Fase 1: Fundamentos de JavaScript (Detalle Extendido)

Esta fase se centra en construir una base sólida en JavaScript. No te preocupes si algunos conceptos te parecen nuevos o desafiantes al principio; la práctica constante es la clave.

1. Variables, Tipos de Datos y Operadores

- **Variables:**
  - Declaración con `var`, `let` y `const`. Entender las diferencias de scope (alcance) entre ellas (especialmente la diferencia entre `var` y `let` / `const`). Se recomienda usar `let` y `const` en lugar de `var`.
  - Convenciones de nombres (camelCase).
- **Tipos de Datos:**
  - Primitivos: `Number`, `String`, `Boolean`, `Null`, `Undefined`, `Symbol`, `BigInt`.
  - Objetos: Entender que los arrays y las funciones son también objetos en JavaScript.
- **Operadores:**
  - Aritméticos: `+`, `-`, `*`, `/`, `%` (módulo), `**` (exponenciación).
  - De asignación: `=`, `+=`, `-=`, `*=`, `/=`.
  - De comparación: `==` (igualdad no estricta), `===` (igualdad estricta), `!=` (desigualdad no estricta), `!==` (desigualdad estricta), `>`, `<`, `>=`, `<=`. **Priorizar el uso de `===` y `!==`.**
  - Lógicos: `&&` (AND), `||` (OR), `!` (NOT).
  - De cadena: Concatenación con `+` y template literals (backticks `` ` ``).
  - Operador ternario: `condicion ? valorSiVerdadero : valorSiFalso`.
- **Ejercicios:**
  - Declarar variables de diferentes tipos y realizar operaciones entre ellas.
  - Comparar valores usando los diferentes operadores de comparación.
  - Usar template literals para crear cadenas complejas.
  - Resolver ejercicios de lógica con operadores lógicos.

2. Estructuras de Control

- **Condicionales:**
  - `if`, `else if`, `else`.
  - `switch`.
- **Bucles:**
  - `for`: bucle tradicional.
  - `while`: bucle condicional.
  - `do...while`: bucle condicional que se ejecuta al menos una vez.
  - `for...in`: iterar sobre las propiedades de un objeto.
  - `for...of`: iterar sobre valores iterables (arrays, strings, maps, sets, etc.). **Muy importante.**
- **Ejercicios:**
  - Crear un programa que determine si un número es par o impar.
  - Imprimir la tabla de multiplicar de un número.
  - Recorrer un array e imprimir sus elementos.
  - Implementar un juego sencillo como "adivina el número".

3. Funciones

- **Declaraciones de función:** `function nombreFuncion(parametros) { ... }`.
- **Expresiones de función:** `const nombreFuncion = function(parametros) { ... };`.
- **Funciones flecha (arrow functions):** `const nombreFuncion = (parametros) => { ... };`. Entender las diferencias en el manejo de `this`. **Muy importante.**
- **Parámetros y argumentos.**
- **Retorno de valores con `return`.**
- **Scope de las funciones (ámbito local y global).**
- **Closures (clausuras):** Un concepto avanzado pero importante.
- **Ejercicios:**
  - Crear funciones para realizar operaciones matemáticas.
  - Crear funciones que manipulen arrays.
  - Crear funciones que retornen otras funciones (funciones de orden superior).

4. Objetos y Arrays

- **Objetos:**
  - Creación de objetos literales: `const objeto = { propiedad1: valor1, propiedad2: valor2 };`.
  - Acceso a propiedades: `objeto.propiedad1`, `objeto['propiedad1']`.
  - Añadir, modificar y eliminar propiedades.
  - Métodos (funciones dentro de objetos).

- `this` en objetos.
- Destructuring de objetos.
- Spread operator (`...`) para objetos.

- **Arrays:**
  - Creación de arrays: `const array = [elemento1, elemento2, elemento3];` .
  - Acceso a elementos: `array[indice]` .
  - Métodos de arrays: `push` , `pop` , `shift` , `unshift` , `splice` , `slice` , `concat` , `join` , `indexOf` , `includes` , `forEach` , `map` , `filter` , `reduce` , **crucial.**
  - Iteración de arrays con `for` , `for...of` y métodos de arrays.
  - Destructuring de arrays.
  - Spread operator (`...`) para arrays.
- **Ejercicios:**
  - Crear un objeto que represente a una persona con sus propiedades y métodos.
  - Manipular arrays usando los diferentes métodos.
  - Combinar objetos y arrays para crear estructuras de datos más complejas.

5. Programación Orientada a Objetos (POO) Básica

- **Prototipos:** Entender cómo funciona la herencia prototípica en JavaScript.
- **Clases (ES6):** Sintaxis para crear objetos con propiedades y métodos.
- **Constructor:** Función especial para inicializar objetos.
- **`this` en clases.**
- **Herencia con `extends` .**
- **Métodos estáticos.**
- **Ejercicios:**
  - Crear una clase para representar un vehículo con sus propiedades y métodos.
  - Crear clases que hereden de otras clases.

6. Manejo del DOM Básico (Opcional pero Recomendado)

- Aunque te enfocarás en backend, entender cómo funciona el DOM te ayudará a comprender mejor cómo interactúa el frontend con el backend.
- Selección de elementos del DOM.
- Manipulación de elementos del DOM (cambiar contenido, estilos, atributos).
- Eventos del DOM.
- **Ejercicios:**
  - Crear una página web simple con JavaScript que interactúe con el DOM.

7. Asincronía en JavaScript (CRUCIAL para Node.js)

- **Callbacks:** Funciones que se ejecutan después de que una operación asíncrona se completa. Entender sus limitaciones (callback hell).
- **Promesas:** Objetos que representan el resultado eventual de una operación asíncrona. Métodos `then` , `catch` y `finally` .
- **`async` y `await` :** Sintaxis más moderna para trabajar con promesas, que hace el código asíncrono más legible y fácil de entender. **Priorizar el uso de `async/await` .**
- **`fetch` API:** Para realizar peticiones HTTP desde el navegador (útil para practicar con APIs públicas).
- **Ejercicios:**
  - Simular operaciones asíncronas con `setTimeout` .
  - Realizar peticiones a APIs públicas con `fetch` y manejar la respuesta con promesas y `async/await` .
  - Manejar errores en operaciones asíncronas.