

# JavaScript desde cero

Módulo 4

# Funciones

# Funciones

## Concepto

En JavaScript, una **función** es un **bloque de código reutilizable que realiza una tarea específica**. Puede aceptar entradas, llamadas argumentos, y devuelve un resultado.

Es la forma de organizar y estructurar el código para poder usar la misma lógica en diferentes partes de tu programa.



## Definición

Para definir una función se usa la palabra reservada **function**, se le da un nombre (en el ejemplo de la derecha, `saludar`), se abre y cierra paréntesis `()`.

Luego, se abren **llaves de bloque** `{}`, se escribe **el cuerpo** (el código que queremos ejecutar cuando sea invocada) y, por último, se cierra la llave de bloque `}`.

```
function saludar() {  
  
    alert("Hola! Soy una funcion")  
  
}
```



## Características de una función

- Debe expresar una **acción**.
- Debe comenzar con un **verbo**.
- Debe expresarse en **modo imperativo** (o sea, dando una orden).
- Puede ser de **nombre simple**, o **compuesto** (en formato *camelCase*).



```
function saludarUsuario() {  
  console.log("Bienvenid@, ", username);  
}
```

### Nombres para una función:



#### Correctos:

saludar() o saludarUsuario().



#### Incorrectos:

saludo() o saludoUsuario().

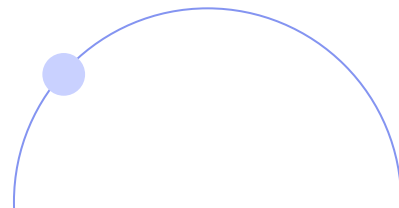


## Ejecución

A diferencia del resto del código que escribimos hasta aquí, **las funciones no se ejecutan si no se las llama**.

Para llamarlas, simplemente **escribimos su nombre junto a los paréntesis**, en alguna parte del archivo JS, y esta será invocada.

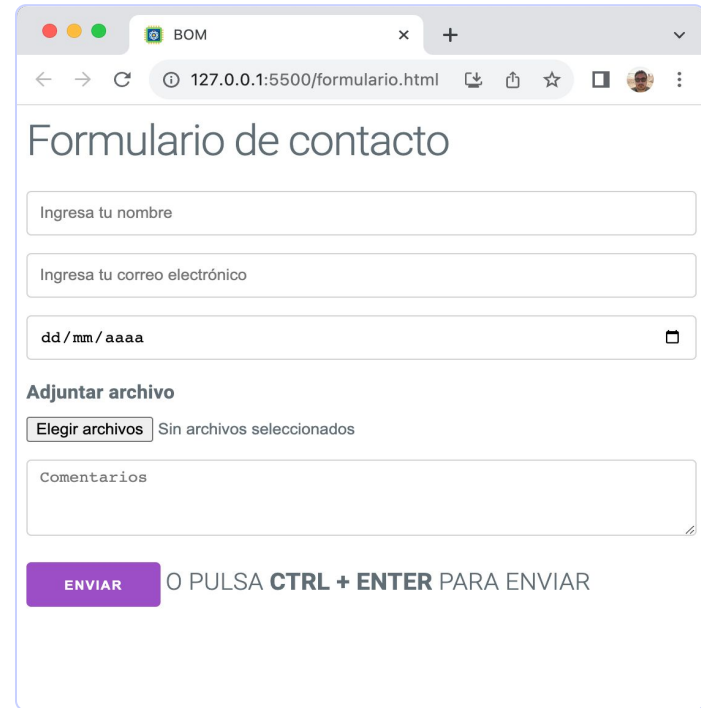
```
function saludarUsuario() {  
    console.log("Bienvenid@,", username);  
}  
  
saludarUsuario();
```



## Ventajas

Si, por ejemplo, tenemos un formulario web con el botón **ENVIAR**, y también queremos darle la opción al usuario que pulse **CTRL + ENTER** para enviar el formulario, **solo debemos escribir una única función**, que recopilará el texto de los campos y enviará los datos al servidor donde se deben alojar.

De igual forma, cuando tenemos una barra de navegación superior para acceder a otras secciones de un sitio, y los puntos de menú al pie de página que también nos permiten ir a otras secciones: **el código será uno sólo y se podrá reutilizar en ambos lugares.**



The screenshot shows a web browser window with the address bar displaying "127.0.0.1:5500/formulario.html". The page title is "Formulario de contacto". The form contains the following fields:

- Text input: "Ingresa tu nombre"
- Text input: "Ingresa tu correo electrónico"
- Date input: "dd/mm/aaaa" with a calendar icon
- Section: "Adjuntar archivo" with a button "Elegir archivos" and the text "Sin archivos seleccionados"
- Text area: "Comentarios"

At the bottom of the form, there is a purple button labeled "ENVIAR" and a text instruction: "O PULSA **CTRL + ENTER** PARA ENVIAR".

# Funciones con parámetro(s)



## Funciones con parámetro(s)

A todas las funciones se les puede pasar uno o más parámetros. Estos no son más que **variables** que existirán dentro de dicha función, y que acarrean valores que dicha función deberá manipular/utilizar internamente.

Así, podemos **armar funciones dinámicas** que, siguiendo la lógica que queramos, pueden **generar distintos resultados al recibir diferentes valores**.

```
function saludarUsuario(username) {  
    console.log("Bienvenid@,", username);  
}  
  
saludarUsuario("July"); //imprime: Bienvenid@, July.  
  
saludarUsuario("Nico"); //imprime: Bienvenid@, Nico.  
  
saludarUsuario("Greta"); //imprime: Bienvenid@, Greta.
```



**No existe un límite de parámetros para definir en una función:** puede ser uno, dos, cinco, etcétera.

Pero sí deberíamos realizar un mínimo análisis para no exagerar en el total de datos que debemos enviarle como parámetros.

Si son demasiados, seguramente se puedan re-definir de otra manera, tal vez atomizando en varias funciones con menos parámetros.



## Definir parámetros descriptivos

La definición del nombre de los parámetros, debe tener la misma lógica que cuando creamos variables: poseer **nombres descriptivos, que guíen al desarrollador** (nosotros u otro miembro

del equipo), para que **sepa al llamar a esta función, qué valor(es) se espera(n) pasarle como parámetro(s)**.

```
function calcularIVA(importe, valorDeIVA) {  
    console.log("Importe final: ", importe * valorDeIVA)  
}  
  
calcularIVA(1500, 1.21);
```



## Controlar posibles valores no definidos

Es muy probable que en algún momento específico, se invoque una función y se olvide de pasarle un parámetro o, el parámetro que le indicamos, proviene de una variable que está vacía (`null`, `undefined`).


Ante estas situaciones, se puede **establecer, en el parámetro de la función, un valor predeterminado, por si el argumento que recibe no posee un dato válido.**

```
function saludarUsuario(username = "invitada/o") {  
    console.log("Bienvenid@,", username);  
}  
  
saludarUsuario("Greta"); //imprime: Bienvenid@, Greta  
saludarUsuario();       //imprime: Bienvenid@, invitada/o
```

# Parámetro(s) versus Argumento(s)


## Parámetro

Es una especie de **variable** que se crea solamente dentro de los paréntesis en la declaración de función (es un término para el **momento de la declaración**).



## Argumento

Es el **valor** que es pasado a la función cuando esta es llamada (es el término utilizado en el **momento en que se llama a la función** y se le pasa uno o más parámetros).



# Funciones con retorno

# Funciones con retorno

Las funciones con retorno permiten **realizar una operación dentro de la función en sí, y retornar un posible valor** o resultado, a través de la misma función.

Para poder retornar un resultado, se debe utilizar la **palabra reservada return**, seguido del valor a retornar.

```
function retornarPrecioConIVA(importe, valorDeIVA) {  
    return importe * valorDeIVA;  
}
```



## Características

Las funciones con retorno **pueden, o no, recibir parámetros** y, cuando son invocadas, el valor que retorne puede ser capturado en una variable, constante, o directamente mostrado en un elemento HTML con el que nos enlazamos desde JS.

Cuando se crean funciones con retorno, es importante también que **su nombre exprese el tipo de función que es**.





## La palabra reservada *return*

Oficia como **punto de interrupción dentro de la función con retorno**, por lo tanto, **todo código que se escriba después de la palabra return, no se ejecutará nunca**.

Cumple el mismo rol de interrupción, que la palabra reservada **break**, que vimos oportunamente en ciclos de iteración y con la cláusula **switch**.

```
function retornarPrecioConIVA(importe, valorDeIVA) {  
    return importe * valorDeIVA;  
  
    console.log("Este mensaje nunca se verá");  
}
```

## Ventaja: reutilización del código

El uso de las funciones con retorno, permite **abstraernos al máximo de las operaciones y lógica de una aplicación de *software*.**

Mientras mejor pensamos a las funciones, con el uso de return, **más reutilizable será el código**, y menos tendremos que escribir cuando una aplicación crezca de forma considerable en funcionalidades y complejidad.



## Ejemplos

Veamos en el siguiente slide dos ejemplos de cómo aprovechar las funciones con retorno:

- Almacenar el resultado en una **variable** e imprimirlo en la consola JS.
- También, imprimir el resultado directamente en un **tag HTML**, con el que nos enlazamos previamente usando el método llamado **querySelector()**.

```
function retornarPrecioConIVA(importe, valorDeIVA) {  
    return importe * valorDeIVA;  
}  
  
//EJEMPLO ALMACENANDO EL VALOR DE RETORNO EN UNA VARIABLE  
let resultado = retornarPrecioConIVA(1500, 1.21);  
console.log("Precio final del producto:", resultado)  
  
//EJEMPLO IMPRIMIENDO EL VALOR DE RETORNO EN UN TAG HTML  
const spanTotal = document.querySelector("span#total");  
spanTotal.textContent = "Total: $ " + retornarPrecioConIVA(1500, 1.21);
```

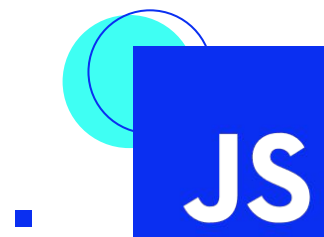
# Ámbito de la variable (*scope*)

# Variables globales y locales

En JS existen **variables del tipo global**, que se declaran al inicio de toda aplicación web, y que se pueden utilizar libremente, dentro de toda la aplicación.

También existen **variables del tipo local**, que sólo se pueden utilizar dentro de, por ejemplo, una función, o un método, en la aplicación web.

A este tipo de usos se lo denomina: **'scope'**

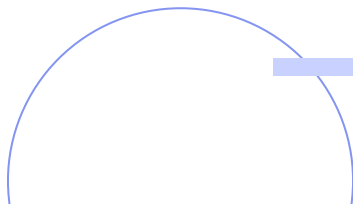


Cuando se crea una **variable de ámbito local**, **esta no podrá ser utilizada por fuera del ámbito donde fue creada**; en este ejemplo, dentro de la función `saludarUsuario()`.

Si se intenta acceder a su valor, por fuera de dicho ámbito, obtendremos un **error** de que la variable no ha sido definida.

```
funcion saludarUsuario() {  
  let usuario = "Invitada/o";  
  console.log("Bienvenid@", usuario);  
}  
  
console.log("Bienvenid@", usuario);
```

✖ ▶ Uncaught ReferenceError: usuario is not defined




**Pero, cuando una aplicación de *software* crece considerablemente en funcionalidades, es muy factible que se repitan nombres de las variables/constantes.**

Algunas, tal vez, estén declaradas de forma global y otras declaradas de forma local.

Lo bueno de JS, es que permite repetir el nombre de variables dentro de un scope local, sin que se afecte el valor de una misma variable declarada de forma global.

Veamos un ejemplo en el siguiente slide.






```
let nombre = "Greta";

console.log("Bienvenid@,", nombre); //imprime: Bienvenid@, Greta

funcion saludarUsuario() {
  let nombre = "Invitada/o";
  console.log("Bienvenid@,", nombre); //imprime: Bienvenid@, Invitada/o
}

console.log("Bienvenid@,", nombre); //imprime: Bienvenid@, Greta
```





# Revisión

- Repasar el concepto de *función*.
- Estudiar cómo **trabajar con parámetros** desde una **función**.
- Entender cómo definir correctamente el **ámbito de una variable**.
- Trabajar en el **Proyecto integrador**.
- Realizar las preguntas necesarias a la/el docente antes de continuar.



**¡Sigamos  
trabajando!**