## **¿Qué son las Funciones con Sobrecarga por Defecto en TypeScript?**

En TypeScript, la sobrecarga de funciones nos permite definir múltiples declaraciones para una misma función, cada una con diferentes tipos de parámetros. Esto proporciona una interfaz más intuitiva para los desarrolladores y permite al compilador realizar comprobaciones de tipos más estrictas.

**La sobrecarga por defecto** entra en juego cuando se especifica un conjunto de sobrecargas y, luego, se proporciona una implementación que cubre todos los casos posibles. Esta implementación predeterminada se utiliza cuando ninguna de las sobrecargas específicas coincide con los argumentos proporcionados.

**Ejemplo 1: Sumar números o concatenar cadenas**

TypeScript

// sumaNumerosOConcatenarCadenas.tsfunction sumaNumerosOConcatenarCadenas(a: number, b: number): number; function sumaNumerosOConcatenarCadenas(a: string, b: string): string; function sumaNumerosOConcatenarCadenas(a: any, b: any): any { // Implementación por defecto (puede ser más específica si es necesario) return a + b; } const resultado1 = sumaNumerosOConcatenarCadenas(1, 2); // Resultado: 3const resultado2 = sumaNumerosOConcatenarCadenas("Hola", " Mundo"); // Resultado: "Hola Mundo"

Use code [with caution.](file:///C:/faq#coding)

**Explicación:**

* **Múltiples declaraciones:** Se definen dos sobrecargas para la función, una para números y otra para cadenas.
* **Implementación por defecto:** La tercera declaración con tipos any actúa como una implementación general, pero puede ser refinada para manejar casos más específicos.
* **Invocación:** Al llamar a la función, TypeScript selecciona la sobrecarga adecuada según los tipos de los argumentos.

**Ejemplo 2: Crear un elemento HTML con diferentes atributos**

TypeScript

// crearElementoHTML.tsfunction crearElementoHTML(tagName: string, attributes?: { [key: string]: string }): HTMLElement; function crearElementoHTML(tagName: string, textContent: string): HTMLElement; function crearElementoHTML(tagName: string, ...args: any[]): HTMLElement { const element = document.createElement(tagName); if (typeof args[0] === 'object') { for (const key in args[0]) { element.setAttribute(key, args[0][key]); } } else if (typeof args[0] === 'string') { element.textContent = args[0]; } return element; } const divConId = crearElementoHTML('div', { id: 'myDiv' }); const parrafoConTexto = crearElementoHTML('p', 'Hola, mundo!');

Use code [with caution.](file:///C:/faq#coding)

**Explicación:**

* **Flexibilidad:** La función permite crear elementos HTML con o sin atributos.
* **Implementación adaptable:** La implementación por defecto usa un rest parameter (...args) para manejar diferentes tipos de argumentos.
* **Versatilidad:** Se pueden crear elementos HTML con diferentes configuraciones.

**Ejemplo 3: Loggear mensajes con diferentes niveles**

TypeScript

// loggearMensaje.tsfunction loggearMensaje(message: string, level?: 'info' | 'warn' | 'error'): void; function loggearMensaje(message: string, ...args: any[]): void { const prefix = level ? `[${level.toUpperCase()}]` : ''; console.log(`${prefix} ${message}`, ...args); } loggearMensaje('Operación exitosa', 'info'); loggearMensaje('Error inesperado', 'error', new Error('Algo salió mal'));

Use code [with caution.](file:///C:/faq#coding)

**Explicación:**

* **Personalización:** Se puede especificar el nivel de log (info, warn, error).
* **Flexibilidad:** La implementación por defecto permite agregar argumentos adicionales para más detalles.
* **Uso práctico:** Ideal para crear funciones de logging personalizadas.

**En resumen:**

La sobrecarga de funciones por defecto en TypeScript proporciona una forma elegante y flexible de definir funciones que pueden aceptar diferentes tipos de argumentos. Esto mejora la legibilidad y la seguridad del código.

**¿Te gustaría explorar más ejemplos o tienes alguna otra pregunta?**

**Consideraciones adicionales:**

* **Orden de las sobrecargas:** El orden en que se declaran las sobrecargas puede afectar la selección de la implementación.
* **Sobrecarga con genéricos:** Puedes combinar la sobrecarga con genéricos para crear funciones aún más versátiles.
* **Limitaciones:** La sobrecarga de funciones tiene algunas limitaciones, como la dificultad de sobrecargar funciones con diferentes números de argumentos opcionales.