Modulo 3 - Operadores y flujos de datos

Operadores

Los operadores permiten manipular el valor de las variables, realizar operaciones matemáticas con sus valores y comparar diferentes variables. Son las herramientas para realizar cálculos complejos y tomar decisiones en base a comparaciones y otros tipos de decisiones.

Existen muchos tipos de operadores en javascript:

- Operadores de asignación
- Operadores de incremento/decremento
- Matemáticos
- Operadores lógicos: -- Negación -- AND -- OR -- Igualdad -- Relacionales (comparaciones)

Asignación

- **=** operador de asignación

Este operador se utiliza para guardar un valor específico en una variable:

```
const myVariable = 5;
```

Se asigna a la variable de la izquierda, el valor (o el resultado de evaluar la expresión) de la derecha:

```
const a = 2;
const b = 3;
const myVariable = a + b; // myVariable tendrá valor 5
```

Incremento y decremento

- **++** operador de incremento
- **--** operador de decremento

Solamente son válidos para las variables numéricas, y su efecto es incrementar/decrementar en una unidad el valor de la variable asociada:

```
let a = 3;
let b = 3;
++a; //a tendrá valor 4
--b; //b tendrá valor 2
```

También se puede hacer post-incremento y post-decremento:

```
let a = 3;
let b = 3;
a++; //a tendrá valor 4
b--; //b tendrá valor 2
```

Aunque ambas hacen lo mismo (incrementar/decrementar), el pre-incremento y post-incremento tienen sentido cuando se utilizan en expresiones:

Ejemplo de pre-incremento:

```
let a = 1;
let b = ++a + 3;
// a valdrá 2 y b 5
```

Ejemplo de post-incremento:

```
let a = 1;
let b = a++ + 3;
// a valdrá 2 y b 4
```

Matemáticos

- **+** operador de suma
- **-** operador de resta
- **/** operador de división
- ***** operador de multiplicación
- **%** operador módulo (resto)

Los operadores matemáticos realizan operaciones sobre los operandos (valores, variables o expresiones):

```
const a = 4 + 4; // a tiene valor 8
const b = 1 - 1; // b tiene valor 0
const c = 10 / 2; // c tiene valor 5
const d = 2 * 3; // d tiene valor 6
const e = 12 % 5; // e tiene valor 2
```

Ejemplos con expresiones:

```
const a = 1;
const b = 1;
const c = a + b; // c tiene valor 2
const d = a + b + c; // d tiene valor 4
const e = (a + b) * (c + d); // e tiene valor 12
```

Lógicos

El resultado de cualquier operación que utilice operadores lógicos siempre es un valor lógico o booleano.

Negación

- **!** operador de negación

Se utiliza para obtener el valor contrario al valor de la variable:

```
const a = true;
const b = !a; // b tiene valor false
```

El funcionamiento de este operador se resume en la siguiente tabla:

variable	!variable
true	false
false	true

Se utiliza mucho para valorar decisiones:

```
if (!esCorrecto) {
  avisar_al_usuario();
}
```

AND

- **&&** operador AND lógico

La operación AND evalúa dos variables (o expresiones) booleanas, y resulta *true* sólo si ambas son *true*:

a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

```
const a = true;
const b = false;
```

```
const result1 = a && b; // result1 tiene valor false
const result2 = a && !b; // !b es true, por tanto result2 tiene valor true
```

OR

- **||** operador OR lógico

La operación OR evalúa dos variables (o expresiones) booleanas, y resulta *true* si alguna de las dos es *true*:

| a | b | a || b | |-------|-------|--------| | true | true | true | | true | false | true | | false | true | true | | false | false | false |

```
const a = true;
const b = false;

const result1 = a || b; // result1 tiene valor true
const result2 = !a || b; // !a es false, por tanto result2 tiene valor
false
```

Igualdad

- **==** operador de igualdad
- **!=** operador de desigualdad
- **===** operador de igualdad estricto
- **!==** operador de desigualdad estricta

Los operadores de igualdad evalúan si ambas variables o expresiones son el mismo valor:

```
const a = 2;

a == 2; // devuelve true
a != 2; // devuelve false
```

Los estrictos, además del valor, tienen en cuenta el tipo:

```
const a = "2";

a == 2; // devuelve true
a === 2; // devuelve false (String vs Number)
```

Hay que tener en cuenta las comparaciones entre objetos y arrays (son punteros):

```
const a = [1, 2, 3];
const b = [1, 2, 3];
```

```
a == b; // retorna false
a === b; // retorna false
```

```
const a = { name: "John", age: 28 };
const b = { name: "John", age: 28 };
a == b; // retorna false;
```

```
const a = [1, 2, 3];
const b = a;
a == b; // true
```

Relacionales (comparación)

- **>** operador *mayor que*
- **<** operador *menor que*
- **>=** operador *mayor o igual a*
- **<=** operador *menor o igual a*

Los operadores relacionales con los que podemos trabajar son idénticos a los que definen las matemáticas:

Ejemplo de operador *mayor que*:

```
const a = 5;
const esMayorQue5 = a > 5; // esMayorQue5 tendrá valor false
const esMayorOIgualA5 = a >= 5; // esMayorOIgualA5 tendrá valor true
const esMenorQue5 = (a - 1) < 5; // esMenorQue5 tendrá valor true;
const esMenorOIgualA5 = a <= 5; // esMenorOIgualA5 tendrá valor true;
```

Serán muy utiles a la hora de tomar decisiones en base a condiciones:

```
const a = 5;

if (a > 0) {
  alert("Es mayor que 0");
}
```

Además de los anteriores, podemos considerar operadores relacionales los dos siguientes:

- **in**: determina si un objeto tiene una propiedad dada:

```
const auto1 = { motor: "2.0", ruedas: "2", nombre: "moto" };
const auto2 = { motor: "1.6", bateria: "4000", ruedas: "4", nombre: "coche
```

```
híbrido"}

const tieneMotor = "motor" in auto1; // tieneMotor es true
const esHibrido1 = "motor" in auto1 && "bateria" in auto1; // esHibrido es
false
const esHibrido2 = "motor" in auto2 && "bateria" in auto2; // esHibrido es
true
```

- **instanceof** determina si un objeto es una instancia de otro:

```
const myVar = new Number(5);
const esNumerico = myVar instanceof Number; // esNumerico es true
```

Flujos de datos

Lo más normal es que durante el desarrollo necesitemos tomar decisiones en base a una condición y realizar operaciones diferentes en cada caso. Por ejemplo, si queremos mostrar un mensaje al usuario si la contraseña es incorrecta o continuar el flujo de la aplicación si es correcta. Para este control de flujo javascript nos ofrece estructuras para definir el camino a tomar dependiendo de la evaluación de la condición.

If

La estructura *if* se utiliza para tomar decisiones en base a una condición. Para entenderla podemos imaginarnos el siguiente pseudocódigo:

```
si (se cumple condicion) {
  hacer_algo();
} si no {
  hacer_otra_cosa();
}
```

En Javascript:

```
if (condicion) {
  ...
}
```

El bloque de código se ejecutara siempre y cuando la condición se evalúe como cierta (es decir, un valor *true*). Si no se evalúa como *true*, el bloque de código contenido en *if* no se ejecuta, y el programa sigue su ejecución.

Por ejemplo, si queremos mostrar un mensaje al usuario si el valor introducido es mayor que 10:

```
if(valorIntroducido > 10) {
  alert("Error: el valor introducido es mayor que 10");
}
```

En este caso, si al evaluar la condición (valorIntroducido > 10) se devuelve *true* se ejecutará el bloque de codigo que contiene el *alert* de aviso al usuario.

La condición puede contener expresiones combinando comparaciones o valores pero siempre se evaluará finalmente como un booleano con valor *true* o *false* para ejecutar o no el bloque de código. Por ejemplo, podemos ampliar el caso anterior para evaluar que el valor introducido esté entre 5 y 10:

```
if(valorIntroducido < 5 || valorIntroducido > 10) {
  alert("Error: el valor introducido no está comprendido entre 5 y 10");
}
```

If...else

Podemos contemplar la ejecución de un bloque de código asociado a la evaluación *false* de la condición del *if*, es decir, si no se cumple. En pseudocódigo, sería un claro ejemplo:

```
si (se cumple condicion) {
  hacer_algo();
} si no {
  hacer_otra_cosa();
}
```

Así, podemos controlar el flujo tanto si la condición se cumple como si no. Imaginemos que queremos dividir 100 entre el valor introducido por el usuario, controlando que no es 0 (para evitar el error de dividir por cero):

```
let resultado;
if(valorIntroducido != 0) {
  resultado = 100 / valorIntroducido;
} else {
  resultado = 0;
}
```

Ternario

Javascript dispone de una estructura que realiza la misma función que if...else pero de forma más abreviada: los ternarios:

```
condicion ? si_verdadero : si_falso
```

Sólo acepta una instrucción en los bloques a ejecutar si se cumple la condición o no:

```
const a = 5;
const b = a > 5 ? a * 2 : 0;
```

Con el último ejemplo de if...else:

```
const resultado = valorIntroducido != 0 ? 100 / valorIntroducido : 0;
```

Switch

La estructura *switch* es muy útil cuando la evaluación de la condición puede tomar muchos valores diferentes. Si utilizaramos para esto la estructura *if...else* tendríamos que repetir la condición múltiples veces:

```
if(dia == 1) {
  console.log("Hoy es lunes.");
} else if(dia == 2) {
  console.log("Hoy es martes.");
} else if(dia == 3) {
  console.log("Hoy es miércoles.");
} else if(dia == 4) {
  console.log("Hoy es jueves.");
} else if(dia == 5) {
  console.log("Hoy es viernes.");
} else if(dia == 6) {
  console.log("Hoy es sábado.");
} else if(dia == 0) {
  console.log("Hoy es domingo.");
}
```

En estos casos, es mejor utilizar la estructura *switch* ya que nos permite dejar un código más limpio, mantenible, y nos ahorra trabajo:

```
switch(dia) {
  case 1: console.log("Hoy es lunes."); break;
  case 2: console.log("Hoy es martes."); break;
  case 3: console.log("Hoy es miércoles."); break;
  case 4: console.log("Hoy es jueves."); break;
  case 5: console.log("Hoy es viernes."); break;
  case 6: console.log("Hoy es sábado."); break;
  case 0: console.log("Hoy es domingo."); break;
  default: console.log("No es un día de la semana válido"); break;
}
```

Es importante no olvidar la instrucción *break* al final del bloque de código de cada *case*. Por el contrario, se seguiría ejecutando el bloque de código del siguiente *case*.

Si el valor resultado de evaluar la condición no coincide con ningún valor especificado en los *case*, se ejecutará, si existe, el bloque *default*.

No se puede indicar otras condiciones en los *case*, si no valores/expresiones finales que deberán coincidir con la condición del *switch*.