Material descargable

egg

# Teoría **JavaScript**

## Funciones en Javascript

Una función, en JavaScript es, al igual que en otro lenguajes, una serie de instrucciones que englobamos dentro de un mismo proceso. Este proceso se podrá luego ejecutar desde cualquier otro sitio con solo llamarlo.

Algunos de los beneficios de hacer uso de funciones son:

- **Reutilización de código:** Las funciones permiten encapsular un bloque de código y ejecutarlo en múltiples lugares del programa. Esto promueve la reutilización del código, lo que facilita el mantenimiento y reduce la duplicación.
- **Abstracción:** Las funciones permiten abstraer detalles de implementación. Puedes definir una función con un propósito específico y utilizarla sin preocuparte por cómo se realiza internamente.
- **Organización del código:** El uso de funciones mejora la estructura y organización del código. Divide el código en bloques más pequeños y manejables, facilitando la comprensión y el mantenimiento del programa.
- **Parámetros y retorno:** Las funciones permiten pasar valores como parámetros y devolver resultados. Esto facilita la creación de código modular y facilita la comunicación entre diferentes partes del programa.
- **Manejo de eventos y callbacks:** Las funciones son esenciales para manejar eventos y proporcionar callbacks. Por ejemplo, puedes definir funciones que se ejecuten cuando un botón es clicado o cuando se completa una operación asincrónica.
- **Recursividad:** Las funciones en JavaScript pueden llamarse a sí mismas, lo que permite implementar la recursividad. Esto es útil en situaciones donde una tarea se puede dividir en subproblemas más pequeños.

Por ejemplo, en una página web puede haber una función para cambiar el color del fondo y desde cualquier punto de la página podríamos llamarla para que nos cambie el color cuando lo deseemos.

La sintaxis de una función en JavaScript es la siguiente:

```
function nombrefuncion (){
 instrucciones de la función
 ...
}
```

Primero se escribe la palabra function, reservada para este uso. Seguidamente, se escribe el nombre de la función, que como los nombres de variables puede tener números, letras y algún carácter adicional como en guion bajo y los paréntesis donde irán nuestros parámetros. A continuación se colocan entre llaves las distintas instrucciones de la función. Las llaves en el caso de las funciones no son opcionales, además es útil colocarlas siempre como se ve en el ejemplo, para que se reconozca fácilmente la estructura de instrucciones que engloba la función.

Un ejemplo de función que escribe hola mundo en la consola sería:

```
function holaMundo(){
console.log('Hola Mundo');
}
```

Después, para llamar a esta función, hay que invocarla mediante su nombre holaMundo();

## Dónde colocamos las funciones Javascript

En principio, podemos colocar las funciones en cualquier parte de la página, siempre entre etiquetas **<script>**. No obstante, existe una limitación a la hora de colocarla con relación a los lugares desde donde se la llame. Usualmente, lo más fácil es colocar la función antes de cualquier llamada a la misma y así seguro que nunca nos equivocaremos.

Existen dos opciones posibles para colocar el código de una función:

**a) Colocar la función en el mismo bloque de script:** En concreto, la función se puede definir en el bloque <script> donde esté la llamada a la función, aunque es

indiferente si la llamada se encuentra antes o después del código de la función, dentro del mismo bloque <script>.

```
<script>
miFuncion();
function miFuncion(){
 //hago algo...
 console.log("Esto va bien");
}
</script>
```

**b) Colocar la función en otro bloque de script:** También es válido que la función se encuentre en un bloque <SCRIPT> anterior al bloque donde está la llamada.

```
<html>
<head>
 <title>MI PÁGINA</title>
<script>
function miFuncion(){
 //hago algo...
 console.log("Esto va bien");
}
</script>
</head>
<body>
<script>
miFuncion()
</script>
```

```
</body>

</html>
```

## Parámetros de las Funciones

Los parámetros se usan para mandar valores a las funciones. Una función trabajará con los parámetros para realizar las acciones. Por decirlo de otra manera, **los parámetros son los valores de entrada que recibe una función.**

Los parámetros en JavaScript, son iguales que en Java, la única diferencia es que, a diferencia de Java, no tenemos que especificar el tipo de dato que recibe la función, sino que va a ser el tipo de dato que enviemos como parámetro.

```
function escribirBienvenida(nombre){

console.log('Hola ' + nombre);

}

escribirBienvenida('Agustin');

// O podemos hacerlo con una variable

let nombre = 'Agustin';

escribirBienvenida(nombre);
```

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

const resultado = sumar(2, 3);
console.log(resultado); // Imprime 5
```

En este ejemplo, la función sumar recibe dos parámetros, a y b. Cuando llamamos a la función sumar(2, 3), los valores 2 y 3 se asignan a los parámetros

a y b respectivamente. Dentro de la función, podemos utilizar esos parámetros para realizar alguna operación, en este caso, sumar los dos valores.

El retorno de una función es el valor que la función "devuelve" al ser ejecutada. Utilizamos la palabra clave return seguida del valor que queremos retornar. El retorno de una función nos permite obtener un resultado y utilizarlo en otras partes de nuestro programa.

Los parámetros y retornos de funciones nos permiten hacer que nuestras funciones sean más flexibles y reutilizables. Podemos pasar diferentes valores como argumentos a los parámetros de una función y recibir diferentes resultados a través de los retornos. Esto nos permite modularizar nuestro código y realizar operaciones específicas en diferentes contextos.

## Devolver valores en Funciones

**Las funciones en Javascript también pueden retornar valores.** De hecho, ésta es una de las utilidades más esenciales de las funciones.

Veamos un ejemplo de función que calcula la media de dos números. La función recibirá los  dos números y retornará el valor de la media.

```
function media(valor1,valor2){
 let resultado;
 resultado = (valor1 + valor2) / 2;
 return resultado;
}
```

Para especificar el valor que retornará la función se utiliza la palabra *return* seguida del valor que se desea devolver. En este caso se devuelve el contenido de la variable resultado, que contiene la media calculada de los dos números.

Pero, cómo podemos recibir un dato que devuelve una función. Cuando una función devuelve un valor, simplemente se sustituye la llamada a la función por ese valor que devuelve. Así pues, para almacenar un valor de devolución de una

función, tenemos que asignar la llamada a esa función como contenido en una variable, y eso lo haríamos con el operador de asignación =.

```
let miMedia

miMedia = media(12,8);

console.log(miMedia);
```

## Funciones Flecha

Hay otra sintaxis muy simple y concisa para crear funciones, que a menudo es mejor que las expresiones de funciones.

Se llama *"funciones de flecha"*, porque se ve así:

```
let func = (arg1, arg2, ..., argN) => expresion
```

Esto crea una función func (nombre de la función) que acepta parámetros arg1..argN, luego evalúa la expresion de la derecha con su uso y devuelve su resultado.

En otras palabras, es la versión más corta de:

```
let func = function(arg1, arg2, ..., argN) {

return expresion;

};
```

Veamos un ejemplo completo:

```
let sum = (a, b) => a + b;

/* Esta función de flecha es una forma más corta de:

let sum = function(a, b) {

 return a + b;

};

*/
```

```
console.log(sum(1, 2)); // 3
```

Como puedes ver (a, b) => a + b significa una función que acepta dos parámetros llamados a y b. Tras la ejecución, evalúa la expresión a + b devuelve el resultado.

Si solo tenemos un argumento, se pueden omitir paréntesis alrededor de los parámetros, lo que lo hace aún más corto.

```
let double = n => n * 2;
```

Esto sería equivalente a escribir:

```
let double = function(n) { return n * 2 }

console.log(double(3)); //6
```

Si no hay parámetros, los paréntesis estarán vacíos (pero deben estar presentes):

```
let saludar = () => console.log("Hola!");

saludar();
```

Este tema lo hemos visto para que sepan que existe y si lo encuentran en alguna ocasión sepan identificarlo y cómo funciona. Recomendamos que antes de pasar a las funciones flecha, aprendamos a trabajar las funciones “normales”.

## Funciones Anónimas

Una función anónima es una función que no tiene un nombre específico asociado. Estas funciones se crean de forma inline (en línea) y pueden asignarse a variables o pasarse como argumentos a otras funciones. A diferencia de las funciones con nombre, las funciones anónimas no tienen un identificador que las distinga.

Las funciones anónimas son útiles en situaciones en las que solo necesitas la función en un lugar específico y no planeas reutilizarla en otras partes del código. También son comunes en el manejo de eventos, devoluciones de llamada (callbacks) y ciertas técnicas de programación asincrónica.

Hay varias maneras de definir funciones anónimas en JavaScript. Aquí tienes algunos ejemplos:

- **Expresión de función anónima:**

```
let suma = function(a, b) {
  return a + b;
};
```

- **Función anónima como argumento:**

```
setTimeout(function() {
  console.log("¡Han pasado 2 segundos!");
}, 2000);
```

- **IIFE (Immediately Invoked Function Expression):**

```
(function() {
  console.log("Esta función es invocada inmediatamente");
})();
```

Es importante tener en cuenta que las funciones anónimas pueden dificultar la depuración y el mantenimiento del código si se utilizan en exceso, ya que carecen de un nombre descriptivo. En la actualidad, con la introducción de las funciones de flecha (arrow functions) en ECMAScript 6 (ES6), se proporciona una sintaxis más concisa para funciones anónimas, lo que ha llevado a un uso más extendido de estas funciones en el desarrollo moderno de JavaScript.

# Template strings

Las Template Strings (o plantilla de cadena de texto en castellano) son una manera de declarar cadenas de texto que permite la **interpolación** de expresiones, es decir, incrustar expresiones dentro de una cadena de texto.

Las cadenas de texto se pueden declarar con comillas simples o dobles, pero los template strings se declaran exclusivamente con comillas invertidas.

## Concatenación de Variables

En un programa realizado en JavaScript, y en cualquier lenguaje de programación en general, es normal crear cadenas en las que tenemos que juntar cadenas con los valores tomados desde las variables.

```
var sitioWeb = "DesarrolloWeb.com";
```

```
var mensaje = 'Bienvenido a ' + sitioWeb;
```

Eso es muy fácil de leer, pero a medida que el código se complica y en una cadena tenemos que concatenar el contenido de varias variables, el código comienza a ser más enrevesado.

```
var nombre = 'Miguel Angel';

var apellidos = 'Alvarez';

var profesion = 'desarrollador';

var perfil = ' ' + nombre + ' ' + apellidos + ' es ' + profesion;
```

Quizás estás acostumbrado a ver esto así. El código está bien y no tiene ningún problema, pero podría ser mucho más elegante si usas los template strings.

## Crear un Template String

Para crear un template string simplemente tienes que usar un carácter que se usa poco, como apertura y cierre de la cadena. Es el símbolo del acento grave. (`)

```
var cadena = `Esto es un template String`;
```

Sintaxis Básica:

```
Unset
const nombre = "Usuario";
const saludo = `¡Hola, ${nombre}!`;
console.log(saludo);
```

En el ejemplo anterior, las comillas invertidas permiten incorporar variables directamente dentro de la cadena de texto utilizando la sintaxis ${variable}.

## Usos de los Template Strings

Los template strings tienen varias características interesantes que, cómo decíamos, facilitan la sintaxis. Veremos a continuación algunos de ellos con código de ejemplo. Lo más interesante es el caso de la concatenación que genera un código poco claro hasta el momento. Echa un vistazo al código siguiente que haría lo mismo que el que hemos visto anteriormente del perfil.

```
var nombre = 'Miguel Angel';

var apellidos = 'Alvarez' ;

var profesion = 'desarrollador';

var perfil = `<b>${nombre} ${apellidos}</b> es ${profesion}`;
```

Como puedes comprobar, dentro de un template string es posible colocar expresiones encerradas entre llaves y precediendo de un símbolo "$". Algo como ${ expresion }.

En las expresiones podemos colocar código que queramos volcar, dentro de la cadena. Las usamos generalmente para colocar valores de variables, pero también servirían para colocar operaciones matemáticas, por ejemplo.

```
var suma = `45 + 832 = ${45 + 832}`;
```

O bien algo como esto:

```
var operando1 = 7;

var operando2 = 98;

var multiplicacion = `La multiplicación entre ${operando1} y ${operando2} equivale a ${operando1 * operando2}`;
```

## Interpolación

A continuación se detallan los usos más comunes de la interpolación:

- Interpolación de expresiones: Puedes insertar variables y expresiones dentro de una cadena de texto utilizando la sintaxis **${expression}**, donde expression puede ser cualquier expresión JavaScript válida.

```
let nombre = "Mundo";
console.log(`Hola, ${nombre}!`);  // "Hola Mundo!"
```

- Interpolación para evaluar:

```
let condicion = true;
let texto = `Podrías decirme la hora? ${condicion ? "Claro!" : "Imposible"}`
console.log(texto);  // "Podrías decirme la hora? Claro!"
```

- Función de interpolación: Puedes definir funciones que dependan de parámetros a interpolar y que retornen una cadena de texto “armada”.

```
function saludar(nombre) {
return `Hola, ${nombre}! Cómo estás?`
};
saludar("Juan") // devuelve "Hola Juan! Cómo estás?"
saludar("María") // devuelve "Hola María! Cómo estás?"
```

## Conclusión

Los Template Strings en JavaScript ofrecen una forma más elegante y poderosa de trabajar con cadenas de texto, permitiendo una sintaxis más clara, mayor flexibilidad y facilidad de mantenimiento en comparación con métodos tradicionales. Su versatilidad los hace una herramienta esencial en el desarrollo moderno de JavaScript.