Alexander Ocoro

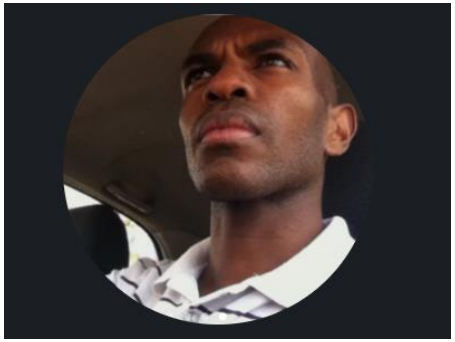
Mentor

TIC

TALENTO TECH VALLE

DESARROLLO *WEB*
FULL STACK
NIVEL BÁSICO

Warner Fdo. Valencia



Ejecutor

JavaScript Avanzado I

Objetivo: Aprender características avanzadas de **ES6 (ECMAScript 2015)** que mejoran la eficiencia, claridad y legibilidad del código JavaScript.

Arrow Functions

Objetivo: Comprender los conceptos básicos de JavaScript: variables, tipos de datos y operadores.

Arrow Functions

Teoría: ¿Qué son las Arrow Functions?

1. **Sintaxis**:
   - Las arrow functions son una forma más corta de escribir funciones en JavaScript.

```
const suma = (a, b) => a + b;
```

2. **Características principales**:
   - No tienen su propio `this`: Utilizan el `this` del contexto donde fueron definidas.
   - Más concisas, ideales para callbacks y funciones de una sola línea.
3. **Uso común en métodos de array**:
   - `map`: Transforma cada elemento de un array.
   - `filter`: Filtra elementos según una condición.
   - `reduce`: Reduce los valores de un array a un único resultado.

Ejemplo Práctico: Convertir Funciones Tradicionales en Funciones Flecha

1. Convierte esta función tradicional:

```
function cuadrado(num) {
  return num * num;
}
```

A una función flecha:

```
const cuadrado = (num) => num * num;
```

2. Usar `map` con funciones flecha:

```
const numeros = [1, 2, 3, 4];
const cuadrados = numeros.map((num) => num * num);
console.log(cuadrados); // [1, 4, 9, 16]
```

Actividad Práctica

Crear funciones flecha: Los estudiantes escribirán una función flecha que calcule el área de un círculo dada su radio, y usarán `filter` para obtener los números mayores a 5 de un array.

1. **Función flecha para calcular el área de un círculo:**

```
const calcularAreaCirculo = (radio) => Math.PI * Math.pow(radio, 2);
```

2. **Usar `filter` para obtener números mayores a 5 de un array:**

```
const filtrarMayoresA5 = (array) => array.filter(numero => numero > 5);
```

Ejemplo de uso:

```
// Calcular el área de un círculo
const radio = 7;
console.log(`El área del círculo con radio ${radio} es:
${calcularAreaCirculo(radio)}`);

// Filtrar números mayores a 5
const numeros = [1, 3, 5, 7, 9];
const mayoresA5 = filtrarMayoresA5(numeros);
console.log(`Números mayores a 5: ${mayoresA5}`);
```

Salida esperada:

```
El área del círculo con radio 7 es: 153.93804002589985
Números mayores a 5: [7, 9]
```

Template Strings

Objetivo: Aprender a usar template strings para crear cadenas dinámicas de manera más clara y concisa.

Teoría: ¿Qué son las Template Strings?

1. **Sintaxis**:
   - Usan comillas invertidas (`` ` ``) en lugar de comillas simples o dobles.
   - Permiten insertar variables y expresiones con `${}`.

```
const nombre = "Juan";
const saludo = `Hola, ${nombre}!`;
console.log(saludo); // Hola, Juan!
```

2. **Ventajas**:
   - Más fáciles de leer cuando se combinan variables y texto.
   - Ideales para construir cadenas largas, como HTML dinámico.

Ejemplo Práctico: Crear Mensajes con Template Strings

1. Crear un saludo personalizado:

```
const nombre = "María";
const edad = 30;
const mensaje = `Hola, ${nombre}. Tienes ${edad} años.`;
console.log(mensaje);
```

2. Generar HTML dinámico:

```
const producto = "Laptop";
const precio = 1500;
const html = `<div>
    <h2>Producto: ${producto}</h2>
    <p>Precio: $${precio}</p>
</div>`;
console.log(html);
```

Actividad Práctica

Crear mensajes de bienvenida personalizados usando template strings con variables dinámicas como nombre y ciudad.

```
// Solicitar datos del usuario
const nombre = "Juan";
const ciudad = "Cali";

// Crear mensaje personalizado
const mensaje = `¡Bienvenido, ${nombre}! Nos alegra que estés visitando nuestra
hermosa ciudad de ${ciudad}.`;

// Mostrar el mensaje en la consola
console.log(mensaje);
```

Destructuring

Objetivo: Aprender destructuración para acceder a valores de objetos y arrays de forma directa y concisa.

Teoría: ¿Qué es Destructuring?

1. **Destructuring de Objetos**:
   - Permite extraer propiedades de un objeto en variables individuales.

```
const persona = { nombre: "Luis", edad: 28 };
const { nombre, edad } = persona;
console.log(nombre, edad); // Luis 28
```

2. **Destructuring de Arrays**:
   - Extrae elementos de un array en variables.

```
const numeros = [10, 20, 30];
const [primero, segundo] = numeros;
console.log(primero, segundo); // 10 20
```

Ejemplo Práctico: Usar Destructuring

1. **Destructuración de un objeto**:

```
const producto = { nombre: "Tablet", precio: 300,
disponible: true };
const { nombre, precio } = producto;
console.log(`Producto: ${nombre}, Precio:
$${precio}`);
```

2. **Destructuración con arrays**:

```
const colores = ["rojo", "verde", "azul"];
const [primero, , tercero] = colores;
console.log(primero, tercero); // rojo azul
```

Actividad Práctica

Usar destructuring para extraer valores de un objeto `persona` y un array de números.

Spread y Rest

Objetivo: Aprender a usar los operadores **spread** y **rest** para combinar, copiar y manipular arrays y objetos.

Teoría: Spread y Rest

1. **Spread Operator (`...`)**:
   - Expande elementos de un array u objeto.

```
const numeros = [1, 2, 3];
const nuevosNumeros = [...numeros, 4, 5];
console.log(nuevosNumeros); // [1, 2, 3, 4, 5]
```

2. **Rest Parameter (`...`)**:
   - Agrupa múltiples valores en un único array o parámetro.

```
function sumar(...numeros) {
  return numeros.reduce((acc, num) => acc +
num, 0);
}
console.log(sumar(1, 2, 3, 4)); // 10
```

Ejemplo Práctico

1. Combinar objetos con spread:

```
const persona = { nombre: "Ana", edad: 25 };
const direccion = { ciudad: "Lima", pais: "Perú" };
const perfil = { ...persona, ...direccion };
console.log(perfil);
```

2. Usar rest en una función:

```
function mostrarNumeros(...numeros) {
  console.log("Números:", numeros);
}
mostrarNumeros(1, 2, 3, 4);
```

Actividad Práctica

Combinar arrays y objetos usando spread y crear funciones que reciban varios argumentos con rest.

```
// Combinar arrays con spread
const frutas = ["manzana", "plátano", "naranja"];
const verduras = ["zanahoria", "brócoli", "espinaca"];
const alimentos = [...frutas, ...verduras];
console.log("Alimentos combinados:", alimentos);

// Combinar objetos con spread
const persona = { nombre: "Carlos", edad: 28 };
const trabajo = { puesto: "Desarrollador", empresa:
"TechCorp" };
const perfil = { ...persona, ...trabajo };
console.log("Perfil combinado:", perfil);

// Usar rest en una función
function sumarNumeros(...numeros) {
    const suma = numeros.reduce((total, num) => total +
num, 0);
    console.log("Suma de los números:", suma);
}

// Probar la función con diferentes argumentos
sumarNumeros(1, 2, 3, 4, 5);
```

Proyecto de Interactividad

Objetivo: Usar lo aprendido para añadir un botón que, al hacer clic, cambie el estilo de un elemento en la página.

Práctica Guiada: Crear Interactividad con Eventos

1. **Paso a Paso**: Añadir un botón en el HTML que desencadene un cambio de estilo al hacer clic.

    1. **Paso 1**: En `index.html`, añade un botón y un elemento para cambiar su estilo.

```
<button id="cambiarEstilo">Cambiar
Estilo</button>
<div
  id="caja"
  style="width: 100px; height: 100px;
background-color: lightblue;"
></div>
```

2. **Paso 2**: En `script.js`, selecciona el botón y la caja, y añade un evento `click` para cambiar el estilo de la caja.

```
let boton =
document.getElementById("cambiarEstilo");
let caja = document.getElementById("caja");

boton.addEventListener("click", function () {
  caja.style.backgroundColor =
    caja.style.backgroundColor === "lightblue" ?
"salmon" : "lightblue";
});
```

Actividad Práctica

Cambio de Contenido Dinámico: Los estudiantes practicarán seleccionando y modificando el contenido de varios elementos en la página.