UD 2. Flujo y Funciones

1. CONTROL DE FUJO

1.1. Condicionales

- IF

Ejecuta un bloque de código si la condición se cumple

```
if (condición) {
   // código a ejecutar si la condición es true
}
```

Un ejemplo seria:

```
let edad = 18;

if (edad >= 18) {
   console.log("Eres mayor de edad.");
}
```

- IF...ESLE

Ejecuta un bloque si la condición es verdadera, y otro si es falsa.

```
if (condición) {
   // se ejecuta si es true
} else {
   // se ejecuta si es false
}
```

Un ejemplo seria:

```
let hora = 14;

if (hora < 12) {
   console.log("Buenos días");
} else {
   console.log("Buenas tardes");
}
```

- IF...ESE IF...ELSE

  Permite evaluar múltiples condiciones en cadena

```
if (condición1) {
   // código si condición1 es true
} else if (condición2) {
   // código si condición2 es true
} else {
   // código si ninguna condición anterior fue true
}
```

  Ejemplo:

```
let nota = 85;

if (nota >= 90) {
   console.log("Excelente");
} else if (nota >= 70) {
   console.log("Aprobado");
} else {
   console.log("Reprobado");
}
```

- Operador ternario (? :)

  Forma abreviada de escribir un if...else simple. Solo para asignaciones o expresiones cortas.

```
condición ? valorSiTrue : valorSiFalse;
```

  Ejemplo:

```
let esMayor = (edad >= 18) ? "Sí" : "No";
console.log("¿Es mayor de edad? " + esMayor);
```

1.2. Switch y bucles

Evalúa una expresión y ejecuta el bloque de código que coincida con el valor. Útil cuando tienes muchas condiciones con valores específicos.

```
switch (expresión) {
  case valor1:
    // código
    break;
  case valor2:
    // código
    break;
  default:
    // código si no coincide ningún case
}
```

Ejemplo

```
let dia = 3;

switch (dia) {
  case 1:
    console.log("Lunes");
    break;
  case 2:
    console.log("Martes");
    break;
  case 3:
    console.log("Miércoles");
    break;
  default:
    console.log("Día no válido");
}
```

2. FUNCIONES

2.1. Funciones declaradas y de flecha

Las **funciones declaradas** y las **funciones flecha** (arrow functions) son dos formas muy comunes de definir funciones en JavaScript, pero tienen diferencias importantes en cuanto a **sintaxis, comportamiento y uso**.

- FUNCIÓN DECLARADA

```
function nombre(param1, param2) {
  // cuerpo de la función
  return resultado;
}
```

Ejemplo

```
function sumar(a, b) {
  return a + b;
}
```

- FUNCION FLECHA

```
const nombre = (param1, param2) => {
  // cuerpo de la función
  return resultado;
};
```

Ejemplo

```
const sumar = (a, b) => {
  return a + b;
};
```

2.2. closures

Un **closure** es una **función que recuerda y accede al ámbito (scope) en el que fue creada**, incluso después de que ese ámbito haya terminado de ejecutarse.

Es decir: **una función interna que tiene acceso a las variables de su función externa, incluso después de que la función externa haya retornado**.

```
function funcionExterna() {
  let variableExterna = "¡Hola desde afuera!";

  function funcionInterna() {
    console.log(variableExterna); // ← ¡Esto es el closure!
  }

  return funcionInterna;
}

const miFuncion = funcionExterna();
miFuncion(); // ¡Hola desde afuera!
```

Ejemplo:

```
function crearContador() {
  let contador = 0;

  return function() {
    contador++;
    return contador;
  };
}

const contar = crearContador();

console.log(contar()); // 1
console.log(contar()); // 2
console.log(contar()); // 3
```

3. ARRAYS

Un **array** es un objeto especial que permite almacenar **múltiples valores en una sola variable**, organizados por índices numéricos (empezando en 0).

```
let nombre_Array = [Elem1, Elem2,..., Elemn];
```

- CREAR ARRAYS

```
let numeros = [1, 2, 3, 4, 5];
let mezcla = ["texto", 42, true, null, {nombre: "Ana"}];
let vacio = [];
```

- ACCEDER A ELEMENTOS

```
let frutas = ["manzana", "plátano", "naranja"];

console.log(frutas[0]); // "manzana"
console.log(frutas[1]); // "plátano"
console.log(frutas[frutas.length - 1]); // "naranja" ← último element
```

- MODIFICAR ELEMENTOS

```
frutas[1] = "kiwi";
console.log(frutas); // ["manzana", "kiwi", "naranja"]
```

- AÑADIR ELEMENTOS

```
frutas.push("uva"); // AL FINAL
frutas.unshift("fresa"); // AL PRINCIPIO
```

- ELEMINAR ELEMENTOS

```
let ultima = frutas.pop(); // DEL FINAL
let primera = frutas.shift(); // DEL PRINCIPIO
```

- METODOS DE BUSQUEDA

```
let animales = ["perro", "gato", "perro", "loro"];

console.log(animales.indexOf("perro"));  // 0
console.log(animales.lastIndexOf("perro")); // 2
console.log(animales.includes("gato")); // true
```

- RECORRER UN ARRAY

```
let numeros = [5, 12, 8, 130, 44];

numeros.forEach((num, index) => {
  console.log(` Índice ${index}: ${num}` );
});
```

- ORDENAR UN ARRAY

```
let letras = ["c", "a", "b"];
letras.sort(); // ["a", "b", "c"]

let nums = [10, 2, 30];
nums.sort(); // [10, 2, 30] ← ¡Orden lexicográfico!

// Para ordenar números correctamente:
nums.sort((a, b) => a - b); // [2, 10, 30]
```

EJEMPLO PRÁCTICO : CARRITO DE LA COMPRA

```
let carrito = [
  { producto: "Camiseta", precio: 20 },
  { producto: "Pantalón", precio: 40 },
  { producto: "Zapatos", precio: 60 }
];

// Total a pagar
let total = carrito.reduce((suma, item) => suma + item.precio, 0);
console.log(`Total: $${total}`); // Total: $120

// Productos caros (>30)
let caros = carrito.filter(item => item.precio > 30);
console.log(caros); // [{producto: "Pantalón", ...}, {producto: "Zapatos", ...}]

// Nombres de productos
let nombres = carrito.map(item => item.producto);
console.log(nombres); // ["Camiseta", "Pantalón", "Zapatos"]
```

4. OBJETOS

Un **objeto** es una colección de **propiedades**, donde cada propiedad es un par **clave-valor**.

```
let persona = {
  nombre: "Ana",
  edad: 25,
  ciudad: "Madrid",

 saludar: function() {
     console.log(` Hola, soy ${this.nombre}` );
  }
};
```

- NOTACIÓN DE PUNTO

```
console.log(persona.nombre); // "Ana"
persona.saludar(); // "Hola, soy Ana"
```

- NOTACIÓN DE CORCHETES

```
console.log(persona["edad"]); // 25

let clave = "ciudad";
console.log(persona[clave]); // "Madrid"
```

- MODIFICAR Y AGREGAR PROPIEDADES

```
persona.edad = 26; // modificar
persona.profesion = "Desarrolladora"; // agregar nueva propiedad

console.log(persona);
// { nombre: "Ana", edad: 26, ciudad: "Madrid", profesion: "Desarrolladora",
saludar: f() }
```

- DEVUELVE UN ARRAY CON LAS CLAVES

```
console.log(Object.keys(persona));
// ["nombre", "edad", "profesion", "saludar"]
```

- DEVUELVE UN ARRAY CON LOS VALORES

```
console.log(Object.values(persona));
// ["Ana", 26, "Desarrolladora", ƒ]
```

- THIS EN OBJETOS

```
let coche = {
  marca: "Toyota",
  mostrarMarca: function() {
    console.log(this.marca); // "Toyota"
  }
};

coche.mostrarMarca();
```

EJEMPLO PRÁCTICO: BIBLIOTECA DE LIBROS

```
let biblioteca = {
  libros: [
    { titulo: "Cien años de soledad", autor: "García Márquez", prestado: false },
    { titulo: "1984", autor: "George Orwell", prestado: true }
  ],

  agregarLibro(titulo, autor) {
    this.libros.push({ titulo, autor, prestado: false });
  },

  listarLibros() {
    this.libros.forEach(libro => {
      console.log(` ${libro.titulo} - ${libro.autor} (${libro.prestado ? 'Prestado' : 'Disponible'})` );
    });
  },

  prestarLibro(titulo) {
    let libro = this.libros.find(l => l.titulo === titulo);
    if (libro && !libro.prestado) {
      libro.prestado = true;
      console.log(` Has prestado: ${titulo}` );
    } else {
      console.log(` No se puede prestar: ${titulo}` );
    }
  }
};

biblioteca.agregarLibro("El principito", "Saint-Exupéry");
biblioteca.listarLibros();
biblioteca.prestarLibro("El principito");
```

5. DESESTRUCTURACIÓN Y SPREAD

5.1. DESESTRUCTURACIÓN

Permite **extraer valores de arrays u objetos** y asignarlos a variables de forma más directa.

- **Desestructuración de OBJETOS** Extraes propiedades de un objeto y las asignas a variables con el **mismo nombre**.

```
const usuario = {
  nombre: "Ana",
  edad: 25,
  ciudad: "Madrid"
};

// Antes:
// const nombre = usuario.nombre;
// const edad = usuario.edad;

// Ahora (destructuring):
const { nombre, edad, ciudad } = usuario;

console.log(nombre); // "Ana"
console.log(edad);   // 25
```

- **Desestructuración de ARRAYS** Extraes elementos de un array según su **posición**.

```
const [a, , c] = colores; // ← saltamos el segundo
console.log(a); // "rojo"
console.log(c); // "azul"
```

5.2. SPREAD

- **Spread en ARRAYS**

```
const numeros = [1, 2, 3];
const copia = [...numeros]; // [1, 2, 3]

// ¡Importante! Es una copia, no la misma referencia
console.log(numeros === copia); // false
```

- **Spread en OBJETOS**

```
const persona = { nombre: "Luis", edad: 30 };
const copia = { ...persona };

console.log(copia); // { nombre: "Luis", edad: 30 }
console.log(persona === copia); // false ← ¡son objetos distintos!
```

EJEMPLOS PRÁCTICOS COMBINADOS

```
const usuario = {
  id: 1,
  nombre: "Carlos",
  perfil: {
    rol: "usuario",
    activo: true
  }
};

// Queremos actualizar solo el rol, sin mutar el original
const actualizado = {
  ...usuario,
  perfil: {
    ...usuario.perfil,
    rol: "admin"
  }
};

console.log(actualizado.perfil.rol); // "admin"
console.log(usuario.perfil.rol);    // "usuario" ← original intacto
```