

{JS}

Clase 04

JS

<índice>

Funciones en Javascript

¿Qué es una función?

Invocación de una función

Return o no return

Parámetros

Paso por referencia y por valor

Cómo declarar funciones

¿Qué es una función?

¿Qué es una función?

- ¿Qué es una función?
- Ventajas del uso de funciones

¿Qué es una función?

¿Qué es una función?

Bloques de código ejecutable a los que podemos pasar parámetros y operar con

algo.

Además, las funciones podrán devolvernos un resultado

¿Qué es una función?

Ventajas del uso de funciones

¿Qué es una función?

Reutilización de Código

Las funciones permiten reutilizar código, evitando la duplicidad. Esto significa que puedes escribir una función para realizar una tarea específica una vez y luego llamarla desde diferentes partes del programa tantas veces como sea necesario.

¿Qué es una función?

Organización y Legibilidad

Las funciones ayudan a organizar el código en bloques lógicos y manejables. Esto hace que el código sea más fácil de leer y entender, lo que es especialmente útil cuando se trabaja en proyectos grandes o en equipo.

¿Qué es una función?

Facilidad de Mantenimiento

Si necesitas modificar la lógica de un proceso específico que se encuentra en una función, solo tienes que hacerlo en un lugar. Esto simplifica el mantenimiento y reduce el riesgo de errores.

¿Qué es una función?

Abstracción y Enfoque

Las funciones permiten abstraer detalles de implementación. Puedes usar una función sin necesidad de saber cómo realiza su tarea internamente. Esto te permite centrarte en lo que hace la función y no en cómo lo hace.

¿Qué es una función?

Reducción de Errores

Al tener un código organizado en funciones, es menos probable que cometas errores.

Además, si un error ocurre en una función, generalmente es más fácil de localizar y corregir.

¿Qué es una función?

Facilidad para Pruebas

Las funciones hacen más fácil probar tu código. Puedes probar cada función individualmente, lo que ayuda a identificar y solucionar problemas de manera más eficiente.

¿Qué es una función?

Facilidad para Pruebas

Las funciones hacen más fácil probar tu código. Puedes probar cada función individualmente, lo que ayuda a identificar y solucionar problemas de manera más eficiente.

¿Qué es una función?

```
Inicio de la definición Identificador Parámetros  
↓ ↓ ↓  
function calcularCuadrado (numero) {  
    let resultado = 0;  
    resultado = numero ** 2;  
    return resultado;  
}  
↑  
Valor de retorno
```

Cuerpo

¿Qué es una función?

Ejemplos

- Función saludar
- Función multiplicar sin parámetros

Invocación de una función

¿Qué es una función?

¿Cómo ejecutamos o llamamos a una función?

Return o no return

Return o no return

- Ejemplo de una función sin return
- ¿Qué devuelve una función sin return?
- Ejemplo de una función con return

Parámetros

Parámetros

- Definición de parámetro
- ¿Para qué sirve un parámetro?
- Múltiples parámetros
- Orden de parámetros
- Parámetros por defecto (ES6+)
- Arguments (argumentos variables)

Paso por referencia y por valor

Parámetros

- Ámbito de variables y tipos de parámetros
- ¿Qué es el paso por valor?
- ¿Qué es el paso por referencia?
- ¿Cómo lo hace Javascript?

Parámetros

Ámbito de variables y tipos de parámetros

```
var mensaje = "Fuera de la función";
function mostrarAnuncio() {
    var mensaje = "Dentro de la función";
    console.log(mensaje);
}
mostrarAnuncio();
console.log(mensaje);
```

Parámetros

Ámbito de variables y tipos de parámetros

```
function mostrarAnuncio() {  
    var mensaje = "Dentro de la función";  
    console.log(mensaje);  
}  
mostrarAnuncio();  
console.log(mensaje);
```

Parámetros

Nota mental, evitar el uso de variables globales

Ámbito de una función

Ámbito de una función

¿Qué es el ámbito de una función?

Ámbito de una función

¿Os acordáis de let, const y var?

Ahora cobra más sentido

Ámbito de una función

```
1 var valor = "global";
2 function funcionlocal() {
3     var valor = "local";
4     return valor;
5 }
6 console.log(valor); // "global"
7 console.log(funcionLocal()); // "local"
8 console.log(valor); // "global"
```

Ámbito de una función

Es por esto por lo que se recomienda
encarecidamente dejar de usar var y usar
let y const

Formas de declarar funciones en Javascript

Declarar funciones

Funciones por declaración

Probablemente, la forma más popular de estas tres, y a la que estaremos acostumbrados si venimos de otros lenguajes de programación, es la primera, a la creación de funciones por declaración.

Declarar funciones

```
function saludar() {  
    return "Hola";  
}  
  
saludar(); // 'Hola'  
typeof saludar; // 'function'
```

Declarar funciones

Funciones por expresión

Sin embargo, en Javascript es muy habitual encontrarse códigos donde los programadores «guardan funciones» dentro de variables, para posteriormente «ejecutar dichas variables»

Declarar funciones

```
// El segundo "saludar" (nombre de la función) se suele omitir: es redundante
const saludar = function saludar() {
    return "Hola";
};

saludo(); // 'Hola'
```

Declarar funciones

Con este nuevo enfoque, estamos creando una función en el interior de una variable, lo que nos permitirá posteriormente ejecutar la variable (como si fuera una función). Observa que el nombre de la función (en este ejemplo: saludar) pasa a ser inútil, ya que si intentamos ejecutar saludar() nos dirá que no existe y si intentamos ejecutar saludo() funciona correctamente.

Declarar funciones

Funciones como objetos

Como curiosidad, debes saber que se pueden declarar funciones como si fueran objetos. Sin embargo, es un enfoque que no se suele utilizar en producción. Simplemente es interesante saberlo para darse cuenta que en Javascript todo pueden ser objetos:

Declarar funciones

```
const saludar = new Function("return 'Hola';");

saludar(); // 'Hola'
```

Declarar funciones

Funciones anónimas

Las funciones anónimas o funciones lambda son un tipo de funciones que se declaran sin nombre de función y se alojan en el interior de una variable y haciendo referencia a ella cada vez que queramos utilizarla:

Declarar funciones

```
// Función anónima "saludo"
const saludo = function () {
    return "Hola";
};

saludo; // f () { return 'Hola'; }
saludo(); // 'Hola'
```

Declarar funciones

Callbacks

A grandes rasgos, un callback (llamada hacia atrás) es pasar una función B por parámetro a una función A, de modo que la función A puede ejecutar esa función B de forma genérica desde su código, y nosotros podemos definirlas desde fuera de dicha función

Declarar funciones

```
// fB = Función B
const fB = function () {
    console.log("Función B ejecutada.");
};

// fA = Función A
const fA = function (callback) {
    callback();
};

fA(fB);
```

Declarar funciones

Funciones autoejecutables o IIFE

Pueden existir casos en los que necesites crear una función y ejecutarla sobre la marcha. En Javascript es muy sencillo crear funciones autoejecutables.

```
// Función autoejecutable
(function () {
    console.log("Hola!!");
})();
```

Declarar funciones

Funciones anidadas

Dentro del bloque de definición de una función pueden declararse a su vez otras funciones.

Si una función, devuelve otra función, para ejecutarla deberemos utilizar el doble ()() a no ser que se ejecute dentro de la primera

Declarar funciones

Funciones anidadas

```
1 function funcionExterna() {
2     var variableExterna = "Hola, soy una variable de la función externa";
3
4     function funcionInterna() {
5         var variableInterna = " y yo soy una variable de la función interna.";
6         console.log(variableExterna + variableInterna); // Accede a ambas variables
7     }
8
9     funcionInterna(); // Llamamos a la función interna dentro de la función externa
10 }
11
12 funcionExterna(); // Llamamos a la función externa
```

Declarar funciones

Funciones anidadas devolviendo la interna

```
1 function funcionExterna() {
2     var variableExterna = "Hola, soy una variable de la función externa";
3
4     function funcionInterna() {
5         var variableInterna = "y yo soy una variable de la función interna.";
6         console.log(variableExterna + variableInterna); // Accede a ambas variables
7     }
8
9     return funcionInterna; // Llamamos a la función interna dentro de la función externa
10 }
11
12 funcionExterna(); // Llamamos a la función externa
```

¿Qué es un Clousure?

¿Qué es un Clousure?

Los closures pueden ser utilizados para crear estructuras de datos privadas y controlar el acceso a ciertas variables, manteniendo una interfaz limpia y bien definida.

¿Qué es un Clousure?

Se crean cada vez que una función es creada

¿Qué es un Clousure?

A veces se usan sin saberlo.

Cada vez que una función cualquiera accede a una variable fuera de su contexto, estás usando un closure

¿Qué es un Clousure?

```
// Ejemplo de Closure con un contador sin usar clases
function crearContador() {
    let contador = 0; // Variable que mantiene el estado del contador

    return {
        incrementar: function() {
            contador++;
            console.log(contador);
        },
        decrementar: function() {
            contador--;
            console.log(contador);
        },
        obtenerValor: function() {
            return contador;
        }
    };
}
```

¿Qué es un Clousure?

- La función crearContador define una variable contador y retorna un objeto con tres métodos: incrementar, decrementar, y obtenerValor.
- contador es privada para el ámbito de crearContador y solo puede ser accedida y modificada a través de estos métodos.
- Al llamar a crearContador, se crea un nuevo closure que mantiene su propio estado privado para contador.
- Puedes usar miContador para interactuar con este contador, incrementando, decrementando y obteniendo su valor actual.

<Despedida>

Email

bienvenidosaez@gmail.com

Instagram

@bienvenidosaez

Youtube

youtube.com/bienvenidosaez