«Talento Tech»

Desarrollo Web 2

Clase 04

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Clase N° 4 |

Temario:

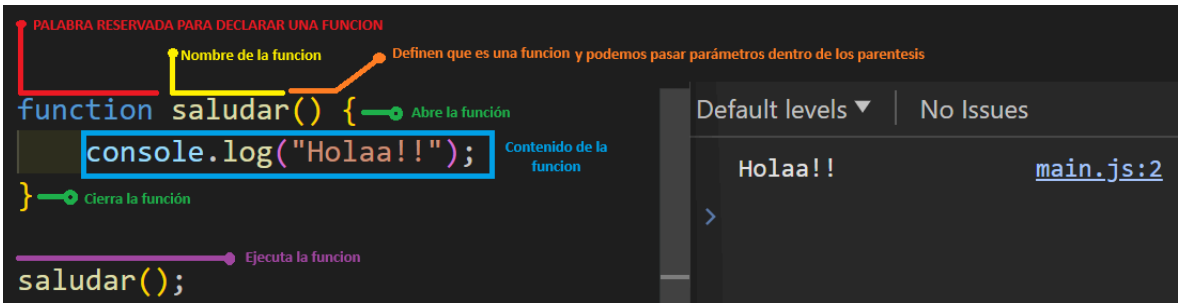
- Funciones
- Parámetro
- Arrow Function
- Return
- Scope de variables

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Funciones

Las funciones son bloques de código que se definen una vez y se pueden llamar en cualquier momento para realizar una tarea específica. Es una parte fundamental del lenguaje y permiten que las re-utilicemos, lo que hace que el código sea más fácil de entender y mantener.

Vamos a crear una función saludar():



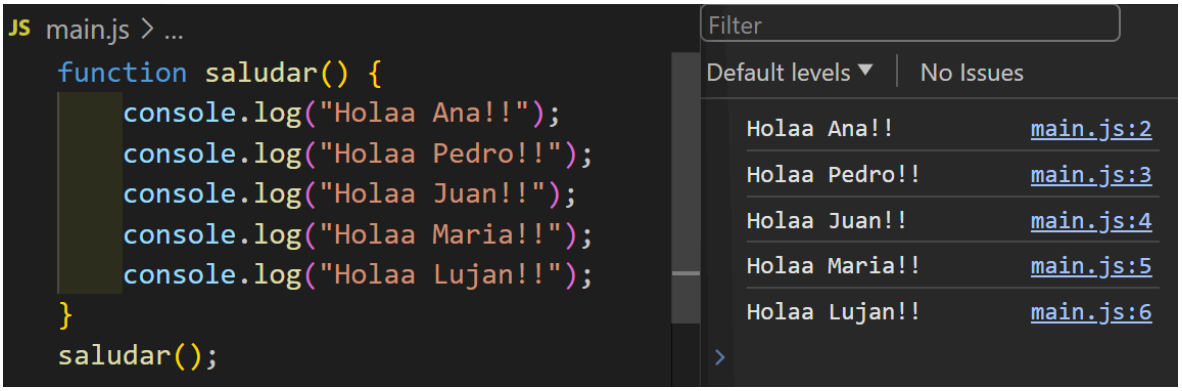
Funciones con parámetros

Los parámetros en una función de JavaScript son variables que se utilizan para recibir valores de entrada cuando se invoca la función. Estos son los que nos permiten que el código sea reutilizable, cuando la función recibe los parámetros sólo los utiliza dentro de sí misma.

Por ejemplo: Si quiero una función que salude a todos mis amigos/as, lo más sencillo es armar una función que salude a todos de una vez.

```
function saludar(){
    console.log("Hola Ana");
    console.log("Hola Pedro");
    console.log("Hola Juan");
    console.log("Hola Maria");
    console.log("Hola Lujan")
}
```

Respuesta:
Hola Ana
Hola Pedro
Hola Juan
Hola Lujan



Pero qué pasaría si necesito saludar a cada uno por separado. Lo podemos pensar así:

```
function saludar(nombre){
    console.log(`Hola ${nombre}`)
}
saludar("Ana");
let mejorAmiga = "Carla";
saludar(mejorAmiga);
```

Respuesta:
Hola Ana
Hola Carla

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Lo que hicimos fue reutilizar una función simple de saludar pasando como parámetro una variable “nombre”. Ésta se reemplazará por el valor que quiera utilizar..

Otro ejemplo:

Tambien lo podriamos usar para hacer una función que sume dos números, sabemos que queremos sumar diferentes números, por lo tanto, lo que tenemos que pasar como parametros son las dos variables que se van a sumar:

```
function suma (a,b){
    let resultado = a + b;
    console.log(resultado);
}
suma(2.4, 5);
suma(20, 1);
```

Resultado:
7.4
21

Función arrow

Más conocidas como función flecha son una manera más corta y simple de hacer funciones en JavaScript. Son populares y útiles, especialmente para funciones cortas. Ofrecen una forma más compacta de escribir código que las funciones tradicionales.

Vamos a ver la diferencia entre una función tradicional y una función flecha:

TRADICIONAL	FUNCION FLECHA

`function saludar(nombre) {`
`  console.log("Holaa " + nombre +"!!" );`
`}`
 	
`saludar("Ana");`
	`let saludar = (nombre) => {`
	`  console.log("Holaa " + nombre +"!!" );`
	`}`
	
	`saludar("Ana");`

- **Tradicional:**

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

- **Función flecha:**

```
let saludar = (nombre) => {
    console.log("Hola " + nombre + "!!");
}
saludar("Ana");
```

La única diferencia entre las dos es que en la Función Flecha el nombre de la función pasa a ser una variable y la palabra reservada “function” pasa a ser representada por el símbolo “ =>”.

Return

En **JavaScript**, return es una palabra clave que se utiliza dentro de una función para especificar el valor que la función debe devolver. Cuando una función encuentra una declaración de return, se detiene en ese punto y devuelve el valor indicado. El uso del return es importante para obtener resultados de una función y utilizarlos en otras partes del código. Puede devolver cualquier tipo de dato, incluyendo números, cadenas de texto, objetos, arreglos o incluso otras funciones.Si no hay un return, la función devuelve 'undefined' sin decir nada.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Cuando hablamos de que la función va a “devolver” un resultado nos referimos a que la función va terminar con un valor como resultado pero dependerá de nosotros guardarlo en una variable o mostrarlo por pantalla.

FUINCION SIN RETURN

```
function sumar(a,b) {
    let resultado = a + b;
    console.log(resultado);
}

sumar(3,5);
```

FUNCION CON RETURN

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

let resultado = sumarConReturn(3, 5);
console.log(resultado);
```

- **Función sin return:**

```
function sumar(a,b){
    let resultado = a + b;
    console.log(resultado);
}
sumar(3,5);
```

- **Función con return:**

```
function sumarConReturn(a,b){
    return a + b;
}
let resultado = sumarConReturn(3, 5);
console.log(resultado)
```

En este ejemplo, la función **sumarConReturn** toma dos parámetros (a y b), los suma utilizando el operador +, y devuelve el resultado con la sentencia **return a + b**;. Cuando se llama a la función sumarConReturn(3, 5), retorna la suma de 3 y 5 (que es 8), y ese valor se almacena en la variable resultado.

Las funciones con return son importantes porque te dan el resultado de una tarea para que puedas usarlo donde quieras en tu programa. Esto hace que puedas reutilizar ese resultado en distintos lugares, lo que hace tu código más flexible y fácil de usar.

Scope de variables

El "**scope**" o alcance de las variables en **JavaScript** se refiere a dónde y cuándo una variable es accesible dentro de un programa. Determina las partes del código donde una variable es visible y puede ser utilizada.
En **JavaScript**, hay diferentes tipos de "**scopes**":

Scope Global: Las variables declaradas fuera de cualquier función se consideran variables globales y están disponibles en todo el código. Esto significa que se pueden acceder a ellas desde cualquier parte del código, ya sea dentro de funciones, bloques de código o incluso en otros archivos si se han incluido.
Las variables globales son accesibles desde cualquier lugar y pueden ser modificadas y accedidas en cualquier momento.

Scope Local: Las variables declaradas dentro de una función sólo son accesibles dentro de esa función, lo que se conoce como alcance local. Esto significa que solo pueden ser accedidas desde dentro de la función en la que se declararon. Estas variables son conocidas como variables locales y cada función crea su propio scope local, lo que significa que las variables declaradas en una función no son accesibles desde fuera de esa función.

```
// Variable Global
let variableGlobal = "Soy una Variable Global"
const miFuncion = () =>{
    // Variable Local
    let variableLocal = "Soy una Variable Local"
    console.log(variableGlobal); // Acceso de Variable Global dentro de la función
    console.log(variableLocal); // Acceso de Variable Local dentro de la función
}
console.log(variableGlobal); // Acceso de Variable Global fuera de la función
// console.log(variableLocal); // Esto producira un error, porque la variable local no esta disponible fuera de la función
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

📝Desafío #4

Crea una función llamada **sumarProductos()** que acepte dos parámetros: **precioUnitario** y **cantidadDeseada**.

La función debe **calcular y devolver el total gastado en un producto multiplicando el precio por la cantidad comprada**.


Buenos Aires
aprende
Buenos
Aires
Ciudad