**Tipos de operadores (de un ejemplo de cada uno)**

hay varios tipos de operadores que se utilizan para realizar diferentes operaciones, como operaciones aritméticas, comparaciones, asignaciones y lógicas. Aquí tienes ejemplos de cada uno de los tipos de operadores:

1. **Operadores Aritméticos:**
   * Se utilizan para realizar operaciones matemáticas.

// Ejemplo de operadores aritméticos

let x = 5;

let y = 2;

console.log(x + y); // Suma: Imprime 7

console.log(x - y); // Resta: Imprime 3

console.log(x \* y); // Multiplicación: Imprime 10

console.log(x / y); // División: Imprime 2.5

console.log(x % y); // Módulo: Imprime 1 (resto de la división)

console.log(++x); // Incremento: Imprime 6

console.log(--y); // Decremento: Imprime 1

1. **Operadores de Comparación**:

* Se utilizan para comparar dos valores y devolver un resultado booleano.

// Ejemplo de operadores de comparación

let a = 5;

let b = 10;

console.log(a == b); // Igual a: Imprime false

console.log(a != b); // No igual a: Imprime true

console.log(a > b); // Mayor que: Imprime false

console.log(a < b); // Menor que: Imprime true

console.log(a >= b); // Mayor o igual que: Imprime false

console.log(a <= b); // Menor o igual que: Imprime true

1. **Operadores de Asignación**:

* Se utilizan para asignar valores a variables.

// Ejemplo de operadores de asignación

let c = 10;

let d = 5;

c += d; // Equivalente a c = c + d;

console.log(c); // Imprime 15

d \*= 2; // Equivalente a d = d \* 2;

console.log(d); // Imprime 10

1. **Operadores Lógicos**:

* Se utilizan para combinar expresiones condicionales y devolver un resultado booleano.

// Ejemplo de operadores lógicos

let llueve = true;

let haceFrio = false;

console.log(llueve && haceFrio); // AND lógico: Imprime false

console.log(llueve || haceFrio); // OR lógico: Imprime true

console.log(!llueve); // NOT lógico: Imprime false

1. **Operadores de Concatenación de Cadenas**:

* Se utilizan para concatenar cadenas de texto.

/ Ejemplo de operador de concatenación de cadenas

let nombre = "Juan";

let apellido = "Pérez";

let nombreCompleto = nombre + " " + apellido;

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

**Condicionales (de un ejemplo)**

Los condicionales en JavaScript se utilizan para ejecutar diferentes bloques de código según una condición específica. El condicional más común es la instrucción **if**, que ejecuta un bloque de código si una condición dada es verdadera. También se pueden utilizar instrucciones **else if** y **else** para especificar bloques de código alternativos que se ejecutan cuando la condición **if** es falsa.

// Ejemplo de condicional if

let edad = 20;

if (edad >= 18) {

console.log("Eres mayor de edad.");

} else {

console.log("Eres menor de edad."

/ Ejemplo de condicional else if

let puntaje = 85;

if (puntaje >= 90) {

console.log("Aprobaste con una A.");

} else if (puntaje >= 80) {

console.log("Aprobaste con una B.");

} else if (puntaje >= 70) {

console.log("Aprobaste con una C.");

} else {

console.log("Has reprobado.");

}

**Ciclos (Loops) (de un ejemplo de cada uno)**

hay varios tipos de ciclos o bucles que se utilizan para repetir la ejecución de un bloque de código hasta que se cumple una condición específica. Los tres tipos principales de ciclos son: for, while y do...while.

**Ciclo for**:

* Se utiliza para iterar sobre una secuencia de valores.

// Ejemplo de ciclo for

for (let i = 0; i < 5; i++) {

console.log("Iteración " + (i + 1));

}

**Ciclo while**:

* Se utiliza para ejecutar un bloque de código mientras una condición específica sea verdadera.

// Ejemplo de ciclo while

let contador = 0;

while (contador < 5) {

console.log("Iteración " + (contador + 1));

contador++;

}

**Ciclo do...while**:

* Similar al ciclo **while**, pero garantiza que el bloque de código se ejecute al menos una vez antes de verificar la condición.

// Ejemplo de ciclo do...while

let x = 0;

do {

console.log("Iteración " + (x + 1));

x++;

} while (x < 5);

**Manejo de errores (de un ejemplo)**

// Función que divide dos números y maneja cualquier error que pueda ocurrir

function dividir(a, b) {

try {

if (b === 0) {

throw new Error("División por cero no permitida");

}

return a / b;

} catch (error) {

// Captura y maneja el error

console.error("Ha ocurrido un error:", error.message);

return null; // Devuelve null para indicar que hubo un error

}

}

// Ejemplo de uso de la función dividir

console.log(dividir(10, 2)); // Imprime: 5

console.log(dividir(8, 0)); // Imprime: Ha ocurrido un error: División por cero no permitida

console.log(dividir(15, 3)); // Imprime: 5

**Break & Continue (de un ejemplo de cada uno)**

las palabras clave break y continue se utilizan dentro de bucles para controlar el flujo de ejecución

**Ejemplo de break**:

* La palabra clave **break** se utiliza para salir de un bucle antes de que se complete su iteración normal.

// Ejemplo de break

for (let i = 0; i < 5; i++) {

console.log("Iteración " + (i + 1));

if (i === 2) {

break; // Salir del bucle cuando i sea igual a 2

}

}

**Ejemplo de continue**:

* La palabra clave **continue** se utiliza para saltar una iteración del bucle y continuar con la siguiente iteración

// Ejemplo de continue

for (let i = 0; i < 5; i++) {

if (i === 2) {

continue; // Saltar la iteración cuando i sea igual a 2

}

console.log("Iteración " + (i + 1));

}

**Que es la destructuración (de un ejemplo)**

La destructuración es una característica de JavaScript que permite descomponer o extraer valores de objetos y arreglos en variables individuales de una manera más concisa y legible. Esto simplifica la asignación de valores a variables, especialmente cuando trabajas con objetos o arreglos complejos.

// Ejemplo de destructuración de objetos

let persona = {

nombre: 'Juan',

edad: 30,

ocupacion: 'Desarrollador'

};

// Extrayendo valores del objeto persona en variables individuales

let { nombre, edad, ocupacion } = persona;

console.log(nombre); // Imprime: Juan

console.log(edad); // Imprime: 30

console.log(ocupacion); // Imprime: Desarrollador

En este ejemplo, la destructuración se utiliza para extraer las propiedades **nombre**, **edad** y **ocupacion** del objeto **persona** y asignarlas a variables del mismo nombre. Esto elimina la necesidad de acceder a las propiedades del objeto utilizando la notación de puntos (**persona.nombre**, **persona.edad**, etc.).

// Ejemplo de destructuración de arreglos

let numeros = [1, 2, 3];

// Extrayendo valores del arreglo numeros en variables individuales

let [num1, num2, num3] = numeros;

console.log(num1); // Imprime: 1

console.log(num2); // Imprime: 2

console.log(num3); // Imprime: 3

En este ejemplo, la destructuración se utiliza para extraer los valores individuales del arreglo **numeros** y asignarlos a variables **num1**, **num2** y **num3**.

**Que son los objetos literales (de un ejemplo)**

Los objetos literales, también conocidos como objetos de notación literal, son una forma de definir y crear objetos en JavaScript utilizando una sintaxis simple y concisa. Los objetos literales se definen entre llaves {} y consisten en pares de clave-valor separados por comas.

// Ejemplo de objeto literal

let persona = {

nombre: 'Juan',

edad: 30,

ocupacion: 'Desarrollador',

saludar: function() {

console.log('¡Hola! Soy ' + this.nombre + ', tengo ' + this.edad + ' años y soy ' + this.ocupacion + '.');

}

};

// Accediendo a propiedades y métodos del objeto literal

console.log(persona.nombre); // Imprime: Juan

console.log(persona.edad); // Imprime: 30

console.log(persona.ocupacion); // Imprime: Desarrollador

persona.saludar(); // Imprime: ¡Hola! Soy Juan, tengo 30 años y soy Desarrollador.

En este ejemplo, el objeto literal **persona** tiene cuatro propiedades: **nombre**, **edad**, **ocupacion** y **saludar**, donde **nombre**, **edad** y **ocupacion** son pares de clave-valor que representan información sobre una persona, y **saludar** es un método que imprime un mensaje de saludo utilizando las propiedades del objeto.

**Que son los parámetros REST y Operador Spread (de un ejemplo de cada uno)**

Los parámetros REST y el operador Spread son características de JavaScript que se utilizan para trabajar con un número variable de argumentos en funciones y para clonar objetos y arreglos de una manera más conveniente, respectivamente.

**Parámetros REST**:

* Los parámetros REST permiten a una función aceptar un número variable de argumentos como un arreglo.

// Ejemplo de parámetros REST

function sumar(...numeros) {

let total = 0;

for (let num of numeros) {

total += num;

}

return total;

}

console.log(sumar(1, 2, 3, 4, 5)); // Imprime: 15

console.log(sumar(10, 20, 30)); // Imprime: 60

En este ejemplo, la función **sumar** acepta un número variable de argumentos utilizando el parámetro REST **...numeros**, que recopila todos los argumentos pasados a la función en un arreglo llamado **numeros**. La función luego suma todos los números dentro del arreglo **numeros** y devuelve el resultado.

1. **Operador Spread**:
   * El operador Spread se utiliza para expandir los elementos de un arreglo o los pares clave-valor de un objeto en un lugar donde se esperan múltiples elementos.

// Ejemplo de operador Spread con arreglos

let arreglo1 = [1, 2, 3];

let arreglo2 = [...arreglo1, 4, 5, 6];

console.log(arreglo2); // Imprime: [1, 2, 3, 4, 5, 6]

// Ejemplo de operador Spread con objetos

let objeto1 = { a: 1, b: 2 };

let objeto2 = { ...objeto1, c: 3, d: 4 };

console.log(objeto2); // Imprime: { a: 1, b: 2, c: 3, d: 4 }

**Que son las arrow functions (de un ejemplo)**

Las arrow functions, también conocidas como funciones flecha, son una característica introducida en ECMAScript 6 (también conocido como ES6) de JavaScript. Proporcionan una sintaxis más concisa para definir funciones en comparación con las funciones tradicionales, y tienen un comportamiento especial con respecto al alcance (this). Aquí tienes un ejemplo de una arrow function:

// Ejemplo de arrow function

const sumar = (a, b) => {

return a + b;

};

console.log(sumar(3, 5)); // Imprime: 8

En este ejemplo, **sumar** es una arrow function que toma dos parámetros **a** y **b**, y devuelve la suma de ambos. La sintaxis de la arrow function es **(parametros) => { cuerpo de la función }**, donde los parámetros están entre paréntesis y el cuerpo de la función está entre llaves. En este caso, el cuerpo de la función es una expresión que devuelve la suma de **a** y **b**.