

OPERADORES EN JAVASCRIPT

OPERADORES ARITMÉTICOS

- Todos los lenguajes de programación tienen operadores y, generalmente, suelen coincidir (salvo los operadores incremento y decremento que aparecieron con C++, siendo uno de sus rasgos más significativos).

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo
++	Incremento
--	Decremento

OPERADORES DE ASIGNACIÓN

- Otros operadores básicos son los operadores de asignación.

Operador	Ejemplo de uso
=	x = y;
+=	x += y; // igual que (x = x + y)
-=	x -= y; // igual que (x = x - y)
*=	x *= y; // igual que (x = x * y)
/=	x /= y; // igual que (x = x / y)
%=	x %= y; // igual que (x = x % y)

OPERADORES DE MANEJO DE STRINGS

- En JavaScript, se utilizan los operadores + y += para concatenar strings. Véase un ejemplo de uso:

```
var = "hola" + " " + "mundo";
```

- O lo que sería igual:

```
var = "hola";  
var += " ";  
var += "mundo";
```

OPERADORES LÓGICOS Y DE COMPARACIÓN

- Tanto los operadores lógicos como los de comparación son profusamente utilizados por los programadores. En el siguiente cuadro se muestran los operadores tanto lógicos como de comparación para los programadores JavaScript.

Operador	Descripción
==	Igual que
===	Igual valor y tipo
!=	Distinto
!==	Distinto valor o distinto tipo
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
?	Operador ternario

OPERADORES DE TIPO

- Los operadores de tipo permiten conocer si un objeto es una instancia de un tipo concreto o bien conocer el tipo de una variable. A continuación, se muestran los operadores de tipo disponibles en JavaScript:
 - `typeof`. Devuelve el tipo de una variable.
 - `instanceof`. Devuelve true si un objeto es una instancia de un tipo de objeto.

**SENTENCIA
IF...ELSE**

- El if...else es un tipo de instrucción condicional que ejecutará un bloque de código cuando la condición de la instrucción if sea verdadera. Si la condición es falsa falsy, se ejecutará el bloque else.

```
if (condicion es verdadera) {  
    // el código se ejecuta  
} else {  
    // el código se ejecuta  
}
```

EJEMPLOS DE SENTENCIAS IF...ELSE EN JAVASCRIPT

- En este ejemplo, la condición para la instrucción if es verdadera true, por lo que el mensaje impreso en la consola sería "Nick es un adulto."
- Pero si cambio la variable de edad a menos de 18, entonces la condición sería falsa false y el código ejecutaría el bloque else en su lugar.

```
const edad = 18;  
  
if (edad >= 18) {  
  console.log("Nick es un adulto.");  
} else {  
  console.log("Nick es un niño.");  
}
```

```
const edad = 12;  
  
if (edad >= 18) {  
  console.log("Nick es un adulto.");  
} else {  
  console.log("Nick es un niño.");  
}
```

EJEMPLOS DE MÚLTIPLES CONDICIONES (SENTENCIAS IF...ELSE IF... ELSE)

- Habrá momentos en los que desees probar múltiples condiciones. Ahí es donde entra el bloque else if.
- Cuando la sentencia if es falsa false, la computadora pasará a la sentencia else if. Si eso también es falso false, entonces se moverá al bloque else.
- En este ejemplo, el bloque else if se ejecutará porque Alice tiene entre 18 y 21 años de edad.

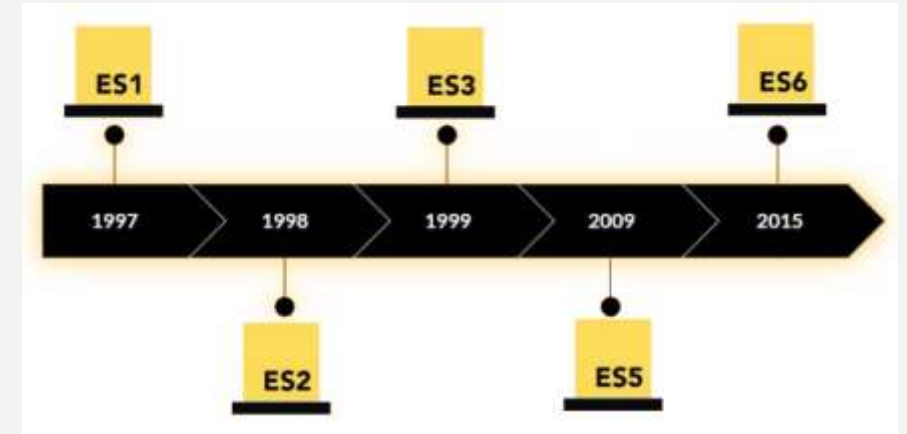
```
if (la condición 1 es verdadera) {  
    // el código se ejecuta  
} else if (la condición 2 es verdadera) {  
    // el código se ejecuta  
} else {  
    // el código se ejecuta  
}
```

```
const edad = 18;  
  
if (edad < 18) {  
    console.log("Alice es menor de 18 años.");  
} else if (edad >= 18 && edad <= 21) {  
    console.log("Alice tiene entre 18 y 21 años de edad.");  
} else {  
    console.log("Alice tiene mas de 21 años.");  
}
```



VARIABLES

ÁMBITO DE VARIABLES



- El **ámbito**, significa esencialmente **dónde están disponibles** estas variables para su uso. Las declaraciones var tienen un ámbito global o un ámbito de función/local.
- El **ámbito es global** cuando una variable var se declara **fuera de una función**. Esto significa que cualquier variable que se declare con var fuera de una función **está disponible para su uso en toda la pantalla**.
- var tiene un **ámbito local** cuando se **declara dentro de una función**. Esto significa que está disponible y solo se puede **acceder a ella dentro de esa función**.

EJEMPLO

```
var saludar = "hey, hola";  
  
function nuevaFuncion() {  
    var hola = "hola";  
}
```

- Aquí, saludar tiene un ámbito global porque existe fuera de la función mientras que hola tiene un ámbito local. Así que no podemos acceder a la variable hola fuera de la función. Así que si realizamos esto:

```
var tester = "hey, hola";  
  
function nuevaFuncion() {  
    var hola = "hola";  
}  
console.log(hola); // error: hola is not defined
```

- Obtendremos un error que se debe a que hola no está disponible fuera de la función.

LAS VARIABLES CON VAR SE PUEDEN VOLVER A DECLARAR Y MODIFICAR

- Esto significa que podemos hacer esto dentro del mismo ámbito y no obtendremos un error.

```
var saludar = "hey, hola";  
var saludar = "dice Hola tambien";
```

- y esto también

```
var saludar = "hey, hola";  
saludar = "dice Hola tambien";
```

HOISTING DE VAR

- Hoisting es un mecanismo de JavaScript en el que las variables y declaraciones de funciones se mueven a la parte superior de su ámbito antes de la ejecución del código. Esto significa que si hacemos esto:

```
console.log (saludar);  
var saludar = "dice hola"
```

- se interpreta así:

```
var saludar;  
console.log(saludar); // saludar is undefined  
saludar = "dice hola"
```

Entonces las variables con var se elevan a la parte superior de su ámbito y se inicializan con un valor de undefined.

PROBLEMA CON VAR

```
var saludar = "hey, hola";  
var tiempos = 4;  
  
if (tiempos > 3) {  
    var saludar = "dice Hola tambien";  
}  
  
console.log(saludar) // "dice Hola tambien"
```

- Por lo tanto, como tiempos > 3 devuelve true, saludar se redefine para "dice Hola tambien". Aunque esto no es un problema si quieres redefinir saludar a conciencia, se convierte en un problema cuando no te das cuenta de que la variable saludar ha sido definida antes.
- Si has utilizado saludar en otras partes de tu código, puede que te sorprenda la salida que puedes obtener. Esto probablemente causará muchos errores en tu código. Por eso son necesarios let y const.

LET

- **let tiene un ámbito de bloque**
- Un bloque es un trozo de código delimitado por {}. Un bloque vive entre llaves. Todo lo que está dentro de llaves es un bloque.
- Así que una variable declarada en un bloque con `let` solo está disponible para su uso dentro de ese bloque. Permíteme explicar esto con un ejemplo:

```
let saludar = "dice Hola";  
let tiempos = 4;  
  
if (tiempos > 3) {  
    let hola = "dice Hola tambien";  
    console.log(hola); // "dice Hola tambien"  
}  
console.log(hola) // hola is not defined
```

Vemos que el uso de `hola` fuera de su bloque (las llaves donde se definió) devuelve un **error**. Esto se debe a que las variables `let` tienen un alcance de bloque.

LET PUEDE MODIFICARSE PERO NO VOLVER A DECLARARSE.

- Al igual que var, una variable declarada con let puede ser actualizada dentro de su ámbito. A diferencia de var, una variable let no puede ser re-declarada dentro de su ámbito.

```
let saludar = "dice Hola";  
let saludar = "dice Hola tambien"; // error: Identifier 'saludar' has already been declared
```

HOISTING DE LET

- Al igual que `var`, las declaraciones `let` se elevan a la parte superior. A diferencia de `var` que se inicializa como `undefined`, la palabra clave `let` no se inicializa. Sí que si intentas usar una variable `let` antes de declararla, obtendrás un `Reference Error`.

CONST

- Las declaraciones `const` tienen un ámbito de bloque
- `const` no puede modificarse ni volver a declararse

```
const saludar = "dice Hola";  
saludar = "dice Hola tambien";// error: Assignment to constant variable.
```

```
const saludar = "dice Hola";  
const saludar = "dice Hola tambien";// error: Identifier 'saludar' has already been declared
```

COMPORTAMIENTO CON OBJETOS

- Mientras que un objeto `const` no se puede actualizar, las propiedades de este objeto sí se pueden actualizar.

```
const saludar = {  
  mensaje: "dice Hola",  
  tiempos: 4  
}
```

- mientras que no podemos hacer esto:

```
saludar = {  
  palabras: "Hola",  
  numero: "cinco"  
} // error: Assignment to constant variable.
```

- Si podemos hacer esto:
- Esto actualizará el valor de `saludar.mensaje` sin devolver errores.

```
saludar.mensaje = "dice Hola tambien";
```

HOISTING DE CONST

- Al igual que let, const las declaraciones const se elevan a la parte superior, pero no se inicializan.

DIFERENCIA ENTRE LAS 3

TIPO DE DATOS

TIPOS DE DATOS. ASIGNACIONES Y EXPRESIONES

- JavaScript es un lenguaje bastante simplificado en lo que a tipos de datos se refiere y, por ello, solamente tiene cinco tipos de datos:
- 1. String.
- 2. Number.
- 3. Boolean.
- 4. Array.
- 5. Object.

EJEMPLO DE UTILIZACIÓN DE CADA UNO

```
var edad = 25; // Number
var nombre = "Dimas"; // String
var asignatura = ["lengua", "mate", "cono"]; // Array
var persona = {nombre:"Dimas", apellido:"Moreno"}; // Objeto
```

- "Dimas" o "lengua" son literales y, por eso, aparecen entrecomillados. Para JavaScript, las siguientes dos líneas de código son iguales:

```
var dato = "Ronaldo " + 10;
var dato = "Ronaldo " + "10";
```

- JavaScript interpretará el número como un string.

EL VALOR NULL

- Null para los lenguajes de programación es nada o algo que no existe. Generalmente, cuando se asigna null a una variable, es porque es o será un objeto.

```
var persona = null;  
persona = {nombre:"Dimas", apellido:"Moreno"};
```

CONVERSIÓN ENTRE TIPOS DE DATOS

- JavaScript es un lenguaje interpretado, por lo tanto, la conversión entre un tipo de dato y otro es transparente al programador. JavaScript asignará el tipo de datos más acertado a la variable que el programador esté utilizando.
- No obstante, existen funciones de conversión como `String()` y `Number()` para convertir a cadenas de caracteres y números, respectivamente.
- También existen las funciones:

```
parseInt();//convierte una cadena en un numero entero  
parseFloat(); //convierte una cadena en un numero decimal;
```

- Para más info: <https://es.javascript.info/type-conversions>