Resumen de Javascript

Semana 3

Logical operators

Logical operators are typically used with Boolean (logical) values; when they are, they return a Boolean value. However, the && and || operators actually return the value of one of the specified operands, so if these operators are used with non-Boolean values, they may return a non-Boolean value. The logical operators are described in the following table.

Table 3.6 Logical operators

Operator	Usage	Description
&&	`expr1 && expr2`	(Logical AND) Returns `expr1` if it can be converted to false; otherwise, returns `expr2`. Thus, when used with Boolean values, && returns true if both operands are true; otherwise, returns false.
||	`expr1 || expr2`	(Logical OR) Returns `expr1` if it can be converted to true; otherwise, returns `expr2`. Thus, when used with Boolean values, || returns true if either operand is true; if both are false, returns false.
!	`!expr`	(Logical NOT) Returns false if its single operand can be converted to true; otherwise, returns true.

Examples of expressions that can be converted to false are those that evaluate to null, 0, NaN, the empty string ("""), or undefined.

The following code shows examples of the && (logical AND) operator.

```
var a1 =  true && true;     // t && t returns true
var a2 =  true && false;    // t && f returns false
var a3 = false && true;     // f && t returns false
var a4 = false && (3 == 4); // f && f returns false
var a5 = "Cat" && "Dog";    // t && t returns Dog
var a6 = false && "Cat";    // f && t returns false
var a7 = "Cat" && false;    // t && f returns false
```

```
typeof undefined              // undefined
typeof null                   // object
null === undefined            // false
null == undefined             // true
```

Try it Yourself »

JavaScript has the following expression categories:

- Arithmetic: evaluates to a number, for example 3.14159. (Generally uses arithmetic operators.)
- String: evaluates to a character string, for example, "Fred" or "234". (Generally uses string operators.)
- Logical: evaluates to true or false. (Often involves logical operators.)
- Object: evaluates to an object. (See special operators for various ones that evaluate to objects.)

Operator	Description	Examples returning true

Equal (==)	Returns true if the operands are equal.	`3 == var1`
		`"3" == var1`
		
		`3 == '3'`

Not equal (!=)	Returns true if the operands are not equal.	`var1 != 4` `var2 != "3"`
Strict equal (===)	Returns true if the operands are equal and of the same type. See also `Object.is` and sameness in JS.	`3 === var1`
Strict not equal (!==)	Returns true if the operands are not equal and/or not of the same type.	`var1 !== "3"` `3 !== '3'`
Greater than (>)	Returns true if the left operand is greater than the right operand.	`var2 > var1` `"12" > 2`
Greater than or equal (>=)	Returns true if the left operand is greater than or equal to the right operand.	`var2 >= var1` `var1 >= 3`
Less than (<)	Returns true if the left operand is less than the right operand.	`var1 < var2` `"2" < "12"`
Less than or equal (<=)	Returns true if the left operand is less than or equal to the right operand.	`var1 <= var2` `var2 <= 5`

delete

The `delete` operator deletes an object, an object's property, or an element at a specified index in an array. The syntax is:

```
delete objectName;
delete objectName.property;
delete objectName[index];
delete property; // legal only within a with statement
```

Arithmetic operators

Arithmetic operators take numerical values (either literals or variables) as their operands and return a single numerical value. The standard arithmetic operators are addition (+), subtraction (-), multiplication (*), and division (/). These operators work as they do in most other programming languages when used with floating point numbers (in particular, note that division by zero produces `Infinity`). For example:

```
console.log(1 / 2); /* prints 0.5 */
console.log(1 / 2 == 1.0 / 2.0); /* also this is true */
```

In addition, JavaScript provides the arithmetic operators listed in the following table.

Table 3.3 Arithmetic operators

Operator	Description	Example
% (Modulus)	Binary operator. Returns the integer remainder of dividing the two operands.	12 % 5 returns 2.
++ (Increment)	Unary operator. Adds one to its operand. If used as a prefix operator (++x), returns the value of its operand after adding one; if used as a postfix operator (x++), returns the value of its operand before adding one.	If x is 3, then ++x sets x to 4 and returns 4, whereas x++ returns 3 and, only then, sets x to 4.
-- (Decrement)	Unary operator. Subtracts one from its operand. The return value is analogous to that for the increment operator.	If x is 3, then --x sets x to 2 and returns 2, whereas x-- returns 3 and, only then, sets x to 2.
- (Unary negation)	Unary operator. Returns the negation of its operand.	If x is 3, then -x returns -3.

Edit This Code: See Result »

```
<html>
<body>

<p>The internal clock in JavaScript starts at midnight
January 1, 1970.</p>
<p>Click the button to display the number of milliseconds
since midnight, January 1, 1970.</p>

<button onclick="myFunction()">Try it</button>

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

<script>
function myFunction() {
    var d1 = new Date();
    var mil1 = d1.getTime();

    for(var i=0;i<10000000;i++){
         var r = 5+5;
    }
    var d2 = new Date();
    var mil2 = d2.getTime();
     document.getElementById("demo").innerHTML = mil2 +"<br>
"+mil1;
}
</script>

</body>
</html>
```

Result:

The internal clock in JavaScript starts at midnight January 1, 1970.

Click the button to display the number of milliseconds since midnight, January 1, 1970.

Try it

1422121422378
1422121422367

JSFIDDLE α

Run | Save | TidyUp | JSHint | Collaboration | Login/Sign up

Frameworks & Extensions

No-Library (pure JS)

onLoad

Fiddle Options

External Resources

Languages

Ajax Requests

Legal, Credits and Links

Keyboard shortcuts

HTML

```html
<body>
    <button onclick="alert('Inline Scripting')">Click Me</button>
    <button ondblclick="alert('doble Click')">Doble</button>


    <button id="btn-conEvento">Tengo un evento agregado</button>
    <input id="in-test" placeholder="Input vacio"/>
</body>
```

JavaScript

```javascript
document.getElementById('btn-conEvento').addEventListener("click",
function(event) {
     alert('algo');
});

document.getElementById('in-test').addEventListener("blur",
function(event) {
     alert('fuera de foco');
});
```

CSS

Result

Click Me | Doble

Tengo un evento agregado

jkh

JSFIDDLE α

Run | Save | TidyUp | JSHint | Collaboration | Login/Sign up

Frameworks & Extensions

No-Library (pure JS)

onLoad

Fiddle Options

External Resources

Languages

Ajax Requests

Legal, Credits and Links

Keyboard shortcuts

HTML

CSS

JavaScript

```
function getType(valor){
    return typeof valor;

}
var sinValor;
var conValor = "CETAV";

//limpiar/ borrar
delete conValor;

console.log('34 ->'+getType(34));
console.log('true ->'+getType(true));
console.log('sinvalor ->'+getType(sinValor));
console.log('null ->'+getType(null));
console.log('objeto ->'+getType({'nombre':'JavaScript Dev'}));
console.log('conValor ->'+getType(conValor));
```

Result

Elements | Network | Sources | Timeline | Profiles | Resources | Audits | Console

html | body | #show-result | #content | fieldset.column.left | div#panel_js.window.bottom | div.CodeMirror.cm-s-default.CodeMirror-wrap | div.CodeMirror-scroll | div

Styles | Computed »

Console | Search | Emulation | Rendering

<top frame> | Preserve log

```
conValor ->string                (index):38
34 ->number                      (index):33
true ->boolean                   (index):34
sinvalor ->undefined             (index):35
null ->object                    (index):36
objeto ->object                  (index):37
conValor ->string                (index):38
```

Summary

Operator precedence determines the order in which operators are evaluated. Operators with higher precedence are evaluated first.

A common example:

```
3 + 4 * 5 // returns 23
```

The multiplication operator ("*") has higher precedence than the addition operator ("+") and thus will be evaluated first.

Associativity

Associativity determines the order in which operators of the same precedence are processed. For example, consider an expression:

a OP b OP c

Left-associativity (left-to-right) means that it is processed as (a OP b) OP c, while right-associativity (right-to-left) means it is interpreted as a OP (b OP c). Assignment operators are right-associative, so you can write:

```
a = b = 5;
```

With the expected result that a and b get the value 5. This is because the assignment operator returns the value that it assigned. First, b is set to 5. Then the a is set to the value of b.

The following table is ordered from highest (19) to lowest (0) precedence.

Precedence	Operator Type	Associativity	Individual Operators
19	Grouping	n/a	(…)
18	Member Access	left-to-right	… . …
	Computed Member Access	left-to-right	… […]
	new(with argument list)	n/a	new … (…)
17	Function call	left-to-right	… (…)
	new(with argument list)	right-to-left	new ...
16	Postfix Increment	n/a	… ++
	postfix Decrement	n/a	… --

15	Logical NOT	right-to-left	! …
	Bitwise NOT	right-to-left	~ …
	Unary Plus	right-to-left	+ …
	Unary Negation	right-to-left	- …
	Prefix Increment	right-to-left	++ …
	Prefix Decrement	right-to-left	-- …
	typeof	right-to-left	typeof …
	void	right-to-left	void …
	delete	right-to-left	delete …
14	Multiplication	left-to-right	… * …
	Division	left-to-right	… / …
	Remainder	left-to-right	… % …

13	Addition	left-to-right	… + …
	Subtraction	left-to-right	… - …
12	Bitwise Left Shift	left-to-right	… << …
	Bitwise Right Shift	left-to-right	… >> …
	Bitwise Unsigned Right Shift	left-to-right	… >>> …
11	Less Than	left-to-right	… < …
	Less Than or Equal	left-to-right	… <= …
	Greater Than	left-to-right	… > …
	Greater Than or Equal	left-to-right	… >= …
	in	left-to-right	… in …
	instanceof	left-to-right	… instanceof …

10	Equality	left-to-right	… == …
	Inequality	left-to-right	… != …
	Strict Equality	left-to-right	… === …
	Strict Inequality	left-to-right	… !== …
9	Bitwise AND	left-to-right	… & …
8	Bitwise XOR	left-to-right	… ^ …
7	Bitwise OR	left-to-right	… | …
6	Logical AND	left-to-right	… && …
5	Logical OR	left-to-right	… || …
4	Conditional	right-to-left	… ? … : …

3	Assignment	right-to-left	… = …
			… += …
			… -= …
			… *= …
			… /= …
			… %= …
			… <<= …
			… >>= …
			… >>>= …
			… &= …
			… ^= …
			… |= …

2	yield	right-to-left	yield ...
1	Spread	n/a	... …
0	Comma / Sequence	left-to-right	… , …