

JAVASCRIPT

Lógica a su servicio

The logo consists of the letters 'JS' in a bold, dark blue, sans-serif font. These letters are centered within a light yellow square, which is itself centered on a larger yellow background.

JS

Operadores

Los **operadores** nos ayudan a **modificar, reasignar y comprobar** el **valor** de las variables con una sintaxis más sencilla y acotada.

Aritméticos

Estos **operadores** se utilizan para **realizar las operaciones matemáticas** tradicionales.

Gracias a ellos podemos **sumar**, **restar**, **multiplicar**, **dividir** y obtener el **resto**.



```
1 let numero = 20;
2 let numero2 = 8;
3
4 var resultado = numero + numero2; // Suma: 28
5 var resultado = numero - numero2; // Resta: 12
6 var resultado = numero * numero2; // Multiplicación: 160
7 var resultado = numero / numero2; // División: 2.5
8 var resultado = numero % numero2; // Resto: 4
```

Asignación

Son los que **nos permiten** **asignar** o **reasignar** valores a nuestras variables.




```
1  var numero = 17;  
2  
3  numero += 10; // 27  
4  numero -= 10; // 7  
5  numero *= 10; // 170  
6  numero /= 10; // 1.7  
7  numero %= 10; // 7  
8  numero **= 10; // 2015993900449
```

Incremento y Decremento

Estos **operadores** son útiles cuando **necesitamos modificar** el valor de *nuestra variable numérica* a razón de **una unidad cada vez**.


Dependiendo la **posición del operador** es el comportamiento obtenido. Por ejemplo, si lo usamos a la **izquierda** entonces **primero incrementa o decrementa** el valor y luego lo asigna a la variable y si se encuentra a **la derecha realiza lo inverso**.



```
1  var numero = 10;
2
3  numero++; // primero lee la variable (10) y luego la incrementa (11)
4  ++numero; // primero incrementa la variable (11) y luego la lee (11)
5
6  numero--; // primero lee la variable (10) y luego la decrementa (9)
7  --numero; // primero decrementa la variable (9) y luego la lee (9)
```

Comparación

Nos permiten generar condiciones basadas en comparaciones o asignar valores.



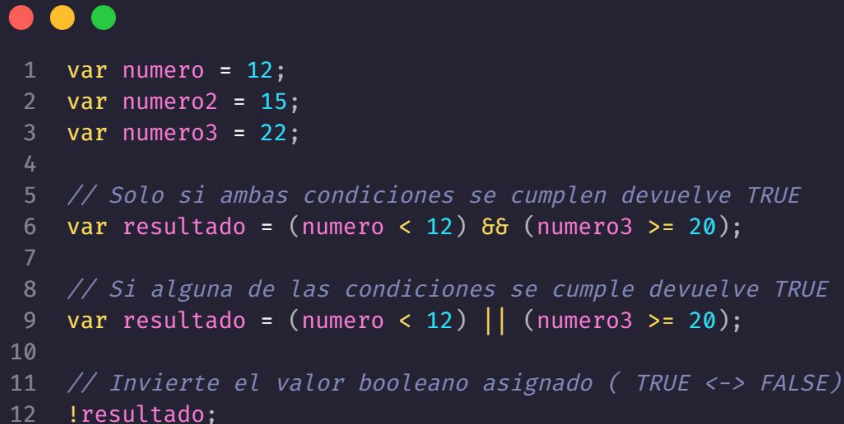
```
1 var resultado = 12 > 10; // Mayor que
2 var resultado = 6 < 10; // Menor que
3
4 var resultado = 125 >= 125; // Mayor o igual que
5 var resultado = 10 <= 10; // Menor o igual que
6
7 // Valida solo valor
8 var resultado = '10' == 10; // Igualdad Simple
9 var distinto = 20 != '20'; // Desigualdad Simple
10
11 // Valida valor y tipo de dato
12 var resultado = 10 === 10; // Igualdad Estricta
13 var distinto = 20 !== 'Hola'; // Inigualdad Estricta
```

Lógicos

Utilizados para **conjugar condiciones** lógicas y **obtener** un valor booleano como respuesta.

Similares a los de comparación pero con una **notación más lógica** y combinable con estos últimos.

Estos operadores son el del conjunción **AND (&&)**, disyunción **OR (||)** o negación **NOT(!)**.



```
1  var numero = 12;
2  var numero2 = 15;
3  var numero3 = 22;
4
5  // Solo si ambas condiciones se cumplen devuelve TRUE
6  var resultado = (numero < 12) && (numero3 >= 20);
7
8  // Si alguna de las condiciones se cumple devuelve TRUE
9  var resultado = (numero < 12) || (numero3 >= 20);
10
11 // Invierte el valor booleano asignado ( TRUE <-> FALSE)
12 !resultado;
```

Condicionales

Las **sentencias condicionales** se utilizan para **realizar** diferentes acciones basadas en **condiciones** manipulando el flujo de nuestro programa.

En JavaScript tenemos las siguientes declaraciones condicionales:

- **if** - bloque de código que **se ejecutará**, **si una condición** especificada **es verdadera**.
- **else** - bloque de código que **se ejecutará**, **si la condición if** resulta **falsa**.
- **switch** para especificar muchos bloques **condicionales** de código que se ejecutarán.

IF

El **código** definido dentro de un bloque IF sólo se ejecutará si la condición de la sentencia es verdadera.



```
1  if (10 > 6) {  
2      // El código se ejecuta solo si 10 es mayor que 6  
3  }  
4
```

ELSE

Corresponde al **bloque de código** que deberá **ejecutarse si** la condición IF precedente resultó ser **falsa**. Esta sentencia **es opcional**, ya que podemos tener IF sin ELSE.



```
1  if (10 > 6) {  
2      // El código se ejecuta solo si 10 es mayor que 6  
3  } else {  
4      // En caso que no hará lo que esté aquí adentro  
5  }
```

ELSE IF

Son **condiciones adicionales** intermedias entre la principal y la condición por defecto. Se utilizan para **validar más de una condición** en un mismo bloque.



```
1  if (10 > 6) {  
2      // El código se ejecuta solo si 10 es mayor que 6  
3  } else if (3 < 7) {  
4      // Se ejecuta solo si la primer condición es falsa  
5      // y esta es verdadera  
6  } else {  
7      // En caso que no hará lo que esté aquí adentro  
8  }
```

Operador Ternario

Si bien pertenece al apartado del tema anterior, este operador funciona como un condicional simplificado u operador condicional.

Es de utilidad cuando tenemos condiciones del tipo if/else, es decir, de **dos valores posibles**.



```
1 let numero = 23;  
2  
3 let resultado = numero >= 13 ? 'Es mayor que 13' : 'Es menor que 13';
```

Lo que viene después del ? será el **valor de retorno** si la condición es verdadera y lo que sigue luego de : cuando la condición resulte falsa.

Otros operadores condicionales

Así como disponemos del **operador ternario**, también contamos con otros que nos permiten evaluar nuestras condiciones sin tener que **definir estructuras** o sentencias robustas como el if o el switch.

Nombre	Operador	Descripción
Operador lógico AND	<code>a && b</code>	Devuelve <code>a</code> si es <code>false</code> , sino devuelve <code>b</code> .
Operador ternario ?:	<code>a ? b : c</code>	Si <code>a</code> es <code>true</code> , devuelve <code>b</code> , sino devuelve <code>c</code> .
Operador lógico OR	<code>a b</code>	Devuelve <code>a</code> si es <code>true</code> , sino devuelve <code>b</code> .
Operador lógico Nullish coalescing	<code>a ?? b</code>	Devuelve <code>b</code> si <code>a</code> es <code>null</code> o <code>undefined</code> , sino devuelve <code>a</code> .
Operador de asignación lógica nula ??=	<code>a ??= b</code>	Es equivalente a <code>a ?? (a = b)</code>

SWITCH

Funciona como un **bloque condicional** if/else if/else que sintácticamente se ve **más claro** y nos permite **comportamientos adicionales**.

case: define el **valor** con el cual debe igualar la condición.

default: es el **resultado por defecto** si ninguna de las anteriores cumple la condición.

break: indica al bloque que **debe dejar de validar condiciones**, si no se coloca entonces los case debajo del que cumple la condición también se ejecutarán.

continue: se usa para **saltar una condición**.

```
1  let condition = 'blue';
2
3  switch (condition) {
4
5      case 'red':
6          console.log('El saco es rojo');
7          break;
8      case 'blue':
9          console.log('El saco es azul');
10         break;
11     case 'yellow':
12         console.log('El saco es amarillo');
13         break;
14     default:
15         console.log('No tengo en ese color');
16         break;
17 }
```

Bucles o Ciclos

Son **estructuras** que nos permiten **realizar iteraciones** o repeticiones de bloques de código en particular.

Existen **2 tipos** de estructuras:

- Basadas en una **cantidad finita y predefinida** de repeticiones.
- Basadas en una **condición** que repetirá el ciclo hasta que este se cumpla.

WHILE

En este caso, **el ciclo depende de una condición** que pueda ser o no numérica. Una vez cumplida la condición **el ciclo dejará de funcionar** o **puede que nunca comience** si la condición se encuentra cumplida antes de ser evaluada por la estructura.

```
1 let condition = Number(prompt('1. Ver saldo\n 0. Salir'));
2
3 while (condition != 0) {
4
5     switch (condition){
6         case 1:
7             alert('Su saldo es $ 0.00.-');
8             condition = 0;
9             break;
10        default:
11            alert('La opción es incorrecta, vuelva a elegir.')
12            break;
13    }
14    condition = Number(prompt('1. Ver saldo\n 0. Salir'));
15 }
16
```


DO WHILE

Es similar a la estructura WHILE, solo que **en este caso** el ciclo **se ejecutará al menos una vez** aunque la condición **no se cumpla**.

```
1  do {
2      var condition = Number(prompt('1. Ver saldo\n 0. Salir'));
3
4      switch (condition){
5          case 1:
6              alert('Su saldo es $ 0.00.-');
7              condition = 0;
8              break;
9          case 0:
10             alert('Gracias, vuelva pronto!');
11             condition = 0;
12             break;
13         default:
14             alert('La opción es incorrecta, vuelva a elegir.')
15             break;
16     }
17
18 } while (condition != 0);
```

FOR

Es el ciclo por excelencia, la cantidad de veces que se repite se encuentra definida por un número finito de veces.

Este ciclo consta de 3 partes: (un **contador**; un **límite**; una **actualización**).



```
1  for (let i = 0; i < 10; i++) {  
2      console.log('El valor de i es: ' + i);  
3  }
```

```
>  
for (let i = 0; i < 10; i++) {  
  console.log('El valor de i es: ' + i);  
}  
El valor de i es: 0  
El valor de i es: 1  
El valor de i es: 2  
El valor de i es: 3  
El valor de i es: 4  
El valor de i es: 5  
El valor de i es: 6  
El valor de i es: 7  
El valor de i es: 8  
El valor de i es: 9
```