1. Comentarios de una sola línea:

Estos comentarios comienzan con dos barras diagonales (`//`) y se extienden hasta el final de la línea. Se utilizan para agregar notas breves sobre el código que está en la misma línea.

Ejemplo:

JavaScript

```
// Esta variable almacena el nombre del usuario
var nombre = "Juan Pérez";

// Esta función muestra un mensaje de alerta
function mostrarAlerta() {
  alert("¡Hola, " + nombre + "!");
}
```

2. Comentarios de múltiples líneas:

Estos comentarios comienzan con una barra diagonal y un asterisco (`/*`) y terminan con un asterisco y una barra diagonal (`*/`). Se utilizan para agregar notas más extensas sobre el código, como explicaciones de cómo funciona una función o un algoritmo.

Ejemplo:

JavaScript

```
/*
 * Esta función calcula el promedio de una lista de números.
 *
 * @param {number[]} numeros La lista de números a promediar.
 * @return {number} El promedio de la lista de números.
 */
```

Los comentarios tienen muchos beneficios, entre ellos:

- **Mejoran la legibilidad del código:** Los comentarios ayudan a explicar lo que hace el código, lo que lo hace más fácil de entender para otros programadores, incluido usted mismo en el futuro.
- **Facilitan la depuración:** Los comentarios pueden ayudar a identificar la causa de un error en el código.
- **Documentan el código:** Los comentarios pueden usarse para documentar el código, lo que puede ser útil para otros programadores que necesiten entender cómo funciona.

Tipos de operadores en JavaScript

1. Operadores aritméticos:

- **Suma (+):** Suma dos valores numéricos.
- **Resta (-):** Resta un valor numérico de otro.
- **Multiplicación (*):** Multiplica dos valores numéricos.
- **División (/):** Divide un valor numérico por otro.
- **Resto (%):** Obtiene el resto de la división entre dos valores numéricos.
- **Exponenciación ():** Eleva un valor numérico a la potencia de otro.

```
const suma = 1 + 2; // 3

console.log(suma); // Imprime 3
```

```
const resto = 11 % 3; // 2

console.log(resto); // Imprime 2
```

2. Operadores de asignación:

- **Asignación simple (=):** Asigna un valor a una variable.
- **Suma y asignación (+=):** Suma un valor a una variable y luego le asigna el resultado.
- **Resta y asignación (-=):** Resta un valor a una variable y luego le asigna el resultado.
- **Multiplicación y asignación (=):* Multiplica un valor por una variable y luego le asigna el resultado.
- **División y asignación (/=):** Divide un valor por una variable y luego le asigna el resultado.
- **Resto y asignación (%=):** Obtiene el resto de la división entre un valor y una variable y luego le asigna el resultado.

```
let nombre = "Juan"; // Asigna el valor "Juan" a la variable nombre

console.log(nombre); // Imprime "Juan"
```

```
let numero = 10;

numero += 5; // Suma 5 al valor de la variable numero

console.log(numero); // Imprime 15
```

```
let cantidad = 20;

cantidad -= 3; // Resta 3 al valor de la variable cantidad

console.log(cantidad); // Imprime 17
```

```
let precio = 5;

precio *= 2; // Multiplica el valor de la variable precio por 2

console.log(precio); // Imprime 10
```

```
let distancia = 100;

distancia /= 4; // Divide el valor de la variable distancia por 4

console.log(distancia); // Imprime 25
```

```
let numero2 = 11;

numero2 %= 3; // Obtiene el resto de la división de la variable numero2

console.log(numero2); // Imprime 2
```

3. Operadores de comparación:

- **Igualdad (==):** Comprueba si dos valores son iguales.
- **Igualdad estricta (===):** Comprueba si dos valores son iguales y del mismo tipo.
- **Desigualdad (!=):** Comprueba si dos valores son diferentes.
- **Desigualdad estricta (!==):** Comprueba si dos valores son diferentes o de diferente tipo.
- **Mayor que (>):** Comprueba si un valor es mayor que otro.
- **Mayor o igual que (>=):** Comprueba si un valor es mayor o igual que otro.
- **Menor que (<):** Comprueba si un valor es menor que otro.
- **Menor o igual que (<=):** Comprueba si un valor es menor o igual que otro.

```
const num1 = 10;
const num2 = "10";

console.log(num1 == num2);
```

```
const num3 = 10;
const num4 = "10";

console.log(num3 === num4);
```

```
const num5 = 5;
const num6 = 10;

console.log(num5 != num6);
```

```
const num7 = 5;
const num8 = "5";

console.log(num7 !== num8);
```

```
const num13 = 5;
const num14 = 10;

console.log(num13 < num14);
```

4. Operadores lógicos:

- **And (&&):** Devuelve `true` si ambos operandos son `true`.
- **Or (||):** Devuelve `true` si al menos uno de los operandos es `true`.
- **Not (!):** Invierte el valor booleano de un operando.

```
const edad = 18;
const isEstudiante = true;

if (edad >= 18 && isEstudiante) {
  console.log("Puedes acceder al descuento para estudiantes");
} else {
  console.log("No cumples con los requisitos para el descuento");
}
```

```
const tienePermisoA = false;
const tienePermisoB = true;

if (tienePermisoA || tienePermisoB) {
  console.log("El usuario tiene acceso al sistema");
} else {
  console.log("El usuario no tiene acceso al sistema");
}
```

```
const estaDisponible = false;

if (!estaDisponible) {
  console.log("El producto no está disponible");
} else {
  console.log("El producto está disponible");
}
```

5. Operadores ternarios:

- **(condición) ? valor_si_verdadero : valor_si_falso**

Este operador evalúa una condición y devuelve un valor si la condición es verdadera y otro valor si la condición es falsa.

```
const edad = 17;
const mensaje = edad >= 18 ? "Eres mayor de edad" : "Eres menor de edad";
console.log(mensaje); // Imprime "Eres menor de edad"
```

6. Operadores de cadena:

- **Concatenación (+):** Concatena dos cadenas de texto.

```
const nombre = "Juan";
const apellido = "Pérez";
const nombreCompleto = nombre + " " + apellido;

console.log(nombreCompleto); // Imprime "Juan Pérez"
```

7. Operadores de tipo especial:

- **typeof:** Devuelve el tipo de dato de un valor.
- **instanceof:** Comprueba si un valor es una instancia de un tipo de dato.

```
const numero = 10;
const cadena = "Hola";
const booleano = true;

console.log(typeof numero); // Imprime "number"
console.log(typeof cadena); // Imprime "string"
console.log(typeof booleano); // Imprime "boolean"
```

8. Operadores unarios:

- **Incremento (++)**: Incrementa el valor de una variable en uno.
- **Decremento (--):** Decrementa el valor de una variable en uno.

```
let numero = 10;
// Incrementar el valor de la variable en 1
numero++;

console.log(numero); // Imprime 11

// Decrementar el valor de la variable en 1
numero--;

console.log(numero); // Imprime 10
```

Precedencia de operadores

La precedencia de operadores en JavaScript define el orden en que se evalúan las operaciones dentro de una expresión. Es importante comprender este orden para obtener el resultado esperado en tus scripts.

Reglas básicas:

1. **Mayor a menor:** Los operadores con mayor precedencia se evalúan primero.
2. **Igual precedencia:** Si dos operadores tienen la misma precedencia, se evalúan de izquierda a derecha.
3. **Paréntesis:** Los paréntesis fuerzan la evaluación de una expresión en el orden deseado, independientemente de la precedencia.

Orden	Operador	Descripción
1	(), [], ., new	Agrupación, acceso a propiedades, operador new
2	++, --	Incremento y decremento (pre y postfijo)
3	!, ~, typeof	Negación, complemento a bit, tipo de dato
4	*, /, %	Multiplicación, división, módulo
5	+, -	Suma y resta
6	<, <=, >, >=, ==,	Comparaciones
7	&&	Conjunción lógica (AND)
8	? :	Operador ternario (condicional)
9	operador =	Asignación

En un lenguaje de programación, una expresión es una combinación de variables, constantes, operadores y funciones que se evalúa para producir un valor. Las expresiones se utilizan para realizar cálculos, tomar decisiones y controlar el flujo de un programa.

- **Expresiones aritméticas:** Se utilizan para realizar operaciones matemáticas como suma, resta, multiplicación y división.
- **Expresiones lógicas:** Se utilizan para evaluar condiciones y tomar decisiones.
- **Expresiones de comparación:** Se utilizan para comparar dos valores y determinar si son iguales, diferentes, mayores o menores que.
- **Expresiones de asignación:** Se utilizan para asignar un valor a una variable.
- **Expresiones de cadena:** Se utilizan para trabajar con cadenas de texto.

Ejemplos de expresiones:

- `2 + 3` es una expresión aritmética que evalúa a 5.
- `x > 10` es una expresión lógica que evalúa a `true` si el valor de `x` es mayor que 10.
- `"Hola" + " mundo"` es una expresión de cadena que concatena las dos cadenas para formar "Hola mundo".

Precondiciones: son las condiciones que **deben cumplirse** antes de iniciar un proceso o algoritmo. Son como los requisitos previos que garantizan un funcionamiento correcto.

Ejemplo: para preparar un pastel, necesitas tener todos los ingredientes necesarios (harina, huevos, leche, etc.). Si no tienes todos los ingredientes, no puedes empezar a cocinar.

Entrada: son los datos o información que se **proporcionan** a un proceso o algoritmo para que funcione. Son como los ingredientes que se utilizan para obtener un resultado específico.

Ejemplo: en la receta del pastel, la entrada serían las cantidades específicas de cada ingrediente (2 tazas de harina, 3 huevos, etc.).

Poscondiciones: son las condiciones que **deben cumplirse** después de que un proceso o algoritmo haya finalizado. Son como el resultado final que se espera obtener.

Ejemplo: después de preparar un pastel, este debe estar cocido, esponjoso y tener un buen sabor.

Salida: son los datos o información que se **obtienen** como resultado de un proceso o algoritmo. Son como el producto final que se genera.

Ejemplo: en la receta del pastel, la salida sería el pastel terminado listo para ser disfrutado.