Capacitarte

Material Imprimible

Curso de JavaScript

Módulo 3

Operadores aritméticos de JavaScript

Los operadores aritméticos se utilizan para realizar operaciones aritméticas en números:

Operador	Descripción
+	Adición
-	Sustracción
*	Multiplicación
**	Exponenciación
/	División
%	Modulus (Remanente de la división)
++	Incremento
--	Decremento

Operadores de asignación de JavaScript

Los operadores de asignación asignan valores a las variables de JavaScript.

Operador	Ejemplo	Igual que
=	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 de JavaScript

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

var txt1 = "John";

var txt2 = "Doe";

var txt3 = txt1 + " " + txt2;

El resultado de txt3 será: John Doe

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

Ejemplo

var txt1 = "Qué gran ";

txt1 + = "día hoy";

El resultado de txt1 será:

Que gran dia hoy

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

Agregar cadenas y números

Agregar dos números devolverá la suma, pero agregar un número y una cadena devolverá una cadena:

var x = 5 + 5;

var y = "5" + 5;

var z = "Hello" + 5;

El resultado de x , y y z será:

10

55

Hello5

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

Operadores de comparación de JavaScript

Operador	Descripción
==	Igual a
===	igual valor y igual tipo
!=	Distinto
!==	no igual valor y no igual tipo
>	Mayor que
<	Menor que
>=	Mayor o igual a
<=	Menor o igual a
?	Operador ternario

Operadores lógicos de JavaScript

Operador	Descripción
&&	lógico and Y
||	lógico or O
!	lógico not NO

Operadores de tipo JavaScript

Operador	Descripción
typeof	Devuelve el tipo de una variable.
instanceof	Devuelve verdadero si un objeto es una instancia de un tipo de objeto

Tipos de datos de JavaScript

Las variables de 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 programación, los tipos de datos son un concepto importante.

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

Sin tipos de datos, una computadora no puede resolver esto con seguridad:

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

¿Tiene sentido agregar "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.

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

var x = 16 + 4 + "Volvo";

Resultado? ……

var x = "Volvo" + 16 + 4;

Resultado?....

Números de JavaScript

JavaScript tiene solo un tipo de números.

Los números se pueden escribir con o sin decimales:

var x1 = 34.00; // con decimales

var x2 = 34; // sin decimales

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

var y = 123e5; // 12300000

var z = 123e-5; // 0.00123

JavaScript booleanos

Los booleanos solo pueden tener dos valores: trueo false.

var x = 5;

var y = 5;

var z = 6;

(x == y) // Returns true

(x == z) // Returns false

<! DOCTYPE html>

<html>

<body>

<h2> Booleanos de JavaScript </h2>

 Los booleanos pueden tener dos valores: verdadero o falso:

<script>

```
var x = 5;
var y = 5;
var z = 6;
document.getElementById ("demo"). innerHTML =
(x == y) + "<br>" + (x == z);
</script>
</body>
</html>
```

Los booleanos se usan a menudo en pruebas condicionales.

Matrices JavaScript

Las matrices de JavaScript se escriben entre 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):

```
<! DOCTYPE html>
<html>
<body>
<h2> Matrices de JavaScript </h2>
<p> Los índices de matriz están basados en cero, lo que significa que el primer elemento es [0].
</p>
<p id = "demo"> </p>
<script>
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById ("demo"). innerHTML = cars [0];
</script>
</body>
</html>
```

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

Objetos JavaScript

Los objetos de JavaScript se escriben con llaves {}.

Las propiedades de los objetos se escriben como nombre: pares de valores, separados por comas.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Objects</h2>
<p id="demo"></p>
<script>
var person = {
 firstName : "John",
 lastName  : "Doe",
 age     : 50,
 eyeColor  : "blue"
};
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>
</body>
</html>
```

El objeto (persona) en el ejemplo anterior tiene 4 propiedades: nombre, apellido, edad y Color de ojos.

El tipo de operador

Puede usar el typeofoperador de JavaScript para encontrar el tipo de una variable de JavaScript.

El typeofoperador devuelve el tipo de una variable o una expresión:

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

```
<h2> JavaScript typeof </h2>
<p> El operador typeof devuelve el tipo de una variable o una expresión. </p>
<p id = "demo"> </p>
<script>
document.getElementById ("demo"). innerHTML =
typeof "" + "<br>" +
typeof "John" + "<br>" +
typeof "John Doe";
</script>
</body>
</html>

<! DOCTYPE html>
<html>
<body>

<h2> JavaScript typeof </h2>
<p> El operador typeof devuelve el tipo de una variable o una expresión. </p>
<p id = "demo"> </p>
<script>
document.getElementById ("demo"). innerHTML =
typeof 0 + "<br>" +
typeof 314 + "<br>" +
typeof 3.14 + "<br>" +
typeof (3) + "<br>" +
typeof (3 + 4);
</script>
</body>
</html>
```

Indefinido

En JavaScript, una variable sin valor tiene el valor undefined. El tipo también es undefined.

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

<h2>JavaScript</h2>

<p>The value (and the data type) of a variable with no value is <b>undefined</b>.</p>

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

<script>
var car;
document.getElementById("demo").innerHTML =
car + "<br>" + typeof car;
</script>

</body>
</html>
```

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

```
<! DOCTYPE html>
<html>
<body>
<h2> JavaScript </h2>
<p> Las variables se pueden vaciar si establece el valor en <b> indefinido </b>. </p>
<p id = "demo"> </p>
<script>
var car = "Volvo";
```

```
car = undefined;
document.getElementById ("demo"). innerHTML =
car + "<br>" + typeof car;
</script>

</body>
</html>
```

Valores vacíos

Un valor vacío no tiene nada que ver con undefined.

Una cadena vacía tiene tanto un valor legal como un tipo.

```
<! DOCTYPE html>
<html>
<body>
<h2> JavaScript </h2>
<p> Una cadena vacía tiene un valor legal y un tipo: </p>
<p id = "demo"> </p>
<script>
var car = "";
document.getElementById ("demo"). innerHTML =
"El valor es:" +
car + "<br>" +
"El tipo es:" + typeof car;
</script>
</body>
</html>
```

Nulo

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

Desafortunadamente, en JavaScript, el tipo de datos nulles un objeto.

Puede considerarlo un error en JavaScript que typeof nulles un objeto. Debería ser null.

Puede vaciar un objeto configurándolo en null:

```
<! DOCTYPE html>
<html>
<body>
<h2> JavaScript </h2>
<p> Los objetos se pueden vaciar estableciendo el valor en <b> null </b>. </p>
<p id = "demo"> </p>
<script>
var person = {firstName: "John", lastName: "Doe", edad: 50, eyeColor: "blue"};
persona = null;
document.getElementById ("demo"). innerHTML = typeof person;
</script>
</body>
</html>
```

También puede vaciar un objeto configurándolo en undefined:

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

<h2> JavaScript </h2>
<p> Los objetos se pueden vaciar estableciendo el valor en <b> undefined </b>. </p>
<p id = "demo"> </p>
<script>
var person = {firstName: "John", lastName: "Doe", edad: 50, eyeColor: "blue"};
person = undefined;
document.getElementById ("demo"). innerHTML = person;
</script>
```

```
</body>
</html>
```

Diferencia entre indefinido y nulo

undefinedy nullson iguales en valor pero diferentes en tipo:

```
<! DOCTYPE html>
<html>
<body>
<h2> JavaScript </h2>
<p> undefined y null son iguales en valor pero diferentes en tipo: </p>
<p id = "demo"> </p>

<script>
document.getElementById ("demo"). innerHTML =
typeof undefined + "<br>" +
typeof null + "<br> <br>" +
(null === undefined) + "<br>" +
(null == undefined);
</script>
</body>
</html>
```

Datos primitivos

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

El typeofoperador puede devolver uno de estos tipos primitivos:

- string
- number
- boolean

- undefined

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

<h2> JavaScript typeof </h2>
<p> El operador typeof devuelve el tipo de una variable o una expresión. </p>
<p id = "demo"> </p>
<script>
document.getElementById ("demo"). innerHTML =
typeof "juan" + "<br>" +
typeof 3.14 + "<br>" +
typeof true + "<br>" +
typeof false + "<br>" +
typeof x;
</script>
</body>
</html>
```

Datos complejos

El typeofoperador puede devolver uno de dos tipos complejos:

- function
- object

El typeofoperador devuelve "objeto" para objetos, matrices y nulos.

El typeofoperador no devuelve "objeto" para las funciones.

```
<! DOCTYPE html>
<html>
<body>
<h2> JavaScript typeof </h2>
<p> El operador typeof devuelve objetos para ambos objetos, matrices y nulos. </p>
```

 El operador typeof no devuelve objetos para funciones.

<script>
document.getElementById ("demo"). innerHTML =
typeof {nombre: 'john', edad: 34} + "
" +
typeof [1,2,3,4] + "
" +
typeof null + "
" +
typeof function myFunc () {};
</script>

</body>
</html>

El typeofoperador devuelve " object" para las matrices porque en JavaScript las matrices son objetos.

Ejercicio

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

Number String Object

Funciones de JavaScript

Una función de JavaScript es un bloque de código diseñado para realizar una tarea en particular.

Una función de JavaScript se ejecuta cuando "algo" lo invoca (lo llama).

<! DOCTYPE html>
<html>
<body>

```
<h2> Funciones de JavaScript </h2>
<p> Este ejemplo llama a una función que realiza un cálculo y devuelve el resultado: </p>
<p id = "demo"> </p>
<script>
function myFunction (p1, p2) {
  return p1 * p2;
}
document.getElementById ("demo"). innerHTML = myFunction (4, 3);
</script>

</body>
</html>
```

Sintaxis de la función de JavaScript

Una función de JavaScript se define con la palabra clave function, seguida de un **nombre** , seguido de paréntesis **()** .

Los nombres de funciones pueden contener letras, dígitos, guion bajo y signos de pesos (las mismas reglas que las variables).

Los paréntesis pueden incluir nombres de parámetros separados por comas:

(*parámetro1, parámetro2, ...*)

El código que se ejecutará, por la función, se coloca entre llaves: **{}**

```
 function name(parameter1, parameter2, parameter3) {
  // código para ser ejecutado
}
```

Los **parámetros de la** función se enumeran entre paréntesis () en la definición de la función.

Los **argumentos de la** función son los **valores** recibidos por la función cuando se invoca.

Dentro de la función, los argumentos (los parámetros) se comportan como variables locales.

Una función es muy parecida a un procedimiento o una subrutina, en otros lenguajes de programación.

Invocación de funciones

El código dentro de la función se ejecutará cuando "algo" **invoque** (llame) la función:

- Cuando ocurre un evento (cuando un usuario hace clic en un botón)
- Cuando se invoca (llama) desde el código JavaScript
- Automáticamente (auto invocado)

Función de retorno (RETURN)

Cuando JavaScript alcanza una declaración return, la función dejará de ejecutarse.

Si la función fue invocada desde una declaración, JavaScript "regresará" para ejecutar el código después de la declaración de invocación.

Las funciones a menudo calculan un **valor de retorno** . El valor devuelto se "devuelve" a la "persona que llama":

```
<! DOCTYPE html>
<html>
<body>
<h2> Funciones de JavaScript </h2>
<p> Este ejemplo llama a una función que realiza un cálculo y devuelve el resultado: </p>
<p id = "demo"> </p>
<script>
var x = myFunction (4, 3);
document.getElementById ("demo"). innerHTML = x;

function myFunction (a, b) {
  return a * b;
}
</script>

</body>
</html>
```

¿Por qué funciones?

Puede reutilizar el código: defina el código una vez y úselo muchas veces.

Puede usar el mismo código muchas veces con diferentes argumentos, para producir resultados diferentes.

```
<! DOCTYPE html>
<html>
<body>
<h2> Funciones de JavaScript </h2>
<p> Este ejemplo llama a una función para convertir de Fahrenheit a Celsius: </p>
<p id = "demo"> </p>
<script>
function toCelsius (f) {
  return (5/9) * (f-32);
}
document.getElementById ("demo"). innerHTML = toCelsius (77);
</script>
</body>

</html>
```

El operador () invoca la función

Usando el ejemplo anterior, se toCelsiusrefiere al objeto de función, y se toCelsius()refiere al resultado de la función.

Acceder a una función sin () devolverá el objeto de función en lugar del resultado de la función.

```
<! DOCTYPE html>
<html>
<body>
<h2> Funciones de JavaScript </h2>
```

```
<p> Acceder a una función sin () devolverá la definición de la función en lugar del resultado de la función: </p>
<p id = "demo"> </p>
<script>
function toCelsius (f) {
  return (5/9) * (f-32);
}
document.getElementById ("demo"). innerHTML = toCelsius;
</script>

</body>
</html>
```

Fuentes:

https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions_and_Operators

https://developer.mozilla.org/es/docs/Web/API/Document/getElementById

https://tutorialesenlinea.es/1937-funciones-predefinidas-por-el-lenguaje-js.html