

# Funciones

Las funciones son bloques de código que tienen la característica de ser llamados o invocados.

## Sintaxis

La siguiente sería una sintaxis de una función que por ejemplo contiene una alerta

```
<script>
function miFuncion(){
  alert('Hola estoy dentro de una función')
}
</script>
```

Si levantamos esto en nuestro navegador, básicamente no obtendremos nada, porque esta función no ha sido invocada. Para invocar debemos hacer lo siguiente,

```
<script>
function miFuncion(){
  alert('Hola estoy dentro de una función')
}

miFuncion()
</script>
```

De esta manera, estamos llamando a nuestra función. Si queremos hacer un llamado dos veces, haremos lo siguiente,

```

<script>

function miFuncion(){

alert('Hola estoy dentro de una función')

}

miFuncion()
miFuncion()

</script>

```

El resultado, será que la siguiente alerta aparecerá dos veces,

Esta página dice  
 Hola estoy dentro de una función

Aceptar

## Funciones + Return + Parámetros

Por ejemplo, en el siguiente caso hemos generado una función que tomará parámetros, es decir pasamos a esta función determinados datos,

```

<p id="caja"></p>

<script>
function miOperacion(p1, p2) {
  return p1 * p2;
}
document.getElementById("caja").innerHTML = miOperacion(4, 3);
</script>

```

En este caso vemos cómo return nos permite retornar un valor, y de esta forma también podemos ir variando los datos que pasamos en la misma función. Es decir una misma función nos sirve para todo.

Por ejemplo, podríamos volver a utilizarla, de la siguiente forma,

```

<p id="cajaA"></p>
<p id="cajaB"></p>

<script>
function miOperacion(p1, p2) {
  return p1 * p2;
}
document.getElementById("cajaA").innerHTML = miOperacion(4, 3);
document.getElementById("cajaB").innerHTML = miOperacion(2, 18);
</script>

```

En el caso anterior, utilizamos la misma función, pero obtenemos dos resultados distintos



Es decir,

```

function ejemplo(parametro1, parametro2, parametro3)
{

//lo que hace a función

}

```

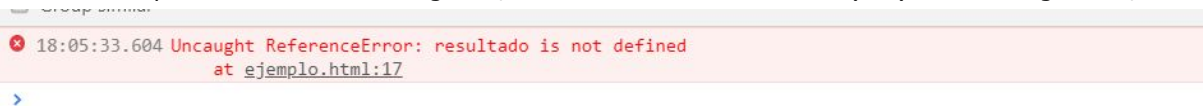
## Ámbito de las variables

Algo importante a tener en cuenta, es qué las variables pueden ser locales o globales. Si por ejemplo yo hago esto,

```
<script>
function miOperacion() {
var resultado = 30;

}
alert(resultado)
</script>
```

El resultado que se verá en mi navegador, será un error en la consola (f12) como el siguiente,



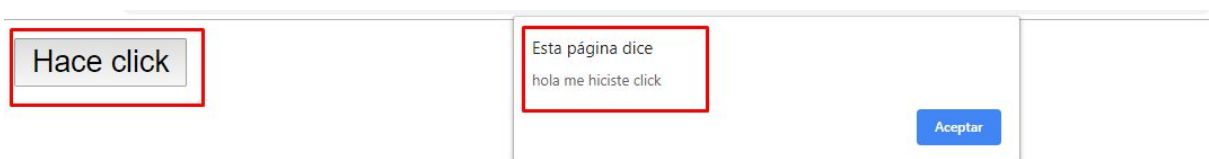
Esto es que básicamente la variable resultado está dentro del ámbito de **miOperación()** y no puedo utilizarla por fuera.

## Eventos

Los eventos son algo qué les pasa a los elementos del HTML. Por ejemplo,

```
<button onclick="alert('hola me hiciste click')"> Hace click </button>
```

El resultado en el navegador será el siguiente,



De esta forma, hemos dicho que al hacerle clic a ese botón disparará una alerta. Tenemos diferentes tipos de eventos,

```
<button onclick="alert('hola me hiciste click')"> Hace click </button>
<button onmouseover="alert('hola me pasaste el mouse por encima')"> Acerca el cursor </button>
<button onmouseout="alert('Alejá el cursor')"> Aleja el cursor </button>
<button onmousemove="alert('Mové el cursor del mouse sobre mí')"> Mové el cursor </button>
```

Por otro lado, también podemos usar eventos relacionados con el teclado, por caso, generaremos una caja de texto (input) y haremos lo siguiente,

```
<input type="text" onchange="alert('haz hecho un cambio en la caja')">
<input type="text" onblur="alert('haz dejado de hacer foco en la caja')">
<input type="text" onfocus="alert('haz hecho foco en la caja')">
<input type="text" onkeypress="alert('haz presionado una tecla dentro de la caja')">
<input type="text" onkeyup="alert('haz dejado de presionar una tecla dentro la caja')">
```

También a los eventos se los puede llamar directamente desde el código,

```
<button> Haceme click </button>

<script>

document.querySelector('button').onclick = alerta;

</script>
```

En este caso, estamos esperando a una función alerta(),

```
<button> Haceme click </button>

<script>

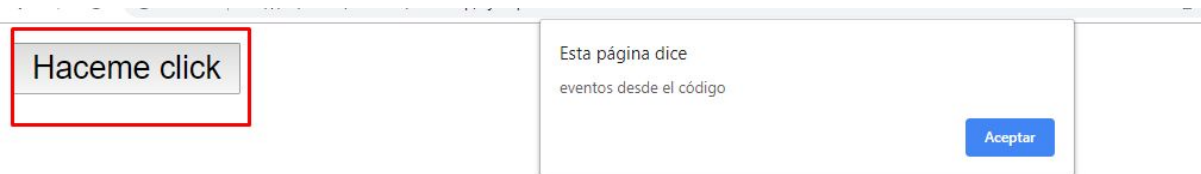
document.querySelector('button').onclick = alerta;
function alerta(){

alert('eventos desde el código')

}

</script>
```

El resultado será el siguiente,



Por otro lado, ahora toma especial interés lo antes visto, porque podemos pasar parámetros y por ejemplo hacer lo siguiente,

```
<button onclick="saludarA('Ana')"> Ana </button>

<script>

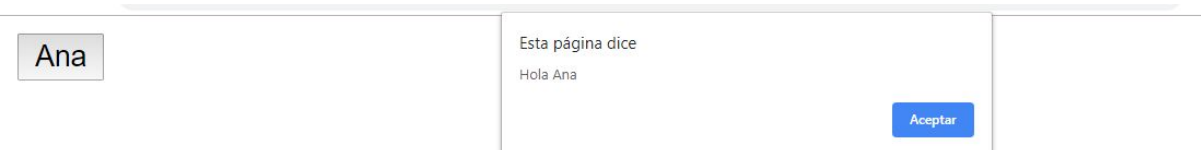
function saludarA(nombre){

alert('Hola ' + nombre)

}

</script>
```

El resultado al hacerle click al botón será el siguiente,



Lo interesante, es qué podemos reutilizar esa misma función para diferentes botones y pasarle a la misma distintos datos según lo necesitemos,

```
<button onclick="saludarA('Ana')"> Ana </button>
<button onclick="saludarA('Pedro')"> Pedro </button>
<button onclick="saludarA('Juan')"> Juan </button>
<button onclick="saludarA('María')"> María </button>
```

El código anterior, sigue intacto porque la función es una sola, no la hemos cambiado, el resultado será que por ejemplo al botón de Juan al hacerle click nos otorga el siguiente resultado,



También podemos hacer lo siguiente,

```
<button> Ana </button>
<button> Pedro </button>
<button> Juan </button>
<button> María </button>
<script>

var boton = document.getElementsByTagName('button')

for(var i = 0; i <2; i++){

  boton[i].onclick = saludar

}

function saludar(){

  alert('hola')

}
```

No se preocupen, pasaremos a explicar lo que hemos realizado, vamos por parte. Primero generamos cuatro(4) botones. Como sería muy tedioso en caso de una interfaz donde estos vayan variando, modificar el HTML, es decir no queremos trabajar desde los botones, lo haremos desde el código.

```
<button> Ana </button>
<button> Pedro </button>
<button> Juan </button>
<button> María </button>
```

En JS, estos botones son partes del DOM, y como ponerle a cada uno un ID o CLASS es poco práctico si es que estos botones serán más o menos de forma dinámica, hemos utilizado una

forma de acceso no utilizada aún qué es `document.getElementsByTagName`, es decir tomamos a los elementos por su nombre dentro del HTML,

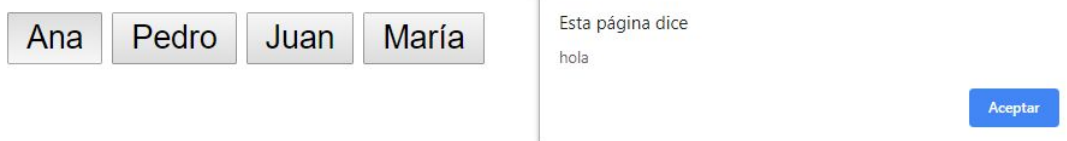
```
var boton = document.getElementsByTagName('button')
```

La pregunta es por que los guardamos en una variable?. Claro el tema es que en realidad esta variable guarda un arreglo (los botones o aquellos elementos llamados button). Lo cual no me sirve. Es decir me sirve porque simplifica la forma de acceso pero ahora necesito individualizar los. Es así qué quiero que al hacerle clic a cada uno, estos hagan individualmente algo (aunque este algo sea lo mismo).

De esta forma, surge algo ya visto qué es el for. El bucle for me ayudará a recorrer estos botones, y así poder individualizar los. He aquí la siguiente parte del código,

```
for(var i = 0; i < 2; i++){  
  
    boton[i].onclick = saludar  
  
}
```

Ahora que hemos entendido esto, vemos que al hacerle clic a cada botón el resultado será el siguiente,



The image shows a web interface with four buttons labeled 'Ana', 'Pedro', 'Juan', and 'María'. A modal dialog box is open, displaying the text 'Esta página dice hola' and an 'Aceptar' button.

No está mal. Pero la verdad, no queremos esto. La idea es que a su vez cada botón pase el valor de su contenido. Lo vamos a modificar un poco para lograr nuestro cometido.



```

<button> Ana </button>
<button> Pedro </button>
<button> Juan </button>
<button> María </button>
<script>

var boton = document.getElementsByTagName('button')

for(var i = 0; i < boton.length; i++){

  boton[i].onclick = function(){ saludarA(this.innerHTML)}
}

function saludarA(nombre){

  alert(nombre)

}

```

Vamos a explicar esta última parte. La idea es qué pasemos el texto contenido en cada botón. Por esa razón lo hacemos con innerHTML qué de esta manera trabaja obteniendo el texto del boton. Cómo lo hacemos dentro del for, el innerHTML será diferente según sobre qué botón hagamos clic.

Por otro lado, debemos pasar esto como parámetro. Por lo tanto, lo hacemos con la siguiente sintaxis,

```
function(){ saludarA(this.innerHTML)}
```

Por eso, si hacemos clic a Juan, el resultado será,

