


## 2.5. Funciones

Las funciones representan el corazón de la programación en JavaScript.

Son de tal importancia que es prácticamente imposible encontrar un programa escrito en este lenguaje que no incluya un gran número de ellas.

Su **versatilidad, eficiencia y flexibilidad** las convierten en una potente herramienta para desarrollar aplicaciones web complejas.

- 
- Desarrollo de una aplicación → Tenemos una lista de tareas → Una tarea se repite de manera constante a lo largo de todo el programa.
  - Habría que reescribir el código que resuelve esa tarea muchas veces en distintos puntos del programa.
  - Si se detectara un error en esa solución, o se quisiera mejorar → Habría que buscar todas las porciones del código donde aparece la solución y actualizarla.
  - Lo cual es una tarea tediosa e improductiva que consume muchos recursos de forma innecesaria.

Las funciones resuelven este problema.

- Se crea una función (asignarle un nombre) y dentro de ella resolvemos la tarea en cuestión.
- En todos los sitios del programa donde hubiera que incorporar la solución, simplemente se haría referencia a ese nombre.
- Si por cualquier motivo se tuviera que modificar esa pieza de código, se haría en un único punto del programa (donde se definió la función) y el cambio repercutiría a todo el programa.

Este concepto de encapsulamiento y reutilización es la base de la **modularidad**.

Se trata de crear piezas de código (**funciones**) que resuelven problemas concretos, más pequeños que la solución global, y alcanzar esta última combinando varias de esas piezas.

Al mismo tiempo, si en otro programa se presentara la necesidad de resolver la misma tarea u otra muy parecida, puede reutilizarse el código ya probado de la otra aplicación, lo que ahorra una cantidad enorme de tiempo de desarrollo.

→ Las funciones permiten **reutilizar** el código, mejorar la **eficiencia**, aumentar su **legibilidad** y reducir los **costes** de mantenimiento de los programas.

```
function functionName(Parameter1, Parameter2, ..)
{
    // Function body
}
```

La función puede devolver un valor usando **return** (si no tiene return es como si devolviera *undefined*).



Si se llama una función con menos **parámetros** de los declarados el valor de los parámetros no pasados será *undefined*.

```
function potencia(base, exponente) {  
    console.log(base);           // muestra 4  
    console.log(exponente);      // muestra undefined  
    let valor=1;  
    for (let i=1; i<=exponente; i++) {  
        valor=valor*base;  
    }  
    return valor;  
}  
  
potencia(4);    // devolverá 1 ya que no se ejecuta el for
```

Podemos poner **parámetros por defecto**.

```
function dividir(numerador, denominador=1) {  
    return (numerador/denominador);  
}  
  
console.log(dividir(4)); // muestra 4  
console.log(dividir(4,2)); // muestra 2  
console.log(dividir()); // muestra NaN
```

Función con **parámetros variables**: es posible acceder a los parámetros desde el array **arguments[]**.

```
function suma () {  
    var result = 0;  
    for (var i=0; i<arguments.length; i++)  
        result += arguments[i];  
    return result;  
}
```

```
console.log(suma(4, 2));           // mostrará 6  
console.log(suma(4, 2, 5, 3, 2, 1, 3)); // mostrará 20
```



Podemos pasarlas **por argumento** o **asignarlas** a una variable.

```
const cuadrado = function(value) {  
    return value * value  
}  
  
function aplica_fn(dato, funcion_a_aplicar) {  
    return funcion_a_aplicar(dato);  
}  
  
aplica_fn(3, cuadrado);    // devolverá 9 (3^2)
```

Definir una función **sin darle un nombre**. Dicha función, **función anónima**, puede asignarse a una variable, autoejecutarse o asignarse a un manejador de eventos.


```
let holaMundo = function() {  
    alert('Hola mundo!');  
}
```

```
holaMundo();           // se ejecuta la función
```

Y, por último, tenemos las **Arrow functions (funciones flecha)**.

- Eliminamos la palabra function
- Si sólo tiene 1 parámetro podemos eliminar los paréntesis de los parámetros
- Ponemos el símbolo =>
- Si la función sólo tiene 1 línea podemos eliminar las { } y la palabra return

```
let potencia = function(base, exponente) {  
  let valor=1;  
  for (let i=1; i<=exponente; i++) {  
    valor=valor*base;  
  }  
  return valor;  
}
```



```
let potencia = (base, exponente) => {  
  let valor=1;  
  for (let i=1; i<=exponente; i++) {  
    valor=valor*base;  
  }  
  return valor;  
}
```



```
let cuadrado = function(base) {  
    return base * base;  
}
```



```
let cuadrado = base => base * base;
```

