4.4. Funciones y eventos

LENGUAJES DE PROGRAMACIÓN NIVEL 3. UNIDAD 4

JavaScript - LENGUAJE DE PROGRAMACIÓN PARA LA INTERACCIÓN EN EL FRONTEND

Financiado por la Unión Europea

Plan de Recuperación, Transformación y Resiliencia

Contenido

Introducción

En el universo de JavaScript, las funciones y los eventos son pilares que permiten no solo estructurar el código de manera eficiente y reutilizable, sino también crear aplicaciones web dinámicas e interactivas.

Este módulo te introducirá a estos conceptos esenciales, que son fundamentales para cualquier desarrollador que aspire a construir interfaces de usuario ricas y responsivas.

La importancia de las funciones

Las funciones son el corazón de JavaScript, permitiendo agrupar instrucciones en bloques reutilizables que pueden ser invocados en cualquier momento y en cualquier lugar de tu programa.

Esta encapsulación no solo promueve la reutilización del código, reduciendo la redundancia y facilitando el mantenimiento, sino que también mejora la modularidad de tus aplicaciones. Al dividir el código en funciones específicas, se logra una separación clara de preocupaciones, donde cada función se encarga de una tarea única, haciendo que tu código sea más legible, manejable y escalable.

Introducción a los eventos

Los eventos, por otro lado, son señales que el navegador envía para indicar que algo ha sucedido, como un clic del mouse, una pulsación de tecla o la carga completa de una página web. En JavaScript, puedes escuchar estos eventos y definir manejadores de eventos (funciones que se ejecutan en respuesta a un evento) para añadir interactividad a tus páginas web.

Esta capacidad de responder a acciones del usuario en tiempo real es lo que hace que las aplicaciones web sean tan atractivas y dinámicas.

Juntos, funciones y eventos crean interactividad

Al combinar funciones con el manejo de eventos, puedes crear comportamientos complejos e interactivos en tus aplicaciones web. Por ejemplo, podrías tener una función que actualiza la interfaz de usuario

basada en los datos ingresados en un formulario, y esa función podría ser llamada cada vez que el usuario presiona una tecla o hace clic en un botón. Este enfoque no solo hace que tus aplicaciones web sean más interactivas y responsivas, sino que también encapsula la lógica de interacción en bloques de código reutilizables, manteniendo tu código organizado y eficiente.

Preparándonos para el viaje

A lo largo de este módulo, exploraremos la sintaxis y los patrones para definir y utilizar funciones, aprenderemos a manejar eventos comunes del navegador y descubriremos cómo estas herramientas se complementan entre sí para crear experiencias web ricas y dinámicas. Prepárate para sumergirte en el fascinante mundo de las funciones y eventos en JavaScript, donde la lógica y la interactividad se unen para dar vida a tus aplicaciones web.

Funciones en JavaScript

Las funciones son uno de los conceptos más fundamentales y poderosos en JavaScript, proporcionando la base para la creación de código modular y reutilizable.

Este módulo se sumerge en el mundo de las funciones en JavaScript, explorando su sintaxis, usos y las mejores prácticas asociadas con ellas. Al entender y aprovechar las funciones, puedes escribir código más limpio, más eficiente y mantenible.

Definición y sintaxis

Las funciones en JavaScript son bloques de código diseñados para realizar una tarea específica, y pueden ser invocadas tantas veces como sea necesario.

La sintaxis básica de una función abarca la definición de la función, los parámetros que puede recibir y el cuerpo de la función donde se ejecuta el código:

- Palabra clave function: Se utiliza en los dos tipos de función que veremos más adelante (excepto en las funciones flecha).
- Nombre de la función: Aquí hay diferencias. En las declaraciones de función, el nombre es obligatorio y sigue inmediatamente después de la palabra clave function. En las expresiones de función, el nombre es

opcional (puede ser una función anónima) y la función se asigna a una variable.

- Parámetros: Los parámetros se definen entre paréntesis () y se separan por comas. Los parámetros actúan como variables locales dentro de la función, y los valores se pasan a la función en el momento de la llamada (argumentos). Las funciones pueden tener desde cero hasta muchos parámetros.
- Cuerpo de la función: El cuerpo de la función se encierra entre llaves {} y contiene las instrucciones que se ejecutan cuando se llama a la función. El cuerpo puede contener cualquier número de instrucciones, incluidas otras declaraciones de función (funciones anidadas).

```
function saludar(nombre) {
    console.log(`Hola, ${nombre}!`);
}
```

[1]

Consideraciones

- return: Dentro del cuerpo de la función, puedes usar la sentencia return para devolver un valor al código que llamó a la función. Si no se especifica un return, o si la función alcanza el final de su cuerpo sin encontrar una sentencia return, se devuelve undefined.
- Funciones Flecha: Introducidas en ES6, las funciones flecha ofrecen una sintaxis más concisa y no tienen su propio this, arguments, super, o new.target. Son especialmente útiles para funciones cortas que se pasan como argumentos o callbacks.

[1] Accesibilidad:
function saludar(nombre) {
console.log("¡Hola, $(nombre)!");
}

Tipos de función

Hay dos maneras principales de definir funciones:

Declaraciones de función:

Define una función con el nombre específico.

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

[2]

Se caracteriza por el uso de la palabra clave function seguida de un nombre para la función, paréntesis () que pueden contener parámetros, y un bloque de código entre llaves {} que define el cuerpo de la función.

Principales características:

- Hoisting: Las declaraciones de función son completamente elevadas (hoisted) al principio del ámbito en el que se definen, lo que significa que puedes llamar a una función antes de que sea declarada en el código.

```
saludar(); // Funciona correctamente, imprime "¡Hola, mundo!"

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

[3]

[2] Accesibilidad:
function saludar(nombre) {
console.log("¡Hola, mundo!");
}

[3] Accesibilidad:
saludar(); // Funciona correctamente, imprime "Hola, mundo!"

function saludar() {
console.log("Hola, mundo!");
}

- Ámbito de función: El ámbito de una función declarada está vinculado al ámbito en el que se define, ya sea global o local dentro de otro bloque o función.

Expresiones de función:

Una expresión de función implica definir una función dentro de una expresión. Puede ser anónima (sin nombre) o nombrada y se asigna generalmente a una variable. La función en sí misma puede ser una función anónima o una función flecha.

```
const despedirse = function() {
    console.log("¡Adiós, mundo!");
};
```

[4]

Características:

- No hoisting: A diferencia de las declaraciones de función, las expresiones de función no son elevadas. La variable que almacena la función se eleva, pero sin su definición. Esto significa que no puedes llamar a una expresión de función antes de definirla en el código.
- Funciones anónimas y flecha: Las expresiones de función permiten la creación de funciones anónimas y funciones flecha, lo que es útil para los callbacks y las funciones que se pasan como argumentos a otras funciones.

¿Cuándo usar una u otra?

- Declaraciones de función: Son preferibles cuando necesitas organizar tu código con funciones que pueden ser llamadas desde cualquier parte de tu script, gracias al hoisting. Son ideales para definir funciones que forman la base de tu lógica de programación y que necesitan ser referenciadas en múltiples lugares.
- Expresiones de función: Son útiles cuando la función necesita ser pasada como un valor, por ejemplo, como un argumento a otra

[4] Accesibilidad:
const despedirse = function() {
    console.log("Adios, mundo!");
};

función, o cuando la definición de la función depende de una condición. Las expresiones de función también son la elección para definir funciones que no necesitan ser llamadas antes de su definición en el código, lo que puede ayudar a mantener la legibilidad y la estructura del código.

Parámetros y argumentos

Las funciones pueden recibir datos de entrada conocidos como parámetros, y los valores reales pasados a estos parámetros se llaman argumentos.

Parámetros

Los parámetros son variables nombradas enumeradas en la definición de la función, que actúan como placeholders para los valores que la función recibirá cuando sea invocada. Los parámetros se definen entre los paréntesis en la declaración de la función y funcionan dentro de la función como variables locales.

```
function sumar(a, b) {  // 'a' y 'b' son parámetros
    return a + b;
}
```

5

En este ejemplo, a y b son parámetros de la función sumar.

[5] Accesibilidad:
function sumar(a, b) { // a y b son parámetros
return a+b;
}

Argumentos

Los argumentos son los valores reales pasados a la función cuando es invocada. Cada argumento corresponde a un parámetro de la función, en el orden en que se definen.

```
let resultado = sumar(3, 5);  // '3' y '5' son argumentos
```

6

Aquí, 3 y 5 son los argumentos pasados a la función sumar.

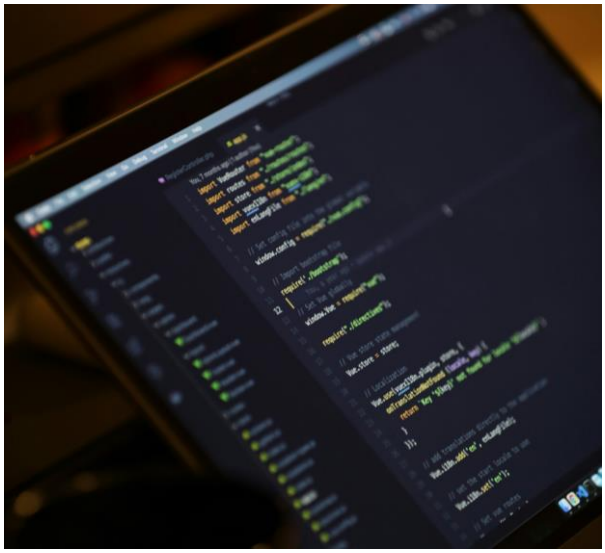
Prácticas recomendadas

Aunque el hoisting permite cierta flexibilidad, es generalmente considerado una buena práctica declarar todas tus funciones al inicio de su ámbito antes de invocarlas.

Esto mejora la legibilidad del código y evita confusiones relacionadas con el hoisting, especialmente para quienes están menos familiarizados con estos aspectos de JavaScript.

[6] Accesibilidad
let resultado = sumar(3, 5); //Los números 3 y 5 son argumentos

Eventos en JavaScript



Los eventos son fundamentales en la creación de páginas web interactivas, permitiendo que el código responda a acciones del usuario como clics, pulsaciones de teclas y cambios en el formulario. Este módulo explora cómo JavaScript puede detectar y reaccionar a estos eventos, transformando páginas estáticas en aplicaciones web dinámicas y reactivas.

Concepto

Los eventos son acciones o sucesos que ocurren en el navegador que el código puede responder; por ejemplo, cuando un usuario hace clic en un botón o escribe en un campo de texto.

En JavaScript, estos eventos pueden ser escuchados y manejados mediante funciones, permitiendo una interacción rica y dinámica entre el usuario y la aplicación.

Manejadores de eventos

Los manejadores de eventos son funciones en JavaScript que se ejecutan en respuesta a un evento específico en un elemento del DOM. Puedes asignar manejadores de eventos a elementos utilizando atributos de evento en HTML o mediante JavaScript usando addEventListener. Veamos cada uno de ellos:

Atributos de evento en HTML:

Un atributo de evento en HTML es una especificación dentro de una etiqueta HTML que permite ejecutar código JavaScript en respuesta a determinados eventos en ese elemento. Estos eventos pueden incluir acciones del usuario como hacer clic en un elemento, pasar el cursor sobre él, presionar una tecla, etc.

La sintaxis de un atributo de evento consiste en el nombre del evento precedido por on, seguido de = y luego comillas que contienen el código JavaScript que se ejecutará cuando se dispare el evento. Este código puede ser una llamada a función, una expresión JavaScript o cualquier código JavaScript válido.

```
<elemento onnombredelevento="acción">
```

[7]

Por ejemplo:

```
<button onclick="alert('¡Hola, mundo!')">Haz clic en mí</button>
```

[8]

En el ejemplo, se define un botón HTML con un atributo onclick. Este atributo onclick está configurado para ejecutar un fragmento de código JavaScript

[7] Accesibilidad:
<elemento onnombredelevento=”acción”>
[8] Accesibilidad:
<button onclick=”alert(¡Hola, mundo!')"»Haz clic en mí</button>

cuando el usuario hace clic en el botón. El código dentro del atributo onclick, alert('¡Hola, mundo!'), es una llamada a la función alert(), que muestra un cuadro de diálogo emergente con el mensaje "¡Hola, mundo!".

Esto ilustra una forma sencilla de agregar interactividad a un elemento de página web, en este caso, haciendo que el botón responda a los clics del usuario y ejecute una acción específica (mostrar un mensaje) cuando se produce el evento de clic.

Usando addEventListener:

addEventListener es un método que se utiliza en JavaScript para adjuntar un manejador de eventos a un elemento del Document Object Model (DOM). Este método permite definir más claramente la separación entre la estructura de la página (HTML) y el comportamiento (JavaScript), y ofrece una mayor flexibilidad y control sobre los eventos en comparación con los atributos de evento en línea.

La sintaxis básica de addEventListener es la siguiente:

```
elemento.addEventListener(tipoDeEvento, manejadorDeEvento, useCapture);
```

[9]

- elemento: Es el elemento del DOM al que se le quiere asignar el manejador de evento. Puede ser obtenido mediante métodos como document.getElementById(), document.querySelector(), etc.
- tipoDeEvento: Es una cadena de texto que especifica el tipo de evento al que se quiere reaccionar, sin el prefijo "on". Por ejemplo, para eventos de clic, se usa "click".
- manejadorDeEvento: Es la función que se ejecutará cuando ocurra el evento. Puede ser una función nombrada o una función anónima.
- useCapture (opcional): Es un valor booleano que indica la fase durante la cual el manejador de evento debe ser activado. Si es true, el manejador se activa durante la fase de captura; si es false (o se omite), durante la fase de burbujeo.

[9] Accesibilidad:
elemento. addEventListenez(tipoDeEvento, manejadorDeEvento, useCapture);

Por ejemplo

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

10

En este ejemplo, se selecciona un elemento botón por su ID y se le asigna un manejador de evento para el evento click usando addEventListener. Cuando el botón es clickeado, se mostrará una alerta.

Tipos de eventos comunes

Los tipos de eventos comunes abarcan una amplia gama de interacciones del usuario y del navegador, permitiendo una interactividad rica y dinámica en las aplicaciones web. Aquí te detallo algunos de los tipos de eventos más comunes en JavaScript:

Eventos del Mouse

- **click:** Se dispara cuando el usuario hace clic en un elemento.
- **dblclick:** Ocurre cuando el usuario hace doble clic en un elemento.
- **mouseover:** Se activa cuando el cursor del mouse se mueve sobre un elemento.
- **mouseout:** Ocurre cuando el cursor del mouse sale del área de un elemento.
- **mousedown:** Se dispara cuando el usuario presiona un botón del mouse sobre un elemento.
- **mouseup:** Se activa cuando el usuario suelta un botón del mouse sobre un elemento.

[10] Accesibilidad:

```
const boton = document. getElementtById( “miBoton');
boton.addEventListener('click', function() {
    alert(' ¡Botón clickeado!');
});
```

Eventos del teclado

- **keydown:** Se dispara cuando el usuario presiona una tecla.
- **keyup:** Ocurre cuando el usuario suelta una tecla presionada.
- **keypress:** Se activa cuando el usuario presiona una tecla que produce un carácter (obsoleto en muchos entornos).

Eventos de la interfaz

- **load:** Se dispara cuando un objeto, como el documento (página web), ha sido completamente cargado.
- **unload:** Ocurre antes de que el documento esté a punto de descargarse completamente.
- **resize:** Se activa cuando se cambia el tamaño de la ventana del navegador.
- **scroll:** Se dispara cuando se desplaza el documento o un elemento.

Eventos del formulario

- **submit:** Ocurre cuando se envía un formulario.
- **change:** Se dispara cuando el valor de un elemento de formulario (como <input>, <select>) cambia.
- **focus:** Se activa cuando un elemento recibe el foco.
- **blur:** Ocurre cuando un elemento pierde el foco.

Eventos de toque (Touch Events)

- **touchstart:** Se dispara cuando el usuario toca la pantalla.
- **touchmove:** Ocurre cuando el usuario mueve el dedo por la pantalla.
- **touchend:** Se activa cuando el usuario levanta el dedo de la pantalla.

Eventos de drag & drop

- **dragstart:** Se dispara cuando el usuario comienza a arrastrar un elemento.
- **drag:** Ocurre mientras el elemento está siendo arrastrado.
- **dragend:** Se activa cuando el usuario suelta el elemento.

Eventos de transición y animación

- **transitionend:** Se dispara cuando una transición CSS ha completado.
- **animationstart:** Ocurre cuando comienza una animación CSS.
- **animationend:** Se activa cuando una animación CSS termina.

Estos eventos se pueden utilizar para crear efectos interactivos, validaciones de formularios, animaciones, respuestas a la entrada del usuario y mucho más, haciendo que las aplicaciones web sean más dinámicas e interesantes para los usuarios. La habilidad para manejar adecuadamente estos eventos es una parte esencial del desarrollo front-end.

Conclusión

Al concluir nuestro viaje a través del módulo de "Funciones y Eventos" en JavaScript, hemos desentrañado cómo estos elementos esenciales no solo enriquecen las aplicaciones web, sino que son fundamentales para la creación de experiencias de usuario dinámicas e interactivas. Las funciones nos brindan la capacidad de encapsular código para tareas específicas, promoviendo la reutilización y la modularidad, mientras que el manejo de eventos nos permite responder a acciones del usuario, como clics, entradas de teclado y más, haciendo que nuestras páginas web respondan y se sientan vivas.

Las funciones y el manejo de eventos juntos forman la columna vertebral de la interactividad en el desarrollo web. Mediante el uso de funciones, podemos definir comportamientos específicos que se activan en respuesta a eventos del usuario, lo que permite que nuestras aplicaciones web no solo muestren contenido estático, sino que interactúen y respondan a los usuarios en tiempo real. Esta combinación es lo que transforma un documento HTML estándar en una experiencia web rica y envolvente.

A medida que continúes tu camino en el desarrollo web, te animamos a experimentar y practicar con las funciones y el manejo de eventos en tus propios proyectos. Recuerda, la práctica constante y la exploración de nuevos desafíos son clave para dominar estos conceptos y convertirte en un desarrollador web competente y creativo. Estamos emocionados de ver cómo aplicarás estas herramientas para crear aplicaciones web aún más interactivas y dinámicas.

¡Hasta la próxima, y feliz codificación!

Recursos adicionales

Para reforzar y expandir tu comprensión de los temas tratados en este módulo, aquí tienes una selección de recursos adicionales. Estos enlaces te llevarán a documentación oficial y tutoriales en línea que te proporcionarán una visión más detallada de las variables, los tipos de datos y los operadores en JavaScript.

Documentación oficial

- MDN Web Docs - JavaScript: Una de las mejores fuentes de documentación para desarrolladores web. Ofrece explicaciones detalladas, ejemplos de código y buenas prácticas.
  https://developer.mozilla.org/en-US/docs/Web/JavaScript

Tutoriales y guías

- JavaScript.info: Un recurso moderno y completo que cubre JavaScript desde los fundamentos hasta temas avanzados, con ejemplos claros y prácticos.
  https://javascript.info/
- freeCodeCamp: Ofrece un extenso curso de JavaScript como parte de su currículum de desarrollo web, completo con ejercicios prácticos y proyectos.
  https://www.freecodecamp.org/

Plataformas interactivas

- Codecademy: Proporciona un curso interactivo de JavaScript que es excelente para principiantes absolutos, ayudándote a aprender mediante la práctica.
  https://www.codecademy.com/

Blogs y artículos

- W3Schools JavaScript Tutorial: Un recurso educativo gratuito que ofrece tutoriales sobre una amplia gama de tecnologías web, incluyendo JavaScript. Es conocido por sus tutoriales interactivos que permiten a los usuarios experimentar con el código directamente en el navegador.
  https://www.w3schools.com/js/