JAVASCRIPT

Lección 4: Operadores en JS

eip ESCUELA INTERNACIONAL POSGRADOS

ÍNDICE

Operadores en JS

PRESENTACIÓN Y OBJETIVOS

En esta unidad se estudiarán los diferentes tipos de operadores existentes en JS.

También vamos a ver las nuevas formas (desde 2015) de declarar una variable.

Objetivos

En esta unidad aprenderás:

- ✓ **Que string, number y boolean, que hasta ahora sabíamos que son primitivas, también existen como objetos**
- ✓ **Los tipos de operadores en JS**
- ✓ **Las nuevas palabras clave para declarar una variable**
- ✓ **La diferencia entre cada palabra clave**

1 STRING / NUMBER / BOOLEAN COMO OBJETOS

JS hace diferencia entre un objeto String y una primitiva String. Lo mismo aplica para los Booleans y Numbers.

```
var stringObject = new String("Roxana") //object
var primitiveString = "Roxana" // string

var numberObject = new Number(2) // object
var primitiveNumber = 2 // number

var booleanObject = new Boolean(true) // object
var primitiveBoolean = true // boolean
```

Para acceder al valor primitivo de un object string / number / boolean se puede usar el método *valueOf().*

```
var stringObject = new String("Roxana") //object
console.log(stringObject) // String { "Roxana" }
console.log(stringObject.valueOf()) // Roxana

var numberObject = new Number(2) // object
console.log(numberObject) // Number { 2 }
console.log(numberObject.valueOf()) // 2

var booleanObject = new Boolean(true) // object
console.log(booleanObject) // Boolean { true }
console.log(booleanObject.valueOf()) // true
```

El uso de String / Number / Boolean object no se recomienda. Complica el código y ralentiza la velocidad de ejecución. También puede producir resultados inesperados, especialmente cuando se trate de verificar la igualdad entre variables.

Las primitivas no tienen métodos. Cuando intentamos llamar a una propiedad o a un método en una primitiva, JS envuelve el valor en un objeto (wrapper object) y llama el método en el wrapper.

2 OPERADORES EN JS

Hay diferentes tipos de operadores en JS.

2.1 Operadores aritméticos

Operador	Descripción
+	Adición
-	Sustracción
*	Multiplicación
/	División
**	Exponenciación
%	Modulo
++	Incremento
--	Decremento

2.1.1 Coerción (Type Coercion) y Conversión (Type Conversion)

El termino coerción se refiere a la conversión implícita de valores de un tipo de dato a otro.

Es similar a la conversión de tipo (**Type Conversion** o Type Casting). La conversión de tipo puede ser explicita.

```
String(123) // explicito
// devuelve '123'
123 + '1' // implícito - 123 se convierte automáticamente en cadena
// devuelve '1231'
```

2.1.2 Operadores aritméticos – detalles

Usar el operador de adición (+) con diferentes tipos de datos:

- STRING – tiene efecto de concatenación
- NUMBER – tiene efecto de adición
- STRING & (NUMBER / BOOLEAN / OBJECT / NULL/ UNDEFINED) – ocurre una coerción; JS convierte automáticamente y de manera implícita (type coercion) el number/boolean/object/null/undefined en una cadena y lo concatena con la primera cadena.

- +STRING & NUMBER – el operador (+) fuerza la cadena a ser un numero (usando Number(string)); Si el numero obtenido es válido, se hace la adición de los 2 números; Si no, devuelve un NaN.

Usar los operadores (-, /, *) con diferentes tipos de datos:

- STRING – JS usa coerción y convierte las cadenas en números (usando Number(string)); Si los numero son válidos, se hace la sustracción / multiplicación / división; Si no, devuelve un NaN.
- NUMBER – tiene efecto de sustracción / división / multiplicación

2.1.3 Precedencia de los operadores

Como en matemática, los paréntesis (), tienen más precedencia que multiplicación (*) y división (/), que tiene más precedencia que adición (+) y sustracción (-).

2.2 Operadores de comparación

Operador	Descripción
==	Igual
===	Estrictamente igual
!=	Distinto
!==	Estrictamente distinto
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que

Presta atención

Cuando comparamos (<, <=, >, >=) dos cadenas de texto, la comparación se hace alfabéticamente.

Tricky Example
"2" > "12" // devuelve true porque alfabéticamente el 2 está después del 1.

Asegúrate siempre que los tipos de datos que estas comparando, son los deseados.

La diferencia entre igual (==) y estrictamente igual (===) es que la última tiene en cuenta el tipo de dato también.

```
1 == '1 ' // true
1 === '1 ' // false
```

2.2.1 El operador igual (==)

El operador chequea si dos operandos son iguales. El resultado retornado es un booleano (true o false). Antes de comparar, intenta convertir los tipos de datos que sean distintos.

Reglas de comparación:

- Si los operandos son del mismo tipo, se comparan así
  - STRING: true sólo si tienen los mismos caracteres en el mismo orden
  - NUMBER / BIGINT: true sólo si tienen el mismo valor; +0 y -0 se consideran como el mismo valor; Si uno de los operandos es NaN, devuelve false (NaN nunca es igual a NaN)
  - BOOLEAN: true sólo si los dos operandos son true o sólo si los dos operandos son false
  - OBJECT: true si los dos operandos referencian el mismo objeto
  - SYMBOL: true si los dos operandos referencian el mismo símbolo

- Si los 2 operandos son null o undefined, devuelve true

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

- Si los 2 operandos son String y Number, la cadena se convierte en número; si eso falla, devuelve un NaN, que comparado con cualquier cosa devuelve false

2.2.2 El operador estrictamente igual (===)

A diferencia del operador igual (==), el (===) no hace ninguna conversión de tipos. El resultado es un boolean (true o false).

Reglas de comparación:

- Si los operandos son de tipos diferentes, devuelve false
- STRING: true sólo si tienen los mismos caracteres en la misma orden
- NUMBER / BIGINT: true sólo si tienen el mismo valor; +0 y -0 se consideran como el mismo valor; Si uno de los operandos es NaN, devuelve false (NaN nunca es igual a NaN)
- BOOLEAN: true sólo si los dos operandos son true o sólo si los dos operandos son false
- OBJECT: true si los dos operandos referencian el mismo objeto
- SYMBOL: true si los dos operandos referencian el mismo símbolo
- Si los 2 operandos son null o si los 2 son undefined, devuelve true

2.3 Operadores lógicos

Operador	Descripción
&&	Conjunción (y)
||	Disyunción (o)
!	Negación

Operador	Operación	Resultado
&&	true && true	true
	true && false	false
	false && true	false
	false && false	false
||	true || true	true
	true || false	true
	false || true	true
	false || false	false
!	!true	false
	!false	true

2.3.1 Evaluación de cortocircuito

Las expresiones lógicas se evalúan de la izquierda a la derecha. A veces, hay situaciones de cortocircuito.

Evaluación de cortocircuito

false && cualquiercosa

Se evalúa en false automáticamente, sin revisar el valor de "cualquiercosa"

true || cualquiercosa

Se evalúa en true automáticamente, sin revisar el valor de "cualquiercosa"

** Mucho cuidado porque la expresión "cualquiercosa" nunca llega a ser ejecutada.

2.4 Operadores de asignación

Operador	Ejemplo	Explicación
=	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

3 VALORES TRUTHY Y FALSY

En JS existen valores truthy (verdaderos) o falsy (falsos) cuando estamos considerando un contexto booleano.

Todos los valores son por defecto truthy, excepto los siguientes que son falsy.

Valores falsy:

> False
> 0
> -0
> +0
> 0n
> null
> undefined
> NaN
> `""`

Ejemplos **de valores truthy** en contexto booleano:

> true
> 42
> "false"
> 3.43
> "Roxana"

Existe una técnica, usando doble negación (!!), para forzar la conversión de cualquier valor a un boleado primitivo (true o false). Esta conversión está basada en la truthyness o falsyness del valor.

```
!!"Roxana" // true
 !!"false" // true
 !!{} // true
 !![] // true
 !!0 // false
 !!"" // false
 !!2 // true
 !!{name: "Roxana"} // true
```

4 VAR vs LET vs CONST

Var, let, y const son palabras reservadas para la declaración de variables.

Desde 1995 hasta 2015, sólo existía var. En 2015 (ES6) fueron añadidas 2 palabras nuevas para la declaración de variables: let y const.

4.1 VAR

- La forma antigua de declarar una variable
- Dependiendo de dónde se ejecuta, puede tener un ámbito global o sólo para la función que la contiene
- Puede ser re-declarada y reasignada

El hecho de que una variable puede ser re-declarada, puede causar problemas. Por eso... LET IS THE NEW VAR

4.2 LET

- La forma nueva de declarar una variable
- Resuelve el problema de var
- Aparecido en ES6, en 2015
- Ámbito o alcance de bloque (block-scoped)
- Puede ser reasignada pero no puede ser re-declarada

4.3 CONST

- La otra forma nueva de declarar una variable
- Se usa para constantes (valores que no van a cambiar)
- Aparecido en ES6, en 2015
- Ámbito o alcance de bloque (block-scoped)
- No puede ser reasignada, ni re-declarada
- Necesita ser inicializada desde el principio

```
const age = 20 // correct
const age // Uncaught SyntaxError: Missing initializer in const declaration
```

5 PUNTOS CLAVE

- String, Number, Boolean pueden ser primitivas, pero también objetos
- JS trata de forma diferente String como primitiva y String como objeto
- Esto aplica para Number y Boolean también
- Existen diferentes tipos de operadores en JS:
  - Aritméticos
  - Lógicos
  - De asignación
  - De comparación
- == y === son diferentes; el === tiene en cuenta también el tipo de dato
- Cuando JS compara dos tipos de datos diferentes (usando ==), automáticamente hace una coerción de tipos
- En el contexto booleano, existen valores truthy y falsy
- Hay un número limitado de valor falsy; todas los otros se consideran truthy
- Además de var, las variables se pueden declarar también con let y const
- let es el nuevo var
- const se usa para constantes