Sintaxis y Conceptos Útiles en JavaScript (Ejemplos Genéricos)

Esta sección está diseñada para recordarte o presentarte la sintaxis básica y los métodos comunes de JavaScript que podrías necesitar para resolver los ejercicios. Los ejemplos son independientes y no están relacionados directamente con las consignas de los ejercicios.

---

1. Declaración y Uso de Funciones

Las funciones son bloques de código reutilizables.

- **Sintaxis básica de declaración:**

JavaScript

```
function nombreDeLaFuncion(parametro1, parametro2) {
  // Código a ejecutar
  return valorDeRetorno; // Opcional: devuelve un valor
}
```

- **Invocar/Llamar a una función:**

JavaScript

```
let resultado = nombreDeLaFuncion(argumento1, argumento2);
// console.log(resultado); // Para ver el valor retornado
```

- **Ejemplo Genérico:**

JavaScript

```
function duplicar(numero) {
  return numero * 2;
}

let miNumero = 5;
let dobleDeMiNumero = duplicar(miNumero); // dobleDeMiNumero es 10
// console.log(dobleDeMiNumero);
```

---

2. Operadores Aritméticos

Para realizar cálculos matemáticos.

- **Suma:** `+`
- **Resta:** `-`

- **Multiplicación:** `*`
- **División:** `/`
- **Módulo (resto de la división):** `%`
- **Incremento:** `++` (agrega 1)
- **Decremento:** `--` (resta 1)
- **Ejemplo Genérico:**

JavaScript

```
let a = 10;
let b = 3;

let suma = a + b;      // 13
let resto = a % b;     // 1 (10 dividido 3 es 3 con resto 1)
a++;                   // a ahora es 11
```

---

3. Operadores de Comparación

Para comparar valores, retornan `true` o `false`.

- **Igual a (valor):** `==`
- **Estrictamente igual a (valor y tipo):** `===` (preferido)
- **Diferente de (valor):** `!=`
- **Estrictamente diferente de (valor y tipo):** `!==`
- **Mayor que:** `>`
- **Menor que:** `<`
- **Mayor o igual que:** `>=`
- **Menor o igual que:** `<=`
- **Ejemplo Genérico:**

JavaScript

```
let x = 10;
let y = "10";
let z = 20;

console.log(x == y);   // true (compara solo valor)
console.log(x === y);  // false (compara valor y tipo)
console.log(x < z);    // true
console.log(x !== z);  // true
```

---

4. Operadores Lógicos

Para combinar o invertir condiciones booleanas.

- **AND lógico:** `&&` (ambas condiciones deben ser `true`)
- **OR lógico:** `||` (al menos una condición debe ser `true`)
- **NOT lógico:** `!` (invierte el valor booleano)
- **Ejemplo Genérico:**

JavaScript

```
let esActivo = true;
let tienePermiso = false;
let estaLogueado = true;

console.log(esActivo && estaLogueado);  // true
console.log(tienePermiso || estaLogueado); // true
console.log(!tienePermiso);             // true
```

---

5. Condicionales (`if`, `else if`, `else`)

Para ejecutar código basado en condiciones.

- **Sintaxis:**

JavaScript

```
if (condicion1) {
  // Código si condicion1 es true
} else if (condicion2) {
  // Código si condicion1 es false Y condicion2 es true
} else {
  // Código si todas las condiciones anteriores son false
}
```

- **Ejemplo Genérico:**

JavaScript

```
let temperatura = 25;

if (temperatura > 30) {
  console.log("Hace mucho calor.");
} else if (temperatura > 20) {
  console.log("Temperatura agradable.");
} else {
```

```
  console.log("Hace frío.");
}
```

---

6. Bucles (`for` , `while` , `forEach`)

Para repetir bloques de código.

- `for` **loop (para un número conocido de iteraciones):**

JavaScript

```
for (let i = 0; i < limite; i++) {
  // Código a repetir
}
```

  - **Ejemplo Genérico:**

JavaScript

```
for (let i = 1; i <= 5; i++) {
  console.log("Número: " + i); // Imprimirá 1, 2, 3, 4, 5
}
```

- `while` **loop (mientras una condición sea verdadera):**

JavaScript

```
while (condicion) {
  // Código a repetir
  // ¡Asegúrate de que la condición eventualmente se vuelva falsa!
}
```

  - **Ejemplo Genérico:**

JavaScript

```
let contador = 0;
while (contador < 3) {
  console.log("Contador: " + contador);
  contador++;
}
// Imprimirá: Contador: 0, Contador: 1, Contador: 2
```

- `forEach` **(para iterar sobre arrays):**

JavaScript

```
miArray.forEach(function(elemento, indice, arrayCompleto) {
  // Código a ejecutar para cada elemento
  // 'elemento' es el valor del elemento actual
  // 'indice' es la posición del elemento (opcional)
  // 'arrayCompleto' es el array sobre el que se itera (opcional)
});
```

- **Ejemplo Genérico:**

JavaScript

```
let frutas = ["manzana", "banana", "cereza"];
frutas.forEach(function(fruta) {
  console.log("Me gusta la " + fruta);
});
// Imprimirá: Me gusta la manzana, Me gusta la banana, Me gusta la cereza
```

---

7. Manipulación Básica de Strings

Los strings tienen métodos útiles.

- `.length` **(propiedad, no método):** Obtener la longitud del string.

JavaScript

```
let texto = "Hola";
console.log(texto.length); // 4
```

- `.toLowerCase()` **:** Convertir a minúsculas.
- `.toUpperCase()` **:** Convertir a mayúsculas.

JavaScript

```
let nombre = "MiNombre";
console.log(nombre.toLowerCase()); // "minombre"
console.log(nombre.toUpperCase()); // "MINOMBRE"
```

- `.includes(substring)` **:** Verificar si un string contiene otro substring.

JavaScript

```
let frase = "El perro ladra.";
console.log(frase.includes("perro")); // true
console.log(frase.includes("gato"));  // false
```

- `+` **(concatenación):** Unir strings.

JavaScript

```
let saludo = "Hola";
let nombre = "Mundo";
let mensaje = saludo + " " + nombre + "!"; // "Hola Mundo!"
```

---

8. Manipulación Básica de Arrays

Los arrays son listas ordenadas de elementos.

- **Declaración:** `[]`
- **Acceso a elementos (por índice):** `miArray[indice]` (el primer elemento está en el índice 0).

JavaScript

```
let listaNumeros = [10, 20, 30, 40];
console.log(listaNumeros[0]); // 10
console.log(listaNumeros[listaNumeros.length - 1]); // Accede al último elemen
```

- `.length` **(propiedad):** Obtener la cantidad de elementos.

JavaScript

```
let miLista = ["uno", "dos"];
console.log(miLista.length); // 2
```

- `.push(elemento)` : Agrega un elemento al final.
- `.pop()` : Elimina y retorna el último elemento.

JavaScript

```
let frutas = ["manzana", "banana"];
frutas.push("naranja"); // frutas es ["manzana", "banana", "naranja"]
```

```
let ultimaFruta = frutas.pop(); // ultimaFruta es "naranja", frutas es ["manza
```

- `.unshift(elemento)` : Agrega un elemento al principio.
- `.shift()` : Elimina y retorna el primer elemento.

JavaScript

```
let colores = ["rojo", "verde"];
colores.unshift("azul"); // colores es ["azul", "rojo", "verde"]
let primerColor = colores.shift(); // primerColor es "azul", colores es ["rojo
```

- `.indexOf(elemento)` : Retorna el índice de la primera aparición del elemento, o `-1` si no se encuentra.

JavaScript

```
let numeros = [1, 5, 2, 5];
console.log(numeros.indexOf(5));  // 1
console.log(numeros.indexOf(9));  // -1
```

- `.includes(elemento)` : Retorna `true` si el array contiene el elemento, `false` en caso contrario.

JavaScript

```
let animales = ["gato", "perro"];
console.log(animales.includes("gato"));  // true
console.log(animales.includes("pez"));   // false
```

---

9. Manipulación Básica de Objetos Literales

Los objetos almacenan datos en pares `clave: valor` .

- **Declaración:** `{}`
- **Acceso a propiedades (notación de punto o corchetes):**

JavaScript

```
let persona = {
  nombre: "Carlos",
  edad: 40,
  ciudad: "México"
```

```
};

// Notación de punto (cuando la clave es fija y válida como identificador)
console.log(persona.nombre); // "Carlos"

// Notación de corchetes (cuando la clave es dinámica, o incluye espacios/cara
let propiedadDinamica = "edad";
console.log(persona[propiedadDinamica]); // 40
```

- **Modificar/Agregar propiedades:**

JavaScript

```
let coche = { marca: "Toyota" };
coche.modelo = "Corolla"; // Agrega o modifica
coche["año"] = 2022;     // Agrega o modifica
console.log(coche); // { marca: "Toyota", modelo: "Corolla", año: 2022 }
```

- **Eliminar propiedades:** `delete objeto.propiedad`

JavaScript

```
let articulo = { id: 1, nombre: "Libro", precio: 25 };
delete articulo.precio;
console.log(articulo); // { id: 1, nombre: "Libro" }
```

- `Object.keys(objeto)` **:** Retorna un array con las claves (nombres de propiedades) del objeto.

JavaScript

```
let miObjeto = { a: 1, b: 2 };
console.log(Object.keys(miObjeto)); // ["a", "b"]
```

- `Object.values(objeto)` **:** Retorna un array con los valores de las propiedades del objeto.

JavaScript

```
let miObjeto = { a: 1, b: 2 };
console.log(Object.values(miObjeto)); // [1, 2]
```

- `Object.entries(objeto)` **:** Retorna un array de arrays, donde cada array interno es un par `[clave, valor]` .

JavaScript

```
let miObjeto = { a: 1, b: 2 };
console.log(Object.entries(miObjeto)); // [["a", 1], ["b", 2]]
```

---

10. Manejo de Errores Básicos (`return` de valores específicos)

A veces, una función debe indicar que algo salió mal.

- En lugar de arrojar un error (que es un tema más avanzado), puedes simplemente retornar un valor específico que indique un problema, como `null`, `undefined`, `0`, `-1`, o un string de error.
- **Ejemplo Genérico:**

JavaScript

```
function dividir(dividendo, divisor) {
  if (divisor === 0) {
    return "Error: División por cero.";
  }
  return dividendo / divisor;
}

console.log(dividir(10, 2)); // 5
console.log(dividir(10, 0)); // "Error: División por cero."
```

Espero que esta guía de sintaxis y ejemplos genéricos te sea de gran utilidad para enfrentar los ejercicios. ¡Mucho éxito en tu práctica!