Diplomatura en

Programación FullStack Developer

GLOBAL ACADEMY

JavaScript
Funciones

Funciones

Definición de funciones:
En JavaScript, una función es un bloque de código reutilizable que realiza una tarea específica cuando es invocada o llamada.

Las funciones en JavaScript son objetos de primera clase, lo que significa que pueden ser asignadas a variables, pasadas como argumentos y devueltas como valores de otras funciones.

Importancia en programación:
Las funciones son fundamentales en la programación debido a su capacidad para modularizar el código.

Permite dividir tareas complejas en pasos más pequeños y manejables, lo que facilita la comprensión y el mantenimiento del código.

La reutilización de funciones a lo largo del código ayuda a reducir la duplicación de código y mejora la eficiencia en el desarrollo.

Las funciones también promueven la legibilidad del código al proporcionar una forma de nombrar y encapsular acciones específicas.

Sintaxis básica:
En JavaScript, una función se declara utilizando la palabra clave function, seguida del nombre de la función y paréntesis (), que pueden contener parámetros separados por comas.

El cuerpo de la función está encerrado entre llaves {} y contiene las instrucciones que definen el comportamiento de la función.

Ejemplos de declaración de funciones:

```
// Declaración de una función sin parámetros
function saludar() {
    console.log("¡Hola, mundo!");
}

// Declaración de una función con parámetros
function sumar(a, b) {
    return a + b;
}
```

Explicación:
La función saludar() simplemente imprime "¡Hola, mundo!" en la consola cuando es llamada.

La función sumar(a, b) toma dos parámetros a y b, y devuelve la suma de estos dos valores.

Notas:
Las funciones declaradas pueden ser llamadas en cualquier parte del código, incluso antes de que la declaración de la función aparezca en el código (debido al hoisting de funciones en JavaScript).

Es importante tener en cuenta que las funciones declaradas no terminan con un punto y coma **;**, a diferencia de las expresiones de función.

El hoisting es un comportamiento en JavaScript donde las declaraciones de variables y funciones se mueven al inicio de su contexto de ejecución durante la fase de compilación, antes de que se ejecute el código. Esto significa que, en términos prácticos, puedes usar una variable o llamar a una función antes de que estén declaradas en el código, y JavaScript no arrojará un error. Sin embargo, es importante tener en cuenta que solo la declaración en sí es elevada (hoisted), no la inicialización de la variable (si hay una), lo que puede llevar a comportamientos inesperados si no se entiende correctamente.

Expresiones de funciones

Sintaxis y diferencias con la declaración de funciones:
En JavaScript, además de la declaración de funciones, también se pueden crear funciones mediante expresiones de funciones.

La principal diferencia radica en que las expresiones de funciones pueden asignarse a variables y no son elevadas (hoisted) al principio del contexto de ejecución como las declaraciones de funciones.

Ejemplos de expresiones de funciones:

```
// Expresión de función anónima
let suma = function(a, b) {
    return a + b;
};

// Expresión de función con nombre (nombre opcional)
let resta = function restaFunc(a, b) {
    return a - b;
};
```

Explicación:
En el primer ejemplo, se asigna una función anónima a la variable suma. Esta función toma dos parámetros a y b y devuelve la suma de estos valores.

En el segundo ejemplo, se asigna una función con nombre restaFunc a la variable resta. Aunque el nombre de la función es opcional en las expresiones de función, puede ser útil para hacer referencia a la función dentro de sí misma (por ejemplo, en la recursión).

Notas:
Las expresiones de funciones son especialmente útiles cuando se necesitan funciones como argumentos para otras funciones (por ejemplo, en los callbacks) o cuando se desea asignar una función a una variable.

Es importante recordar que las expresiones de funciones no son elevadas (hoisted) como las declaraciones de funciones, por lo que deben declararse antes de ser utilizadas en el código.

Un callback en JavaScript es una función que se pasa como argumento a otra función y se ejecuta después de que cierto proceso ha finalizado o cuando ocurre un evento específico. Los callbacks son comunes en situaciones donde se manejan operaciones asíncronas, como las solicitudes de red o las interacciones del usuario. Permiten que el código continúe ejecutándose de manera síncrona mientras se espera que ocurra algún evento externo, y luego se activa el callback para manejar la respuesta o el resultado. Esto facilita la escritura de código asíncrono y la gestión de flujos de control complejos en JavaScript.

Diferencia entre parámetros y argumentos:
En JavaScript, es importante distinguir entre parámetros y argumentos en el contexto de las funciones.

Los parámetros son variables declaradas en la definición de la función y representan los valores que la función espera recibir cuando es invocada.

Los argumentos son los valores reales que se pasan a la función cuando esta es llamada.

Ejemplos de uso de argumentos en funciones:

```
// Función que suma los argumentos recibidos
function sumar(a, b) {
    return a + b;
}

// Llamada a la función sumar con argumentos
let resultado = sumar(5, 3);
```

En el ejemplo, la función sumar(a, b) espera dos parámetros a y b.

Cuando se llama a la función sumar(5, 3), 5 y 3 son los argumentos que se pasan a la función y se asignan a los parámetros a y b, respectivamente.

La función devuelve la suma de los dos argumentos, que en este caso es 8.

Concepto de valor de retorno en funciones:
En JavaScript, las funciones pueden devolver valores mediante la declaración de return.

El valor devuelto por una función puede ser de cualquier tipo de dato, incluyendo números, cadenas, booleanos, objetos u otros valores.

Ejemplos de funciones que devuelven valores:

```
// Función que devuelve el cuadrado de un número
function cuadrado(x) {
    return x * x;
}

// Función que verifica si un número es par
function esPar(num) {
    return num % 2 === 0;
}
```

En el primer ejemplo, la función cuadrado(x) toma un parámetro x y devuelve el resultado de multiplicar x por sí mismo, es decir, su cuadrado.

En el segundo ejemplo, la función esPar(num) toma un parámetro num y devuelve true si num es par, y false en caso contrario.

Alcance global y local en funciones:

En JavaScript, el alcance se refiere a la accesibilidad de las variables dentro del código.

Las variables pueden tener un alcance global o local dentro de una función.

```
let globalVar = "Soy global"; // Variable global

function miFuncion() {
    // Variable local
    let localVar = "Soy local";
    console.log(globalVar); // Acceso a variable global
    console.log(localVar); // Acceso a variable local
}

miFuncion();
console.log(globalVar); // Acceso a variable global fuera de la función
//console.log(localVar); // Error: localVar no está definida fuera de la función
```

En el ejemplo, globalVar es una variable global, lo que significa que está disponible en todo el código, incluidas las funciones.

Dentro de la función miFuncion(), se declara la variable localVar, que es local a esa función y no está disponible fuera de ella.

Dentro de la función, se puede acceder tanto a la variable global (globalVar) como a la variable local (localVar), pero fuera de la función, solo se puede acceder a la variable global.

Alcance global y local en funciones:

En JavaScript, el alcance se refiere a la accesibilidad de las variables dentro del código.

Las variables pueden tener un alcance global o local dentro de una función.

```
let globalVar = "Soy global"; // Variable global

function miFuncion() {
    // Variable local
    let localVar = "Soy local";
    console.log(globalVar); // Acceso a variable global
    console.log(localVar); // Acceso a variable local
}

miFuncion();
console.log(globalVar); // Acceso a variable global fuera de la función
//console.log(localVar); // Error: localVar no está definida fuera de la función
```

En el ejemplo, globalVar es una variable global, lo que significa que está disponible en todo el código, incluidas las funciones.

Dentro de la función miFuncion(), se declara la variable localVar, que es local a esa función y no está disponible fuera de ella.

Dentro de la función, se puede acceder tanto a la variable global (globalVar) como a la variable local (localVar), pero fuera de la función, solo se puede acceder a la variable global.

Funciones flecha

Sintaxis de funciones flecha:

- Las funciones flecha son una característica introducida en ECMAScript 6 (ES6) que proporciona una sintaxis más concisa para definir funciones en JavaScript.
- Se definen utilizando una sintaxis de flecha =>.

Ejemplos de funciones flecha:

```
// Función flecha para sumar dos números
let sumar = (a, b) => a + b;

// Función flecha para elevar un número al cuadrado
let cuadrado = x => x * x;

// Función flecha para saludar a una persona
let saludar = nombre => console.log("¡Hola, " + nombre + "!");
```

- La función flecha sumar toma dos parámetros a y b y devuelve la suma de estos valores.
- La función flecha cuadrado toma un parámetro x y devuelve el cuadrado de ese valor.
- La función flecha saludar toma un parámetro nombre y saluda a la persona con ese nombre.

Callbacks

Callbacks:

- En JavaScript, un callback es una función que se pasa como argumento a otra función y se invoca después de que cierto proceso ha finalizado o cuando ocurre un evento específico.
- Los callbacks son fundamentales en la programación asíncrona y son comúnmente utilizados para manejar operaciones que pueden tomar tiempo, como las solicitudes de red o las interacciones del usuario.

```
// Función que simula una solicitud de red
function hacerSolicitud(url, callback) {
    // Simular tiempo de espera
    setTimeout(function() {
        // Simular respuesta de la solicitud
        let datos = "Datos de respuesta";
        // Llamar al callback con los datos recibidos
        callback(datos);
    }, 2000); // 2000 milisegundos (2 segundos) de tiempo de espera
}

// Llamada a la función hacerSolicitud con un callback
hacerSolicitud("https://ejemplo.com/api", function(datos) {
    console.log("Datos recibidos:", datos);
});
```

Explicación:
En el ejemplo, la función hacerSolicitud simula una solicitud de red que toma un tiempo para completarse.

Se pasa una función callback como segundo argumento a hacerSolicitud, que será invocada con los datos recibidos una vez que la solicitud se haya completado.

Dentro del callback, se puede manejar la respuesta de la solicitud, como imprimir los datos recibidos en la consola.

Notas:
Los callbacks son una forma común de manejar la programación asíncrona en JavaScript, ya que permiten especificar qué hacer después de que se complete una tarea.

Sin embargo, el anidamiento excesivo de callbacks puede llevar a un código difícil de leer y mantener, por lo que se recomienda el uso de técnicas como las promesas o async/await para mitigar este problema.

Conclusiones y recomendaciones

Ventajas de usar funciones en JavaScript:

- Las funciones proporcionan una forma de organizar y reutilizar el código en JavaScript.
- Permiten encapsular lógica y comportamiento para realizar tareas específicas.
- Facilitan la escritura de código modular y mantenible.

Aspectos a tener en cuenta al trabajar con funciones:

- Es importante comprender el alcance de las variables y cómo se manejan dentro de las funciones.
- Debe prestarse atención al manejo de parámetros y argumentos para asegurar la correcta ejecución de las funciones.

Recomendaciones para mejorar el uso de funciones:

- Utilizar nombres descriptivos para las funciones y parámetros para mejorar la legibilidad del código.

- Evitar la anidación excesiva de funciones y callbacks para mantener el código limpio y fácil de entender.

Conclusión:

- Las funciones son una parte fundamental de JavaScript y son una herramienta poderosa para escribir código flexible y mantenible.

- Al comprender y dominar el uso de funciones, se puede mejorar significativamente la calidad y la eficiencia del código JavaScript.

Ejercicio

- Diseñar una función en JavaScript que reciba un array de números como parámetro y devuelva el promedio de esos números.

- La función debe calcular el promedio de todos los números en el array y devolverlo como resultado.

Ejemplo de uso:

```
let numeros = [10, 20, 30, 40, 50];
let promedio = calcularPromedio(numeros);
console.log("El promedio es:", promedio); // Resultado esperado: 30
```

Pautas para resolver el ejercicio:

1. Definir una función llamada calcularPromedio que tome un array de números como parámetro.
2. Calcular la suma de todos los números en el array utilizando un bucle o el método reduce().
3. Dividir la suma por la cantidad de elementos en el array para obtener el promedio.
4. Devolver el promedio como resultado de la función.

Ejemplo de implementación:

```
function calcularPromedio(numeros) {
    let suma = numeros.reduce((total, num) => total + num, 0);
    return suma / numeros.length;
}
```

Ejercicio adicional:

1) Descripción del ejercicio:

- Diseña una función en JavaScript que reciba un array de números.

- La función debe devolver un nuevo array que contenga pares de números del array original.

2) Descripción del ejercicio:

- Crear una función que reciba un array de strings y un número entero como parámetros.

- La función debe devolver un nuevo array que contenga solo aquellos strings del array original que tengan una longitud mayor al valor dado.

```
let palabras = ["casa", "perro", "gato", "coche", "bicicleta"];
let palabrasFiltradas = filtrarPorLongitud(palabras, 5);
```