



Ministerio de
**TRABAJO, EMPLEO
Y SEGURIDAD SOCIAL**



*Paraguay
de la gente*

Operadores y Funciones Javascript

Definición

Los operadores permiten manipular el valor de las variables, realizar operaciones matemáticas con sus valores y comparar diferentes variables. De esta forma, los operadores permiten a los programas realizar cálculos complejos y tomar decisiones lógicas en función de comparaciones y otros tipos de condiciones.

Operadores

Asignación

El operador de asignación es el más utilizado y el más sencillo. Este operador se utiliza para guardar un valor específico en una variable. El símbolo utilizado es = (no confundir con el operador == que se verá más adelante):

```
var numero1 = 3;
```

Estos dos Incremento y decremento

operadores solamente son válidos para las variables numéricas y se utilizan para incrementar o decrementar en una unidad el valor de una variable.

Ejemplo:

```
var numero = 5;
```

```
++numero;
```

```
alert(numero); // numero = 6
```

De forma equivalente, el operador decremento (indicado como un prefijo -- en el nombre de la variable) se utiliza para decrementar el valor de la variable:

```
var numero = 5;  
--numero;  
alert(numero); // numero = 4
```

Lógicos

Los operadores lógicos son imprescindibles para realizar aplicaciones complejas, ya que se utilizan para tomar decisiones sobre las instrucciones que debería ejecutar el programa en función de ciertas condiciones.

El resultado de cualquier operación que utilice operadores lógicos siempre es un valor lógico o *booleano*.

- **Negación**

Uno de los operadores lógicos más utilizados es el de la negación. Se utiliza para obtener el valor contrario al valor de la variable:

```
var visible = true;  
alert(!visible); // Muestra "false" y no "true"
```

- **AND**

La operación lógica AND obtiene su resultado combinando dos valores booleanos. El operador se indica mediante el símbolo && y su resultado solamente es true si los dos operandos son true:

```
valor1 = true;  
valor2 = true;  
resultado = valor1 && valor2; // resultado = true
```

- **OR**

La operación lógica OR también combina dos valores booleanos. El operador se indica mediante el símbolo `||` y su resultado es true si alguno de los dos operandos es true:

```
var valor1 = true;  
var valor2 = false;  
resultado = valor1 || valor2; // resultado = true
```

Matemáticos

JavaScript permite realizar manipulaciones matemáticas sobre el valor de las variables numéricas. Los operadores definidos son: suma (+), resta (-), multiplicación (*) y división (/). Ejemplo:

```
var numero1 = 10;  
var numero2 = 5;  
  
resultado = numero1 / numero2; // resultado = 2  
resultado = 3 + numero1;      // resultado = 13  
resultado = numero2 - 4;      // resultado = 1  
resultado = numero1 * numero 2; // resultado = 50
```

El operador módulo en JavaScript se indica mediante el símbolo %, que no debe confundirse con el cálculo del porcentaje:

```
var numero1 = 10;  
var numero2 = 5;  
resultado = numero1 % numero2; // resultado = 0
```

Relacionales

Los operadores relacionales definidos por JavaScript son idénticos a los que definen las matemáticas: mayor que (>), menor que (<), mayor o igual (>=), menor o igual (<=), igual que (==) y distinto de (!=).

Los operadores que relacionan variables son imprescindibles para realizar cualquier aplicación compleja, como se verá en el siguiente capítulo de programación avanzada. El resultado de todos estos operadores siempre es un valor booleano:

```
var numero1 = 3;  
var numero2 = 5;  
resultado = numero1 > numero2; // resultado = false  
resultado = numero1 < numero2; // resultado = true
```

Se debe tener especial cuidado con el operador de igualdad (==), ya que es el origen de la mayoría de errores de programación, incluso para los usuarios que ya tienen cierta experiencia desarrollando scripts. El operador == se utiliza para comparar el valor de dos variables, por lo que es muy diferente del operador =, que se utiliza para asignar un valor a una variable:

```
// El operador "=" asigna valores  
var numero1 = 5;  
resultado = numero1 = 3; // numero1 = 3 y resultado = 3
```

```
// El operador "==" compara variables  
var numero1 = 5;  
resultado = numero1 == 3; // numero1 = 5 y resultado = false
```

Los operadores relacionales también se pueden utilizar con variables de tipo cadena de texto:

```
var texto1 = "hola";  
var texto2 = "hola";  
var texto3 = "adios";
```

```
resultado = texto1 == texto3; // resultado = false  
resultado = texto1 != texto2; // resultado = false  
resultado = texto3 >= texto2; // resultado = false
```

Archivo codigo.js	Archivo index.html	Vista Navegador
<pre>//Demostracion Incremento var x = 5; ++x; document.getElementById("demoincremento").innerHTML = x //Demostracion Logico AND y=true z=true resultado = y && z; document.getElementById("demologico").innerHTML = resultado; //Demostracion Matematico var numero1 = 10; var numero2 = 5; division = numero1 / numero2; // resultado = 2 document.getElementById("demomatematico").innerHTML = division; //Demostracion Relacional var numero3 = 9; var numero4 = 3; m = numero3>numero4 document.getElementById("demorelacional").innerHTML = m;</pre>	<pre><!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta name="viewport" content="width=device-width, initial-scale=1.0"> <title>Document</title> </head> <body> <h2>JavaScript Operadores</h2> <p>Demo Incremento:</p> <p id="demoincremento"></p> <p>Demo Logico:</p> <p id="demologico"></p> <p>Demo Matematica division:</p> <p id="demomatematico"></p> <p>Demo Relacional:</p> <p id="demorelacional"></p> <script src="codigo.js"></script> </body> </html></pre>	<div>JavaScript Operadores</div> <div>Demo Incremento:</div> <div>6</div> <div>Demo Logico:</div> <div>true</div> <div>Demo Matematica division:</div> <div>2</div> <div>Demo Relacional:</div> <div>true</div>

Estructuras

Estructura if

La estructura más utilizada en JavaScript y en la mayoría de lenguajes de programación es la estructura if. Se emplea para tomar decisiones en función de una condición. Su definición formal es:

```
if(condicion) {  
    ...  
}
```

Ejemplo:

```
var mostrarMensaje = true;
```

```
if(mostrarMensaje) {  
    alert("Hola Mundo");  
}
```

Existe una variante de la estructura if llamada if...else. Su definición formal es la siguiente:

```
if(condicion) {
```

```
  ...
```

```
}
```

```
else {
```

```
  ...
```

```
}
```

Si la condición se cumple (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro del if().

Si la condición no se cumple (es decir, si su valor es false) se ejecutan todas las instrucciones contenidas en else { }.

Ejemplo:

```
var edad = 18;
```

```
if(edad >= 18) {
```

```
  alert("Eres mayor de edad");
```

```
}
```

```
else {
```

```
  alert("Todavía eres menor de edad");
```

```
}
```

Archivo codigo.js	Archivo index.html	Vista Navegador
<pre>var x; var z; x = 5; z = 5; if(x==z){ alert("Los numeros son iguales"); }</pre>	<pre><!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta name="viewport" content= "width=device-width, initial- scale=1.0"> <title>Document</title> </head> <body> <h2>Demostracion estructura if</h2> <script src="codigo.js"></script> </body> </html></pre>	<div>Una página incorporada en esta página dice Los numeros son iguales Aceptar</div> <div>Demostracion estructura if</div>

Estructura for

La idea del funcionamiento de un bucle for es la siguiente: "mientras la condición indicada se siga cumpliendo, repite la ejecución de las instrucciones definidas dentro del for. Además, después de cada repetición, actualiza el valor de las variables que se utilizan en la condición".

La "inicialización" es la zona en la que se establece los valores iniciales de las variables que controlan la repetición.

La "condición" es el único elemento que decide si continua o se detiene la repetición.

La "actualización" es el nuevo valor que se asigna después de cada repetición a las variables que controlan la repetición.

var mensaje = "Hola, estoy dentro de un bucle";

```
for(var i = 0; i < 5; i++) {  
    alert(mensaje);  
}
```

Estructura for...in

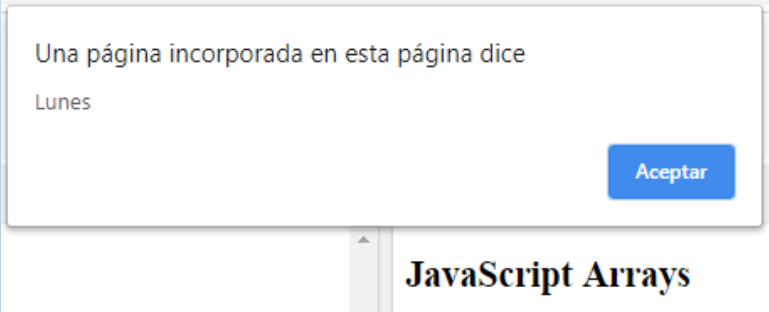
Una estructura de control derivada de for es la estructura for...in. Su definición exacta implica el uso de objetos, que es un elemento de programación avanzada que no se va a estudiar. Por tanto, solamente se va a presentar la estructura for...in adaptada a su uso en arrays. Su definición formal adaptada a los arrays es:

```
for(indice in array) {  
    ...  
}
```

Si se quieren recorrer todos los elementos que forman un array, la estructura for...in es la forma más eficiente de hacerlo, como se muestra en el siguiente ejemplo:

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];
```

```
for(i in dias) {  
    alert(dias[i]);  
}
```

Archivo codigo.js	Archivo index.html	Vista Navegador
<pre>var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"]; for(i in dias) { alert(dias[i]); }</pre>	<pre><!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta name="viewport" content= "width=device-width, initial- scale=1.0"> <title>Document</title> </head> <body> <h2>JavaScript Arrays</h2> <script src="codigo.js"></script> </body> </html></pre>	

Funciones

Las funciones son la base de la modularidad en JavaScript. Son utilizadas para reutilizar código, ocultar información y abstracción. Por norma general, las funciones son utilizadas para especificar el comportamiento de los objetos, aunque pueden definirse funciones al margen de los objetos.

Las funciones en JavaScript se definen mediante la palabra reservada `function`, seguida del nombre de la función.

Sintaxis

FUNCIONES CON PARÁMETROS Y SIN PARÁMETROS

Una función JavaScript puede requerir ser llamada pasándole cierta información o no requerir información.

Definición de una función sin parámetros (no requiere información):

```
//Comentario descriptivo de qué hace la función
```

```
function nombreDeLaFunción () {
```

```
//Código de la función
```

```
}
```

Definición de una función con parámetros (requiere información):

```
//Comentario descriptivo de qué hace la función
```

```
function nombreDeLaFunción (param1, param2, ..., paramN) {
```

```
//Código de la función
```

```
}
```


FUNCIONES QUE DEVUELVEN UN RESULTADO. RETURN.

Una función JavaScript puede devolver un resultado si se introduce la sentencia `return resultado`; donde `resultado` es aquello que queremos devolver (normalmente una variable que contiene un valor numérico, de texto o booleano, pero también podrían ser objetos con mayor complejidad como un array).

Una vez se llega a la sentencia `return` se produce la devolución del resultado y se interrumpe la ejecución de la función. Por ello la sentencia `return` será normalmente la última instrucción dentro de una función.

Definición de una función sin parámetros que devuelve un resultado:

```
//Comentario descriptivo de qué hace la función
function nombreDeLaFunción () {
//Código de la función
return resultado;
}
```

Definición de una función con parámetros que devuelve un resultado:

```
//Comentario descriptivo de qué hace la función
function nombreDeLaFunción (param1, param2, ..., paramN) {
//Código de la función
return resultado;
}
```

Archivo codigo.js	Archivo index.html	Vista Navegador
<pre>function conversionCelsius(f) { return (5/9) * (f-32); } document.getElementById("demo").innerHTMLHTML = conversionCelsius(77);</pre>	<pre><!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta name="viewport" content= "width=device-width, initial- scale=1.0"> <title>Document</title> </head> <body> <h2>Demostracion Funciones Javascript</h2> <p>Conversion de Fahrenheit a Celsius:</p> <p id="demo"></p> <script src="codig o.js"></script> </body> </html></pre>	<div>Demostracion Funciones Javascript Conversion de Fahrenheit a Celsius: 25</div>

Algunas funciones útiles

Funciones útiles para arrays

A continuación se muestran algunas de las funciones más útiles para el manejo de arrays:

length, calcula el número de elementos de un array

```
var vocales = ["a", "e", "i", "o", "u"];
```

```
var numeroVocales = vocales.length; // numeroVocales = 5
```

pop(), elimina el último elemento del array y lo devuelve. El array original se modifica y su longitud disminuye en 1 elemento.

```
var array = [1, 2, 3];
```

```
var ultimo = array.pop();
```

```
// ahora array = [1, 2], ultimo = 3
```

push(), añade un elemento al final del array. El array original se modifica y aumenta su longitud en 1 elemento. (También es posible añadir más de un elemento a la vez)

```
var array = [1, 2, 3];
```

```
array.push(4);
```

```
// ahora array = [1, 2, 3, 4]
```

Fuentes Consultadas

<https://www.aprenderaprogramar.com/>

<https://uniwebsidad.com/libros/javascript>

https://www.w3schools.com/js/js_functions.asp