**Que son las variables var y las variables let (de un ejemplo)**

**var**:

* Antes de la llegada de **let** en ECMAScript 6 (también conocido como ES6), **var** era la forma más común de declarar variables en JavaScript.
* Las variables declaradas con **var** tienen un alcance de función o global, lo que significa que pueden ser accesibles dentro de la función en la que se declararon o en todo el ámbito global del script, dependiendo de dónde se declaren.
* Las variables **var** pueden ser re-declaradas y actualizadas dentro de su ámbito.

Texto

Descripción generada automáticamente

**let**:

* Introducido en ECMAScript 6 (ES6), **let** tiene un alcance de bloque, lo que significa que solo están disponibles dentro del bloque en el que se declaran (por ejemplo, dentro de un bucle **for**, una declaración **if**, un bloque de función, etc.).
* Las variables **let** no pueden ser re-declaradas dentro del mismo ámbito y no permiten la duplicación de nombres de variable dentro de un mismo bloque.
* A diferencia de **var**, las variables **let** no se elevan al principio del ámbito, lo que significa que no puedes acceder a ellas antes de su declaración.

Texto

Descripción generada automáticamente

**Que son las constantes (conts) (de un ejemplo)**

las constantes (const) en JavaScript son variables cuyo valor no puede cambiar después de su declaración inicial. Son útiles para almacenar valores que no deben modificarse durante la ejecución del programa, como constantes matemáticas, valores de configuración, o referencias a objetos inmutables.

// Ejemplo de constante

const PI = 3.14159;

console.log(PI); // Imprime: 3.14159

// Intentar cambiar el valor de una constante generará un error

// PI = 3; // Esto generaría un error

// Las constantes deben asignarse un valor al momento de la declaración

// const MY\_CONSTANT; // Esto generaría un error, ya que no se ha asignado un valor a la constante

// Las constantes pueden ser de cualquier tipo de dato

const SALUDO = "¡Hola, mundo!";

console.log(SALUDO); // Imprime: ¡Hola, mundo!

// Las constantes pueden ser objetos o arreglos, y sus propiedades pueden modificarse

const persona = { nombre: "Juan", edad: 30 };

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

persona.edad = 31; // Esto es válido, ya que no estamos cambiando la referencia de la constante, solo sus propiedades

console.log(persona.edad); // Imprime: 31

// Sin embargo, no podemos asignar una nueva referencia a la constante

// persona = { nombre: "María", edad: 25 }; // Esto generaría un error

**Que son las cadenas de texto (Strings) (de un ejemplo)**

Las cadenas de texto, también conocidas como strings en inglés, son una secuencia de caracteres encerrados entre comillas simples (''), comillas dobles ("") o comillas invertidas (``). En JavaScript, las cadenas de texto son utilizadas para representar datos de texto.

**/ Ejemplo de cadenas de texto**

let mensaje1 = '¡Hola, mundo!';

let mensaje2 = "Esta es una cadena de texto.";

let mensaje3 = `Las comillas invertidas también pueden usarse para crear cadenas de texto.`;

console.log(mensaje1); // Imprime: ¡Hola, mundo!

console.log(mensaje2); // Imprime: Esta es una cadena de texto.

console.log(mensaje3); // Imprime: Las comillas invertidas también pueden usarse para crear cadenas de texto.

**Que son las plantillas de cadenas texto (Template Strings) (de un ejemplo)**

Las plantillas de cadenas de texto, también conocidas como template strings en inglés, son una característica introducida en ECMAScript 6 (ES6) que permite crear cadenas de texto de una manera más legible y flexible, permitiendo la interpolación de variables y expresiones dentro de la cadena utilizando la sintaxis de comillas invertidas (``).

**// Ejemplo de plantillas de cadenas de texto**

let nombre = 'Juan';

let edad = 30;

// Utilizando plantillas de cadenas de texto

let mensaje = `Hola, mi nombre es ${nombre} y tengo ${edad} años.`;

console.log(mensaje); // Imprime: Hola, mi nombre es Juan y tengo 30 años.

**Que son los números (Numbres) (de un ejemplo)**

Los números, también conocidos como números en inglés, son un tipo de datos en JavaScript que representan valores numéricos, ya sean enteros o de punto flotante (números decimales). En JavaScript, los números pueden ser escritos con o sin decimales.

**// Ejemplo de números**

let entero = 42; // Un número entero

let decimal = 3.14; // Un número decimal (de punto flotante)

console.log(entero); // Imprime: 42

console.log(decimal); // Imprime: 3.14

// Se pueden realizar operaciones matemáticas con números

let suma = entero + decimal;

console.log(suma); // Imprime: 45.14

// Los números también pueden ser negativos

let negativo = -7;

console.log(negativo); // Imprime: -7

// Los números en JavaScript pueden representar valores muy grandes o pequeños

let muyGrande = 1.5e10; // 1.5 \* 10^10

let muyPequenio = 2.5e-4; // 2.5 \* 10^(-4)

console.log(muyGrande); // Imprime: 15000000000

console.log(muyPequenio); // Imprime: 0.00025

**Que son los Booleans (de un ejemplo)**

Los booleanos, también conocidos como valores booleanos, son un tipo de datos en JavaScript que representan un valor de verdad, es decir, pueden ser true (verdadero) o false (falso). Estos valores son fundamentales para el control de flujo y la toma de decisiones en programación.

**// Ejemplo de booleanos**

let esMayorDeEdad = true; // Variable booleana que indica si una persona es mayor de edad

let esDiaSoleado = false; // Variable booleana que indica si el día está soleado

console.log(esMayorDeEdad); // Imprime: true

console.log(esDiaSoleado); // Imprime: false

// Los booleanos se utilizan comúnmente en estructuras condicionales

if (esMayorDeEdad) {

console.log("La persona es mayor de edad");

} else {

console.log("La persona es menor de edad");

}

if (esDiaSoleado) {

console.log("Hace sol hoy");

} else {

console.log("No hace sol hoy");

}

**Que son los undefined, null y Nan (de un ejemplo)**

En JavaScript, undefined, null y NaN son valores especiales que representan diferentes conceptos:

1. **undefined:** Este valor se asigna automáticamente a las variables que han sido declaradas pero no inicializadas con un valor. También puede ser el valor de retorno de una función si no se especifica explícitamente lo que debe devolver.

**let variableSinValor;**

console.log(variableSinValor); // Imprime: undefined

function obtenerValor() {

// No se especifica un valor de retorno

}

console.log(obtenerValor()); // Imprime: undefined

1. **null**: Este valor se utiliza para indicar la ausencia intencional de cualquier valor o referencia de objeto. Es diferente de **undefined**, que generalmente indica que una variable no está definida o que una función no tiene un valor de retorno.

let objetoNulo = null;

console.log(objetoNulo); // Imprime: null

1. **NaN** (Not a Number): Este valor se utiliza para representar un valor que no es un número válido en JavaScript. Puede ocurrir, por ejemplo, al intentar realizar operaciones matemáticas inválidas, como dividir una cadena por un número.

let resultado = "texto" / 2;

console.log(resultado); // Imprime: NaN

**Que son las funciones y que tipos de funciones existente en JavaScript (de ejemplos)**

las funciones son bloques de código reutilizable que pueden ser llamados o invocados para realizar una tarea específica. Las funciones pueden aceptar parámetros como entrada y pueden devolver un valor como salida. Son una parte fundamental del lenguaje y se utilizan para organizar y reutilizar el código de manera eficiente.

Existen varios tipos de funciones en JavaScript, incluyendo:

1. **Funciones Declarativas**: También conocidas como funciones nombradas o funciones con nombre, se declaran utilizando la palabra clave **function**. Pueden ser llamadas antes de que se definan en el código, debido al hoisting.

// Ejemplo de función declarativa

function saludar(nombre) {

return "¡Hola, " + nombre + "!";

}

console.log(saludar("Juan")); // Imprime: ¡Hola, Juan!

1. **Expresiones de Funciones**: Son funciones asignadas a una variable o propiedad de objeto. Pueden ser anónimas o nombradas.

// Ejemplo de expresión de función anónima

let sumar = function(a, b) {

return a + b;

};

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

// Ejemplo de expresión de función nombrada

let multiplicar = function multiplicar(a, b) {

return a \* b;

};

console.log(multiplicar(2, 4)); // Imprime: 8

1. **Funciones Flecha (Arrow Functions)**: Son una forma más concisa de escribir funciones en JavaScript introducidas en ES6. Tienen una sintaxis más corta y conservan el contexto **this** del entorno en el que se definen.

// Ejemplo de función flecha

let cuadrado = (x) => {

return x \* x;

};

console.log(cuadrado(3)); // Imprime: 9

1. **Funciones Constructoras**: Son funciones utilizadas junto con el operador **new** para crear objetos. Se utilizan como plantillas para crear múltiples objetos del mismo tipo.

// Ejemplo de función constructora

function Persona(nombre, edad) {

this.nombre = nombre;

this.edad = edad;

}

let persona1 = new Persona("Juan", 30);

let persona2 = new Persona("María", 25);

console.log(persona1); // Imprime: Persona { nombre: 'Juan', edad: 30 }

console.log(persona2); // Imprime: Persona { nombre: 'María', edad: 25 }

1. **Funciones anónimas**: Las funciones anónimas son útiles cuando solo necesitas definir una función en un lugar específico del código y no la vas a reutilizar en otros lugares.

// Ejemplo de función anónima

let saludar = function(nombre) {

return "¡Hola, " + nombre + "!";

};

console.log(saludar("Juan")); // Imprime: ¡Hola, Juan!

**Que son los arreglos (Arrays) (de ejemplos)**

los arreglos (arrays en inglés) son estructuras de datos que se utilizan para almacenar una colección ordenada de elementos. Los elementos de un arreglo pueden ser de cualquier tipo de datos, como números, strings, booleanos, objetos u otros arreglos. Cada elemento en un arreglo tiene una posición numérica conocida como índice, que comienza desde 0 para el primer elemento y se incrementa secuencialmente.

**// Ejemplo de creación de un arreglo con números**

let numeros = [1, 2, 3, 4, 5];

// Ejemplo de creación de un arreglo con strings

let colores = ['rojo', 'verde', 'azul'];

// Ejemplo de creación de un arreglo mixto

let mixto = [1, 'dos', true, [6, 7, 8]];

// También se puede crear un arreglo vacío y luego añadir elementos

let vacio = [];

vacio.push(10);

vacio.push(20);

**// Accediendo a elementos por su índice**

console.log(numeros[0]); // Imprime: 1

console.log(colores[1]); // Imprime: verde

// Accediendo a elementos de un arreglo dentro de otro arreglo

console.log(mixto[3][0]); // Imprime: 6

**// Modificando un elemento del arreglo**

numeros[2] = 30;

// Añadiendo un nuevo elemento al final del arreglo

colores.push('amarillo');

// Eliminando el último elemento del arreglo

mixto.pop();

console.log(numeros.length); // Imprime: 5

console.log(colores.length); // Imprime: 4

**Que son los objetos en JavaScript (de un ejemplo)**

los objetos son estructuras de datos que permiten almacenar datos y funciones relacionadas en un solo lugar. Los objetos pueden representar entidades del mundo real, como personas, productos o cualquier otro concepto, y se componen de pares de clave-valor, donde cada clave es una cadena única que actúa como identificador del valor correspondiente.

// Ejemplo de objeto en JavaScript

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 del objeto

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

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

// Llamando a un método del objeto

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