Apuntes de JavaScript

1.Introducción y contexto

- objetivos

- importancia

- preguntas fundamentales

2.Introducción a node.js

3.Tipos de datos en javascript

4.Variables en javascript

- características

- declaración

- Ámbito (Scope)

5.Operadores y expresiones

- Aritméticos

- Asignación

- Comparación

6.Funciones en javascript

- Declaración

- Uso // Invocación

- Funciones Anónimas

- Funciones Flecha

7. Estructuras de control

8. Eventos y DOM

1. Introducción y Contexto

1.1. Objetivos de esta Sección:

- Comprender la esencia de JavaScript y su ecosistema, incluyendo Node.js.
- Aprender a identificar y utilizar los tipos de datos primitivos y objetos en JavaScript.
- Comprender la funcionalidad y aplicación de operadores, expresiones y funciones en JavaScript.

1.2. La Importancia de JavaScript y Node.js en el Desarrollo Web:

- **JavaScript**: El Lenguaje Esencial para la Web Interactiva.
    - Es el lenguaje de programación nativo de los navegadores web.
    - Permite añadir interactividad dinámica a las páginas HTML, respondiendo a las acciones del usuario.
    - Es fundamental para la manipulación del Document Object Model (DOM), la gestión de eventos y la comunicación asíncrona (como AJAX).
- **Node.js**: JavaScript en el Servidor y Más Allá.
    - Es un entorno de ejecución de JavaScript construido sobre el motor V8 de Google Chrome.
    - Permite ejecutar código JavaScript fuera del navegador,[1] principalmente en el lado del servidor.
    - Facilita la creación de servidores web robustos, APIs (Application Programming Interfaces) y aplicaciones en tiempo real.
    - Ha expandido el uso de JavaScript al desarrollo full-stack, permitiendo a los desarrolladores utilizar un único lenguaje tanto en el frontend como en el backend.

1.3. Preguntas Fundamentales sobre JavaScript:

- ¿Qué es JavaScript?

    - Un lenguaje de programación de alto nivel: Su sintaxis es relativamente cercana al lenguaje humano, facilitando su aprendizaje y uso.
    - Interpretado: El código fuente se ejecuta línea por línea por un intérprete (el motor JavaScript del navegador o Node.js), sin necesidad de una compilación previa a código máquina. Esto permite un desarrollo más rápido y flexible.

Momento Cerebrito: ¿Qué significa interpretado?
Un lenguaje interpretado traduce y ejecuta el código fuente directamente en tiempo de ejecución. El motor (como V8 en Chrome y Node.js) lee cada instrucción, la interpreta y la ejecuta al instante. Esto contrasta con los lenguajes compilados (como C++ o Java), que primero se traducen a un código binario ejecutable (.exe, .class) antes de ser ejecutados por el sistema operativo o una máquina virtual.

    - Dinámico: La tipificación de las variables se realiza en tiempo de ejecución. Una misma variable puede contener diferentes tipos de datos a lo largo de la ejecución del programa. El lenguaje se adapta al tipo de dato que se le asigna en cada momento.

¿Qué significa dinámico? En JavaScript, no necesitas declarar explícitamente el tipo de dato de una variable al crearla. El lenguaje infiere el tipo según el valor que se le asigna. Además, el tipo de una variable puede cambiar durante la ejecución del programa.

    - Usado principalmente en desarrollo web, pero su versatilidad lo ha llevado a otros entornos.
    - Es el lenguaje nativo de la web, ya que todos los navegadores modernos tienen un motor JavaScript integrado.
- **¿Cuál es la función de JavaScript en el desarrollo web?**
    - Frontend (Lado del Cliente):
        - Añade interactividad a las páginas web (animaciones, respuestas a clics, validación de formularios, etc.).
        - Manipulación del DOM (Document Object Model): Permite modificar la estructura, el contenido y el estilo de las páginas HTML dinámicamente.
        - Manejo de eventos: Permite responder a las acciones del usuario (clics, movimientos del ratón, pulsaciones de teclas).
        - Comunicación asíncrona (AJAX / Fetch API): Permite enviar y recibir datos del servidor en segundo plano sin recargar la página.
    - Backend (Lado del Servidor) con Node.js:
        - Creación de servidores web: Permite construir aplicaciones web completas con capacidad de respuesta.

        - Desarrollo de APIs (Application Programming Interfaces): Facilita la comunicación entre diferentes aplicaciones y servicios.
        - Aplicaciones en tiempo real: Permite construir aplicaciones que requieren una comunicación bidireccional instantánea (chats, juegos online).
- **¿Solo se usa para frontend?**
    - No. Gracias a Node.js, JavaScript se utiliza ampliamente en el backend, permitiendo el desarrollo full-stack con un único lenguaje. Esto simplifica el flujo de trabajo y facilita la colaboración entre los equipos de frontend y backend.
- **¿Cuáles son aplicaciones comunes de JavaScript en el desarrollo web (y más allá)?**
    - Desarrollo de interfaces de usuario (UI) dinámicas y reactivas (con frameworks como React, Angular, Vue.js).
    - Desarrollo de APIs RESTful y GraphQL para la comunicación entre el frontend y el backend.
    - Desarrollo de aplicaciones en tiempo real (chats, notificaciones push, juegos multijugador).
    - Automatización de tareas (con Node.js para scripts del lado del servidor y herramientas de automatización del frontend).
    - Desarrollo de aplicaciones móviles (con frameworks como React Native).
    - Desarrollo de aplicaciones de escritorio (con frameworks como Electron).
    - Internet de las Cosas (IoT) y otros entornos.

2. Introducción a Node.js

- **Node.js**: El Poder de JavaScript Fuera del Navegador.
    - Es un entorno de ejecución que permite ejecutar código JavaScript en el servidor y en otros entornos fuera del navegador web.
    - Construido sobre el motor V8 de Google Chrome, el mismo motor de alto rendimiento que impulsa el navegador Chrome.
    - Utiliza una arquitectura orientada a eventos y no bloqueante (I/O no bloqueante), lo que lo hace altamente eficiente para manejar múltiples conexiones simultáneamente, ideal para aplicaciones en tiempo real y con alta concurrencia.

```
JavaScript
```

```
// Primer programa en Node.js: "Hello World"
console.log('Hello, World!');
```

- Este simple código, ejecutado con Node.js en la terminal (`node nombre_del_archivo.js`), mostrará "Hello, World!" en la consola del servidor.

3. Tipos de Datos en JavaScript

JavaScript maneja diferentes tipos de datos para representar información. Se dividen principalmente en dos categorías:

- **Primitivos**: Representan valores individuales y son inmutables (su valor no puede cambiar directamente).
    - `string`: Cadenas de texto, secuencias de caracteres (ej: `'Hola'`, `"Mundo"`).
    - `number`: Números, incluyendo enteros y números de punto flotante (ej: `25`, `3.14`).
    - `boolean`: Valores lógicos que representan verdadero o falso (`true`, `false`).
    - `null`: Un valor especial que representa la ausencia intencional de un valor.
    - `undefined`: Un valor que indica que una variable no ha sido asignada con un valor.
    - `symbol` (introducido en ES6): Un tipo de dato único e inmutable, a menudo usado como clave de propiedad de objeto para evitar colisiones.

- **Objetos**: Representan colecciones de datos más complejas.
    - `object`: Colecciones de pares clave-valor (ej: `{ nombre: 'Juan', edad: 25 }`).
    - `array`: Listas ordenadas de valores (ej: `[1, 2, 3, 4, 5]`). Los arrays en JavaScript pueden contener elementos de diferentes tipos.
    - `function`: Objetos especiales que representan bloques de código reutilizable.

```
JavaScript
```

```
// Primitivos
let nombre = 'Juan'; // string
let edad = 25; // number
let esEstudiante = true; // boolean
```

```
let direccion = null; // null
let telefono; // undefined
const idUnico = Symbol('miSimbolo'); // symbol

// Objetos
let persona = { nombre: 'Juan', edad: 25 }; // object
let numeros = [1, 2, 3, 4, 5]; // array
function saludar() { console.log('Hola'); } // function
```

4. Variables en JavaScript

En programación, una **variable** es un **nombre simbólico** que se utiliza para **almacenar y referenciar un valor** en la memoria del ordenador. Imagina una variable como una **etiqueta** que pegas en una caja. La etiqueta es el nombre de la variable, y lo que hay dentro de la caja es el valor que almacena.

Características Clave de las Variables:

- **Nombre:** Cada variable tiene un nombre único (identificador) que se utiliza para acceder a su valor. Este nombre debe seguir ciertas reglas del lenguaje (por ejemplo, en JavaScript, no puede empezar con un número, no puede contener espacios, etc.).
- **Valor:** La variable almacena un dato específico, que puede ser de diferentes tipos (número, texto, booleano, objeto, etc.). Este valor puede cambiar a lo largo de la ejecución del programa (a menos que la variable se declare con `const`).
- **Tipo (en lenguajes dinámicos como JavaScript):** Aunque en JavaScript no se declara explícitamente el tipo de una variable al crearla, internamente el valor almacenado tiene un tipo. El tipo de una variable puede cambiar si se le asigna un valor de un tipo diferente.
- **Ubicación en memoria:** El nombre de la variable es una forma legible para los humanos de referirse a una ubicación específica en la memoria del ordenador donde se guarda el valor.

En resumen, una variable es un contenedor con un nombre que te permite guardar y manipular información dentro de tu programa.

Se declaran utilizando las palabras clave `var`, `let` y `const`.

- **Declaración de variables:**
    - `var`: Declara una variable con ámbito de función o global, dependiendo de dónde se declare. Tiene un comportamiento de "hoisting" (la declaración se eleva al principio del ámbito, pero la inicialización no).
    - `let` (introducido en ES6): Declara una variable con ámbito de bloque. También tiene hoisting, pero no se puede acceder a la variable antes de su declaración en el código. Permite la reasignación de valores, es una mejor opción para variables que pueden cambiar.

    - `const` (introducido en ES6): Declara una constante con ámbito de bloque. Al igual que `let`, tiene hoisting pero no permite la reasignación de valores después de su inicialización. Aunque el valor de un objeto o array declarado con `const` no se puede reasignar, sus propiedades o elementos sí pueden modificarse.
- **Ámbito de las variables (Scope)**: El ámbito define la accesibilidad de una variable en diferentes partes del código.
    - Ámbito global: Las variables declaradas fuera de cualquier función o bloque tienen un ámbito global y son accesibles desde cualquier parte del código.
    - Ámbito de función (para `var`): Las variables declaradas con `var` dentro de una función son accesibles solo dentro de esa función (y sus funciones anidadas).
    - Ámbito de bloque (para `let` y `const`): Las variables declaradas con `let` o `const` dentro de un bloque (`{ ... }`, como un `if`, `for`, etc.) son accesibles solo dentro de ese bloque.

JavaScript

```
// Declaración de variables
var nombre = 'Juan'; // variable global o de función
let edad = 25; // variable de bloque
const esEstudiante = true; // constante

// Scope
if (true) {
  var variableGlobal = 'Soy global';
  let variableLocal = 'Soy local';
  console.log(variableLocal); // 'Soy local'
}
console.log(variableGlobal); // 'Soy global'
// console.log(variableLocal); // Error: variableLocal is not defined
```

5. Operadores y Expresiones

Los operadores son símbolos especiales que realizan operaciones en uno o más operandos (valores). Las expresiones son combinaciones de operandos y operadores que se evalúan para producir un valor. Las expresiones son cualquier **fragmento de código** que se puede **evaluar** para producir un **valor**.

- Operadores aritméticos: Realizan operaciones matemáticas básicas.
    - `+` (Suma)

    - `-` (Resta)
    - `*` (Multiplicación)
    - `/` (División)
    - `%` (Módulo - devuelve el resto de una división)[2]
    - `**` (Exponenciación - introducido en ES7)
- Operadores de asignación: Asignan valores a las variables.
    - `=` (Asignación simple)
    - `+=` (Suma y asignación)
    - `-=` (Resta y asignación)
    - `*=` (Multiplicación y asignación)
    - `/=` (División y asignación)
    - `%=` (Módulo y asignación)[3]
    - `**=` (Exponenciación y asignación - introducido en ES7)
- Operadores de comparación: Comparan dos operandos y devuelven un valor booleano (`true` o `false`).
    - `==` (Igualdad - compara solo el valor, con conversión de tipo si es necesario)
    - `===` (Igualdad estricta - compara tanto el valor como el tipo)
    - `!=` (Desigualdad)
    - `!==` (Desigualdad estricta)
    - `>` (Mayor que)
    - `<` (Menor que)
    - `>=` (Mayor o igual que)
    - `<=` (Menor o igual que)
- Operadores lógicos: Realizan operaciones lógicas en valores booleanos.
    - `&&` (AND lógico - devuelve `true` si ambos operandos son `true`)
    - `||` (OR lógico - devuelve `true` si al menos uno de los operandos es `true`)
    - `!` (NOT lógico - invierte el valor booleano del operando)

JavaScript

```
// Operadores aritméticos
let suma = 5 + 3; // 8
let resta = 5 - 3; // 2
let multiplicacion = 5 * 3; // 15
let division = 5 / 3; // 1.666...
let modulo = 5 % 3; // 2
let potencia = 2 ** 3; // 8
```

```
// Operadores de comparación
console.log(5 == '5'); // true (comparación de valor)
console.log(5 === '5'); // false (comparación de valor y tipo)
console.log(5 != '5'); // false
console.log(5 !== '5'); // true
console.log(5 > 3); // true
console.log(5 < 3); // false
console.log(5 >= 5); // true
console.log(5 <= 3); // false

// Operadores lógicos
console.log(true && false); // false
console.log(true || false); // true
console.log(!true); // false
console.log(!false); // true
```

6. Funciones en JavaScript

Las funciones son bloques de código reutilizables que realizan tareas específicas. Permiten organizar el código, hacerlo más legible y evitar la repetición.

El uso de funciones las funciones es definido como *"Invocación".* Ejecuta el código dentro de una función, se llama a la función por su nombre seguido de paréntesis (y los argumentos correspondientes si la función tiene parámetros).

. Declaración de Funciones (Function Declaration):

- Se define con la palabra clave `function`, seguida del nombre, los parámetros (opcional) entre paréntesis y el código a ejecutar entre llaves `{}`. Se pueden utilizar en cualquier parte del ámbito donde están definidas.

JavaScript

```
function mostrarMensaje() {
  console.log('¡Hola desde la función!');
}

function saludar(nombre) {
  console.log('¡Hola, ' + nombre + '!');
}

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

mostrarMensaje(); // Invoca la función mostrarMensaje
saludar('Ana');   // Invoca la función saludar con el argumento 'Ana'
let resultadoSuma = sumar(5, 3); // Invoca sumar con argumentos 5 y 3, guarda el resultado
console.log('La suma es:', resultadoSuma); // Muestra: La suma es: 8
```

. Expresión de Funciones (Function Expression):

JavaScript

```
const multiplicar = function(a, b) {
  return a * b;
};

const dividir = function dividirNumeros(a, b) {
  return b === 0 ? 'Error: No se puede dividir por cero' : a / b;
};

let resultadoMultiplicacion = multiplicar(4, 7); // Invoca la función a través de la variable
console.log('La multiplicación es:', resultadoMultiplicacion); // Muestra: La multiplicación es: 28
let resultadoDivision = dividir(10, 2); // Invoca la función a través de la variable
console.log('La división es:', resultadoDivision); // Muestra: La división es: 5
```

Explicación: Una función anónima (o con nombre opcional) se asigna a una variable (`const multiplicar`). Deben definirse antes de ser usadas en el código.

. Funciones Anónimas:

- Son funciones que no tienen un nombre. A menudo se asignan a variables o se utilizan como argumentos de otras funciones. Son útiles como *callbacks* (funciones que se pasan como argumento a otra función y se ejecutan después) o en IIFEs (Immediately Invoked Function Expressions) para crear un ámbito privado.

JavaScript

```
// Ejemplo como callback (típico en eventos)
// document.getElementById('miBoton').addEventListener('click', function() {
//   alert('¡Botón clickeado usando una función anónima!');
// });

// Función anónima auto-ejecutable (IIFE)
(function() {
  let mensajeIIFE = 'Este mensaje se muestra al instante';
  console.log(mensajeIIFE); // Muestra: Este mensaje se muestra al instante
})();
```

. Funciones Flecha (Arrow Functions):

- Proporcionan una sintaxis más concisa para escribir funciones anónimas. No tienen su propio `this` y tienen algunas otras diferencias de comportamiento en comparación con las funciones tradicionales. Posee sintaxis concisa para funciones anónimas. `(parámetros) => expresión` (retorno implícito si es una sola expresión) o `(parámetros) => { cuerpo }` (requiere return explícito para devolver un valor). Tienen un `this` léxico (heredan el this del contexto circundante).

```
JavaScript

const multiplicarFlecha = (a, b) => a * b;
const cuadrado = numero => numero ** 2;
const saludarFlecha = () => '¡Hola desde la función flecha!';
const dividirFlecha = (a, b) => {
  if (b === 0) {
    return 'Error: División por cero';
  }
  return a / b;
};

console.log('Multiplicación flecha:', multiplicarFlecha(3, 5)); // Muestra: Multiplicación flecha: 15
console.log('Cuadrado flecha:', cuadrado(4)); // Muestra: Cuadrado flecha: 16
console.log(saludarFlecha()); // Muestra: ¡Hola desde la función flecha!
console.log('División flecha:', dividirFlecha(10, 5)); // Muestra: División flecha: 2
```

. Parámetros y Argumentos:

```
JavaScript

function ejemploParametros(a, b = 10, ...restOfArgs) {
  console.log('a:', a, 'b (por defecto):', b, 'rest:', restOfArgs);
}

ejemploParametros(5);
ejemploParametros(2, 8);
ejemploParametros(1, undefined, 'extra1', 'extra2');
ejemploParametros(3, 7, 'uno', 'dos', 'tres');
```

Explicación: Los parámetros se definen en la función, los argumentos son los valores pasados al invocarla. Se pueden definir valores por defecto para los parámetros (`b = 10`). El parámetro rest (`...restOfArgs`) permite recibir un número indefinido de argumentos como un array.

7. Estructuras de Control: Permiten controlar el flujo de ejecución del código, permitiendo que diferentes bloques de código se ejecuten bajo ciertas condiciones o se repitan.

Condicionales: Ejecutan diferentes bloques de código basados en si una condición es verdadera o falsa.

- `if`: Especifica un bloque de código que se ejecutará si una condición es verdadera.
- `else if`: Especifica un bloque de código que se ejecutará si la condición `if` es falsa y una nueva condición es verdadera. Pueden haber múltiples `else if`.
- `else`: Especifica un bloque de código que se ejecutará si todas las condiciones `if` y `else if` son falsas.

```
let edad = 19;
if (edad > 18) {
  console.log('Es adulto');
} else {
  console.log('Es menor de edad');
}
```

Bucles (Loops): Permiten repetir un bloque de código varias veces.

- `for`: Se utiliza para iterar un bloque de código un número específico de veces. Requiere una inicialización, una condición y una expresión de incremento/decremento.
- `while`: Se utiliza para iterar un bloque de código mientras una condición especificada sea verdadera. La condición se evalúa antes de cada iteración.
- `do...while`: Similar a `while`, pero el bloque de código se ejecuta al menos una vez antes de que se evalúe la condición. La condición se evalúa después de cada iteración.

```
for (let i = 0; i < 10; i++) {
  console.log(i);
}
```

```
let contador = 0;
while (contador < 5) {
  console.log('Contador:', contador);
  contador++;
}

let j = 0;
do {
  console.log('J:', j);
  j++;
} while (j < 3);
```

Switch: La estructura `switch` proporciona una forma eficiente de ejecutar diferentes bloques de código basados en el valor de una expresión. Evalúa una expresión y compara su valor con múltiples casos (`case`). Cuando se encuentra una coincidencia, se ejecuta el bloque de código asociado a ese `case`.

Sintaxis:

```
JavaScript

switch (expresión) {
  case valor1:
    // Bloque de código a ejecutar si expresión === valor1
    break;
  case valor2:
    // Bloque de código a ejecutar si expresión === valor2
    break;
  // ... más casos
  default:
    // Bloque de código a ejecutar si la expresión no coincide con ningún caso anterior
}
```

Elementos Clave:

- `switch (expresión)`: La expresión que se evalúa. El resultado de esta expresión se compara con el valor de cada `case`.
- `case valorN:`: Define un valor específico que se compara con el resultado de la `expresión`. Se utiliza el operador de igualdad estricta (`===`) para la comparación.
- `// Bloque de código`: El conjunto de instrucciones que se ejecutan si la `expresión` coincide con el `valorN` del `case`.
- `break;`: La instrucción `break` es crucial para salir de la estructura `switch` una vez que se encuentra una coincidencia y se ejecuta su código. Si se omite `break`, la ejecución continuará con el siguiente `case` (comportamiento conocido como "fall-through").
- `default:` (opcional): Define un bloque de código que se ejecuta si la `expresión` no coincide con ninguno de los valores de los `case` anteriores. Si `default` está presente, generalmente se coloca al final de la estructura `switch`, aunque no

es obligatorio. No necesita una instrucción `break` después de su bloque de código.

Ejemplo Básico:

```JavaScript
let dia = 3;
let nombreDia;

switch (dia) {
  case 1:
    nombreDia = "Lunes";
    break;
  case 2:
    nombreDia = "Martes";
    break;
  case 3:
    nombreDia = "Miércoles";
    break;
  case 4:
    nombreDia = "Jueves";
    break;
  case 5:
    nombreDia = "Viernes";
    break;
  case 6:
    nombreDia = "Sábado";
    break;
  case 7:
    nombreDia = "Domingo";
    break;
  default:
    nombreDia = "Número de día inválido";
}

console.log(`Hoy es ${nombreDia}`); // Salida: Hoy es Miércoles
```

Comportamiento "Fall-through":

Si se omite la instrucción `break` en un `case`, la ejecución del código continuará con el siguiente `case`, incluso si la expresión no coincide con el valor de ese siguiente `case`. Esto puede ser útil en ciertas situaciones donde se desea ejecutar el mismo código para múltiples valores, pero a menudo es una fuente de errores si no se usa intencionalmente.

```JavaScript
let nivelAcceso = 2;
```

```
switch (nivelAcceso) {
  case 1:
    console.log("Acceso básico.");
    break;
  case 2:
    console.log("Acceso intermedio.");
    // ¡Sin break! La ejecución continúa al siguiente caso
  case 3:
    console.log("Acceso avanzado.");
    break;
  default:
    console.log("Nivel de acceso desconocido.");
}
// Salida:
// Acceso intermedio.
// Acceso avanzado.
```

Comparación Estricta (===):

Es importante recordar que la comparación entre la expresión del switch y los valores de los case se realiza utilizando el operador de igualdad estricta (===). Esto significa que tanto el valor como el tipo de dato deben ser iguales para que haya una coincidencia.

```
JavaScript

let valor = "2";

switch (valor) {
  case 2:
    console.log("Es el número 2");
    break;
  case "2":
    console.log("Es la cadena '2'");
    break;
  default:
    console.log("No coincide");
}
// Salida: Es la cadena '2'
```

En este ejemplo, aunque el valor de la variable valor parece ser el número 2, es una cadena, por lo que solo coincide con el case "2".

La estructura switch es una alternativa clara y a menudo más legible a múltiples sentencias if...else if...else cuando se compara una única expresión con una serie de valores específicos. Sin embargo, es crucial entender el comportamiento del break y la comparación estricta para utilizarla correctamente.

8. Eventos y DOM (Document Object Model):

- DOM: Es una representación estructurada de un documento HTML o XML como un árbol de objetos. JavaScript puede interactuar con el DOM para modificar la estructura, el contenido y el estilo de las páginas web.
- Eventos: Son acciones que ocurren en el navegador (ej: clic del ratón, carga de la página, envío de un formulario). JavaScript permite "escuchar" estos eventos y ejecutar código en respuesta a ellos.
- La manipulación del DOM y la gestión de eventos son fundamentales para crear interfaces de usuario dinámicas e interactivas en el frontend.

```
// Ejemplo básico de manipulación del DOM y manejo de eventos
// (Requiere un elemento HTML con el id 'miBoton')
// <button id="miBoton">Haz clic</button>

const miBoton = document.getElementById('miBoton');
if (miBoton) {
  miBoton.addEventListener('click', function() {
    alert('Botón clickeado');
  });
}
```