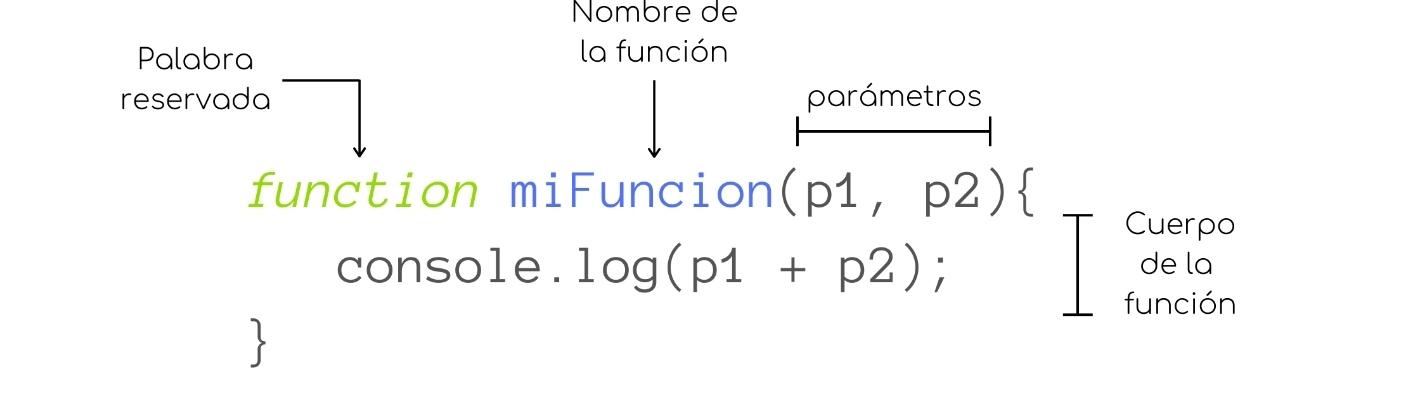
**FUNCIONES EN JAVASCRIPT**

****

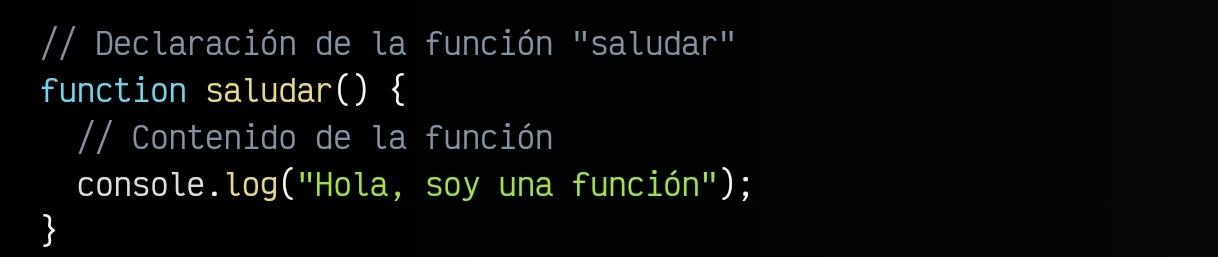
**¿Qué es una función?**

Las funciones nos permiten agrupar líneas de código en tareas con un nombre, para que, posteriormente, podamos hacer referencia a ese nombre para realizar todo lo que se agrupe en dicha tarea. Para usar funciones hay que hacer 2 cosas:

* Declarar la función: Preparar la función, darle un nombre y decirle las tareas que realizará.
* Ejecutar la función: «Llamar» a la función para que realice las tareas de su contenido.
* Se utiliza la palabra reservada Return. Al retornar un valor estamos hablando de una función.

**Declaración de una función en javascript**

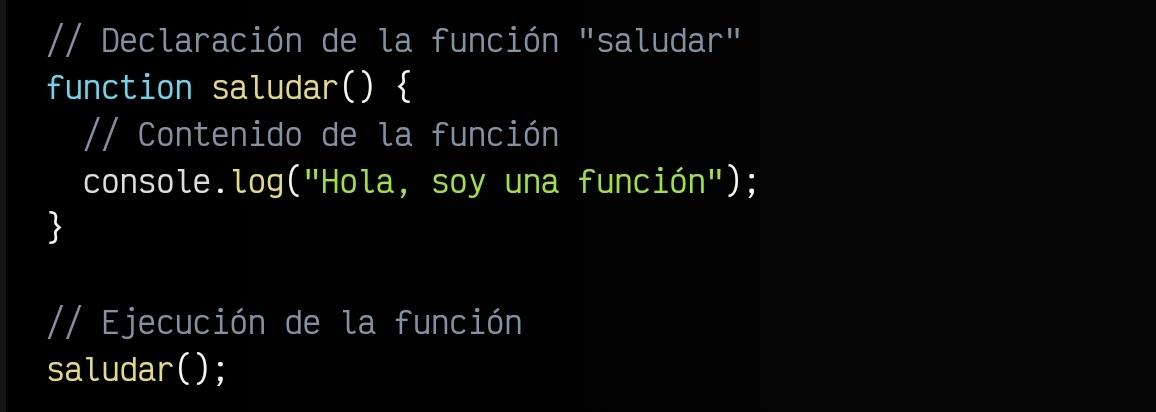
En el siguiente ejemplo veremos la declaración de una función llamada saludar:



El contenido de la función es una línea que mostrará por consola un saludo. Sin embargo, si escribimos estas 4-5 líneas de código en nuestro programa, no mostrará nada por pantalla. Por que nos falta el **segundo paso, ejecutarla, que es realmente cuando se realizan las tareas de su contenido.**

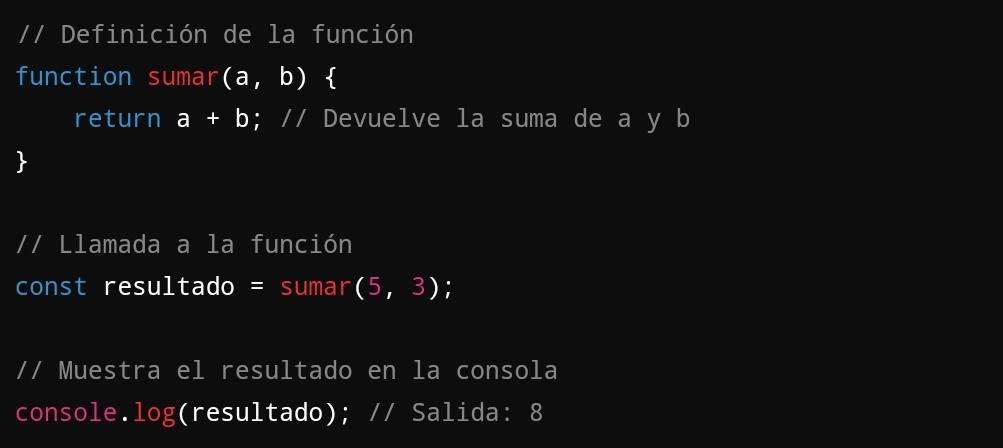
**Ejecución**

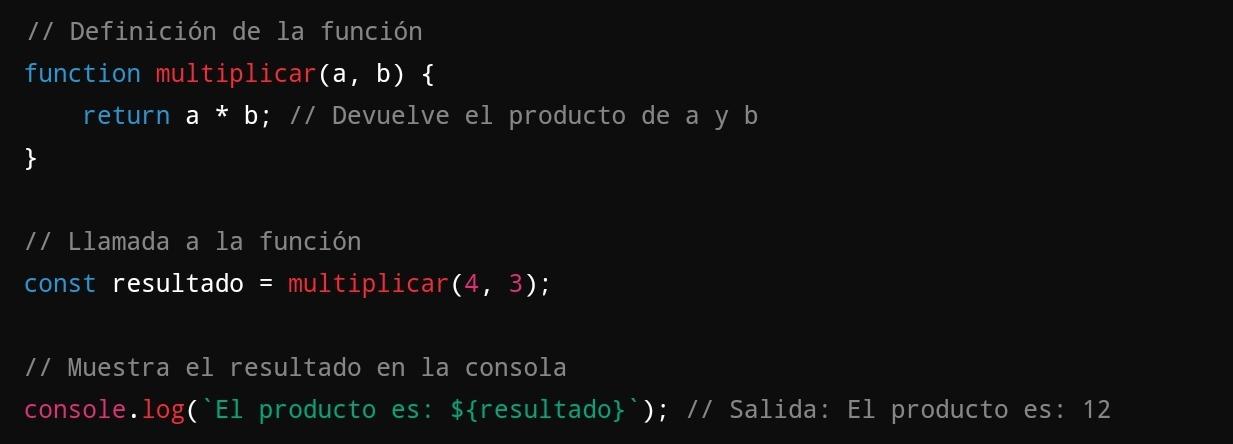
Veamos, ahora sí, el ejemplo completo con declaración y ejecución:



En este ejemplo hemos declarado la función y además,hemos ejecutado la función (en la última línea) llamándola por su nombre y seguida de ambos paréntesis, que nos indican que es una función**.** En este ejemplo, si se nos mostraría en la consola Javascript el mensaje de saludo.

| Definir una función no la ejecuta. Definirla simplemente nombra la función y especifica qué hacer cuando se llama a la función. Llamar a la función (ejecutarla) es llevar a cabo las acciones especificadas con los parámetros indicados. Siguiendo el ejemplo anterior:  saludar (); // Llama a la función, es decir, la ejecuta. |
| --- |





**Estructura de una función en Javascript**

La **definición de función** (también denominada declaración de función o expresión de función) consta de la palabra **clave function**, seguida de:

* El nombre de la función.
* Una lista de parámetros de la función, entre paréntesis y separados por comas.
* Las declaraciones que definen la función, se encierran entre llaves, { ... }.

Por ejemplo, el siguiente código **define una función** simple que se llama square**:**

| **function** square (number) { // Declaración de función con nombre y parámetros.  return number \* number;  } |
| --- |

La función square toma un parámetro, llamado number. La función consta de una declaración que devuelve el *parámetro* de la función (es decir, number) multiplicado por sí mismo.

La instrucción return especifica el valor devuelto por la función:

| return number \* number; // Instrucción “return” especifica el valor devuelto por la función |
| --- |

La declaración de función anterior sintácticamente es una declaración, **las funciones también se pueden crear mediante una expresión function.** Esta función puede ser anónima; no tiene por qué tener un nombre.

Por ejemplo, la función square se podría haber definido como:

| const square = function (number) { //Función sin nombre definido antes de los parámetros  return number \* number;  };  var x = square(4); // x obtiene el valor 16 |
| --- |

Sin embargo, puedes proporcionar un nombre a la función después de utilizar la palabra reservada **function**. Proporcionar un nombre permite que la función se refiera a sí misma y facilita la identificación de la función en el seguimiento de la pila de un depurador:

| const factorial = function fac (n) { // Nombre de la función: “fac”, para referirse a sí misma  return n < 2 ? 1 : n \* fac (n - 1); // Return de parámetro “n” con tareas a realizar  };  console.log (factorial(3)); // Ejecución. |
| --- |

**Las expresiones function son convenientes cuando se pasa una función como argumento a otra función.** El siguiente ejemplo muestra una función map que debería recibir una función como primer argumento y un arreglo como segundo argumento.

| function map (f, a) {  let result = [ ]; // Crea un nuevo arreglo  let i; // Declara una variable  for (i = 0; i != a.length; i++) result[i] = f(a[i]);  return result;  } |
| --- |

En el siguiente código**, la función recibe una función definida por una expresión de función y la ejecuta por cada elemento del arreglo recibido como segundo argumento.**

| function map (f, a) {  let result = [ ]; // Crea un nuevo arreglo  let i; // Declara una variable  for (i = 0; i != a.length; i++) result [i] = f (a [i] );  return result;  }  const f = function (x) { // expresión función  return x \* x \* x;  };  let numbers = [0, 1, 2, 5, 10];  let cube = map (f, numbers);  console.log(cube); |
| --- |

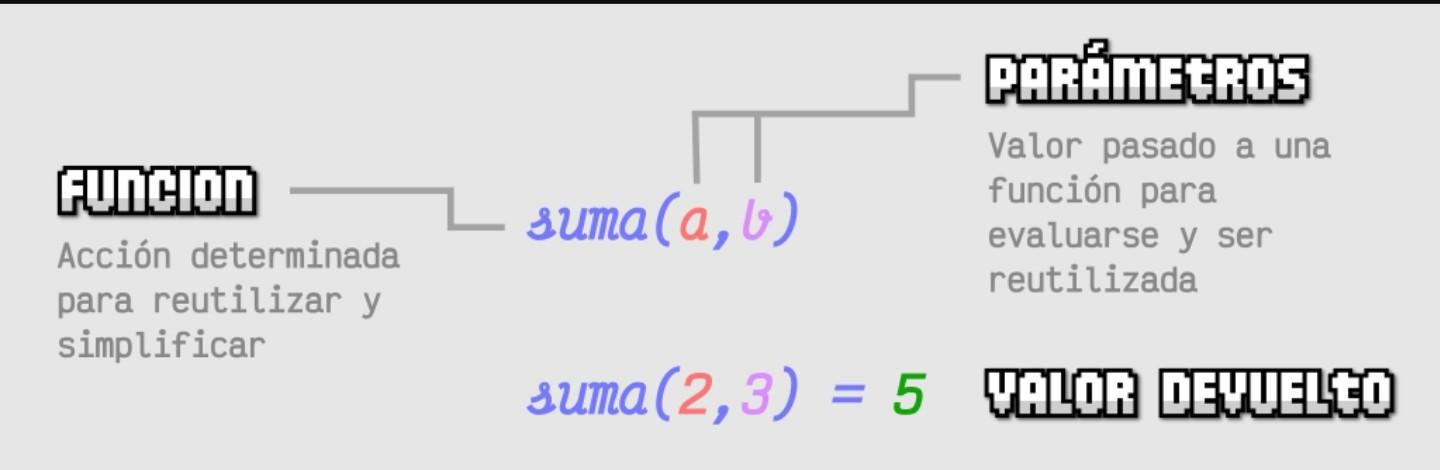
Respuesta: La función devuelve: [0, 1, 8, 125, 1000].

En JavaScript, **una función se puede definir en función de una condición.** Por ejemplo, la siguiente definición de función define myFunc solo si num es igual a 0:

| var myFunc; // Definición de variable  if (num === 0) { // Expresión condicional  myFunc = function (theObject) { // Asigna la función si la condición es true  theObject.make = "Toyota"; // Instrucción a realizar según la función  };  } |
| --- |

**¿Qué son los parámetros?**

A las funciones se les pueden pasar parámetros, que no son más que variables que les pasamos desde fuera hacia dentro de la función. Además, también podemos hacer que la función realice sus tareas y nos devuelva un resultado hacia el exterior de la función.

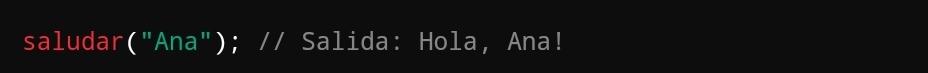


| **Los parámetros de una función son las variables que se definen en la declaración de la función y que se utilizan para recibir valores de entrada cuando se llama a la función.** |
| --- |

**Parámetros**

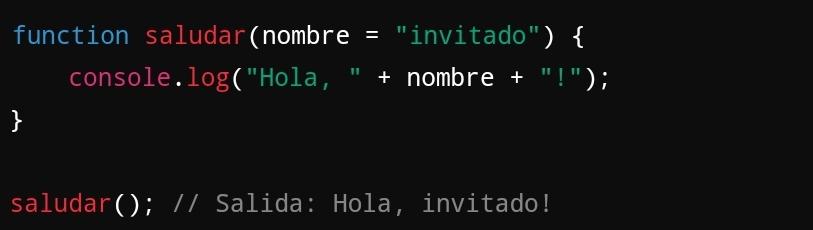
**Llamada a la Función o ejecución de la función**

Cuando llamas a la función, pasas argumentos que corresponden a los parámetros:

****

**Parámetros por Defecto**

Los parámetros por defecto en JavaScript te permiten establecer valores predeterminados para los parámetros de una función. Si al llamar a la función no se pasa un argumento para esos parámetros, se utilizará el valor por defecto. La sintaxis es sencilla: solo tienes que asignar un valor al parámetro durante la definición de la función:



* **Expresiones como Valores por Defecto:** Los valores por defecto pueden ser cualquier expresión, incluso llamadas a funciones:

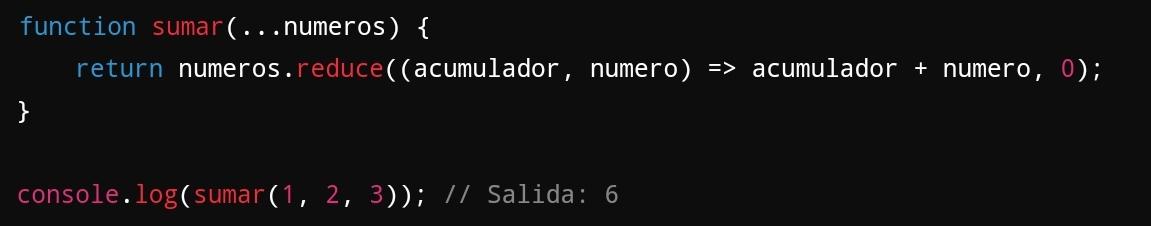


**Parámetros Rest**

Los *parámetros rest* permiten que una función acepte un número variable de argumentos como un array. Esto es útil cuando no sabes de antemano cuántos argumentos se van a pasar a la función. Usos Comunes:

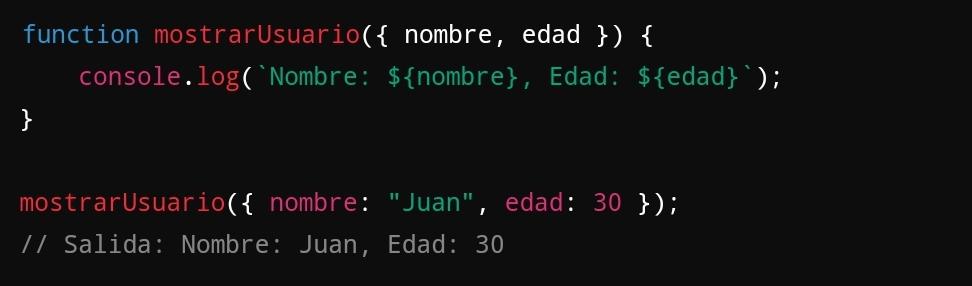
* Funciones matemáticas: Para realizar operaciones con un número indeterminado de valores.
* Manejo de listas: Ideal para funciones que manipulan listas de elementos.

Si quieres pasar un número variable de argumentos, puedes usar el operador …:



**Parámetros en Objetos**

Los parámetros en objetos te permiten pasar múltiples valores a una función mediante un solo argumento que es un objeto. Esto es especialmente útil para funciones que requieren varios parámetros, ya que mejora la legibilidad y la organización del código.



* **Desestructuración:** En el ejemplo anterior, estamos utilizando la desestructuración de objetos, que permite extraer propiedades de un objeto y asignarlas a variables individuales directamente en los parámetros de la función.
* **Valores por Defecto en Objetos:** También puedes establecer valores por defecto para propiedades de un objeto:



**Ventajas**

1. Claridad: Hace que la llamada a la función sea más clara, ya que puedes ver de inmediato qué propiedades se están pasando.
2. Flexibilidad: Puedes pasar solo las propiedades que necesitas, sin preocuparte por el orden.
3. Mantenimiento: Facilita el mantenimiento del código, especialmente si se agregan más propiedades en el futuro.

Aquí hay un ejemplo más completo que muestra cómo **usar objetos como parámetros en una función:**



**Devolución de valores “Return”**

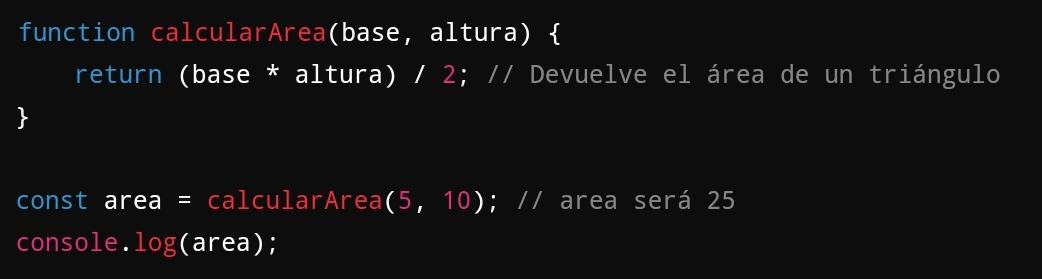
Lo que buscamos es que una función realice una tarea y nos devuelva la información al exterior de la función, para así utilizarla o guardarla en una variable, que utilizaremos posteriormente para nuestros objetivos.

Para ello, se utiliza la palabra clave return, que suele colocarse al final de la función, ya que con dicha devolución terminamos la ejecución de la función (si existe código después, nunca será ejecutado). En otras palabras, en JavaScript, **la palabra clave return se utiliza en una función para devolver un valor. Cuando se alcanza una declaración return, la ejecución de la función se detiene y se devuelve el valor especificado.**

**Características de return**

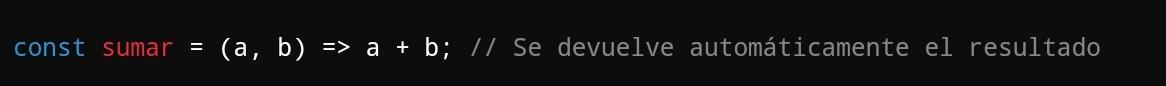
* Detiene la ejecución: Al llegar a return, la función deja de ejecutarse, incluso si hay más código después de esa línea.
* Puede devolver cualquier tipo de valor: Puedes devolver números, cadenas, objetos, arrays, funciones, etc.
* Devolver sin valor: Si no se especifica un valor en return, la función devuelve *undefined*.

Aquí hay un ejemplo que muestra cómo usar return en una función:



**Funciones flecha y return**

En funciones flecha (arrow functions), puedes omitir la palabra clave return si la función consiste en una sola expresión:



**Funciones predefinidas**

JavaScript tiene integradas varias funciones de nivel superior:

* **eval()**, el método eval() evalúa el código JavaScript representado como una cadena.
* **uneval()**, el método uneval() crea una representación de cadena del código fuente de un Object.
* **isFinite()**, la función global isFinite() determina si el valor pasado es un número finito. Si es necesario, el parámetro, primero se convierte en un número.
* **isNaN()**, la función isNaN() determina si un valor es NaN o no. Nota: La coerción dentro de la función isNaN tiene interesantes reglas; también puedes querer usar Number.isNa(), como se define en ECMAScript 2015, o puedes usar typeof para determinar si el valor no es un número (NaN).
* **parseFloat()**, la función parseFloat() procesa un argumento de cadena y devuelve un número de punto flotante.
* **decodeURI()**, la función decodeURI() decodifica un identificador uniforme de recursos (URI) creado previamente por encodeURI o por una rutina similar.
* **encodeURI()**, El método encodeURI() codifica un identificador uniforme de recursos (URI) reemplazando cada instancia de ciertos caracteres por una, dos, tres o cuatro secuencias de escape que representan la codificación UTF-8 del caracter (solo habrá cuatro secuencias de escape para caracteres compuestos por dos caracteres "sustitutos").
* **encodeURIComponent()**, el método encodeURIComponent() codifica un componente Identificador uniforme de recursos (URI) reemplazando cada instancia de ciertos caracteres por una, dos, tres o cuatro secuencias de escape que representan la codificación UTF-8 del caracter (solo habrá cuatro secuencias de escape para caracteres compuestos por dos caracteres "sustitutos").
* **escape()**, el método obsoleto escape() calcula una nueva cadena en la que ciertos caracteres han sido reemplazados por una secuencia de escape hexadecimal. En su lugar usa encodeURI o encodeURIComponent.
* **unescape()**, el método obsoleto unescape() calcula una nueva cadena en la que las secuencias de escape hexadecimales se reemplazan con el caracter que representan. Las secuencias de escape se pueden introducir por medio de una función como escape. Debido a que unescape() está en desuso, usa decodeURI() o decodeURIComponent en su lugar.