

Funciones

Una función es un **bloque de código reutilizable** que realiza una **tarea específica** y **retorna** un valor.

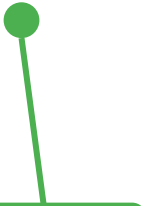
Nos permite **agrupar código** que vayamos a **usar muchas veces**.

Podemos invocarla todas las veces que necesitemos.

ESTRUCTURA BÁSICA

Palabra reservada

Usamos la palabra **function** para indicarle a Javascript que vamos a escribir una función.

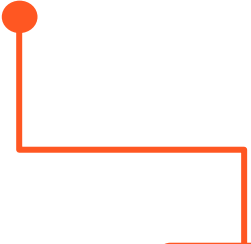


```
function sumar (a,b) {  
    return a + b;  
}
```

ESTRUCTURA BÁSICA

Nombre

Definimos un **nombre** para referirnos a nuestra función al momento de querer invocarla.




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

ESTRUCTURA BÁSICA

Parámetros

Escribimos los paréntesis y dentro de ellos los parámetros de la función.
Si lleva más de uno los separamos usando comas ,.

Si la función no lleva parámetros escribimos los paréntesis sin nada adentro {}.



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

ESTRUCTURA BÁSICA

Parámetros


Dentro de nuestra función podremos acceder a los parámetros como si fueran variables. Con solo escribir los nombres de los parámetros podremos trabajar con ellos.

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

ESTRUCTURA BÁSICA

Cuerpo

Entre las llaves de apertura y de cierre escribimos la lógica de nuestra función, es decir, el código que queremos que se ejecute cada vez que la invoquemos.




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

ESTRUCTURA BÁSICA

El retorno

A la hora de escribir una función queremos devolver al exterior el resultado del proceso.

Para ello utilizamos la palabra reservada **return** seguida de lo que queramos retornar.



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


FUNCIONES DECLARADAS

Son aquellas que se declaran usando la **estructura básica**. Reciben un **nombre formal** a través del cual la invocaremos.

```
{
```

```
function hacerHelado(cantidad) {  
    return '🍦'.repeat(cantidad)  
}
```



Se cargan **antes** de que cualquier código sea ejecutado.

FUNCIONES EXPRESADAS

Son aquellas que se **asignan como valor** a una variable. El nombre de la función será el **nombre** de la **variable** que declaremos.

```
let hacerSushi = function(cantidad) {  
  return '🍣'.repeat(cantidad)  
}
```



*Se cargan cuando el intérprete
alcanza la línea de código donde
se **encuentra** la **función**.*

INVOCANDO UNA FUNCIÓN

La forma de **invocar** (ejecutar) una función es escribiendo su nombre seguido de apertura y cierre de paréntesis.

```
nombreFuncion();
```

En caso de querer guardar el dato que **retorna** será necesario almacenarlo en una variable. Podemos utilizar `console.log()` si necesitamos chequear la información en la consola del navegador.

```
let resultado = nombreFuncion();  
console.log(resultado);
```

INVOCANDO UNA FUNCIÓN

Si la función espera parámetros **debemos** pasarlos dentro de los paréntesis.

Si hay más de un parámetro es **importante respetar el orden** ya que Javascript los asignará en el orden en que fueron declarados.

```
function saludar(nombre, apellido) {  
    return 'Hola ' + nombre + ' ' + apellido;  
}  
  
saludar('Robertito', 'Rodríguez');  
  
// retorna 'Hola Robertito Rodríguez'
```

}

INVOCANDO UNA FUNCIÓN

Si no pasamos los parámetros Javascript les asignará el tipo de dato *undefined*.

```
function saludar(nombre, apellido) {  
    return 'Hola ' + nombre + ' ' + apellido;  
}  
saludar(); // retorna 'Hola undefined undefined'
```



Al no haber recibido los argumento que necesitaba Javascript le asigna el tipo de dato **undefined** a las variables nombre y apellido.



Scope de variables

El **scope** de una variable se refiere al ámbito en donde podemos acceder a ella.

Los scopes **son definidos** principalmente por bloques de código y las **las funciones definen bloques de código**.

Es fundamental dominarlo cuando trabajamos con ellas.

SCOPE LOCAL

En el momento en que declaramos una variable **dentro** de una función o bloque de código la misma pasa a tener **alcance local**.

La variable “vivirá” únicamente **dentro** de esa función o bloque de código.

No es posible hacer uso de esa variable por **fuera** de la función o bloque de código.

```
function miFuncion() {  
    // todo el código que escribamos dentro  
    // de nuestra función, tiene scope local  
}
```

}

Scope de variables

```
{ código }
```

```
function hola() {  
  let saludo = 'Hola ¿qué tal?';  
  return saludo;  
}
```

```
console.log(saludo);
```

Definimos la variable saludo **dentro** de la función *hola()*, por lo tanto su **scope** es **local**.

Sólo dentro de esta función podemos acceder a ella.

Scope de variables

```
{ código }
```

```
function hola() {  
  let saludo = 'Hola ¿qué tal?';  
  return saludo;  
}
```

```
console.log(saludo);
```

Al querer hacer uso de la variable **saludo** por fuera de la función, Javascript no la encuentra y nos devuelve el siguiente error:

Uncaught ReferenceError: saludo is not defined

SCOPE GLOBAL

En el momento en que declaramos una variable **fuera** de cualquier función o bloque de código la misma pasa a tener **alcance global**.

Podemos hacer uso de ella desde cualquier lugar del código en el que nos encontremos, inclusive dentro de una función.

```
// todo el código que escribamos fuera
// de las funciones, es global

let miVariable;
function miFuncion() {
    // Tenemos acceso a las variables globales
}
```

PRACTIQUEMOS

