IT Academy

by KIBERNUM

Programa: Desarrollo de Aplicaciones Front-End

-

Módulo 2: Fundamentos de Desarrollo Front-End

Aprendizaje esperado

Utilizar código Javascript para la personalización de eventos sencillos dentro de un documento html dando solución al problema planteado.

Sentencias Condicionales

🔹 **Definición**

Las sentencias condicionales permiten que el código ejecute diferentes acciones basadas en condiciones específicas.

🔹 **Características**

✅ Evalúan expresiones booleanas (verdadero o falso) para decidir el flujo del programa.

✅ Permiten crear lógica en el programa, haciendo que el flujo de ejecución dependa de ciertas condiciones.

✅ if-else permite múltiples bifurcaciones.

✅ switch es útil para comparar un valor contra múltiples casos.

Sentencias Condicionales

Sentencia IF

- **Definición**

La sentencia if ejecuta un bloque de código solo si la condición especificada se cumple (es verdadera).

- **Sintaxis**

```
if (condición) {
    // Código a ejecutar si la condición es verdadera
}
```

- **Uso común**

Se utiliza para tomar decisiones simples dentro de un programa.

Sentencias Condicionales

Sentencia IF-ELSE

- **Definición**

La estructura if-else permite ejecutar un bloque de código si la condición es verdadera y otro bloque diferente si es falsa.

- **Sintaxis**

```
if (condición) {
    // Código si la condición es verdadera
} else {
    // Código si la condición es falsa
}
```

- **Uso común**

Se usa cuando hay dos caminos posibles en la ejecución del programa.

Sentencias Condicionales

Sentencia ELSE IF

- **Definición**

La sentencia else if permite evaluar múltiples condiciones de forma secuencial.

- **Sintaxis**

```
if (condición1) {
    // Código si condición1 es verdadera
} else if (condición2) {
    // Código si condición2 es verdadera
} else {
    // Código si ninguna condición es verdadera
}
```

- **Uso común**

Ideal cuando hay más de dos escenarios posibles.

Sentencias Condicionales

Sentencia SWITCH CASE

- **Definición**

La estructura switch evalúa una expresión y ejecuta el bloque de código correspondiente al caso que coincida con su valor.

- **Sintaxis**

```
switch (expresión) {
    case valor1:
        // Código si expresión == valor1
        break;
    case valor2:
        // Código si expresión == valor2
        break;
    default:
        // Código si no hay coincidencias
}
```

- **Uso común**

Se utiliza cuando una misma variable puede tomar varios valores posibles y se requiere ejecutar acciones diferentes para cada uno.

Ejemplo: Estructura if y else en JavaScript

¿Para qué sirve if y else?

✅ Se usa if para evaluar una condición.

✅ Si la condición es true, ejecuta un bloque de código.

✅ Si es false, se ejecuta el bloque dentro de else..

```
let edad = 18; // Puedes cambiar este valor para probar diferentes resultados
let mensaje; // Variable para almacenar el mensaje

// Sentencia condicional if-else
if (edad >= 18) {
    mensaje = "Eres mayor de edad"; // Mensaje si la condición es verdadera
} else {
    mensaje = "Eres menor de edad"; // Mensaje si la condición es falsa
}

alert(mensaje); // Muestra el mensaje en una ventana emergente
```

Ejemplo: Estructura Switch en JavaScript

¿Para qué sirve switch?

✅ Permite comparar un valor con múltiples casos posibles.

✅ Se usa case para definir cada opción y break para evitar que el código siga ejecutándose.

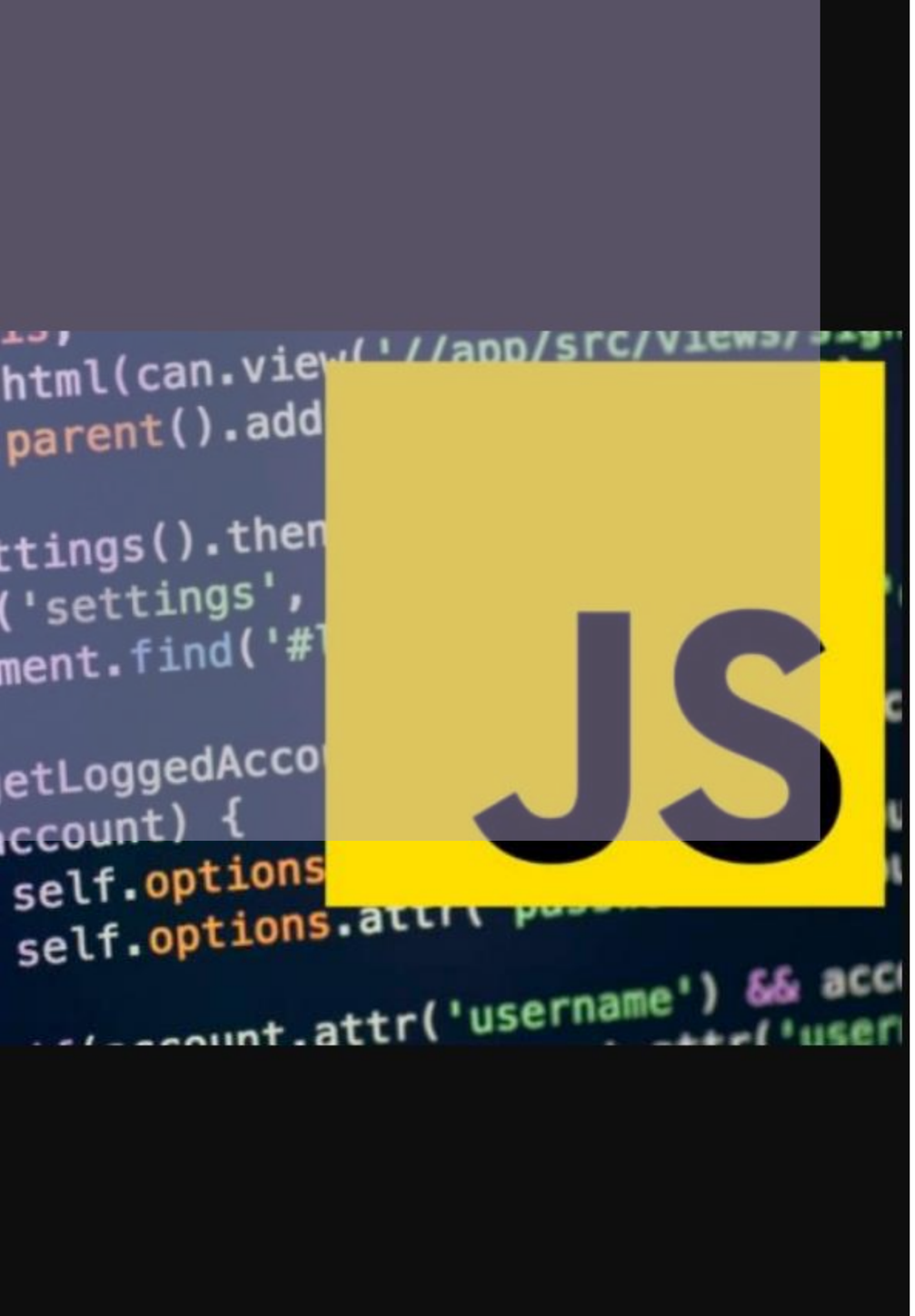
✅ Si ningún caso coincide, se ejecuta default:

```
let color = "rojo"; // Cambia este valor para probar otros colores
let mensajeColor; // Variable para almacenar el mensaje del color

// Sentencia switch
switch (color) {
    case "rojo":
        mensajeColor = "El color es rojo"; // Mensaje si el color es rojo
        break;
    case "azul":
        mensajeColor = "El color es azul"; // Mensaje si el color es azul
        break;
    default:
        mensajeColor = "Color no reconocido"; // Mensaje si el color no coincide
}

alert(mensajeColor); // Muestra el mensaje del color en una ventana emergente
```

¿Cómo podríamos evitar repetir código en un programa?



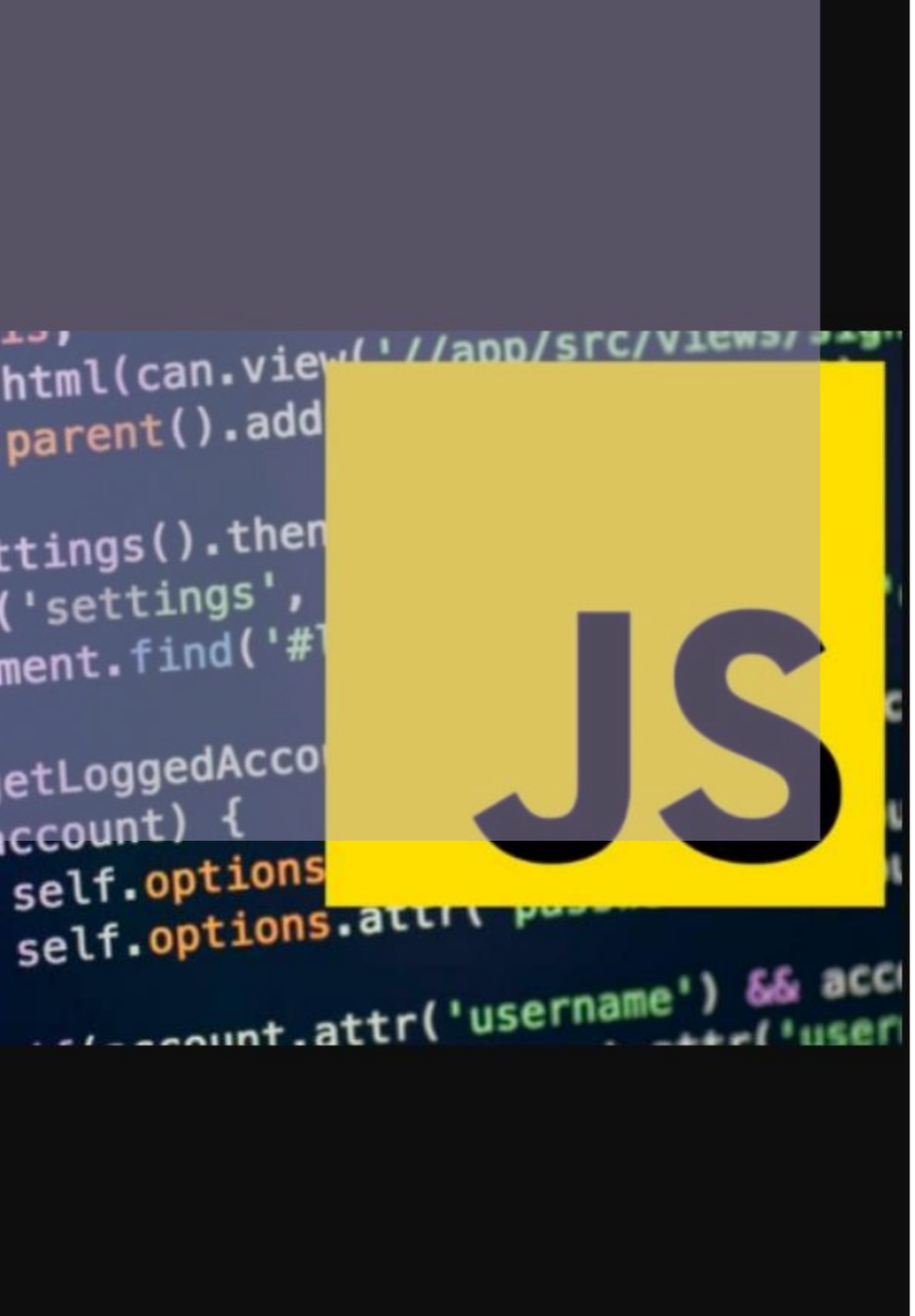
Definición de una función en JavaScript

- **Definición**

Una **función** es un bloque de código reutilizable que realiza una tarea específica.
Permite **dividir el programa en partes más pequeñas**, mejorando la **modularidad** y el **mantenimiento** del código.

- **Sintaxis básica**

```
function nombreFuncion() {
    // Código a ejecutar
}
```



Beneficios de usar funciones

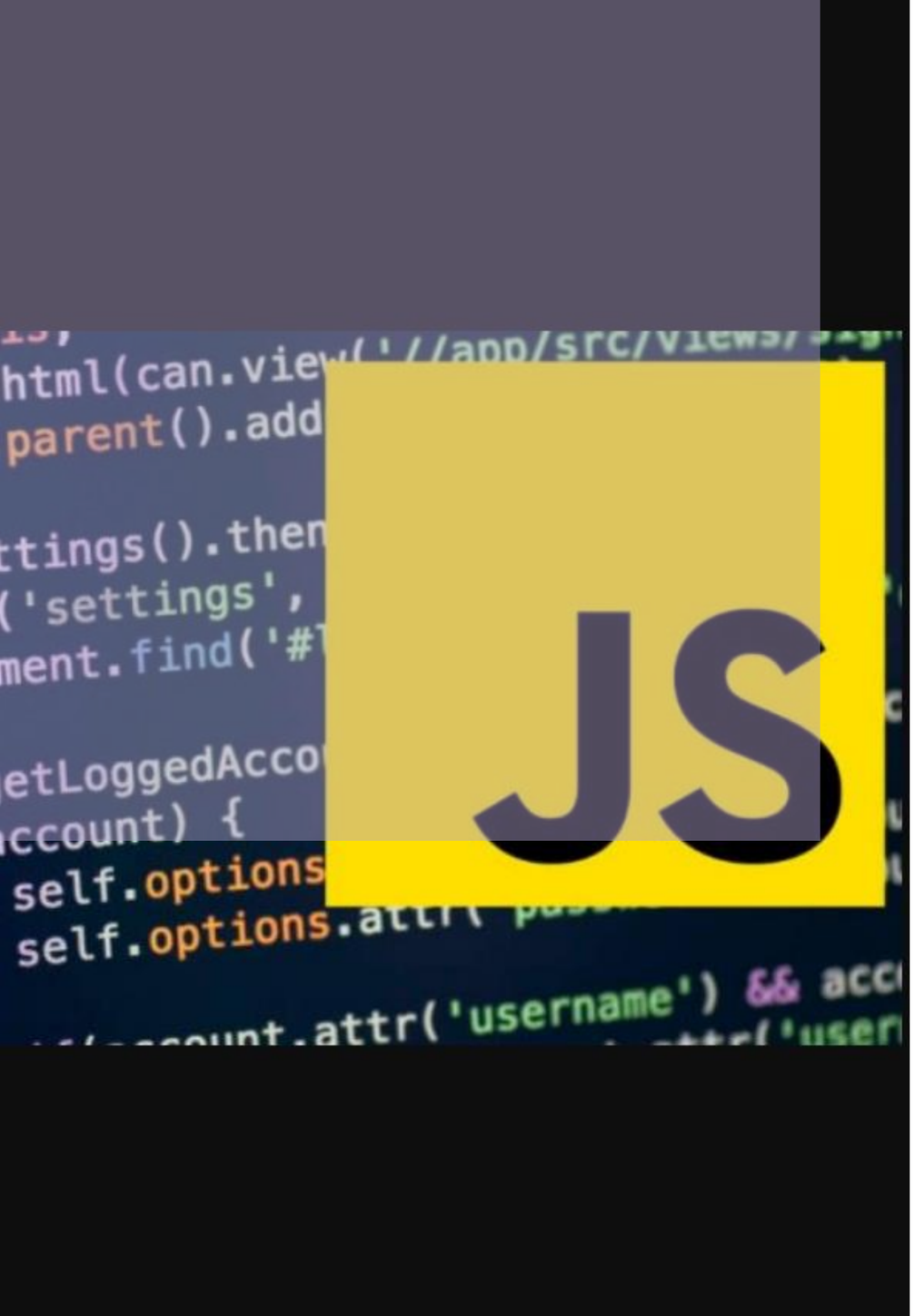
🔹 **Beneficios principales**

✅ **Reutilización del código:** permite ejecutar el mismo bloque en diferentes partes del programa.

✅ **Organización:** facilita la lectura y mantenimiento del código.

✅ **Modularidad:** divide el programa en secciones más manejables.

✅ **Facilidad de prueba:** cada función puede probarse de forma independiente.



Ejemplo de función en JS

- **Ejemplo simple**

```
function saludar() {
    console.log("¡Hola, bienvenido!");
}

saludar(); // Llamada a la función
```

- **Explicación**

•Se **declara** la función con la palabra clave function.
•Dentro de las llaves {} se escribe el **código que ejecutará**.
•Para usarla, se **llama** escribiendo su nombre seguido de ().

Actividad

🔹 "Imagina que tienes que escribir un programa que salude a tres personas. ¿Cómo organizarías tu código para no repetir líneas innecesarias?"

📢 Piensa en una solución y compártela con la clase.

Ejemplo: Funciones Tradicionales en JavaScript

¿Qué es una función?

✅ Un bloque de código reutilizable.

✅ Puede aceptar parámetros y devolver un resultado.

✅ Se usa return para devolver un valor.

```
function saludar(nombre) {
    return "Hola, " + nombre + "!";
}
console.log(saludar("Juan"));
```

¿Qué es una Arrow Function?

🔹 **Definición**

Una **Arrow Function** (función de flecha) es una forma más **concisa y moderna** de escribir funciones en JavaScript, introducida en **ES6 (ECMAScript 2015)**.

🔹 **Características principales**

✅ Utiliza la sintaxis => en lugar de la palabra clave function.
✅ Es ideal para funciones **cortas o de una sola línea**.
✅ Si solo tiene una expresión, **no requiere usar return ni llaves {}.**
✅ Hereda el valor de this del contexto donde se define (a diferencia de las funciones tradicionales).

🔹 **Uso común**

Se emplea con frecuencia en **funciones anónimas**, **callbacks** o **métodos de array** como map(), filter() y forEach().

Ejemplo de Arrow Function

```
// Forma tradicional
function saludar(nombre) {
    return "Hola, " + nombre;
}

// Arrow Function
const saludar = (nombre) => "Hola, " + nombre;

console.log(saludar("Ana")); // Salida: Hola, Ana
```

🔹 **Explicación**

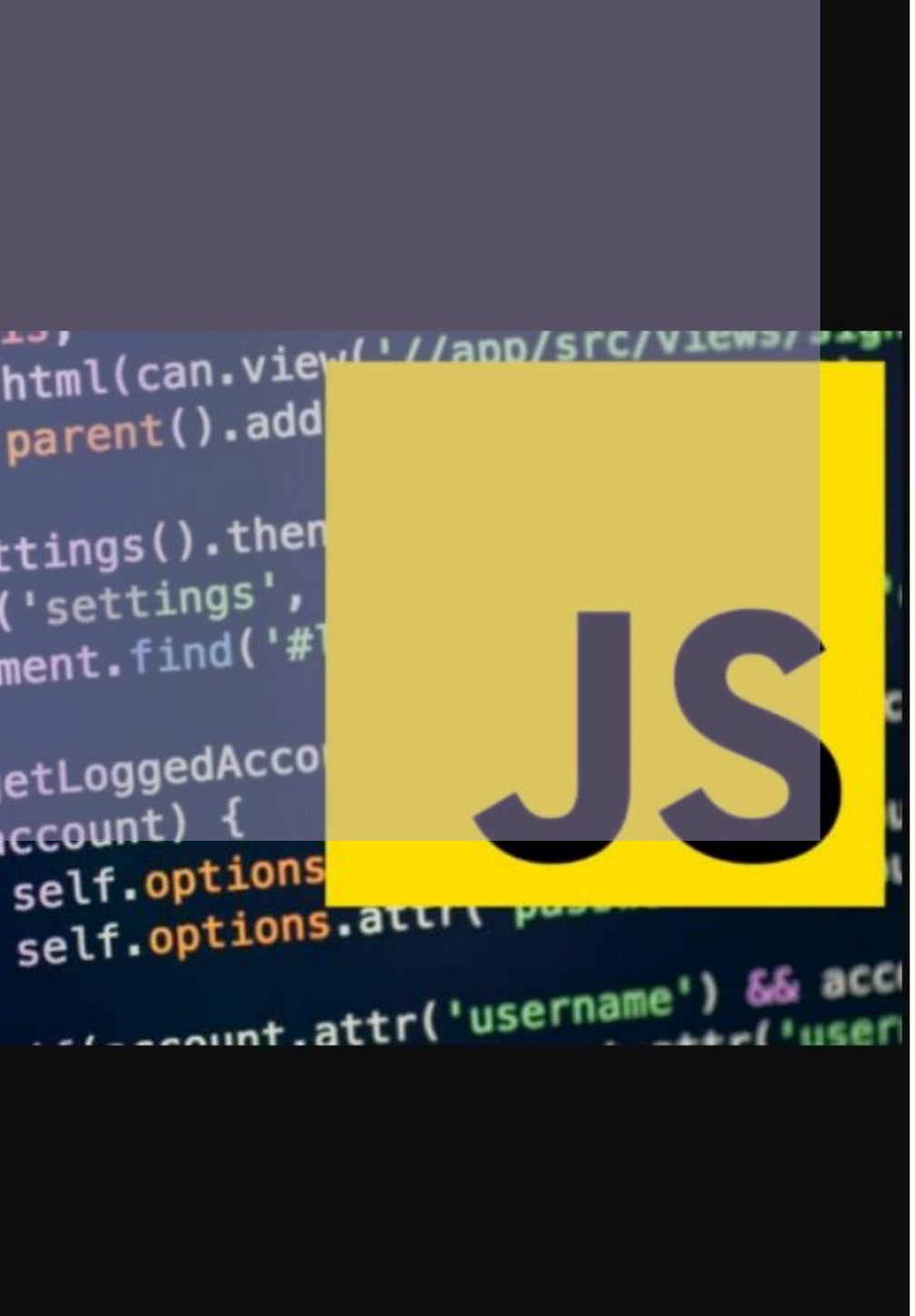
•Se reemplaza function por =>.

•Si hay un solo parámetro, se pueden **omitir los paréntesis**:

```
const saludar = nombre => "Hola, " + nombre;
```

•Si solo hay una línea, **el return es implícito**.

•Son ideales para escribir código **más limpio y legible**.



Ejemplo función tradicional vs Arrow Function

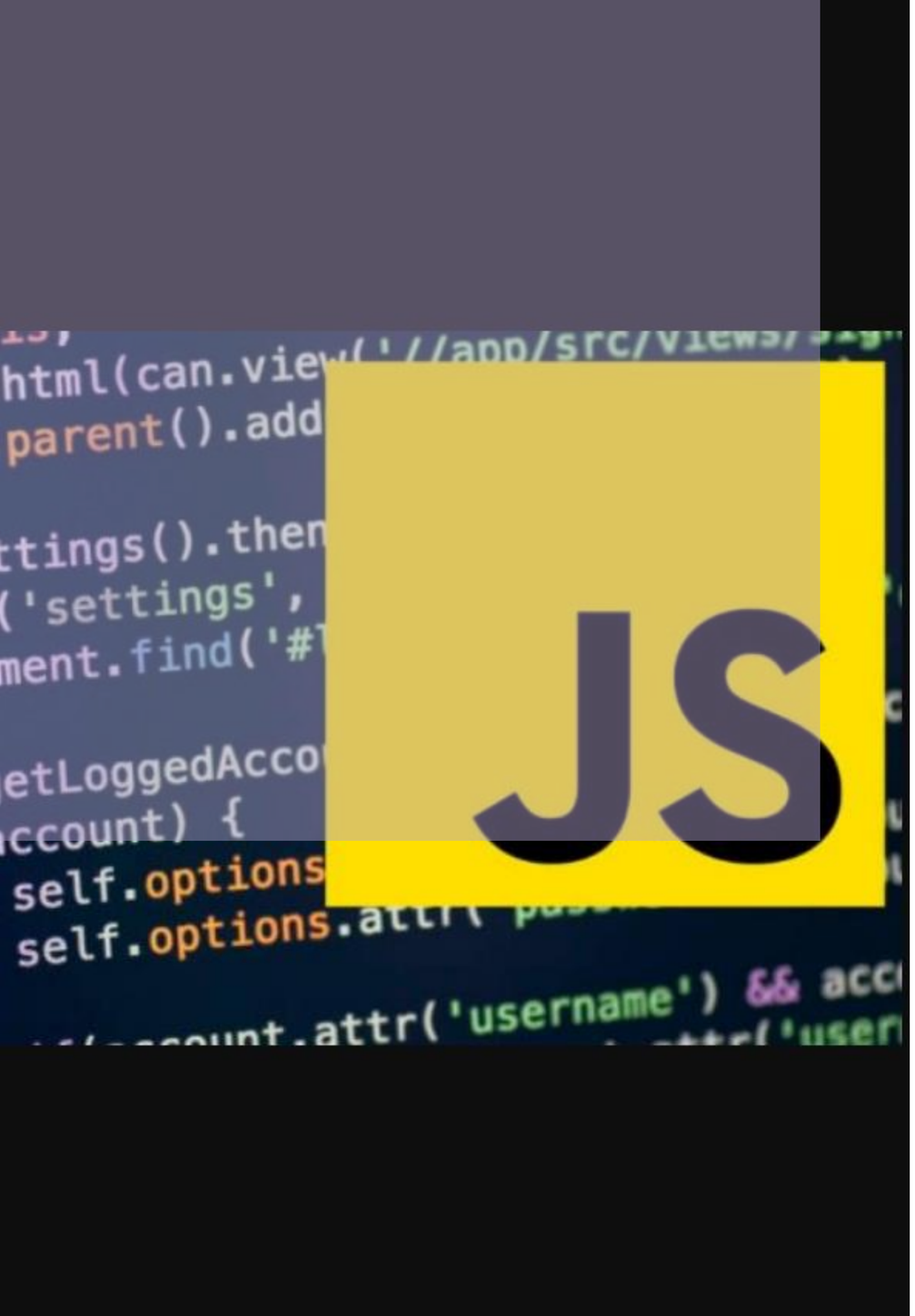
Ejemplo con función tradicional

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

console.log(sumar(5, 3)); // Resultado: 8
```

Paso a paso

1. Se declara con la palabra clave function.
2. Recibe dos parámetros: a y b.
3. Usa return para devolver la suma.
4. Se llama usando su nombre: sumar(5, 3).



Ejemplo función tradicional vs Arrow Function

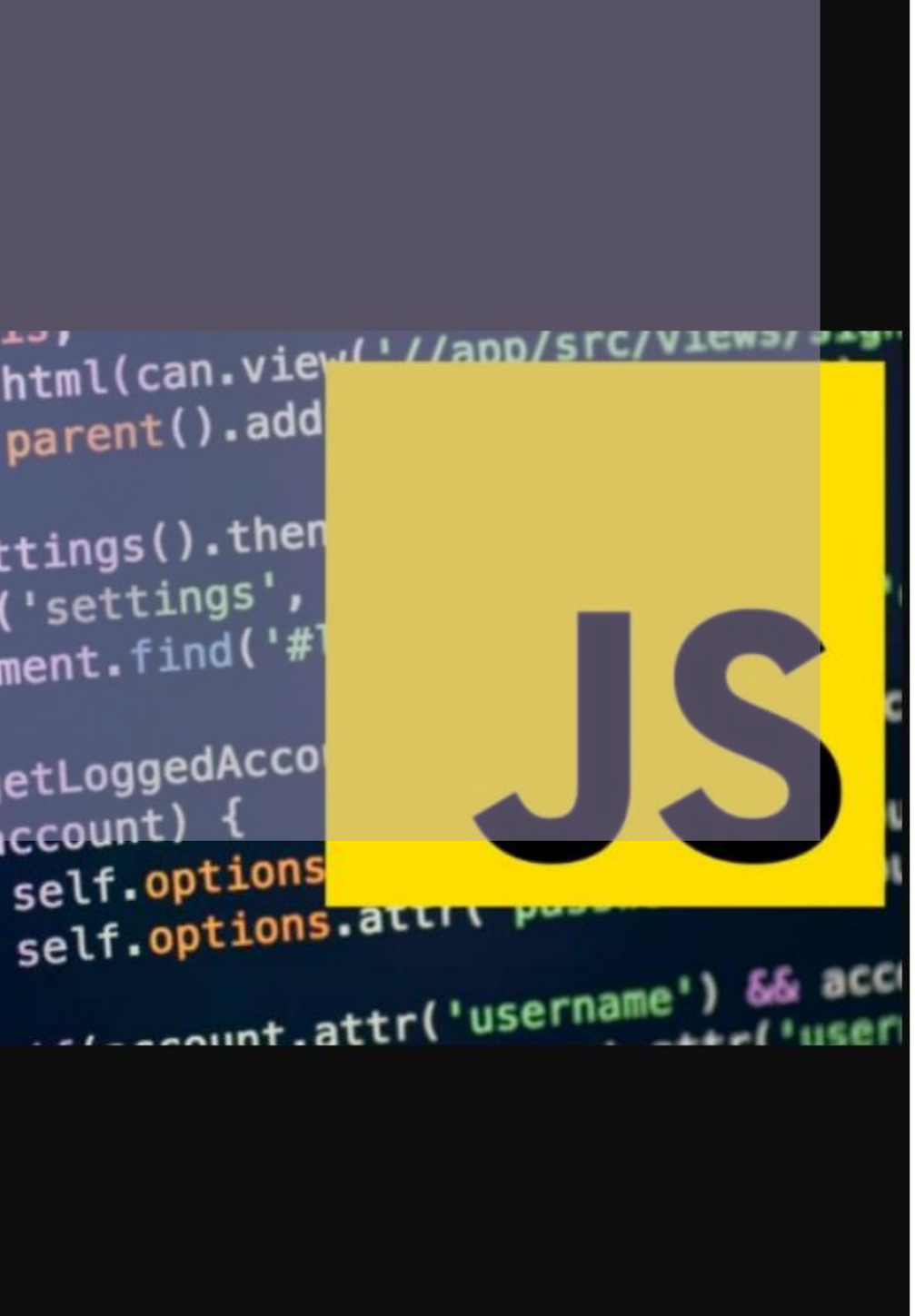
Misma función con Arrow Function

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


console.log(sumar(5, 3)); // Resultado: 8
```

Observaciones

- No usa la palabra function.
- => indica que es una función de flecha.
- El return es implícito (porque es una sola línea).



Comparativa función tradicional vs Arrow Function

Característica	Función Tradicional	Arrow Function
Sintaxis	Más larga, usa function	Más corta, usa =>
Return	Necesario escribirlo	Implícito si es una sola línea
this	Tiene su propio this	Hereda el this del entorno
Uso común	Funciones complejas o con varias líneas	Funciones simples o callbacks
Ejemplo	function sumar(a,b){ return a+b; }	(a,b) => a+b

IT Academy

by KIBERNUM