Fundamentos de Programación con JavaScript

05

Operadores

Operadores

- Operadores de asignación
- Operadores aritméticos
- Operadores de comparación
- Operadores lógicos

- **Operador de cadena**
- **Operador ternario**
- Otros operadores

Operadores de **Asignación**

Nombre	Operador abreviado	Significado
Operadores de asignación	`x = y`	`x = y`
Asignación de adición	`x += y`	`x = x + y`
Asignación de sustracción	`x -= y`	`x = x - y`
Asignación de multiplicación	`x *= y`	`x = x * y`
Asignación de división	`x /= y`	`x = x / y`
Asignación de resto	`x %= y`	`x = x % y`
Asignación de exponenciación	`x **= y`	`x = x ** y`

Operadores **Aritméticos**

Operador	Descripción	Ejemplo
Resto (`%`)	Operador binario correspondiente al módulo de una operación. Devuelve el resto de la división de dos operandos.	`12 % 5` devuelve `2`.
Incremento (`++`)	Operador unario. Incrementa en una unidad al operando. Si es usado antes del operando (`++x`) devuelve el valor del operando después de añadirle 1 y si se usa después del operando (`x++`) devuelve el valor de este antes de añadirle 1.	Si `x` es `3`, entonces `++x` establece `x` a `4` y devuelve `4`, mientras que `x++` devuelve `3` y, solo después de devolver el valor, establece `x` a `4`.
Decremento (`--`)	Operador unario. Resta una unidad al operando. Dependiendo de la posición con respecto al operando tiene el mismo comportamiento que el operador de incremento.	Si `x` es `3`, entonces `--x` establece `x` a `2` y devuelve `2`, mientras que `x--` devuelve `3` y, solo después de devolver el valor, establece `x` a `2`.
Negación Unaria (`-`)	Operación unaria. Intenta convertir a número al operando y devuelve su forma negativa.	`-"3"` devuelve `-3`. `-true` devuelve `-1`.
Unario positivo (`+`)	Operación unaria. Intenta convertir a número al operando.	`+"3"` devuelve `3`. `+true` devuelve `1`.
Exponenciación (`**`)	Calcula la potencia de la base al valor del exponente. Es equivalente a $base^{exponente}$	`2 ** 3` devuelve `8`. `10 ** -1` devuelve `0.1`.

Operadores **Comparación**

Operador	Descripción	Ejemplos: devolviendo true
Igualdad (==)	Devuelve true si ambos operadorandos son iguales.	`3 == "3"` `"3" == var1`
Desigualdad (!=)	Devuelve true si ambos operandos no son iguales.	`var1 != 4` `var2 != "3"`
Estrictamente iguales (===)	Devuelve true si los operandos son iguales y tienen el mismo tipo.	`3 === var1`
Estrictamente desiguales (!==)	Devuelve true si los operandos no son iguales y/o no son del mismo tipo.	`var1 !== "3"` `3 !== "3"`
Mayor que (>)	Devuelve true si el operando de la izquierda es mayor que el operando de la derecha.	`var2 > var1` `"12" > 2`
Mayor o igual que (>=)	Devuelve true si el operando de la izquierda es mayor o igual que el operando de la derecha.	`var2 >= var1` `var1 >= 3`
Menor que (<)	Devuelve true si el operando de la izquierda es menor que el operando de la derecha.	`var1 < var2` `"2" < 12`
Menor o igual que (<=)	Devuelve true si el operando de la izquierda es menor o igual que el operando de la derecha.	`var1 <= var2` `var2 <= 5`

Operadores de **comparación**

JavaScript tiene 4 operadores de comparación

- Menor: **<**
- Menor o igual: **<=**
- Mayor: **>**
- Mayor o igual: **>=**

Utilizar comparaciones solo con números (number)

- poseen una relación de orden bien definida

```
1.2 < 1.3     => true

1 <  1        => false
1 <= 1        => true
1 >  1        => false
1 >= 1        => true

false < true  => true

"a" < "b"     => true
"a" < "aa"    => true
```

No se recomienda utilizar con otros tipos: **string**, **boolean**, **object**, ..

- Las relación de orden en estos tipos existe, pero es muy poco intuitiva

Operadores de **identidad e igualdad**

Identidad o igualdad estricta: **<v1> === <v2>**

- igualdad de tipo y valor:
  - aplicable a: **number**, **boolean** y **strings**
- En objetos es **identidad de referencias**

Desigualdad estricta: **<v1> !== <v2>**

- negación de la igualdad estricta

Igualdad y desigualdad débil: **==** y **!=**

- **No utilizar!** Conversiones impredecibles!

```
// Identidad de tipos básicos

0 === 0          => true
0 === 0.0        => true
0 === 0.00       => true

0 === 1          => false
0 === false      => false

'2' === "2"      => true
'2' === "02"     => false

'' === ""        => true
'' === " "       => false
```

Operadores **Lógicos**

Operador	Uso	Descripción
AND Lógico (&&)	`expr1 && expr2`	Devuelve `expr1` si puede ser convertido a false de lo contrario devuelve `expr2.` Por lo tanto, cuando se usa con valores booleanos, `&&` devuelve `true` si ambos operandos son `true`, en caso contrario devuelve `false`.
OR Lógico (||)	`expr1 || expr2`	Devuelve `expr1` si puede ser convertido a `true` de lo contrario devuelve `expr2.` Por lo tanto, cuando se usa con valores booleanos, `||` devuelve `true` si alguno de los operandos es `true`, o `false` si ambos son `false`.
NOT Lógico (!)	`!expr`	Devuelve `false` si su operando puede ser convertido a `true`, en caso contrario, devuelve `true`.

Operador **y** **&&** **AND**

Operador lógico **y** **(AND)**

- operador binario: **<valor_1> y <valor_2>**
  - Verdadero solo si ambos valores son verdaderos

&& se ha extendido y es más que un operador lógico

- No convierte el resultado a booleano
  - Solo **interpreta <valor_1> como booleano** y según sea **false** o **true**
    - Devuelve como resultado **<valor_1>** o **<valor_2>** sin modificar

```
true  && true    => true
false && true    => false
true  && false   => false
false && false   => false


0 && true        => 0
1 && "5"         => "5"
```

Semántica del operador lógico y (and) de JavaScript: **<valor_1> && <valor_2>**

- si **<valor_1>** se evalúa a **false**
  - devuelve **<valor_1>**, sino devuelve **<valor_2>**

Operador **o** **||** **OR**

Operador lógico **o** **(OR)**

- operador binario: **<valor_1> o <valor_2>**
  - Verdadero solo si ambos valores son verdaderos

```
true  || true    => true
false || true    => true
true  || false   => true
false || false   => false

undefined || 0       => 0
13 || 0              => 13

// Asignar valor por defecto
// si x es undefined o null

x = x || 0;
```

|| se ha extendido y es más que un operador lógico

- No convierte el resultado a booleano
  - Solo **interpreta <valor_1> como booleano** y según sea **false** o **true**
    - Devuelve como resultado **<valor_1>** o **<valor_2>** sin modificar

Semántica del operador lógico o (or) de JavaScript: **<valor_1> || <valor_2>**

- si **<valor_1>** se evalúa a **true**
  - devuelve **<valor_1>**, sino devuelve **<valor_2>**

Tipo **booleano**

El tipo **boolean** (booleano) solo tiene solo 2 valores

- **true**: verdadero
- **false**: falso

!false	=> true
!true	=> false

Operador unitario **negación** (negation): **! <valor booleano>**

- Convierte un valor lógico en el opuesto, tal y como muestra la tabla

Las expresiones booleanas, también se denominan expresiones lógicas

- Se evalúan siempre a verdadero (true) o falso (false)
  - Se utilizan para **tomar decisiones en sentencias condicionales**: if/else, bucles,
    - Por ejemplo: **if (expresión booleana) {Acciones_A} else {Acciones_B}**

Conversión a **booleano**

La regla de conversión de otros tipos a booleano es

- **false: 0, -0, NaN, null, undefined, "", ''**
- **true: resto de valores**

Cuando el operador negación **!** se aplica a otro tipo

- convierte el valor a su equivalente booleano
  - y obtiene el valor booleano opuesto

```
!4          => false
!"4"        => false
!null       => true
!0          => true

!!""        => false
!!4         => true
```

El operador negación aplicado 2 veces permite obtener

- el booleano equivalente a un valor de otro tipo, por ejemplo **!!1 => true**

Operador **ternario** (condicional)

El operador condicional es el único operador de JavaScript que necesita tres operandos. El operador asigna uno de dos valores basado en una condición. La sintaxis de este operador es:

```
condición ? valor1 : valor2
```

Si la condición es `true`, el operador tomará el `valor1`, de lo contrario tomará el `valor2`.
Puedes usar el operador condicional en cualquier lugar que use un operador estándar.

Por ejemplo,

```
var estado = (edad >= 18) ? "adulto" : "menor";
```

Esta sentencia asigna el valor `adulto` a la variable `estado` si `edad` es mayor o igual a `18`, de lo contrario le asigna el valor `menor`.

Operador de asignación condicional ? :

El operador de asignación condicional

?:

- devuelve un valor en función de una **condición** lógica
  - La condición lógica va siempre entre paréntesis

Semántica de la asignación condicional: **(condición) ? <valor_1> : <valor_2>**

- si **condición** se evalúa a **true**
  - devuelve **<valor_1>**, sino devuelve **<valor_2>**

(true) ? 0 : 7	=> 0
(false)? 0 : 7	=> 7

(7) ? 0 : 7	=> 0
(“”) ? 0 : 7	=> 7

Operador de **cadena de caracteres**

Además de los operadores de comparación, que pueden ser usados en cadenas de caracteres, el operador de concatenación `(+)` une dos valores de tipo String, devolviendo otro String correspondiente a la unión de los dos operandos.

Por ejemplo,

```
console.log("mi " + "string"); // lanza el String "mi string" en la consola.
```

La versión acortada de este operador de asignación `(+=)` puede ser usada también para concatenar cadenas de caracteres.

Por ejemplo,

```
var mistring = "alfa";
mistring += "beto"; // devuelve "alfabeto" y asigna este valor a "mistring".
```

Operador **typeof**

El operador typeof es usado de las siguientes maneras:

1.
```
typeof operando
```

2.
```
typeof (operando)
```

El operador `typeof` devuelve una cadena de caracteres indicando el tipo del operando evaluado. En los ejemplos anteriores `operando` hace referencia a la cadena de caracteres, variable, palabra clave u objeto del que se intenta obtener su tipo. Los paréntesis son opcionales.

Operador **coma**

El operador coma `(,)` simplemente evalúa ambos operandos y retorna el valor del último. Este operador es ante todo utilizado dentro de un ciclo *`for`*, permitiendo que diferentes variables sean actualizadas en cada iteración del ciclo.

Por ejemplo, si `a` es un Array bi-dimensional con 10 elementos en cada lado, el siguiente código usa el operador coma para actualizar dos variables al mismo tiempo. El código imprime en la consola los valores correspondientes a la diagonal del Array:

```
for (var i = 0, j = 9; i <= j; i++, j--)
  console.log("a[" + i + "][" + j + "]= " + a[i][j]);
```

Operadores **unarios**

Una operación unaria es una operación que sólo necesita un operando.

`delete` La función del operador `delete` es eliminar un objeto, una propiedad de un objeto, o un elemento en el indice específico de un Array. La sintaxis es la siguiente:

```
1  delete nombreObjeto;
2  delete nombreObjeto.propiedad;
3  delete nombreObjeto[indice];
4  delete propiedad; // solo admitido con una declaración "with"
```

Donde `nombreObjeto` es el nombre de un objeto, `propiedad` el nombre de la propiedad de un objeto, e `indice` un entero que representa la localización de un elemento en un Array.

La cuarta forma es admitida solo dentro de una sentencia with, para eliminar una propiedad de un objeto.

Puedes usar el operador `delete` para eliminar aquellas variables que han sido declaradas implícitamente, pero no aquellas que han sido declaradas con `var`.

Si la operación `delete` finaliza con éxito, establece la propiedad o el elemento a `undefined`.
El operador `delete` devuelve `true` si la operación ha sido posible y `false` en caso contrario.

Operador **void**

1.
```
void (expresion)
```

2.
```
void expresion
```

El operador `void` especifica una expresión que será evaluada y no retornará ningún resultado. En los ejemplos anteriores, `expresion` hace referencia a la expresión que será evaluada. Si bien los paréntesis que envuelven a la expresión son opcionales, en cuanto a estilo del código, es una buena práctica usarlos.

El operador `void` puede ser usado para especificar una expresión como un link de hipertexto. La expresión será evaluada pero no cargará una página en el documento actual.

El siguiente código crea un link de hipertexto que no hace nada cuando el usuario hace click en él. Cuando el usuario hace click sobre el link, `void(0)` será evaluada como `undefined`, lo cual no tiene efecto en **JavaScript**.

```
<a href="javascript:void(0)">Haz click aquí para no hacer nada</a>
```

Fundamentos de Programación con **JavaScript**