## Conceptos fundamentales en JavaScript

### Variables

Son espacios de memoria que almacenan información temporalmente durante la ejecución del programa.

**Estructura:**

let nombreVariable = valor;  
const CONSTANTE = valorFijo;

**Ejemplo:**

let usuario = "Laura";  
const ID\_CLIENTE = 7890;

### Tipos de datos

Categorías que determinan cómo se almacena y procesa la información.

* **Primitivos:** Strings, Números, Booleanos, Null, Undefined, Symbol.
* **No primitivos:** Objetos, Arrays, Funciones.

**Estructura:**

let texto = "cadena";  
let numero = 123;  
let booleano = true;  
let vacio = null;  
let indefinido;  
let objeto = { clave: valor };  
let lista = [elemento1, elemento2];

**Ejemplo:**

let nombre = "Informe";  
let cantidad = 50;  
let activo = true;  
let vacio = null;  
let sinDefinir;  
let empleado = { nombre: "Ana", cargo: "Gerente" };

### Operadores

Permiten realizar cálculos y comparaciones.

**Estructura:**

// Aritméticos  
+, -, \*, /, %  
  
// Asignación  
=, +=, -=  
  
// Comparación  
==, ===, !=, !==, >, <  
  
// Lógicos  
&&, ||, !

**Ejemplo:**

let suma = 5 + 3;  
let division = 20 / 4;  
  
let contador = 10;  
contador -= 2;  
  
console.log(10 === "10"); // false  
console.log(true && false); // false  
console.log(true || false); // true

### Condicionales

Evalúan condiciones para decidir qué bloque de código ejecutar según si se cumple o no una condición lógica. Son esenciales para la toma de decisiones en el flujo del programa.

* **if-else:** Se utiliza cuando se desea ejecutar un bloque de código si una condición se cumple y otro bloque si no se cumple. Útil para decisiones binarias como validar edad mínima o si un campo está vacío.
* **if-else if-else:** Se emplea cuando se deben evaluar múltiples condiciones secuenciales. Ideal para establecer rangos como calificaciones, niveles de acceso o tarifas por tramos.
* **Operador ternario:** Alternativa corta al if-else. Se utiliza para asignar rápidamente valores o ejecutar acciones simples según una condición. Muy útil para renderizados condicionales o ajustes rápidos de estados.

Estos condicionales ayudan a controlar el flujo del programa y adaptar la lógica del sistema en tiempo real según entradas, estados o reglas del negocio. Son esenciales para la toma de decisiones en el flujo del programa.

**Estructura if-else:**

if (condicion) {  
 // código si verdadero  
} else {  
 // código si falso  
}

**Estructura if-else if-else:**

if (condicion1) {  
 // código  
} else if (condicion2) {  
 // otro código  
} else {  
 // código por defecto  
}

**Estructura operador ternario:**

condicion ? valorSiVerdadero : valorSiFalso;

**Ejemplos:**

let edad = 20;  
if (edad >= 18) {  
 console.log("Mayor de edad");  
} else {  
 console.log("Menor de edad");  
}  
  
let puntaje = 85;  
if (puntaje >= 90) {  
 console.log("Excelente");  
} else if (puntaje >= 70) {  
 console.log("Aceptable");  
} else {  
 console.log("Requiere mejorar");  
}  
  
let resultado = puntaje >= 70 ? "Aprobado" : "Reprobado";  
console.log(resultado);

### Switch

Evalúa múltiples casos específicos según un valor.

**Estructura:**

switch (variable) {  
 case valor1:  
 // código  
 break;  
 case valor2:  
 // código  
 break;  
 default:  
 // código por defecto  
}

**Ejemplo:**

let dia = "Lunes";  
switch (dia) {  
 case "Lunes":  
 console.log("Inicio de semana");  
 break;  
 case "Viernes":  
 console.log("Cierre semanal");  
 break;  
 default:  
 console.log("Día regular");  
}

### Bucles

Permiten ejecutar repetidamente un bloque de código mientras se cumpla una condición. Son esenciales para automatizar tareas repetitivas como recorrer listas, esperar eventos o realizar cálculos acumulativos.

* **for:** Recorre un bloque de código un número determinado de veces. Ideal cuando conoces cuántas veces deseas repetir una acción.
* **while:** Ejecuta un bloque mientras una condición sea verdadera. Ideal cuando no sabes cuántas veces repetir, pero tienes una condición clara.
* **do-while:** Similar al while, pero garantiza al menos una ejecución del bloque.
* **for-of:** Recorre directamente los elementos de un array o iterable, útil para trabajar con colecciones sin preocuparte del índice.

**Estructuras:**

**for:**

for (inicialización; condición; incremento) {  
 // código repetido  
}

**while:**

while (condición) {  
 // código  
}

**do-while:**

do {  
 // código  
} while (condición);

**for-of:**

for (const item of array) {  
 // código  
}

**Ejemplos:**

for (let i = 1; i <= 3; i++) {  
 console.log(`Iteración ${i}`);  
}  
  
let contador = 3;  
while (contador > 0) {  
 console.log(contador);  
 contador--;  
}  
  
let intentos = 0;  
do {  
 console.log("Intento número", intentos);  
 intentos++;  
} while (intentos < 2);  
  
const nombres = ["Ana", "Luis", "Pedro"];  
for (const nombre of nombres) {  
 console.log(nombre);  
}

### Funciones

Bloques reutilizables que ejecutan tareas específicas. Las funciones permiten estructurar el código de manera modular y reutilizable. Existen tres tipos comunes:

* **Funciones declaradas:** Se definen con la palabra clave function. Se elevan (hoisting), por lo que pueden usarse antes de ser declaradas.
* **Funciones anónimas:** No tienen nombre. Se asignan normalmente a variables. No se elevan.
* **Funciones flecha (arrow functions):** Sintaxis concisa, ideales para funciones simples. No tienen su propio this.

**Estructuras:**

**Declarada:**

function nombreFuncion(parámetros) {  
 return valor;  
}

**Anónima:**

const nombre = function(parámetros) {  
 return valor;  
};

**Flecha:**

const nombre = (parámetros) => valor;

**Ejemplos:**

function sumar(a, b) {  
 return a + b;  
}  
console.log(sumar(2, 3));

const saludo = function(nombre) {  
 return `Hola ${nombre}`;  
};  
console.log(saludo("Carlos"));

const multiplicar = (x, y) => x \* y;  
console.log(multiplicar(4, 5));

### Estructuras de datos

Organizan eficientemente la información para acceder, modificar o recorrer datos de forma óptima.

**Array:** Colección ordenada de elementos.

**Estructura:**

const arreglo = [valor1, valor2, valor3];

**Métodos comunes:**

* push(): agrega al final
* pop(): elimina el último
* shift(): elimina el primero
* unshift(): agrega al inicio
* length: devuelve la cantidad de elementos
* forEach(), map(), filter(), find()

**Objeto:** Conjunto de pares clave-valor.

**Estructura:**

const objeto = {  
 clave1: valor1,  
 clave2: valor2  
};

**Métodos comunes:**

* Acceso: objeto.clave o objeto["clave"]
* Modificar: objeto.clave = nuevoValor
* Eliminar: delete objeto.clave
* Listar claves/valores: Object.keys(), Object.values()

**Ejemplo:**

const tareas = ["Reunión", "Informe"];  
tareas.push("Entrega");  
  
const producto = { nombre: "Laptop", precio: 1500 };  
producto.precio = 1200;

### Valores Truthy/Falsy

Determinan evaluación lógica de valores en contextos booleanos.

**Estructura Truthy:**

if (valor) {  
 // se ejecuta si valor es truthy  
}

**Estructura Falsy:**

if (!valor) {  
 // se ejecuta si valor es falsy  
}

**Ejemplos Truthy:**

if ("texto") console.log("Truthy: texto no vacío");  
if (42) console.log("Truthy: número distinto de cero");  
if ([]) console.log("Truthy: array vacío");  
if ({}) console.log("Truthy: objeto vacío");

**Ejemplos Falsy:**

if (!"") console.log("Falsy: cadena vacía");  
if (!0) console.log("Falsy: número cero");  
if (!null) console.log("Falsy: valor null");  
if (!undefined) console.log("Falsy: undefined");  
if (!NaN) console.log("Falsy: NaN");