumpla la condición especificada entre paréntesis después de la cláusula “while”
- Si la condición antes de comenzar el bucle no se cumple, **el bucle no se realizará** y programa continuará en la próxima instrucción debajo del bloque de instrucciones que agrupa el bucle
- Si la condición se cumple, el bucle volverá a la primera instrucción del bloque
- Si utilizamos una variable de control que se incrementa o se decrementa, por lo general su valor se cambia como última instrucción en el bucle

Estructura de bucle “while” (2)

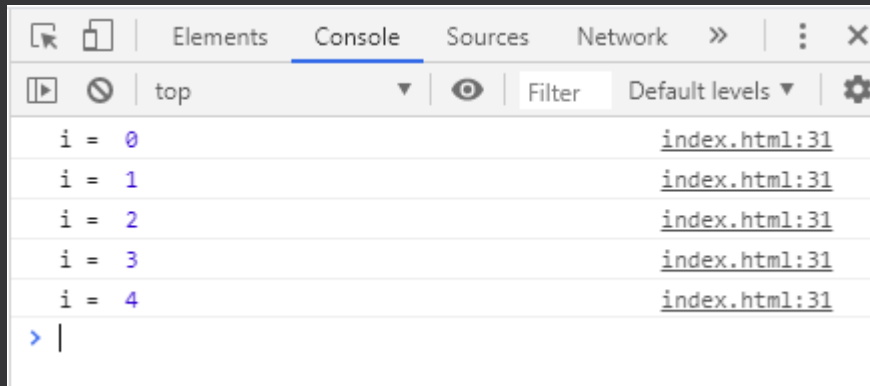
```
var animales = ["oso", "tigre", "leon", "leopardo", "t-rex", "raptor"];  
var i = 0;  
while (animales[i]) {  
  console.log("animal ", i, " = ", animales[i]);  
  i++;  
}
```



- En este ejemplo utilizamos el bucle “while” para recorrer un arreglo de animales
- Con índice $i = 0$, mientras el arreglo indexado en i tenga un valor distinto de “undefined” mostramos el animal, incrementamos el índice i para ver el siguiente y continuamos en el bucle
- Cuando i sobrepase la cantidad de elementos del arreglo, `animales[i]` devolverá indefinido (undefined) que se toma como un valor falso y el bucle “while” será interrumpido

Estructura de bucle “do-while”

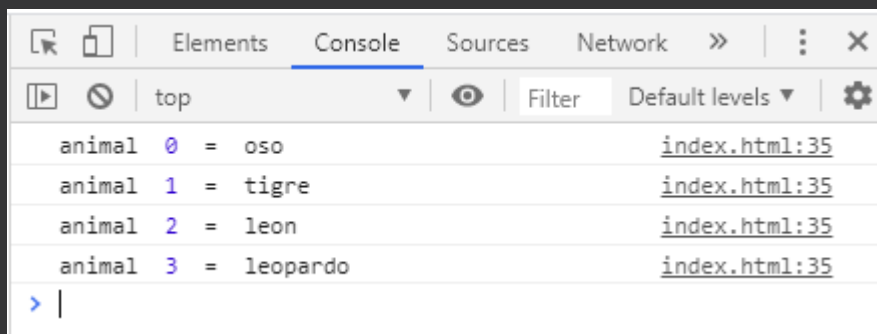
```
var i = 0;
do {
  console.log("i = ", i);
  i++;
} while (i < 5);
```



- El bucle “do-while” repite las instrucciones en su bloque mientras se cumpla la condición especificada entre paréntesis después de la cláusula “while”
- Si la condición antes de comenzar el bucle no se cumple, el bucle se **realizará al menos una vez (siempre entra)** y programa continuará en la próxima instrucción debajo del bloque de instrucciones que agrupa el bucle
- Si la condición se cumple, el bucle volverá a la primera instrucción del bloque cuyo comienzo marca la cláusula “do”
- Si utilizamos una variable de control que se incrementa o se decrementa, por lo general su valor se cambia como última instrucción en el bucle

La instrucción break

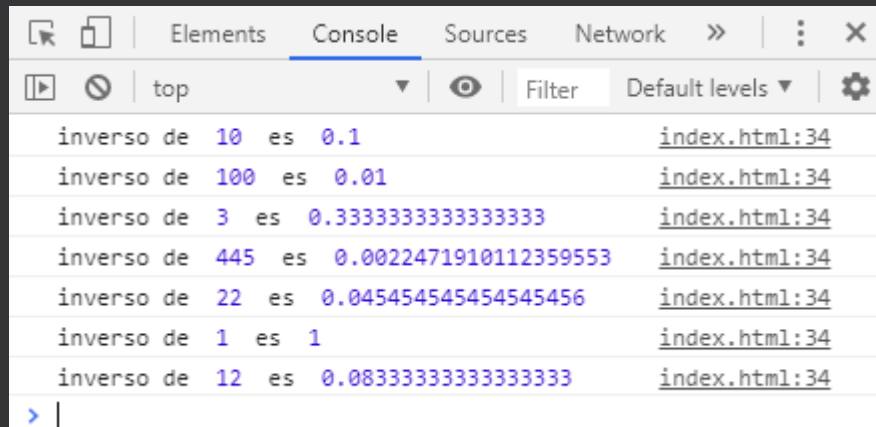
```
var animales = ["oso", "tigre", "leon", "leopardo", "t-rex", "raptor"];
var i = 0;
while (animales[i]) {
  if (animales[i] == "t-rex" || animales[i] == "raptor") {
    break;
  }
  console.log("animal ", i, " = ", animales[i]);
  i++;
}
```



- La instrucción **break**, rompe cualquier bucle si se cumple una condición
- El programa continuará en la próxima instrucción debajo del bloque del bucle

La instrucción continue

```
var enteros = [10, 0, 100, 3, 445, 22, 0, 1, 12];  
for (var i = 0; i < enteros.length; i++) {  
  if (!enteros[i]) {  
    continue;  
  }  
  inverso = 1.0 / enteros[i];  
  console.log("inverso de ", enteros[i], " es ", inverso);  
}
```



Log Message	Source
inverso de 10 es 0.1	index.html:34
inverso de 100 es 0.01	index.html:34
inverso de 3 es 0.3333333333333333	index.html:34
inverso de 445 es 0.0022471910112359553	index.html:34
inverso de 22 es 0.045454545454545456	index.html:34
inverso de 1 es 1	index.html:34
inverso de 12 es 0.08333333333333333	index.html:34

- La instrucción `continue` hace que el lazo comience de nuevo en la próxima iteración
- Como no están definidos el inverso de cero, si un elemento del arreglo `enteros` es cero pasamos a la próxima iteración que es lo mismo que ir al próximo elemento

FUNCIONES

Declaración de funciones

```
function mult(a,b){  
    return a * b;  
}
```

- Las funciones se declaran con la palabra reservada “function”, una lista de parámetros opcionales y un bloque de instrucciones
- Las funciones devuelven valor con la instrucción return
- Las funciones no se ejecutan cuando se declaran

```
<button onclick="alert(mult(10,10))">Multiplica</button>
```

127.0.0.1:5500 dice

100

Aceptar

- Las funciones “quedan reservadas” en nuestro script hasta que son invocadas o llamadas

Funciones como expresiones

```
var mult = function(a, b) { return a * b; };
```

- Las funciones pueden declararse también como expresiones y asignarse a variables
- Una vez que la variable contiene a la expresión que define la función, la variable puede ser utilizada como si fuera una función

```
<button onclick="alert(mult(10,10))">Multiplica</button>
```

127.0.0.1:5500 dice

100

Aceptar

- Igualmente las funciones definidas como expresiones en variables, “quedan reservadas” en nuestro script hasta que son invocadas o llamadas

Funciones auto-invocadas(1)

```
(function(){  
    alert("Funcion auto invocada");  
})();
```

127.0.0.1:5500 dice
Funcion auto invocada

Aceptar

- En JavaScript es posible crear funciones anónimas que se auto-invoquen (también conocidas como **IIFE** o **Immediately Invoked Function Expressions**)
- Primero creamos una construcción de tipo `()`;
- Y luego definimos una función sin nombre dentro del primer paréntesis con las instrucciones que queremos ejecutar

Funciones auto-invocadas(2)

```
var mult = function(a, b) {  
    return a * b;  
};  
  
(function(w) {  
    alert(w.mult(5, 5));  
})(window);
```

127.0.0.1:5500 dice

Funcion auto invocada

Aceptar

- Si vas a llamar a una función definida fuera de la función auto-invocada es necesario pasar el objeto global window que es al que pertenecen o donde se crean todas las funciones que vamos declarando
- Luego usamos el parámetro para invocar la función deseada

```
mult: f (a, b)  
miVariable: "primer valor"  
miEntero: 100  
soyAstronauta: "No"  
enteros: (9) [10, 0, 100, 3, 445, 22, 0, 1, 12]  
i: 9  
holaMundo: f holaMundo(nombre)
```

Fragmento del objeto global window obtenido con un `console.log(window);`

Argumentos de las funciones(1)

```
function demo(nombre, edad) {  
    return nombre + " con edad " + edad;  
}
```

```
var personaEdad = demo("Juan", 24);
```

- Los **parámetros** son los nombres con los que se declara la función
- Los **argumentos** son los valores reales pasados a esos parámetros y recibidos por la función
- En muchos sitios **parámetros** y **argumentos** se utilizan indistintamente con el mismo significado

- Las funciones JavaScript no especifican el tipo de dato para ningún parámetro
- El JavaScript no ejecuta ningún chequeo de tipo en los argumentos pasados al invocar la función
- Las funciones JavaScript no tienen ningún control sobre la cantidad de argumentos recibidos

Argumentos de las funciones(2)

```
function demo(nombre, edad) {  
  return nombre + " con edad " + edad;  
}
```

```
var personaEdad = demo("Juan");
```

- Si una función es invocada con menos argumentos que parámetros han sido definidos en su declaración, estos tendrán el valor indefinido (undefined)
- En este caso nombre será "Juan" y edad "undefined"

Argumentos de las funciones(3)

```
function demo(nombre = "Juan", edad = 22) {  
  return nombre + " con edad " + edad;  
}  
  
var personaEdad1 = demo();  
var personaEdad2 = demo("María");  
var personaEdad3 = demo("Elena", 25);  
var personaEdad4 = demo(undefined, 30);
```

- Esta variante de demo tiene a nombre “Juan” y a edad 22 por defecto
- **Caso 1:** al invocar la función sin argumentos, nombre es “Juan” y edad 22
- **Caso 2:** al invocar la función solo con “María”, nombre es “María” y edad 22
- **Caso 3:** al invocar la función con “Elena” y 25 estos serán los valores respectivos para nombre y edad
- **Caso 4:** al invocar la función con nombre indefinido y edad 30, el nombre será “Juan” y la edad 30

El objeto Arguments(1)

```
function maximoEntero() {  
    var maximo = -Infinity;  
    for (var i = 0; i < arguments.length; i++) {  
        if (arguments[i] > maximo) {  
            maximo = arguments[i];  
        }  
    }  
    return maximo;  
}  
  
var max = maximoEntero(1, 123, 500, 115, 44, 88);  
  
console.log("Maximo = ", max);
```

- Es posible crear una función que lea cualquier número de argumentos y calcule el máximo de ellos
- Solo tenemos que recorrer en un bucle “for” el objeto `arguments`, donde las funciones de JavaScript guardan los argumentos que se pasan cuando son invocadas

El objeto Arguments(2)

```
function suma() {  
    var sum = 0;  
    for (var i = 0; i < arguments.length; i++) {  
        sum += arguments[i];  
    }  
    return sum;  
}  
  
var s = suma(1, 123, 500, 115, 44, 88);  
  
console.log("Suma = ", s);
```

- Con la misma técnica anterior, es posible crear una función que sume el total de todos sus argumentos