Universidad Nacional de Rosario

UNR

Argentina programa 4.0

Programación
Web Full Stack PHP

Lógica y decisiones

¿Cómo implementamos las decisiones?

Operadores de comparación

En JavaScript, los operadores de comparación son fundamentales para tomar decisiones en nuestros programas. Estos operadores nos permiten comparar valores y obtener un resultado booleano (verdadero o falso).

Con los operadores de comparación, podemos evaluar diferentes tipos de datos, como números, caracteres, cadenas de texto y más. Esto nos brinda la capacidad de realizar comparaciones y tomar decisiones basadas en la información disponible.

Un ejemplo práctico de cómo usar los operadores de comparación sería en un sitio web, donde podemos mostrar interfaces de usuario distintas según el tipo de usuario y si han iniciado sesión o no.

```
// Operadores Lógicos

>       // Mayor que
<       // Menor que
>=      // Mayor o igual que
<=      // Menor o igual que
==      // Igualdad
!=      // Desigualdad
===     // Igualdad estricta
!==     // Desigualdad estricta
```

Podemos comparar diferentes tipos de datos en JavaScript. En el caso de los datos numéricos, la comparación funciona de la manera esperada. Sin embargo, cuando se trata de caracteres, es posible encontrar un comportamiento inesperado debido al uso de códigos numéricos llamados "ASCII Codes".

En JavaScript, los caracteres se representan internamente como códigos numéricos según la tabla ASCII. Estos códigos determinan el orden de los caracteres en las comparaciones. Por lo tanto, al comparar caracteres, en realidad estamos comparando los valores numéricos correspondientes a los códigos ASCII.

ASCII printable characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	|
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

theASCIIcode.com.ar

Ejemplo:

```
10 > 1  // true
0.2 < 1 // true+
-10 > 10 // false
1 <= 1 // true
100 >= 50 // true
100 >= 101 // false
"a" < "b" // true
"A" > "a" // false
```

Comparadores de igualdad

JavaScript cuenta con dos comparadores de igualdad/desigualdad. El operador "==" o "!=" valida la igualdad del valor, pero no del tipo de dato. En cambio, los operadores "===", y "!==", validan tanto el valor como el tipo de dato. Siempre es más seguro utilizar la igualdad o desigualdad estricta.

El operador "==" realiza una comparación de igualdad y devuelve verdadero si los valores son iguales, incluso si los tipos de datos son diferentes. Por ejemplo, 5 == "5" devuelve verdadero, ya que ambos valores son iguales, a pesar de que uno es un número y el otro es una cadena de texto.

Por otro lado, el operador "===" realiza una comparación estricta de igualdad y devuelve verdadero solo si los valores y los tipos de datos son iguales. Utilizar "===" es más seguro y preciso en la mayoría de los casos, ya que garantiza que tanto el valor como el tipo sean idénticos. Por ejemplo, 5 === "5" devuelve falso, ya que los tipos de datos son diferentes.

De manera similar, los operadores de desigualdad "!=" y "!==" funcionan de la misma manera que los operadores de igualdad, pero negando la igualdad.

```
1 == 1 // true
1 == "1" // true
1 === "1" // false
1 === 1 // true
0 == "" // true
0 == false // true
0 === false // false
```

Expresiones condicionales

Una declaración condicional es una estructura que nos permite ejecutar un conjunto de comandos si se cumple una condición especificada. Estas declaraciones son fundamentales para controlar el flujo de ejecución de un programa y tomar decisiones basadas en ciertas condiciones.

Expresión If

En la estructura "if", la condición es una expresión que se evalúa como verdadera o falsa. Si la condición es verdadera, se ejecuta el bloque de código dentro del primer conjunto de llaves. Si la condición es falsa, el código continúa su ejecución sin ejecutar el bloque.

```
if (condicion) {
    // Código que se ejecuta si condición es true
}
```

Else

Utilizando el if podremos agregarle la cláusula opcional else para ejecutar una instrucción si la condición es false.

```
if (condicion) {
    // Código que se ejecuta si condición es true
} else {
    // Código que se ejecuta si condición es false
}
```

Else if

Hemos visto que el if evalúa una condición y, si es false, puede terminar allí o, en caso de tener un else, se ejecuta su bloque. Sin embargo, a veces es necesario verificar múltiples condiciones, por lo que es útil la expresión else if.

La expresión "else if" nos permite agregar condiciones adicionales en una estructura condicional "if-else". Nos permite evaluar múltiples condiciones en secuencia y ejecutar un bloque de código correspondiente a la primera condición que se cumpla.

Solo se ejecutará la primera condición lógica que se evalúe como verdadera.

```
if (condicion) {
    // Código que se ejecuta si condición es true
} else if (condicion_2) {
    // Código que se ejecuta si condición_2 es true
} else if (condicion_3) {
    // Código que se ejecuta si condición_3 es true
} else {
    // Código que se ejecuta si ninguna condición es true
}
```

En esta estructura, se evalúa la condición. Si esa condición es verdadera, se ejecuta el bloque de código correspondiente. Si la condición es falsa, se pasa a evaluar la condición_2. Si la condición_2 es verdadera, se ejecuta el bloque de código correspondiente, y lo mismo con la condición_3. Si ninguna de las condiciones anteriores es verdadera, se ejecuta el bloque de código del "else".

Es importante tener en cuenta que la evaluación de condiciones en una estructura "if-else" se realiza de forma secuencial, de arriba hacia abajo. Una vez que se cumple una condición y se ejecuta su bloque de código, las condiciones restantes no se evalúan ni se ejecutan.

Operador ternario

El **operador ternario** es el único operador en JavaScript que utiliza tres operandos: una condición seguida de un signo de interrogación (?), luego una expresión a ejecutar si la condición es verdadera, seguida de dos puntos (:) y, finalmente, la expresión a ejecutar si la condición es falsa. Este operador se utiliza como alternativa a una declaración [if...else].

Este operador es una alternativa compacta y legible a la declaración if...else cuando se necesitan tomar decisiones simples basadas en una condición. Al utilizar el operador ternario, podemos ahorrar líneas de código y simplificar la lógica de nuestros programas.

```
condicion ? /* Código si es true */ : /* Código si es false */
```

Switch Statement

La instrucción **switch** evalúa una expresión, comparando el valor de la expresión con una serie de cláusulas **case**, y ejecuta las declaraciones después de la primera cláusula case con un valor coincidente, hasta que se encuentre una declaración break.

La cláusula **default** de una instrucción que se ejecutará si no hay ninguna cláusula case que coincida con el valor de la expresión.

```
switch(numero) {
    case 1:
        // Código se ejecuta si numero es 1
        break;
    case 50:
        // Código se ejecuta si numero es 50
        break;
    default:
        // Código que se ejecuta si numero no coincide con
        // ningun case
}
```

Valores Truthy y Falsy

Debemos comprender que todos los valores tienen un valor booleano inherente, lo que significa que pueden ser evaluados como verdaderos (Truthy) o falsos (Falsy) en una condición.

Valores Falsy

- false
- undefined
- 0
- null
- “”
- Nan

Estos valores son considerados falsy, lo que significa que en una condición se evaluarán como falsos.

Valores Truthy

Todos los demás valores, incluido los objetos y Strings no vacíos son Truthy, es decir, en una condición son evaluados como true

Es importante comprender estos valores truthy y falsy, ya que nos permiten realizar evaluaciones condicionales más eficientes y concisas en nuestro código. Podemos utilizarlos en declaraciones if, operadores ternarios y otras estructuras de control para tomar decisiones basadas en el valor de una expresión.

```
let nombre = ""; // valor falsy

if (nombre) {
    console.log("El nombre es válido.");
} else {
    console.log("El nombre es inválido.");
}
```

En este ejemplo, la variable 'nombre' tiene asignada una cadena vacía, que es un valor falsy. Al evaluar la condición dentro del if, se considera como falsa y se ejecuta el bloque de código dentro del else, mostrando el mensaje 'El nombre es inválido.' en la consola.

Si nombre tiene un string no vacío, será evaluado con un valor Truthy y ejecutará el bloque de código del if.

Operadores lógicos

Los operadores lógicos en JavaScript se utilizan para realizar operaciones booleanas, es decir, operaciones que involucran valores booleanos (true o false). Estos operadores permiten combinar o modificar valores booleanos y evaluar condiciones más complejas.

Operador AND (&&): devuelve true si ambos operandos son true, de lo contrario, devuelve false. Si el primer operando es false, no se evalúa el segundo operando, ya que no es necesario para determinar el resultado final.

```
console.log(true && true); // true
console.log(true && false); // false
console.log(false && true); // false
console.log(false && false); // false
console.log(10 > 5 && 20 > 10); // true
console.log(10 > 5 && 20 < 10); // false
```

Operador OR (||): devuelve true si al menos uno de los operandos es true. Si el primer operando es true, no se evalúa el segundo operando, ya que no es necesario para determinar el resultado final.

```
console.log(true || true); // true
console.log(true || false); // true
console.log(false || true); // true
console.log(false || false); // false
console.log(10 > 5 || 20 > 10); // true
console.log(10 > 5 || 20 < 10); // true
```

Operador NOT (!): El operador "!" se utiliza para negar el valor booleano de un operando. Si el operando es true, devuelve false; si el operando es false, devuelve true.

```
console.log(!true); // false
console.log(!false); // true
console.log(!(10 > 5)); // false
console.log(!(20 < 10)); // true
```

Además, es importante destacar que los operadores lógicos && y || pueden devolver valores distintos a los booleanos cuando se utilizan con operandos no booleanos. En lugar de devolver un resultado

booleano, estos operadores devolverán el valor de uno de los operandos originales. Esto puede ser útil en ciertos casos para realizar evaluaciones condicionales más concisas. *Este tema lo profundizaremos mas adelante en este curso.*

Precedencia de operadores

La precedencia de los operadores determina el orden en el que se evalúan las operaciones en una expresión. En JavaScript, los operadores se evalúan siguiendo ciertas reglas de precedencia para asegurar que las expresiones se evalúen de manera coherente y predecible.

A continuación, se muestra la precedencia de los operadores en JavaScript, de mayor a menor:

1. **Paréntesis ()**: Los paréntesis se utilizan para agrupar partes de una expresión y establecer el orden de evaluación. Las expresiones dentro de los paréntesis se evalúan primero.
2. **Operadores unarios:** Los operadores unarios son aquellos que actúan sobre un solo operando. Algunos ejemplos de operadores unarios son el operador lógico de negación (!), el operador de incremento (++) y el operador de decremento (--). Estos operadores tienen una alta precedencia y se evalúan antes que los demás operadores.
3. **Operadores aritméticos**: Los operadores aritméticos como la multiplicación (*), la división (/), la suma (+) y la resta (-) tienen una precedencia intermedia y se evalúan después de los operadores unarios.
4. **Operadores de comparación**: Los operadores de comparación, como el operador de igualdad (==), el operador de desigualdad (!=), el operador mayor que (>), el operador menor que (<), entre otros, se evalúan después de los operadores aritméticos.
5. **Operadores lógicos**: Los operadores lógicos, como el operador AND (&&) y el operador OR (||), tienen una precedencia más baja y se evalúan después de los operadores de comparación. La precedencia de los operadores lógicos es la siguiente:
    1. Operador NOT (!): Tiene la mayor precedencia entre los operadores lógicos, al ser también un operador unario (sólo actúa sobre un operando). Se evalúa antes que los demás operadores lógicos.
    2. Operador AND (&&): Tiene una precedencia menor que el operador NOT. Se evalúa después del operador NOT y antes del operador OR.

3. Operador OR (||): Tiene la menor precedencia entre los operadores lógicos. Se evalúa después del operador NOT y del operador AND.

Ejemplo:

```
console.log(!true || false && true); // false
```

En este ejemplo, el operador NOT se evalúa primero y niega el valor de **true**, dando como resultado **false**. Luego, se evalúa el operador AND, que compara **false && true** y también devuelve **false**. Por último, se evalúa el operador OR entre **false** y **false**, y el resultado final es **false**.