Programación con funciones, arrays y objetos definidos por el usuario

Funciones predefinidas del lenguaje

JavaScript ofrece una variedad de funciones predefinidas (que ya están integradas en el lenguaje) que facilitan tareas comunes de programación. Estas funciones integradas proporcionan herramientas poderosas para manipular datos, controlar flujos de ejecución y optimizar el rendimiento de las aplicaciones. Podemos utilizarlas sin tener que declararlas previamente y sin tener que conocer todas las instrucciones que realiza. Simplemente tenemos que conocer el nombre de la función y el resultado final que obtenemos al utilizarla.

- **Matemáticas y Números:** JavaScript proporciona un conjunto de funciones matemáticas como *Math.abs(), Math.round(), Math.random()* para realizar cálculos numéricos de manera eficiente.
- **Manipulación de Cadenas:** Funciones como *String.toUpperCase(), String.replace(), String.split()* facilitan la transformación y el procesamiento de textos.
- **Fecha y Hora:** Funciones como *Date.now(), Date.getDate()* permiten trabajar con fechas y horarios de manera sencilla.
- **Conversión de Tipos:** Funciones como *Number(), Boolean(), String()* ayudan a convertir entre diferentes tipos de datos
- **Depuración:** Funciones como *console.log(), console.error()* son útiles para la depuración y el seguimiento de errores.

Función eval()

La función _eval()_ evalúa una cadena de texto como si fuera código JavaScript. Permite ejecutar código dinámico, ya que puede interpretar expresiones, declaraciones y código completo a partir de una cadena de texto.

```
var input = prompt("Introduce una operación numérica");
var resultado = eval(input);
alert("El resultado de la operación es: " + resultado);
```

Ver ejemplo1.html

Cuidado con esta función. El código anterior es peligroso

Función isNaN()

isNaN es el acrónimo de Not a Number (no es un número). Esta función evalúa si el objeto pasado como argumento es de tipo numérico. Devuelve True si el valor no es un número, o False en caso contrario.

```
var input = prompt("Introduce un valor numérico:");
if( isNaN(input) ){
   alert("El dato ingresado no es numérico");
}else{
   alert("El dato ingresado es numérico");
}
```

Función String()

La función String() convierte el objeto pasado como argumento en una cadena de texto.

```
var fecha = new Date()
var fechaString = String(fecha)
alert("La fecha actual es: " + fechaString);
```

Función Number() y parseInt()

Convierte un objeto pasado como argumento en un número

Si la cadena de texto no es un número, devuelve NaN

Convierte una cadena de caracteres pasada como argumento en un número entero. Puede hacerlo en una base específica si se pasa por argumento (base binaria, base hexadecimal)

Si encuentra algún carácter que no es numérico, devuelve el valor antes de ese carácter.

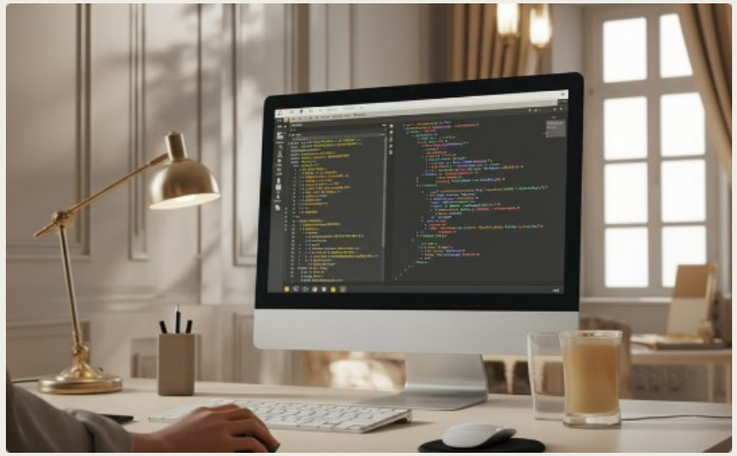
Si el primer valor no es numérico, devuelve NaN

Funciones del usuario

Un grupo de instrucciones relacionadas para realizar una tarea concreta.

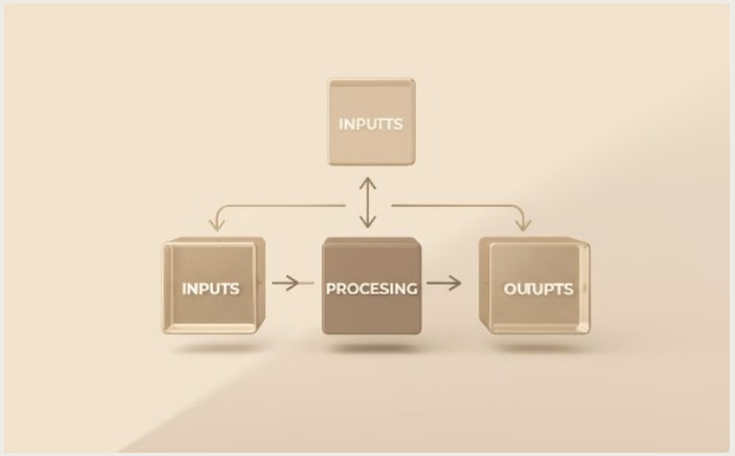
Definición de Funciones

Las funciones definidas por el usuario permiten encapsular lógica reutilizable y modular en bloques de código con parámetros y valores de retorno.



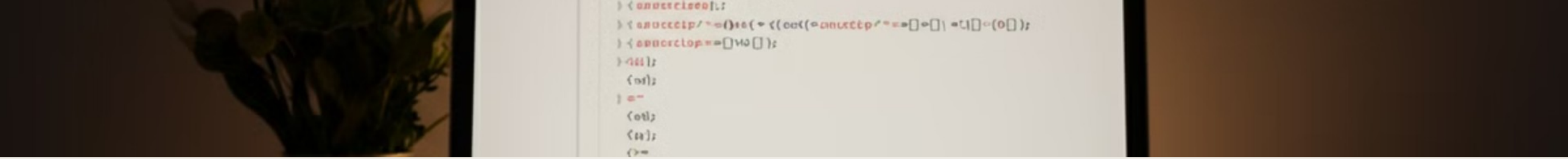
Invocación de Funciones

Las funciones se pueden llamar o invocar en diferentes partes del código para ejecutar su lógica interna y obtener resultados específicos.



Estructura de Funciones

Una función típica consta de un nombre, parámetros de entrada, un bloque de instrucciones y un valor de retorno opcional.



Definición de funciones

Sintaxis básica

1

Las funciones en JavaScript se definen utilizando la palabra clave **function**, seguida del nombre de la función y un par de paréntesis que pueden contener parámetros.

Funciones con y sin parámetros

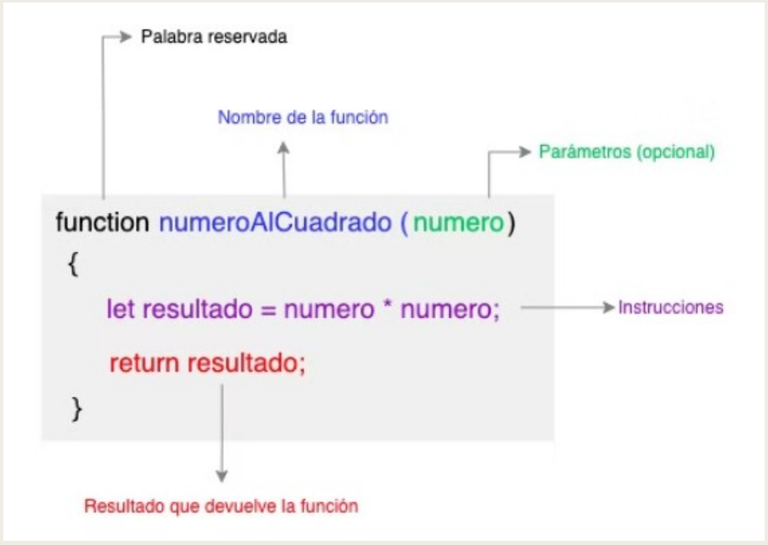
2

Las funciones pueden recibir parámetros, que son variables que se pasan a la función cuando se invoca. También pueden prescindir de parámetros.

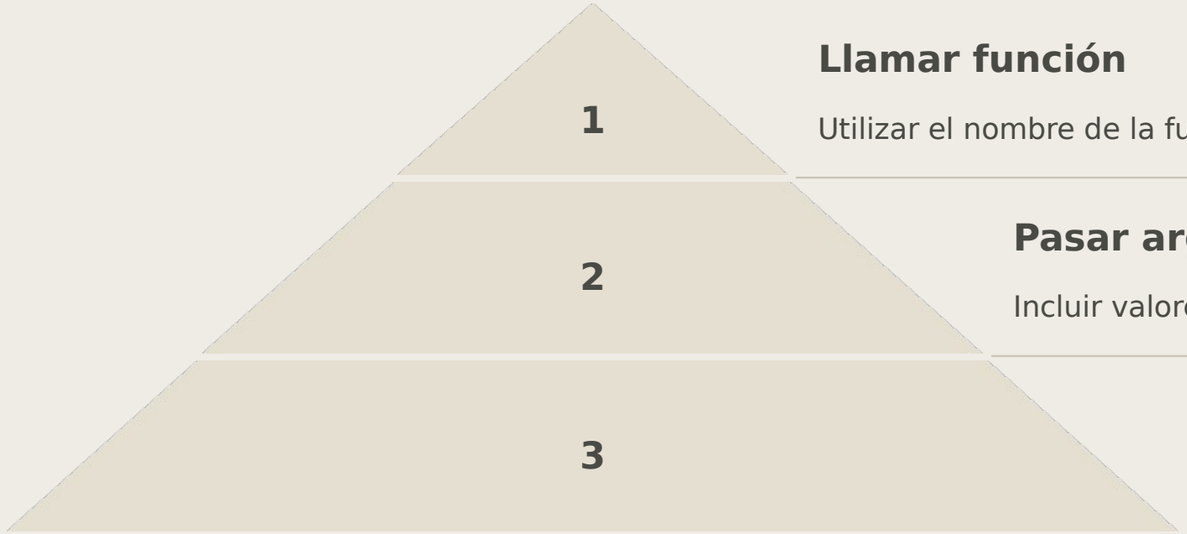
Bloque de código de la función

3

e de código de la función, encerrado entre llaves, contiene las instrucciones a ejecutar cuando se llama a la función.



Invocación de funciones



Llamar función

Utilizar el nombre de la función seguido de paréntesis

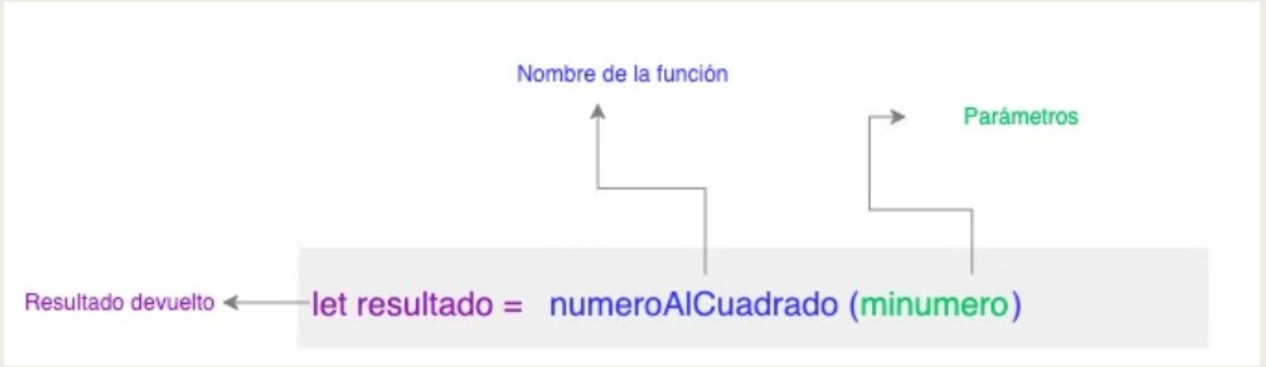
Pasar argumentos

Incluir valores dentro de los paréntesis, separados por comas

Recibir resultados

Asignar el valor de retorno de la función a una variable

Para invocar una función, se utiliza su nombre seguido de paréntesis. Dentro de los paréntesis se pueden pasar argumentos, que serán recibidos por la función. El valor de retorno de la función se puede asignar a una variable para su posterior uso.



Ejemplo: Aplicar IVA a un precio

```
let valorProducto = prompt("Ingresa el precio del producto sin IVA);
let iva = prompt("Ingresa el IVA que se debe aplicar);

let valorFinalProducto = valorProducto * iva;

alert("El precio del producto, aplicando el IVA del " + iva + " es " + valorFinalProducto);
```

Ejemplo: Aplicar IVA a un precio

```
let valorProducto = prompt("Ingresa el precio del producto sin IVA);
let iva = prompt("Ingresa el IVA que se debe aplicar);

let valorFinalProducto = valorProducto * iva;

alert("El precio del producto, aplicando el IVA del " + iva + " es " + valorFinalProducto);
```

```
function aplicarIva(valor, iva)
{
    let valorFinal = valorProducto * iva;
    return valorFinal;
}

let valorProducto = prompt("Ingresa el precio del producto sin IVA);
let iva = prompt("Ingresa el IVA que se debe aplicar);

let valorFinalProducto = aplicarIva(valorProducto,iva);

alert("El precio del producto, aplicando el IVA del " + iva + " es " + valorFinalProducto);
```

Ejercicio 1: Concatenar strings

Objetivo: Crea una función llamada `concatenarStrings` que reciba dos strings y los combine en una sola cadena con un espacio entre ellas.

```
function concatenarStrings(str1, str2) {
  // Tu código aquí
}

console.log(concatenarStrings("Hola", "Mundo")); // Debería imprimir: "Hola Mundo"
```

Ejercicio 2: Calcular factorial

Objetivo: Crea una función llamada `factorial` que reciba un número y devuelva su factorial (el producto de todos los números enteros desde 1 hasta el número).

Si recibe el 1, tiene que calcular el 1! = 1.

Si recibe el 2, tiene que calcular el 2! = 1 × 2

Si recibe el 3, tiene que calcular el 3! = 1×2×3

...

Nota: Tienes que iterar desde 1 hasta N. En cada iteración, el resultado se multiplica por el indice de iteración (normalmente

Ejercicio 3: Contar vocales

Objetivo: Crea una función llamada `contarVocales` que reciba una cadena de texto y devuelva el número de vocales que c

```
function contarVocales(cadena) {
  // Tu código aquí
}

console.log(contarVocales("javascript")); // Debería imprimir: 3
console.log(contarVocales("hola mundo")); // Debería imprimir: 4
```

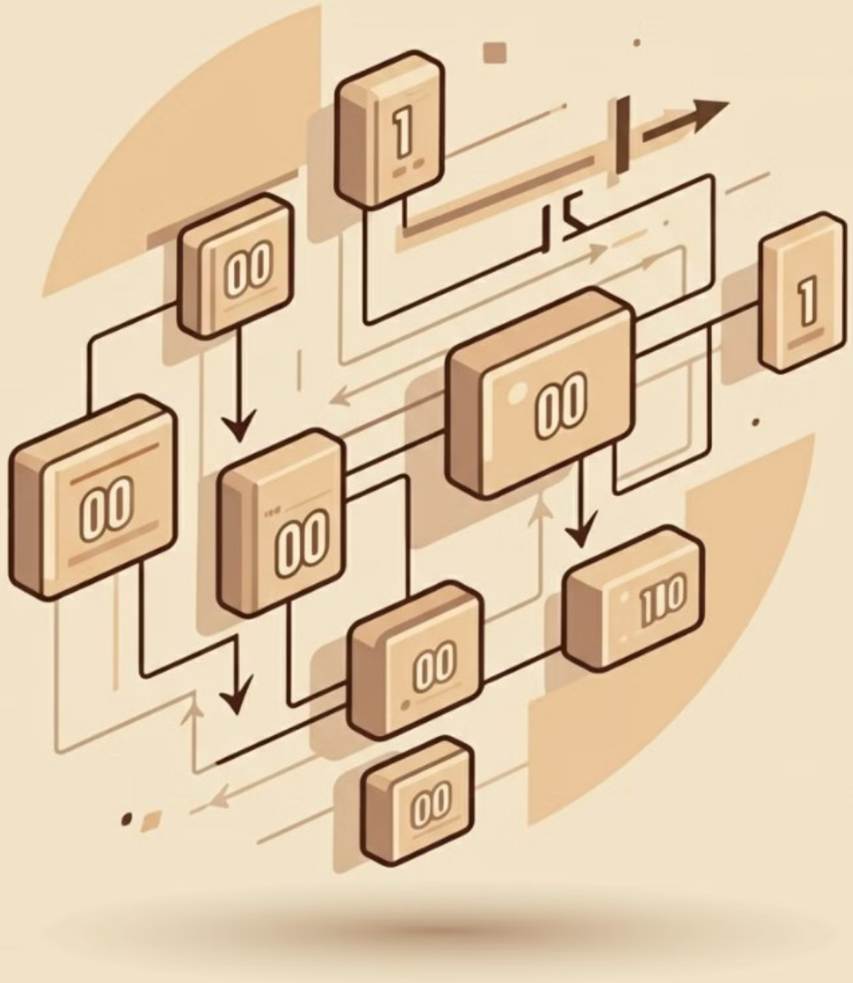
Arrays en JavaScript

La mayor parte de las aplicaciones web están diseñadas para gestionar un número elevado de datos. Si tenemos por ejemplo una aplicación que gestiona una tienda de productos alimenticios, es normal que manipulemos decenas de variables:

```
var producto1 = "Pan";

var producto2 = "Agua";

var producto3 = "Lentejas";

var producto4 = "Naranjas";

var producto 5 = "Cereales";

....
```

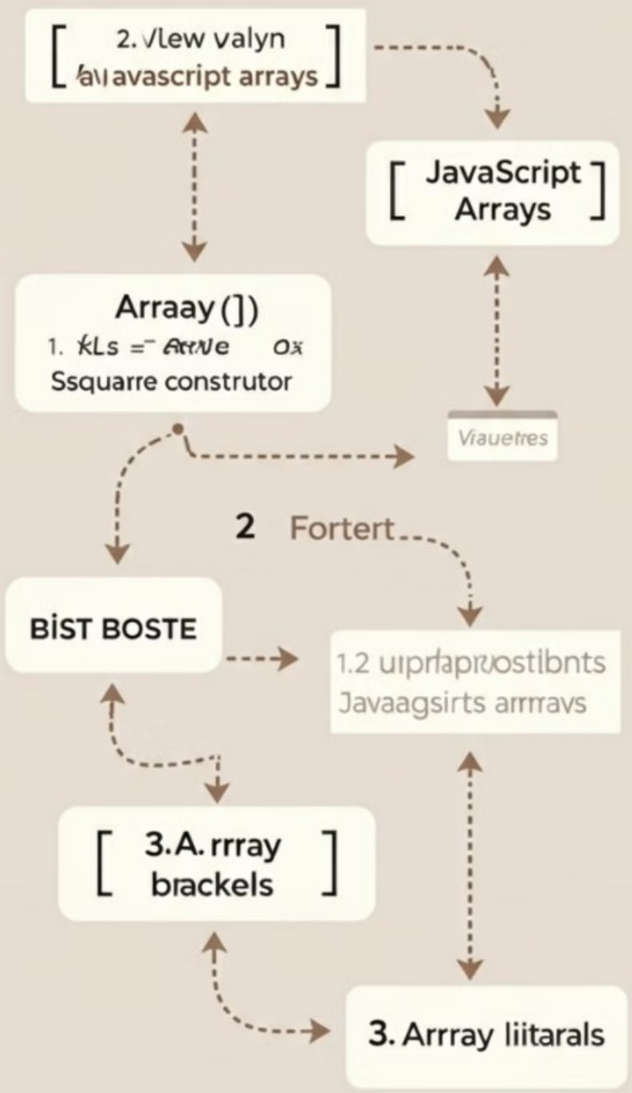
Y si queremos imprimir en pantalla:

```
console.log(producto1);

console.log(producto2);

console.log(producto3);

console.log(producto4);

console.log(producto5);
```



Cómo funciona el array

Indice	Valor
0	"Pan"
1	"Agua"
2	"Lentejas"
3	"Naranjas"
4	"Cereales"



Declaración de arrays

1

Sintaxis básica

Los arrays se declaran con corchetes cuadrados **[]**. Cada elemento se separa con una coma.

2

Elementos heterogéneos

Los arrays pueden contener datos de diferentes tipos, como números, cadenas, booleanos e incluso otros arrays.

3

Inicialización vacía

También se puede declarar un array vacío y luego ir agregando elementos a medida que se necesiten.

Declaración de Arrays. Sintaxis básica

Es necesario declarar un array antes de poder usarlo.

```
var nombre_del_array = new Array();
```

Indicando el número de elementos que tendrá el array

```
var nombre_del_array = new Array(10);
```

Inicializar con valores

```
var nombre_del_array = ["Pera", "Agua","Lentejas", "Naranjas", "Cereales"];
```

Inicialización de arrays

Existen varias formas de inicializar un array en JavaScript.

1.Indicando el valor de una posición específica del array:

nombre_del_array[indice] = valor_del_array;

Ejemplo:

```
var productos = new Array(10);
productos[0] = "Pera";
producots[1] = "Pan";
...
```

2. Inicializar al mismo tiempo que declaramos el array

nombre_del_array = new Array(valor1, valor2, valor3,...)

```
var productos = new Array("Pera", "Pan","Lentejas", "Naranjas", "Cereales");
```

Uso de arrays con bucles

Recorrido con for

Uno de los métodos más comunes para recorrer un array es mediante un bucle **for**. Esto permite acceder a cada elemento del array de forma individual.

Bucle forEach

El método **forEach()** también permite iterar sobre los elementos del array, aplicando una función callback a cada uno.

Otras opciones

También se pueden utilizar otros tipos de bucles, como **for...of** o **for...in**, así como métodos de array como **map()**, **filter()** y **reduce()**.

Manipulación

Estos bucles y métodos permiten no solo recorrer los arrays, sino también manipularlos, como añadir, eliminar o modificar elementos.

Recorrido de arrays con for

Podemos inicializar los arrays con ayuda de un for

```
var codigos_productos = new Array();
for (var i=0;i<10;i++) {
  codigos_productos[i] = "Código_producto_" + i;
}
```

Y para imprimir en pantalla, para evitar escribir 10 veces document.write("Código_producto_0"), nos podemos de ayudar de nuevo de un for para recorrer el array e imprimir en pantalla por cada elemento del array

```
for (var i=0; i<10; i++)
{
   document.write(codigos_productos[i] + "<br>");
}
```

Propiedades de los arrays

Propiedad length

Devuelve el número de elementos que tiene el array.

Acceso a elementos

Los elementos del array se pueden acceder mediante su índice, empezando desde 0.

Agregar y eliminar

Se pueden agregar o eliminar elementos usando métodos como push(), pop(), shift() y splice().

Recorrer el array

Puedes recorrer los elementos de un array usando bucles como for, forEach() o métodos como map() o filter().

Ejercicio 4: Sumar elementos de un array

Objetivo: Define una función llamada `sumarArray` que reciba un array de números y devuelva la suma de todos sus elemen

```
function sumarArray(arr) {
  // Tu código aquí
}

const numeros = [1, 2, 3, 4, 5];
console.log(sumarArray(numeros)); // Debería imprimir: 15
```

Ejercicio 5: Encontrar el mayor número

Objetivo: Define una función llamada `numeroMayor` que reciba un array de números y devuelva el número más grande.

```
function numeroMayor(arr) {
  // Tu código aquí
}

const numeros = [10, 35, 2, 90, 48];
console.log(numeroMayor(numeros)); // Debería imprimir: 90
```

Ejercicio 6: Filtrar números pares

Objetivo: Define una función llamada `filtrarPares` que reciba un array de números y devuelva un nuevo array que contenga solo los números pares.

```
function filtrarPares(arr) {
  // Tu código aquí
}

const numeros = [1, 2, 3, 4, 5, 6, 7, 8];
console.log(filtrarPares(numeros)); // Debería imprimir: [2, 4, 6, 8]
```

Ejercicio 7: Comprobar palíndromo

Objetivo: Crea una función llamada `esPalindromo` que reciba una palabra y devuelva `true` si es un palíndromo (se lee igual al derecho y al revés) y `false` si no lo es.

```
function esPalindromo(palabra) {
  // Tu código aquí
}

console.log(esPalindromo("radar")); // Debería imprimir: true
console.log(esPalindromo("hola")); // Debería imprimir: false
```