

Unidad 4.- Programación con arrays, funciones y objetos definidos por el usuario:

- a) Funciones predefinidas del lenguaje.
- b) Llamadas a funciones. Definición de funciones.
- c) Arrays.
- d) Creación de objetos. Definición de métodos y propiedades.

a) Funciones predefinidas del lenguaje.

Las funciones predefinidas vienen dadas por el lenguaje. Vamos a ver algunas de estas funciones:

parseInt(*cadena* [, *base*]) devuelve un número entero resultante de convertir el número representado por la cadena a entero. Con base se indica la base en la que se expresa el número, si no se indica la base, tomará ésta en función de los primeros caracteres, si empieza por cero la base será 8, octal, si empieza por 0x la base será 16, hexadecimal, y por cualquier otro dígito la base es 10, decimal. Si la cadena empieza por un/os dígito/s y a continuación encuentra un carácter que no es dígito la conversión la realiza con el valor de la cadena hasta el primer carácter no dígito.

parseFloat(*cadena*) devuelve un número en coma flotante, que es el valor representado por la cadena.

isNaN(*valor*) devuelve un valor lógico que indica si el valor es NaN.

eval(*expresión*) devuelve el valor de la expresión, si realizamos la concatenación de cadenas y ésta representa una variable u objeto, va a devolver la referencia a la variable u objeto.

Number(*cadena*) devuelve un número con el valor de la cadena.

String(*valor*) devuelve una cadena con el valor indicado.

isFinite(*valor*) devuelve un valor lógico que nos indica si el valor es finito, devuelve false cuando el valor es infinito, -infinito o NaN.

escape(*cadena*) devuelve una cadena que es una copia de la original, en la cual los caracteres no ASCII aparecen escapados, con \xx.

unescape(*cadena*) devuelve una cadena que es una copia de la original en la cual los caracteres escapados aparecen con su valor.

b) Definición de funciones. Llamadas a funciones.

Para realizar la definición de una función deberemos poner:

```
function nombre-función( [lista-parámetros] ){  
    instrucciones  
}
```

Nos permite definir una función en la cual la lista de parámetros va a ser el nombre de los mismos. Dentro de las instrucciones nos vamos a encontrar con la instrucción **return** (en las funciones vamos a poner una única instrucción return) que seguida de un valor devolverá dicho valor, si no se pone el valor no va a devolver nada, o también se puede poner en este caso para que no devuelva nada **return null**. Admite recursividad. Los parámetros siempre se van a pasar por valor. Una forma de pasar parámetros por referencia es pasar una matriz o un objeto.

001	function sumando(primer, segundo){
002	let suma;
003	suma = primero + segundo;
004	return suma;
005	}

factorial.js

001	function factorial(numero){
002	if (numero==0){
003	return 1;
004	} else {
005	return numero*factorial(numero-1);
006	}
007	}

Para realizar una llamada a una función deberemos poner

nombre-función([valores-parámetros])

La llamada a la función la podemos poner sola en una línea, si no devuelve valores o si los queremos ignorar o bien en una expresión del mismo tipo que el valor devuelto por la función.

001	result=mifuncion(indice,suma);
-----	--------------------------------

Otra forma de declarar una función es:

var nombre-variable=function() {
 cuerpo-función
}

Declarar una función que es asignada a una variable. La diferencia entre la primera y la segunda declaración está en el tratamiento. Mientras que la primera declaración de la función se compila al inicio y se mantiene hasta que se necesita la segunda es compilada y ejecutada según se va leyendo.

001	var mia = function () {
002	console.log(arguments.length);
003	console.log(arguments);
004	for (var i=0; i< arguments.length ; i++){
005	console.log(arguments[i]);
006	}
007	return ;
008	}
009	mia("hola","prueba",13,45);

Otra forma de definir una función es:

window[nombre-función]=new Function(lista-argumento, cuerpo-función);

De esta forma se pueden crear funciones de manera dinámica, ya que tanto el nombre de la función, como los parámetros y el cuerpo de la función pueden estar contenidas en variables.

001	var nombreFuncion='cuadrado';
002	var argumentoFuncion='x';
003	var codigoFuncion='return x * x;'
004	window [nombreFuncion]= new Function(argumentoFuncion,codigoFuncion);
005	alert (window [nombreFuncion](3));

Existe la posibilidad de declarar una función sin parámetros, aunque luego se la pueda pasar un número indeterminado de parámetros. En este caso dentro de la función vamos a tener una variable llamada **arguments**, que va a ser un array.

001	<code>function mia(){</code>
002	<code> console.log(arguments.length);</code>
003	<code> console.log(arguments);</code>
004	<code> for(let i=0; i< arguments.length ; i++){</code>
005	<code> console.log(arguments[i]);</code>
006	<code> }</code>
007	<code> return;</code>
008	<code>}</code>
009	<code>mia("hola","prueba",13,45);</code>

Parámetros opcionales

```
function nombre-función(parámetro1=valor1, ...) {
  cuerpo
}
```

Se deben poner los parámetros opcionales al final

ejemplo-04-06.js

001	<code>function opera(first, second=0, thersty=2){</code>
002	<code> return first + second + thersty;</code>
003	<code>}</code>
004	<code>document.writeln(opera(45)+"
");</code>

Podemos poner que a partir de un determinado parámetro vamos a poder tener un número indeterminado de parámetros más, que se van a agrupar en un parámetro que va a ser tratado como un array (estamos declarando un número mínimo de parámetros, que son los que van al inicio y luego un número indeterminado de parámetros), al poner ese parámetro le vamos a poner un prefijo de tres puntos seguidos. La forma de declararlo será

```
function nombre-función(parámetro1, parámetro2, ...parámetro3) {
  cuerpo
}
```

El parámetro3 recibe todos los parámetros que se le pasan a la función a partir del tercero y dentro del cuerpo de la función se trata como un array.

ejemplo-04-05.js

001	<code>function operaciones(one, two,...other){</code>
002	<code> let sumar = one + two;</code>
003	<code> for(let i=0; i < other.length;i++)</code>
004	<code> sumar += other[i];</code>
005	<code> return sumar;</code>
006	<code>}</code>
007	<code>document.writeln(operaciones(2,4,6,8)+"
");</code>

Funciones simples

function(*parámetros*) *instrucción-del-return*

ejemplo-04-03.js

001	function sumar(primero,segundo){
002	let suma = primero + segundo;
003	return suma
004	}
005	var uno = sumar(12,24);
006	function sumando(primero,segundo) primero+segundo;
007	var dos = sumando(24,12);
008	document .writeln(uno + " ");
009	document .writeln(dos + " ");

Funciones que devuelven varios valores

return [*lista-valores*]

Los valores los puede recibir un array o bien varias variables, en este caso se deben poner los nombres de las variables encerradas entre corchetes.

ejemplo-04-02.js

001	function operaciones(primero,segundo){
002	let suma = primero + segundo;
003	let resta = primero - segundo;
004	let multi = primero * segundo;
005	let divi = primero / segundo;
006	let poten = primero ** segundo;
007	return [suma,resta, multi, divi, poten]
008	}
009	var todos=new Array();
010	todos=operaciones(8,2);
011	[uno,dos,tres, cuatro, cinco]=operaciones(4,2);
012	document .writeln(uno+" ");
013	document .writeln(dos+" ");
014	document .writeln(tres+" ");
015	document .writeln(cuatro+" ");
016	document .writeln(cinco+" ");
017	for (let i=0; i < todos.length ; i++){
018	document .writeln(todos[i]+" ");
019	}

Funciones flecha.

Si tenemos una función del tipo

```
function nombre-función([parámetros]) {
    cuerpo-función;
    return valor;
}
```

Se puede transformar en una función anónima, haciéndola una función flecha, para lo cual deberemos realizar la siguiente transformación

```
var nombre-función= ([parámetros]) => {
    cuerpo-función;
    return valor;
}
```

Estas funciones admiten cualquier tipo de parámetros que hemos visto anteriormente, a excepción de **arguments** que no va a existir de por sí en la función.

Si la función solamente tiene una instrucción **return** no es necesario poner las llaves ni poner la palabra clave **return**.

```
var nombre-función= ([parámetros]) =>expresión;
```

Si solamente se tiene un parámetro no es necesario poner los paréntesis.

```
var nombre-función= parámetro => {  
    cuerpo-función;  
    return valor;  
}
```

o bien

```
var nombre-función= parámetro =>expresión;
```

ejemplo-04-04.js

001	function suma (primero, segundo){
002	return primero + segundo;
003	}
004	var suma1 =(primero, segundo)=>{
005	return primero + segundo;
006	};
007	var suma2 =(primero,segundo)=> primero + segundo;
008	function doble(uno){
009	return uno *2;
010	}
011	var doble1 =(uno)=>{
012	return uno *2;
013	};
014	var doble2 = uno =>{
015	return uno *2;
016	};
017	var doble3 =(uno)=> uno *2;
018	var doble4 = uno => uno *2;
019	
020	function operaciones(one, two,...other){
021	let sumar = one + two;
022	for(let i=0; i < other.length;i++)
023	sumar += other[i];
024	return sumar;
025	}
026	var operaciones1 =(one, two,...other)=>{
027	let sumar = one + two;
028	for(let i=0; i < other.length;i++)
029	sumar += other[i];
030	return sumar;
031	}
032	function opera(first, second=0, thersty=2){
033	var res1=first + second + thersty;
034	var res2=first * thersty;
035	var res3=first / thersty;
036	return[res1,res2,res3];
037	}
038	var opera1=(first, second=0, thersty=2)=>{
039	let res1=first + second + thersty;
040	let res2=first * thersty;
041	let res3=first / thersty;

042	<code>return [res1,res2,res3];</code>
043	<code>}</code>
044	<code>document.writeln(suma(2,4)+"
");</code>
045	<code>document.writeln(suma1(2,4)+"
");</code>
046	<code>document.writeln(suma2(2,4)+"
");</code>
047	<code>document.writeln(doble(5)+"
");</code>
048	<code>document.writeln(doble1(5)+"
");</code>
049	<code>document.writeln(doble2(5)+"
");</code>
050	<code>document.writeln(doble3(5)+"
");</code>
051	<code>document.writeln(doble4(5)+"
");</code>
052	<code>document.writeln(operaciones(2,4,6,8)+"
");</code>
053	<code>document.writeln(operaciones1(2,4,6,8)+"
");</code>
054	<code>[ope1,ope2,ope3]= opera(45)</code>
055	<code>document.writeln(ope1+"
");</code>
056	<code>document.writeln(ope2+"
");</code>
057	<code>document.writeln(ope3+"
");</code>
058	<code>[ope1,ope2,ope3]= opera1(45)</code>
059	<code>document.writeln(ope1+"
");</code>
060	<code>document.writeln(ope2+"
");</code>
061	<code>document.writeln(ope3+"
");</code>

Función de generadora

```
function* nombre-función([lista-parámetros]) {  
    cuerpo-función  
}
```

La forma de hacer referencia a la función es

```
nombre-variable=nombre-función([lista-valores])
```

Para que se ejecute la función y obtener el valor devuelto deberemos poner

```
nombre-variable.next().value
```

El método **next()** hace que se ejecute la función y con la propiedad **value** obtener el valor devuelto.

ejemplo-04-50.js

001	<code>function* sumatorio(){</code>
002	<code>let suma=0;</code>
003	<code>let i;</code>
004	<code>for(i=0;i < arguments.length;i++){</code>
005	<code> suma+=arguments[i];</code>
006	<code>}</code>
007	<code>return suma;</code>
008	<code>}</code>
009	<code>var sumando=sumatorio(12,23,13,45,25,56,37,53,74,83,16,94,84);</code>
010	<code>document.writeln(sumando.next().value);</code>

Dentro de la función podemos poner la instrucción **yield**

```
yield expresión
```

Detiene la ejecución de la función hasta que se vuelva a llamar y devuelve un objeto de tipo **yield** compuesto por dos propiedades, que son: **value** que se corresponde con el valor de la expresión y **done** que nos indica si se ha terminado la función a través de un valor lógico.

Para mandar ejecutar la función y que se vaya reanudando la función tenemos el método **next()**, que además devuelve el objeto **yield** de la función.

ejemplo-04-51.js

001	function* sumatorio(){
002	let suma=0;
003	let i;
004	for (i=0;i < arguments.length;i++){
005	suma+=arguments[i];
006	yield suma;
007	}
008	return suma;
009	}
010	var sumando=sumatorio(12,23,13,45,25,56,37,53,74,83,16,94,84);
011	var dato=sumando.next();
012	while (!dato.done){
013	document .writeln(dato.value);
014	dato=sumando.next();
015	}

Además también podemos utilizar la instrucción

yield* *nombre-función-generadora*(*lista-valores*)

Realiza una llamada a la función generadora con el valor del parámetro

ejemplo-04-52.js

001	function* duplicado(numero){
002	yield numero*2;
003	}
004	function* sumatorio(){
005	let suma=0;
006	var i;
007	let (i=0;i < arguments.length;i++){
008	suma+=arguments[i];
009	yield* duplicado(suma);
010	}
011	return suma;
012	}
013	var sumando=sumatorio(12,23,13,45,25,56,37,53,74,83,16,94,84);
014	var dato=sumando.next();
015	while (!dato.done){
016	document .writeln(dato.value);
017	dato=sumando.next();
018	}

c) Arrays.

Un array es un conjunto de celdas, que almacenan diversos valores y que son nombrados mediante un nombre y la posición que ocupan dentro de la estructura.

En JavaScript los arrays se empiezan a numerar por el 0. Un array puede contener valores de diferentes tipos localizados en distintas posiciones.

En los arrays la dimensión no es importante, ya que en cualquier momento se puede modificar añadiendo un nuevo elemento al array.

Para realizar la declaración de un array podemos utilizar diversos formatos como son:

var *nombre-array*= new Array()

Nos declaramos un array sin dimensión.


```
001 var tabla=new Array();
```

var nombre-array= new Array(*lista-valores*)

Nos declaramos un array, que va a tener tantos elementos como valores se indican, los valores están separados por comas.

```
001 var matriz=new Array("Juan","Pedro",13,true);
```

var nombre-array = new Array(*número-elementos*)

Nos declara un array con tantos elementos como se indican.

```
001 var arreglo=new Array(9);
```

var nombre-array=[]

Nos declaramos un array sin dimensión.

```
001 var datos=[];
```

var nombre-array=[*lista-valores*]

Nos declaramos un array, que va a tener tantos elementos como valores se indican, los valores están separados por comas.

```
001 var conjunto=[13,56,78,"Luis"];
```

Para acceder a un elemento del array deberemos poner

***nombre-array*[posición]**

```
001 conjunto[2]
```

Para añadir un elemento bastará con asignar valor a un elemento que ocupa una posición posterior al último elemento.

```
001 conjunto[9]="Leonor";
```

Los arrays disponen de las siguientes propiedades:

- ♦ **length**: contiene el número de elementos del array, en un array multidimensional devuelve el número de elementos de la primera dimensión.

ejemplo-04-351-length

```
001 var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda",  
"Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz",  
"Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino",  
"Vanessa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis",  
"Paz" );  
002 document.writeln(`Elementos del array <b>${VstNombres}</b> <br />`);  
003 var VitLongitud=VstNombres.length;  
004 document.writeln(`El número de elementos del array es de <b>${VitLongitud}</b> <br /><br />`);
```

Resultado

```
001 Elementos del array Juan,Lucas,María,Angel,Fuenciasla,Yolanda,Julia,Almudena,Félix, María,  
Isabel,Pedro,Inés,Julian,Beatriz,Angel,Consuelo,María,Carlos,Elena,Fernando,Ursula,Marino,  
Vanessa,Guillermo,Rosa,Ismael,María,Soledad,Javier,Olga,Luis,Paz
```

002	El número de elementos del array es de 33
-----	---

Los arrays disponen de los siguientes métodos:

- ♦ **at(posición)**: devuelve el elemento del array, que ocupa la posición indica. Si posición es un número negativo va a devolver el elemento que ocupa la posición correspondiente a retroceder tantos elementos como valor absoluto tiene posición, el valor -1 hacer referencia al último elemento del array.

ejemplo-04-352-at.js

001	<code>var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanessa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</code>
002	<code>document.writeln("Elementos del array \${VstNombres}
");</code>
003	<code>var VstValor=VstNombres.at(5);</code>
004	<code>document.writeln("Elemento del array que ocupa la posición 5 \${VstValor}
");</code>
005	<code>VstValor=VstNombres.at(-4);</code>
006	<code>document.writeln("Elemento del array que ocupa la posición -4 \${VstValor}

");</code>

Resultado

001	Elementos del array Juan, Lucas, María, Angel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Angel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
002	Elemento del array que ocupa la posición 5 Yolanda
003	Elemento del array que ocupa la posición -4 Javier

- ♦ **shift()**: devuelve el valor del primer elemento del array y le elimina.

ejemplo-04-362-shift.js

001	<code>var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanessa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</code>
002	<code>document.writeln("Elementos del array \${VstNombres}
");</code>
003	<code>var VstPrimero=VstNombres.shift();</code>
004	<code>document.writeln("Elementos del array borrado el primero elemento \${VstNombres}
");</code>
005	<code>document.writeln("Elemento eliminado, el primero del array \${VstPrimero}

");</code>

Resultado

001	Elementos del array Juan, Lucas, María, Angel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Angel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
002	Elementos del array borrado el primero elemento Lucas, María, Angel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Angel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
003	Elemento eliminado, el primero del array Juan

- ♦ **pop()**: devuelve el valor del último elemento del array y le elimina.

ejemplo-04-363-pop.js

001	<code>var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanessa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</code>
002	<code>document.writeln("Elementos del array \${VstNombres}
");</code>
003	<code>var VstUltimo=VstNombres.pop();</code>
004	<code>document.writeln("Elementos del array borrado el último elemento \${VstNombres}
");</code>
005	<code>document.writeln("Elemento eliminado, el primero del array \${VstUltimo}

");</code>

Resultado

001	Elementos del array Juan, Lucas, María, Ángel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Ángel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
002	Elementos del array borrado el último elemento Juan, Lucas, María, Ángel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Ángel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis
003	Elemento eliminado, el primero del array Paz

- ♦ **push(lista-valores)**: añade los valores indicados al final del array, cada uno de ellos en una nueva posición.

ejemplo-04-365-push.js

001	<pre>var VstNombres = new Array("Juan", "Lucas", "María", "Ángel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Ángel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanesa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</pre>
002	<pre>document.writeln(`Elementos del array \${VstNombres}
`);</pre>
003	<pre>VstNombres.push("Salvador");</pre>
004	<pre>document.writeln(`Elementos del array habiendo añadido un elemento al final \${VstNombres}

`);</pre>
005	<pre>document.writeln(`Elementos del array \${VstNombres}
`);</pre>
006	<pre>VstNombres.push("Rodrigo", "Manuel", "Leonardo", "Jesús");</pre>
007	<pre>document.writeln(`Elementos del array habiendo añadido cuatro elementos al inicio \${VstNombres}

`);</pre>

Resultado

001	Elementos del array Juan, Lucas, María, Ángel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Ángel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
002	Elementos del array habiendo añadido un elemento al final Juan, Lucas, María, Ángel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Ángel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz, Salvador
003	Elementos del array Juan, Lucas, María, Ángel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Ángel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz, Salvador
004	Elementos del array habiendo añadido cuatro elementos al inicio Juan, Lucas, María, Ángel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Ángel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz, Salvador, Rodrigo, Manuel, Leonardo, Jesús

- ♦ **unshift(lista-valores)**: añade los valores indicados al inicio del array, cada uno de ellos en una nueva posición, desplazando los que había en esas posiciones.

ejemplo-04-364-unshift.js

001	<pre>var VstNombres = new Array("Juan", "Lucas", "María", "Ángel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Ángel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanesa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</pre>
002	<pre>document.writeln(`Elementos del array \${VstNombres}
`);</pre>
003	<pre>VstNombres.unshift("Antonio");</pre>
004	<pre>document.writeln(`Elementos del array habiendo añadido un elemento al inicio \${VstNombres}

`);</pre>
005	<pre>document.writeln(`Elementos del array \${VstNombres}
`);</pre>
006	<pre>VstNombres.unshift("Cesar", "Domingo", "Eliás");</pre>
007	<pre>document.writeln(`Elementos del array habiendo añadido tres elementos al inicio \${VstNombres}

`);</pre>

Resultado

001	Elementos del array Juan, Lucas, María, Ángel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Ángel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
002	Elementos del array habiendo añadido un elemento al inicio Antonio, Juan, Lucas, María, Ángel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Ángel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
003	Elementos del array

	Antonio, Juan, Lucas, María, Angel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Angel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
004	Elementos del array habiendo añadido tres elementos al inicio Cesar, Domingo, Elias, Antonio, Juan, Lucas, María, Angel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Angel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz

- ♦ **splice(*inicio*, *nºelemento* [, *lista-valores*])**: elimina a partir de la posición indicada por inicio tanto elementos como se indican, al mismo tiempo se pueden añadir los valores indicados, cada uno en un elemento, a partir de la posición indicada. Devuelve un array con los elementos eliminados.

ejemplo-04-355-splice.js

001	<pre>var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanesa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</pre>
002	<pre>var VstNombr=VstNombres.slice(0);</pre>
003	<pre>var VstNombres=VstNombr.splice(4,8);</pre>
004	<pre>document.writeln("Elementos del array \${VstNombres}
");</pre>
005	<pre>document.writeln("Elementos del array habiendo borrado desde la posición 4, 8 elementos\${VstNombr}
");</pre>
006	<pre>document.writeln("Elementos borrados del array desde la posición 4, 8 elementos\${VstNombres}

");</pre>
007	<pre>VstNombr=VstNombres.slice(0);</pre>
008	<pre>var VstNombres=VstNombr.splice(12,3, "Alba", "Begoñz", "Carmen", "Elisa");</pre>
009	<pre>document.writeln("Elementos del array \${VstNombres}
");</pre>
010	<pre>document.writeln("Elementos del array habiendo borrado desde la posición 12, 3 elementos y se han insertado 4 elementos \${VstNombr}
");</pre>
011	<pre>document.writeln("Elementos borrados del array desde la posición 12, 3 elementos \${VstNombres}

");</pre>

Resultado

001	Elementos del array Juan, Lucas, María, Angel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Angel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
002	Elementos del array habiendo borrado desde la posición 4, 8 elementos Juan, Lucas, María, Angel, Inés, Julian, Beatriz, Angel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
003	Elementos borrados del array desde la posición 4, 8 elementos Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro
004	Elementos del array Juan, Lucas, María, Angel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Angel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
005	Elementos del array habiendo borrado desde la posición 12, 3 elementos y se han insertado 4 elementos Juan, Lucas, María, Angel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Alba, Begoñz, Carmen, Elisa, Angel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
006	Elementos borrados del array desde la posición 12, 3 elementos Inés, Julian, Beatriz

- ♦ **toSpliced(*inicio*, *nºelemento* [, *lista-valores*])**: devuelve un array que es una copia del inicial en el que se ha eliminado a partir de la posición indicada por inicio tanto elementos como se indican, al mismo tiempo se pueden añadir los valores indicados, cada uno en un elemento, a partir de la posición indicada.

ejemplo-04-379-tospliced.js

001	<pre>var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanesa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</pre>
002	<pre>var VstNombres=VstNombres.toSpliced(4,8);</pre>
003	<pre>document.writeln("Elementos del array \${VstNombres}
");</pre>

004	<code>document.writeln(`Elementos borrados del array desde la posición 4, 8 elementos\${VstNombres}

`);</code>
005	<code>VstNombres=VstNombres.splice(12,3, "Alba", "Begoñz", "Carmen", "Elisa");</code>
006	<code>document.writeln(`Elementos del array \${VstNombres}
`);</code>
007	<code>document.writeln(`Elementos borrados del array desde la posición 12, 3 elementos \${VstNombres}

`);</code>

Resultado

001	Elementos del array Juan, Lucas, María, Ángel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Ángel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
002	Elementos borrados del array desde la posición 4, 8 elementos Juan, Lucas, María, Ángel, Inés, Julian, Beatriz, Ángel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
003	Elementos del array Juan, Lucas, María, Ángel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Ángel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
004	Elementos borrados del array desde la posición 12, 3 elementos Juan, Lucas, María, Ángel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Alba, Begoñz, Carmen, Elisa, Ángel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz

- ♦ **with(posición, valor):** devuelve un nuevo array, que es una copia del inicial en el que se ha sustituido el valor del elemento que ocupa la posición indicada, teniendo como nuevo valor el indicado.

ejemplo-04-378-with.js

001	<code>var VstNombres = new Array("Juan", "Lucas", "María", "Ángel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Ángel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanesa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</code>
002	<code>var VstNombres=VstNombres.with(4, "Segismundo");</code>
003	<code>document.writeln(`Elementos del array \${VstNombres}
`);</code>
004	<code>document.writeln(`Elementos del array sustituido elemento 4 por Segismundo \${VstNombres}

`);</code>

Resultado

001	Elementos del array Juan, Lucas, María, Ángel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Ángel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
002	Elementos del array sustituido elemento 4 por Segismundo Juan, Lucas, María, Ángel, Segismundo, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Ángel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz

- ♦ **reverse():** invierte los elementos del array y devuelve un array con los elementos en orden inverso.

ejemplo-04-353-reverse.js

001	<code>var VstNombres = new Array("Juan", "Lucas", "María", "Ángel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Ángel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanesa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</code>
002	<code>var VstNombr=VstNombres.slice(0);</code>
003	<code>var VstNomuno=VstNombr.reverse();</code>
004	<code>document.writeln(`Elementos del array en orden inverso \${VstNomuno}
`);</code>
005	<code>document.writeln(`Elementos del array en orden inverso \${VstNombr}
`);</code>
006	<code>document.writeln(`Elementos del array \${VstNombres}

`);</code>

Resultado

001	Elementos del array en orden inverso Paz, Luis, Olga, Javier, Soledad, María, Ismael, Rosa, Guillermo, Vanesa, Marino, Ursula, Fernando, Elena, Carlos, María, Consuelo, Ángel, Beatriz, Julian, Inés, Pedro, Isabel, María, Félix, Almudena, Julia, Yolanda, Fuenciasla, Ángel, María, Lucas, Juan
002	Elementos del array en orden inverso Paz, Luis, Olga, Javier, Soledad, María, Ismael, Rosa, Guillermo, Vanesa, Marino, Ursula, Fernando, Elena, Carlos, María, Consuelo, Ángel, Beatriz, Julian, Inés, Pedro, Isabel, María, Félix, Almudena, Julia, Yolanda, Fuenciasla, Ángel, María, Lucas, Juan
003	Elementos del array Juan, Lucas, María, Ángel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Ángel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz

♦ **toReversed()**: devuelve un array con los elementos en orden inverso.

ejemplo-04-385-toReversed.js

001	<code>var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanessa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</code>
002	<code>var VstNono=VstNombres.toReversed();</code>
003	<code>document.writeln("Elementos del array \${VstNombres}
");</code>
004	<code>document.writeln("Elementos del array ordenados \${VstNono}
");</code>

Resultado

001	Elementos del array Juan,Lucas,María,Angel,Fuenciasla,Yolanda,Julia,Almudena,Félix,María,Isabel,Pedro,Inés,Julian,Beatriz,Angel,Consuelo,María,Carlos,Elena,Fernando,Ursula,Marino,Vanessa,Guillermo,Rosa,Ismael,María,Soledad,Javier,Olga,Luis,Paz
002	Elementos del array ordenados Paz,Luis,Olga,Javier,Soledad,María,Ismael,Rosa,Guillermo,Vanessa,Marino,Ursula,Fernando,Elena,Carlos,María,Consuelo,Angel,Beatriz,Julian,Inés,Pedro,Isabel,María,Félix,Almudena,Julia,Yolanda,Fuenciasla,Angel,María,Lucas,Juan

♦ **sort()**: ordena el array y devuelve una copia del array ordenado.

ejemplo-04-354-sort.js

001	<code>var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanessa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</code>
002	<code>var VstNombr=VstNombres.slice(0);</code>
003	<code>var VstNomdos=VstNombr.sort();</code>
004	<code>document.writeln("Elementos del array ordenados \${VstNomdos}
");</code>
005	<code>document.writeln("Elementos del array en ordenados \${VstNombr}
");</code>
006	<code>document.writeln("Elementos del array \${VstNombres}

");</code>

Resultado

001	Elementos del array ordenados Almudena,Angel,Angel,Beatriz,Carlos,Consuelo,Elena,Fernando,Fuenciasla,Félix,Guillermo,Inés,Isabel,Ismael,Javier,Juan,Julia,Julian,Lucas,Luis,Marino,María,María,María,María,Olga,Paz,Pedro,Rosa,Soledad,Ursula,Vanessa,Yolanda
002	Elementos del array ordenados Almudena,Angel,Angel,Beatriz,Carlos,Consuelo,Elena,Fernando,Fuenciasla,Félix,Guillermo,Inés,Isabel,Ismael,Javier,Juan,Julia,Julian,Lucas,Luis,Marino,María,María,María,María,Olga,Paz,Pedro,Rosa,Soledad,Ursula,Vanessa,Yolanda
003	Elementos del array Juan,Lucas,María,Angel,Fuenciasla,Yolanda,Julia,Almudena,Félix,María,Isabel,Pedro,Inés,Julian,Beatriz,Angel,Consuelo,María,Carlos,Elena,Fernando,Ursula,Marino,Vanessa,Guillermo,Rosa,Ismael,María,Soledad,Javier,Olga,Luis,Paz

♦ **toSorted()**: devuelve una copia del array ordenado.

ejemplo-04-380-toSorted.js

001	<code>var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanessa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</code>
002	<code>var VstNono=VstNombres.toSorted();</code>
003	<code>document.writeln("Elementos del array \${VstNombres}
");</code>
004	<code>document.writeln("Elementos del array ordenados \${VstNono}
");</code>

Resultado

001	Elementos del array Juan,Lucas,María,Angel,Fuenciasla,Yolanda,Julia,Almudena,Félix,María,Isabel,Pedro,Inés,Julian,Beatriz,Angel,Consuelo,María,Carlos,Elena,Fernando,Ursula,Marino,Vanessa,Guillermo,Rosa,Ismael,María,Soledad,Javier,Olga,Luis,Paz
002	Elementos del array ordenados Almudena,Angel,Angel,Beatriz,Carlos,Consuelo,Elena,Fernando,Fuenciasla,Félix,Guillermo,Inés,Isabel,Ismael,Javier,Juan,Julia,Julian,Lucas,Luis,Marino,María,María,María,María,Olga,Paz,Pedro,Rosa,Soledad,Ursula,Vanessa,Yolanda

♦ **slice(inicio [, último])**: devuelve un array con los elementos existentes entre el inicio y el final o bien hasta el último, excluido este último.

ejemplo-04-359-slice.js

001	<code>var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanessa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</code>
-----	---

	"Vanessa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");
002	document.writeln(`Elementos del array \${VstNombres} `);
003	var VstSegundo=VstNombres.slice(20);
004	document.writeln(`Elementos a partir de la posición 20 \${VstSegundo} `);
005	var VstTercero=VstNombres.slice(5,12);
006	document.writeln(`Elementos de la posición 5 a la 12 \${VstTercero} `);
<i>Resultado</i>	
001	Elementos del array Juan, Lucas, María, Ángel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Ángel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
002	Elementos a partir de la posición 20 Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
003	Elementos de la posición 5 a la 12 Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro

- ♦ **indexOf(valor [, inicio]):** devuelve la posición que ocupa la primera aparición del valor indicado dentro del array, empezando la búsqueda por el primer elemento o por la posición de inicio; la búsqueda se realiza de inicio a fin. Si no encuentra el valor en el array devuelve -1.

ejemplo-04-356-indexof.js

001	var VstNombres = new Array("Juan", "Lucas", "María", "Ángel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Ángel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanesa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");
002	document.writeln(`Elementos del array \${VstNombres} `);
003	var VitPosuno=VstNombres.indexOf("María");
004	document.writeln(`La posición de la primera aparición de María en el array es \${VitPosuno} `);
005	var VitPosdos=VstNombres.indexOf("María", VitPosuno+1);
006	document.writeln(`La posición de la segunda aparición de María en el array es \${VitPosdos} `);
007	var VitPoscinco=VstNombres.indexOf("Felipe");
008	document.writeln(`La posición de la primera aparición de Felipe en el array es \${VitPoscinco} `);
<i>Resultado</i>	
001	Elementos del array Juan, Lucas, María, Ángel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Ángel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
002	La posición de la primera aparición de María en el array es 2
003	La posición de la segunda aparición de María en el array es 9
004	La posición de la primera aparición de Felipe en el array es -1

- ♦ **lastIndexOf(valor [, inicio]):** devuelve la posición que ocupa la primera aparición del valor indicado dentro del array, empezando la búsqueda por el último elemento o por la posición de inicio; la búsqueda se realiza de final al inicio. Si no encuentra el valor en el array devuelve el valor -1.

ejemplo-04-357-lastindexOf.js

001	var VstNombres = new Array("Juan", "Lucas", "María", "Ángel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Ángel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanesa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");
002	document.writeln(`Elementos del array \${VstNombres} `);
003	var VitPostres =VstNombres.lastIndexOf("María");
004	document.writeln(`La posición de la última aparición de María en el array es \${VitPostres} `);
005	var VitPoscuatro=VstNombres.lastIndexOf("María", VitPostres -1);
006	document.writeln(`La posición de la penúltima aparición de María en el array es \${VitPoscuatro} `);
007	var VitPosseis=VstNombres.lastIndexOf("mercedes");
008	document.writeln(`La posición de la última aparición de Mercedes en el array es

	<code>\${VitPosseis}

`);</code>
	<i>Resultado</i>
001	Elementos del array Juan, Lucas, María, Angel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Angel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
002	La posición de la última aparición de María en el array es 27
003	La posición de la penúltima aparición de María en el array es 17
004	La posición de la última aparición de Mercedes en el array es -1

- ◆ **includes(valor [, inicio]):** devuelve un valor lógico que nos indica si el valor se encuentra en el array.

ejemplo-04-358-includes.js

001	<code>var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanesa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</code>
002	<code>document.writeln(`Elementos del array \${VstNombres}
`);</code>
003	<code>if (VstNombres.includes("Ursula"))</code>
004	<code>document.writeln("El nombre Ursula está en el array
")</code>
005	<code>else</code>
006	<code>document.writeln("El nombre Ursula NO está en el array
");</code>
007	<code>if (VstNombres.includes("Angel", 8))</code>
008	<code>document.writeln("El nombre Angel está en el array a partir de la posición 8
")</code>
009	<code>else</code>
010	<code>document.writeln("El nombre Angel NO está en el array a partir de la posición 8
");</code>
011	<code>if (VstNombres.includes("Nuria"))</code>
012	<code>document.writeln("El nombre Nuria está en el array

")</code>
013	<code>else</code>
014	<code>document.writeln("El nombre Nuria NO está en el array

");</code>
	<i>Resultado</i>
001	Elementos del array Juan, Lucas, María, Angel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Angel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
002	El nombre Ursula está en el array
003	El nombre Angel está en el array a partir de la posición 8
004	El nombre Nuria NO está en el array

- ◆ **concat(array):** devuelve un array que es la concatenación del array del objeto con el array suministrado.

ejemplo-04-360-concar.js

001	<code>var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanesa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</code>
002	<code>var VstNombrado= new Array("Araceli", "Tomas", "Marta", "Cristina", "Ramón", "Sara", "Raquel", "Sergio", "Sandra");</code>
003	<code>var VstNomcuatro=VstNombres.concat(VstNombrado);</code>
004	<code>document.writeln(`Elementos del array \${VstNombres}
`);</code>
005	<code>document.writeln(`Elementos del nuevo array \${VstNombrado}
`);</code>
006	<code>document.writeln(`Elementos del array concatenado \${VstNomcuatro}

`);</code>
	<i>Resultado</i>
001	Elementos del array Juan, Lucas, María, Angel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Angel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
002	Elementos del nuevo array Araceli, Tomas, Marta, Cristina, Ramón, Sara, Raquel, Sergio, Sandra
003	Elementos del array concatenado Juan, Lucas, María, Angel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Angel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz, Araceli, Tomas, Marta, Cristina, Ramón, Sara, Raquel, Sergio, Sandra

- ♦ **join(*carácter*)**: devuelve una cadena con todos los elementos del array separados por el carácter indicado.

ejemplo-04-361-join.js

001	<code>var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanessa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</code>
002	<code>document.writeln(`Elementos del array \${VstNombres}
`);</code>
003	<code>var VstCadena=VstNombres.join("#");</code>
004	<code>document.writeln(`Cadena con los elementos del array separados por "#` \${VstCadena}

`);</code>

Resultado

001	Elementos del array Juan,Lucas,María,Angel,Fuenciasla,Yolanda,Julia,Almudena,Félix,María,Isabel,Pedro,Inés,Julian,Beatriz,Angel,Consuelo,María,Carlos,Elena,Fernando,Ursula,Marino,Vanessa,Guillermo,Rosa,Ismael,María,Soledad,Javier,Olga,Luis,Paz
002	Cadena con los elementos del array separados por "#" Juan#Lucas#María#Angel#Fuenciasla#Yolanda#Julia#Almudena#Félix#María#Isabel#Pedro#Inés#Julian#Beatriz#Angel#Consuelo#María#Carlos#Elena#Fernando#Ursula#Marino#Vanessa#Guillermo#Rosa#Ismael#María#Soledad#Javier#Olga#Luis#Paz

- ♦ **forEach(*función*)**: para cada elemento del array llama a la función con tres parámetros, que son: el valor, la posición y el array completo.

Ejemplo-04-367-foreach.js

001	<code>var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanessa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</code>
002	<code>document.writeln(`Elementos del array \${VstNombres}
`);</code>
003	<code>document.write("Elementos que comienzan con L ");</code>
004	<code>VstNombres.forEach(comienzaL);</code>
005	<code>document.writeln("

");</code>
006	<code>function comienzaL(valor, posicion, todos){</code>
007	<code>let VstResul="";</code>
008	<code>if (valor.startsWith("L"))</code>
009	<code>document.write(valor+" ");</code>
010	<code>}</code>

Resultado

001	Elementos del array Juan,Lucas,María,Angel,Fuenciasla,Yolanda,Julia,Almudena,Félix,María,Isabel,Pedro,Inés,Julian,Beatriz,Angel,Consuelo,María,Carlos,Elena,Fernando,Ursula,Marino,Vanessa,Guillermo,Rosa,Ismael,María,Soledad,Javier,Olga,Luis,Paz
002	Elementos que comienzan con L Lucas Luis

- ♦ **fill(*valor* [, *inicio* [, *final*]])**: devuelve un nuevo array en el que se han rellenados todos los elementos que tiene el array con el valor indicado, si indicamos inicio se indica a partir de qué posición se inicia el relleno y si indicamos final se indica en qué posición se para el relleno, en esa posición no se produce el relleno, se modifica el array.

ejemplo-05-366-fill.js

001	<code>var VstNovedoso= new Array(11);</code>
002	<code>VstNovedoso.fill("Elemento");</code>
003	<code>document.writeln(`Elementos del array 11 elemento con Elemento \${VstNovedoso}
`);</code>
004	<code>VstNovedoso.fill("Número", 3);</code>
005	<code>document.writeln(`Elementos del array a partir de la posición 3 con Número \${VstNovedoso}
`);</code>
006	<code>VstNovedoso.fill("Valor", 5, 9);</code>
007	<code>document.writeln(`Elementos del array a partir de la posición 5 hasta la 9 con Valor \${VstNovedoso}

`);</code>

Resultado

001	Elementos del array 11 elemento con Elemento Elemento,Elemento,Elemento,Elemento,
-----	---

	Elemento,Elemento,Elemento,Elemento,Elemento,Elemento,Elemento
002	Elementos del array a partir de la posición 3 con Número Elemento,Elemento,Elemento, Número,Número,Número,Número,Número,Número,Número
003	Elementos del array a partir de la posición 5 hasta la 9 con Valor Elemento,Elemento, Elemento,Número,Número,Valor,Valor,Valor,Valor,Número,Número

- ♦ **find(nombre-función)**: se ejecuta la función indicada para cada uno de los elementos de la función, esta función va a tener tres parámetros que se corresponden con el valor, la posición y el array. Esta función va a devolver el valor del primer elemento encontrado, si devuelve false ese valor no es tenido en cuenta. La función que ponemos va a devolver true si encuentra el elemento y false en caso contrario.
- ♦ **findIndex(nombre-función)**: se ejecuta la función indicada para cada uno de los elementos de la función, esta función va a tener tres parámetros que se corresponden con el valor, la posición y el array. Esta función va a devolver la posición del primer elemento encontrado, si devuelve false ese valor no es tenido en cuenta. La función que ponemos va a devolver true si encuentra el elemento y false en caso contrario.
- ♦ **findLast(nombre-función)**: se ejecuta la función indicada para cada uno de los elementos de la función, esta función va a tener tres parámetros que se corresponden con el valor, la posición y el array. Esta función va a devolver el valor del primer elemento encontrado empezando por el final, si devuelve false ese valor no es tenido en cuenta. La función que ponemos va a devolver true si encuentra el elemento y false en caso contrario.
- ♦ **findLastIndex(nombre-función)**: se ejecuta la función indicada para cada uno de los elementos de la función, esta función va a tener tres parámetros que se corresponden con el valor, la posición y el array. Esta función va a devolver la posición del primer elemento encontrado empezando por el final, si devuelve false ese valor no es tenido en cuenta. La función que ponemos va a devolver true si encuentra el elemento y false en caso contrario.

ejemplo-04-368-find.js

001	<code>var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanesa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</code>
002	<code>VstA=VstNombres.find(buscaA);</code>
003	<code>VstPosa=VstNombres.findIndex(buscaA);</code>
004	<code>VstUta=VstNombres.findLast(buscaA);</code>
005	<code>VstPosua=VstNombres.findLastIndex(buscaA);</code>
006	<code>function buscaA(valor, posicion, todos){</code>
007	<code> if (valor.startsWith("A"))</code>
008	<code> return true</code>
009	<code> else</code>
010	<code> return false;</code>
011	<code>}</code>
012	<code>document.writeln(`Elementos del array \${VstNombres}
`);</code>
013	<code>document.writeln(`El primer elementos del array que empieza por A es \${VstA}
`);</code>
014	<code>document.writeln(`El primer elementos del array que empieza por A ocupa la posición \${VstPosa}
`);</code>

015	<code>document.writeln('El último elementos del array que empieza por A es \${VstUltima}
');</code>
016	<code>document.writeln('El primer elementos del array que empieza por A ocupa la posición \${VstPosua}

');</code>

Resultado

001	Elementos del array Juan, Lucas, María, Angel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Angel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
002	El primer elementos del array que empieza por A es Angel
003	El primer elementos del array que empieza por A ocupa la posición 3
004	El último elementos del array que empieza por A es Angel
005	El primer elementos del array que empieza por A ocupa la posición 15

- ♦ **entries()**: devuelve un nuevo array iterator que va a tener en cada fila la referencia a los elementos del array inicial y en las columnas va a tener la posición del elemento y el valor del elemento.

ejemplo-04-370-entries.js

001	<code>var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanesa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</code>
002	<code>var VstValores= new Array();</code>
003	<code>VstValores=VstNombres.entries();</code>
004	<code>document.writeln('Elementos del array \${VstNombres}
');</code>
005	<code>var VobVal=VstValores.next();</code>
006	<code>while (!VobVal.done){</code>
007	<code>document.writeln('Elementos del array par valor, clave \${VobVal.value}
');</code>
008	<code>VobVal=VstValores.next();</code>
	<code>}</code>

Resultado

001	Elementos del array Juan, Lucas, María, Angel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Angel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
002	Elementos del array par valor, clave 0, Juan
003	Elementos del array par valor, clave 1, Lucas
004	Elementos del array par valor, clave 2, María
005	Elementos del array par valor, clave 3, Angel
006	Elementos del array par valor, clave 4, Fuenciasla
007	Elementos del array par valor, clave 5, Yolanda
008	Elementos del array par valor, clave 6, Julia
009	Elementos del array par valor, clave 7, Almudena
010	Elementos del array par valor, clave 8, Félix
011	Elementos del array par valor, clave 9, María
012	Elementos del array par valor, clave 10, Isabel
013	Elementos del array par valor, clave 11, Pedro
014	Elementos del array par valor, clave 12, Inés
015	Elementos del array par valor, clave 13, Julian
016	Elementos del array par valor, clave 14, Beatriz
017	Elementos del array par valor, clave 15, Angel
018	Elementos del array par valor, clave 16, Consuelo
019	Elementos del array par valor, clave 17, María
020	Elementos del array par valor, clave 18, Carlos
021	Elementos del array par valor, clave 19, Elena
022	Elementos del array par valor, clave 20, Fernando
023	Elementos del array par valor, clave 21, Ursula
024	Elementos del array par valor, clave 22, Marino
025	Elementos del array par valor, clave 23, Vanesa
026	Elementos del array par valor, clave 24, Guillermo
027	Elementos del array par valor, clave 25, Rosa
028	Elementos del array par valor, clave 26, Ismael
029	Elementos del array par valor, clave 27, María
030	Elementos del array par valor, clave 28, Soledad
031	Elementos del array par valor, clave 29, Javier
032	Elementos del array par valor, clave 30, Olga
033	Elementos del array par valor, clave 31, Luis

034	Elementos del array par valor, clave 32,Paz
-----	---

- ◆ **values()**: devuelve un nuevo array iterador que va a contener los valores de todos los elementos del array inicial.

ejemplo-04-372-values.js

001	<code>var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanessa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</code>
002	<code>var VstValor=VstNombres.values()</code>
003	<code>document.writeln(`Elementos del array \${VstNombres}
`);</code>
004	<code>VobVal=VstValor.next();</code>
005	<code>while (!VobVal.done){</code>
006	<code>document.writeln(`Elementos del array con valor \${VobVal.value}
`);</code>
007	<code>VobVal=VstValor.next();</code>
008	<code>}</code>

Resultado

001	Elementos del array Juan,Lucas,María,Angel,Fuenciasla,Yolanda,Julia,Almudena,Félix,María, Isabel,Pedro,Inés,Julian,Beatriz,Angel,Consuelo,María,Carlos,Elena,Fernando,Ursula,Marino, Vanessa,Guillermo,Rosa,Ismael,María,Soledad,Javier,Olga,Luis,Paz
002	Elementos del array con valor Juan
003	Elementos del array con valor Lucas
004	Elementos del array con valor María
005	Elementos del array con valor Angel
006	Elementos del array con valor Fuenciasla
007	Elementos del array con valor Yolanda
008	Elementos del array con valor Julia
009	Elementos del array con valor Almudena
010	Elementos del array con valor Félix
011	Elementos del array con valor María
012	Elementos del array con valor Isabel
013	Elementos del array con valor Pedro
014	Elementos del array con valor Inés
015	Elementos del array con valor Julian
016	Elementos del array con valor Beatriz
017	Elementos del array con valor Angel
018	Elementos del array con valor Consuelo
019	Elementos del array con valor María
020	Elementos del array con valor Carlos
021	Elementos del array con valor Elena
022	Elementos del array con valor Fernando
023	Elementos del array con valor Ursula
024	Elementos del array con valor Marino
025	Elementos del array con valor Vanessa
026	Elementos del array con valor Guillermo
027	Elementos del array con valor Rosa
028	Elementos del array con valor Ismael
029	Elementos del array con valor María
030	Elementos del array con valor Soledad
031	Elementos del array con valor Javier
032	Elementos del array con valor Olga
033	Elementos del array con valor Luis
034	Elementos del array con valor Paz

- ◆ **keys()**: devuelve un nuevo array iterador que va a contener las posiciones de todos los elementos del array inicial.

ejemplo-04-371-keyss.js

001	<code>var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanessa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</code>
002	<code>var VstValor=VstNombres.keys()</code>
003	<code>document.writeln(`Elementos del array \${VstNombres}
`);</code>
004	<code>VobVal=VstValor.next();</code>

005	<code>while (!VobVal.done){</code>
006	<code>document.writeln(`Elementos del array con posicion/clave \${VobVal.value}
`);</code>
007	<code>VobVal=VstValor.next();</code>
008	<code>}</code>

Resultado

001	Elementos del array Juan, Lucas, María, Angel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Angel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
002	Elementos del array con posicion/clave 0
003	Elementos del array con posicion/clave 1
004	Elementos del array con posicion/clave 2
005	Elementos del array con posicion/clave 3
006	Elementos del array con posicion/clave 4
007	Elementos del array con posicion/clave 5
008	Elementos del array con posicion/clave 6
009	Elementos del array con posicion/clave 7
010	Elementos del array con posicion/clave 8
011	Elementos del array con posicion/clave 9
012	Elementos del array con posicion/clave 10
013	Elementos del array con posicion/clave 11
014	Elementos del array con posicion/clave 12
015	Elementos del array con posicion/clave 13
016	Elementos del array con posicion/clave 14
017	Elementos del array con posicion/clave 15
018	Elementos del array con posicion/clave 16
019	Elementos del array con posicion/clave 17
020	Elementos del array con posicion/clave 18
021	Elementos del array con posicion/clave 19
022	Elementos del array con posicion/clave 20
023	Elementos del array con posicion/clave 21
024	Elementos del array con posicion/clave 22
025	Elementos del array con posicion/clave 23
026	Elementos del array con posicion/clave 24
027	Elementos del array con posicion/clave 25
028	Elementos del array con posicion/clave 26
029	Elementos del array con posicion/clave 27
030	Elementos del array con posicion/clave 28
031	Elementos del array con posicion/clave 29
032	Elementos del array con posicion/clave 30
033	Elementos del array con posicion/clave 31
034	Elementos del array con posicion/clave 32

- ♦ **copyWithin(posición [, inicio [, final]])**: modifica y devuelve el array inicial, en el cual se van a copiar elementos a partir de la posición indicada (positivo se empieza a contar desde el principio, y negativo se empieza a contar desde el final. El primer elemento empezando por la izquierda es cero y el primer elemento empezando por la derecha es -1) e inicialmente los elementos se toman a partir del primer elemento a no ser que se indique la posición, indicada por inicial, en la que se empiezan a copiar. También se puede indicar en qué posición se dejan de coger elementos para copiar, esa posición no se incluye.

ejemplo-04-369-copywhin.js

001	<code>var VstLetrado=new</code> <code>Array("0","1","2","3","4","5","6","7","8","9","10","11","12","13","14","15");</code>
002	<code>document.writeln(`Elementos del array \${VstLetrado}
`);</code>
003	<code>VstLetrado.copyWithin(8);</code>
004	<code>document.writeln(`Elementos del array copia a partir de la posición 8 los elementos</code> <code>iniciales \${VstLetrado}
`);</code>
005	<code>VstLetrado=new</code> <code>Array("0","1","2","3","4","5","6","7","8","9","10","11","12","13","14","15");</code>
006	<code>document.writeln(`Elementos del array \${VstLetrado}
`);</code>
007	<code>VstLetrado.copyWithin(-5);</code>
008	<code>document.writeln(`Elementos del array copia en los últimos 5 elementos los elementos</code>

	<code>iniciales \${VstLetrado}
`);</code>
009	<code>VstLetrado=new Array("0","1","2","3","4","5","6","7","8","9","10","11","12","13","14","15");</code>
010	<code>document.writeln(`Elementos del array \${VstLetrado}
`);</code>
011	<code>VstLetrado.copyWithIn(6,3);</code>
012	<code>document.writeln(`Elementos del array copia a partir de la posición 6 los elementos que hay a partir de la posición 3\${VstLetrado}
`);</code>
013	<code>VstLetrado=new Array("0","1","2","3","4","5","6","7","8","9","10","11","12","13","14","15");</code>
014	<code>document.writeln(`Elementos del array \${VstLetrado}
`);</code>
015	<code>VstLetrado.copyWithIn(4,2,7);</code>
016	<code>document.writeln(`Elementos del array copia a partir de la posición 4 los elementos de las posiciones de la 2 a la 7 \${VstLetrado}

`);</code>

Resultado

001	Elementos del array 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
002	Elementos del array copia a partir de la posición 8 los elementos iniciales 0,1,2,3,4,5,6,7,0,1,2,3,4,5,6,7
003	Elementos del array 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
004	Elementos del array copia en los últimos 5 elementos los elementos iniciales 0,1,2,3,4,5,6,7,8,9,10,0,1,2,3,4
005	Elementos del array 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
006	Elementos del array copia a partir de la posición 6 los elementos que hay a partir de la posición 3 -> 0,1,2,3,4,5,3,4,5,6,7,8,9,10,11,12
007	Elementos del array 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
008	Elementos del array copia a partir de la posición 4 los elementos de las posiciones de la 2 a la 7 -> 0,1,2,3,2,3,4,5,6,9,10,11,12,13,14,15

- ◆ **some(nombre-función | function ([parámetros]){cuerpo})**: devuelve un valor lógico, que nos indica si algún elemento del array cumple una condición, dicha condición se va a poner en la función que ponemos y que devolverá true si se cumple y false en caso contrario (se ejecuta una vez por cada elemento del array).

ejemplo-04-373-some.js

001	<code>var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanessa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</code>
002	<code>var VboLetraF=VstNombres.some(operacion);</code>
003	<code>function operacion(valor){</code>
004	<code>let VboTieneF= true;</code>
005	<code>if (!valor.startsWith("F"))</code>
006	<code>VboTieneF= false;</code>
007	<code>return VboTieneF;</code>
008	<code>}</code>
009	<code>var VboLetraW=VstNombres.some(operac);</code>
010	<code>function operac(valor){</code>
011	<code>let VboTieneF= true;</code>
012	<code>if (!valor.startsWith("W"))</code>
013	<code>VboTieneF= false;</code>
014	<code>return VboTieneF;</code>
015	<code>}</code>
016	<code>document.writeln(`Elementos del array \${VstNombres}
`);</code>
017	<code>document.writeln(`Hay Elementos en el array que empiezan por F \${VboLetraF} <br </>`);</code>
018	<code>document.writeln(`Hay Elementos en el array que empiezan por W \${VboLetraW} <br </>`);</code>

Resultado

001	Elementos del array Juan,Lucas,María,Angel,Fuenciasla,Yolanda,Julia,Almudena,Félix,María, Isabel,Pedro,Inés,Julian,Beatriz,Angel,Consuelo,María,Carlos,Elena,Fernando,Ursula,Marino, Vanessa,Guillermo,Rosa,Ismael,María,Soledad,Javier,Olga,Luis,Paz
002	Hay Elementos en el array que empiezan por F true
003	Hay Elementos en el array que empiezan por W false

- ♦ **every(nombre-función):** devuelve un valor lógico que nos indica si todos los elementos del array cumplen la condición establecida en la función indicada, dicha función devolverá true si se cumple y false en caso contrario. Esa función tiene un parámetro que hace referencia a cada uno de los valores del array. Se llama a la función una vez por cada elemento del array.

ejemplo-04-374-every.js

001	<code>var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanessa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</code>
002	<code>var VboTres=VstNombres.every(longitud);</code>
003	<code>function longitud(valor){</code>
004	<code> return valor.length>=3;</code>
005	<code>}</code>
006	<code>var VboLetrau=VstNombres.every(letrasu);</code>
007	<code>function letrasu(valor){</code>
008	<code> return valor.includes("u");</code>
009	<code>}</code>
010	<code>document.writeln(`Elementos del array \${VstNombres}
`);</code>
011	<code>document.writeln(`Todos los elementos del array tiene una longitud >= 3 \${VboTres}
`);</code>
012	<code>document.writeln(`Todos los elementos del array tiene la letra "u" \${VboLetrau}
`);</code>

Resultado

001	Elementos del array Juan, Lucas, María, Angel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Angel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
002	Todos los elementos del array tiene una longitud >= 3 true
003	Todos los elementos del array tiene la letra "u" false

- ♦ **filter(nombre-función):** devuelve un array con los elementos del array que cumplen la condición establecida en la función indicada, dicha función devolverá true si se cumple y false en caso contrario. Esa función tiene un parámetro que hace referencia a cada uno de los valores del array. Se llama a la función una vez por cada elemento del array.

ejemplo-04-375-filter.js

001	<code>var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanessa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</code>
002	<code>var VarLetrase=VstNombres.filter(lastrado);</code>
003	<code>function lastrado(valor){</code>
004	<code> return valor.includes("e");</code>
005	<code>}</code>
006	<code>document.writeln(`Elementos del array \${VstNombres}
`);</code>
007	<code>document.writeln(`Elementos del array que tienen la letra "e" \${VarLetrase}
`);</code>

Resultado

001	Elementos del array Juan, Lucas, María, Angel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Angel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
002	Elementos del array que tienen la letra "e" Angel, Fuenciasla, Almudena, Isabel, Pedro, Beatriz, Angel, Consuelo, Elena, Fernando, Vanesa, Guillermo, Ismael, Soledad, Javier

- ♦ **map(nombre-función):** devuelve un array con los valores devueltos por la función, la cual se va a ejecutar una vez por cada uno de los valores que tenga el array sobre el que se aplica, esta función tiene un

parámetro que se corresponde con el valor del array que está tratando en ese momento. Si el valor que devuelve le ponemos entre corches nos va a devolver un array bidimensional con tantas filas como elementos tiene el array original y una columna; cada valor que devuelve es un array. Por cada corchete que ponemos en el valor devuelto nos crea un array multidimensional con tantas dimensiones como corchetes ponemos + 1.

ejemplo-04-384-map.js

001	<code>var VarMulti=[2,5,7,11,13,17,23];</code>
002	<code>var VarMul=VarMulti.map(triplicar);</code>
003	<code>function triplicar(valor){</code>
004	<code>return 3 * valor;</code>
005	<code>}</code>
006	<code>document.writeln(`Elementos del array \${VarMulti}
`);</code>
007	<code>document.writeln(`Elementos del nuevo array triplicando los valores del primero array \${VarMul}
`);</code>

Resultado

001	Elementos del array 2,5,7,11,13,17,23
002	Elementos del nuevo array triplicando los valores del primero array 6,15,21,33,39,51,69

- ♦ **flatMap(*nombre-función*)**: devuelve un array con los valores devueltos por la función, la cual se va a ejecutar una vez por cada uno de los valores que tenga el array sobre el que se aplica, esta función tiene un parámetro que se corresponde con el valor del array que está tratando en ese momento. Si el valor que devuelve le ponemos entre corchetes, nos devuelve lo mismo que si no les ponemos. Si el valor que devuelve le ponemos entre doble corches nos va a devolver un array bidimensional con tantas filas como elementos tiene el array original y una columna; cada valor que devuelve es un array. Por cada corchete que ponemos en el valor devuelto nos crea un array multidimensional con tantas dimensiones como corchetes.

ejemplo-04-386-flatmap.js

001	<code>var VarMulti=[2,5,7,11,13,17,23];</code>
002	<code>var VarMul=VarMulti.flatMap(triplicar);</code>
003	<code>var VarMult=VarMulti.map(triplicar);</code>
004	<code>function triplicar(valor){</code>
005	<code>return [3 * valor];</code>
006	<code>}</code>
007	<code>var VarMulo=VarMulti.flatMap(triplico);</code>
008	<code>var VarMulto=VarMulti.map(triplico);</code>
009	<code>function triplico(valor){</code>
010	<code>return [[3 * valor]];</code>
011	<code>}</code>
012	<code>document.writeln(`Elementos del array \${VarMulti}
`);</code>
013	<code>document.writeln(`Elementos del nuevo array triplicando los valores del primero array \${VarMul}
`);</code>
014	<code>document.writeln(`Elementos del array triplicando los valores del primero array \${VarMult}
`);</code>
015	<code>console.log(VarMul);</code>
016	<code>console.log(VarMult);</code>
017	<code>console.log(VarMulti);</code>
018	<code>console.log(VarMulo);</code>
019	<code>console.log(VarMulto);</code>

Resultado

001	Elementos del array 2,5,7,11,13,17,23
002	Elementos del nuevo array triplicando los valores del primero array 6,15,21,33,39,51,69
003	Elementos del array array triplicando los valores del primero array 6,15,21,33,39,51,69

```

▶ Array(7) [ 6, 15, 21, 33, 39, 51, 69 ]
▼ Array(7) [ (1) [–], (1) [–], (1) [–], (1) [–], (1) [–], (1) [–], (1) [–] ]
  ▶ 0: Array [ 6 ]
  ▶ 1: Array [ 15 ]
  ▶ 2: Array [ 21 ]
  ▶ 3: Array [ 33 ]
  ▶ 4: Array [ 39 ]
  ▶ 5: Array [ 51 ]
  ▶ 6: Array [ 69 ]
  length: 7
  ▶ <prototype>: Array []

▶ Array(7) [ 2, 5, 7, 11, 13, 17, 23 ]
▼ Array(7) [ (1) [–], (1) [–], (1) [–], (1) [–], (1) [–], (1) [–], (1) [–] ]
  ▶ 0: Array [ 6 ]
  ▶ 1: Array [ 15 ]
  ▶ 2: Array [ 21 ]
  ▶ 3: Array [ 33 ]
  ▶ 4: Array [ 39 ]
  ▶ 5: Array [ 51 ]
  ▶ 6: Array [ 69 ]
  length: 7
  ▶ <prototype>: Array []

▼ Array(7) [ (1) [–], (1) [–], (1) [–], (1) [–], (1) [–], (1) [–], (1) [–] ]
  ▶ 0: Array [ (1) [–] ]
  ▶ 1: Array [ (1) [–] ]
  ▶ 2: Array [ (1) [–] ]
  ▶ 3: Array [ (1) [–] ]
  ▶ 4: Array [ (1) [–] ]
  ▶ 5: Array [ (1) [–] ]
  ▶ 6: Array [ (1) [–] ]
  length: 7
  ▶ <prototype>: Array []

```

- ◆ **flat([nivel]):** devuelve un array con los elementos del array (array multidimensional) teniendo este nuevo array una dimensión inferior o tantas dimensiones inferiores como indique nivel.

ejemplo-04-383-flat.js

001	var VarMulti=[[1,2,3],[4,5,6],[7,8,9]],[[11,12,13],[14,15,16],[17,18,19]],[[21,22,23],[24,25,26],[27,28,29]]];
002	var VarMult=[[1,2,3],[4,5,6],[7,8,9],[11,12,13],[14,15,16],[17,18,19],[21,22,23],[24,25,26],[27,28,29]]];
003	var VarMul=[[1,2,3],[4,5,6],[7,8,9],[-1,-2,-3]],[[11,12,13],[14,15,16],[17,18,19],[-4,-5,-6],[-7,-8,-9]],[[21,22,23],[24,25,26],[27,28,29]]];
004	document .writeln(`Elementos del array 1 \${VarMulti} `);
005	document .writeln(`Número de elementos del array 1 \${VarMulti.length} `);
006	document .writeln(`Elementos del array 2 \${VarMult} `);
007	document .writeln(`Número de elementos del array 2 \${VarMult.length} `);
008	document .writeln(`Elementos del array 3 \${VarMul} `);
009	document .writeln(`Número de elementos del array 3 \${VarMul.length} `);
010	var VarMulti2=VarMulti.flat();
011	var VarMult2=VarMult.flat();
012	var VarMul2=VarMul.flat(2);
013	document .writeln(`Elementos del array 1, sin una dimensión \${VarMulti2} `);
014	document .writeln(`Número de elementos del array 1, sin una dimensión \${VarMulti2.length} `);
015	document .writeln(`Elementos del array 2, sin dos dimensión \${VarMult2} `);
016	document .writeln(`Número de elementos del array 2, sin dos dimensión \${VarMult2.length} `);
017	document .writeln(`Elementos del array 3, sin una dimensión \${VarMul2} `);
018	document .writeln(`Número de elementos del array 3, sin una dimensión

	<code>\${VarMul2.length}
`);</code>
	<i>Resultado</i>
001	Elementos del array 1 1,2,3,4,5,6,7,8,9,11,12,13,14,15,16,17,18,19,21,22,23,24,25,26,27,28,29
002	Número de elementos del array 1 3
003	Elementos del array 2 1,2,3,4,5,6,7,8,9,11,12,13,14,15,16,17,18,19,21,22,23,24,25,26,27,28,29
004	Número de elementos del array 2 9
005	Elementos del array 3 1,2,3,4,5,6,7,8,9,-1,-2,-3,11,12,13,14,15,16,17,18,19,-4,-5,-6,-7,-8,-9,21,22,23,24,25,26,27,28,29
006	Número de elementos del array 3 3
007	Elementos del array 1, sin una dimensión 1,2,3,4,5,6,7,8,9,11,12,13,14,15,16,17,18,19,21,22,23,24,25,26,27,28,29
008	Número de elementos del array 1, sin una dimensión 9
009	Elementos del array 2, sin dos dimensión 1,2,3,4,5,6,7,8,9,11,12,13,14,15,16,17,18,19,21,22,23,24,25,26,27,28,29
010	Número de elementos del array 2, sin dos dimensión 27
011	Elementos del array 3, sin una dimensión 1,2,3,4,5,6,7,8,9,-1,-2,-3,11,12,13,14,15,16,17,18,19,-4,-5,-6,-7,-8,-9,21,22,23,24,25,26,27,28,29
012	Número de elementos del array 3, sin una dimensión 36


```

▼ Array(3) [ (3) [_, (3) [_, (3) [_] ] ]
  ▼ 0: Array(3) [ (3) [_, (3) [_, (3) [_] ] ]
    ▶ 0: Array(3) [ 1, 2, 3 ]
    ▶ 1: Array(3) [ 4, 5, 6 ]
    ▶ 2: Array(3) [ 7, 8, 9 ]
    length: 3
    <prototype>: Array []
  ▼ 1: Array(3) [ (3) [_, (3) [_, (3) [_] ] ]
    ▶ 0: Array(3) [ 11, 12, 13 ]
    ▶ 1: Array(3) [ 14, 15, 16 ]
    ▶ 2: Array(3) [ 17, 18, 19 ]
    length: 3
    <prototype>: Array []
  ▼ 2: Array(3) [ (3) [_, (3) [_, (3) [_] ] ]
    ▶ 0: Array(3) [ 21, 22, 23 ]
    ▶ 1: Array(3) [ 24, 25, 26 ]
    ▶ 2: Array(3) [ 27, 28, 29 ]
    length: 3
    <prototype>: Array []
    length: 3
    <prototype>: Array []

▼ Array(9) [ (3) [_, (3) [_, (3) [_, (3) [_, (3) [_, (3) [_, (3) [_, (3) [_] ] ] ] ] ] ] ]
  ▶ 0: Array(3) [ 1, 2, 3 ]
  ▶ 1: Array(3) [ 4, 5, 6 ]
  ▶ 2: Array(3) [ 7, 8, 9 ]
  ▶ 3: Array(3) [ 11, 12, 13 ]
  ▶ 4: Array(3) [ 14, 15, 16 ]
  ▶ 5: Array(3) [ 17, 18, 19 ]
  ▶ 6: Array(3) [ 21, 22, 23 ]
  ▶ 7: Array(3) [ 24, 25, 26 ]
  ▶ 8: Array(3) [ 27, 28, 29 ]
  length: 9
  <prototype>: Array []

▼ Array(3) [ (4) [_, (5) [_, (3) [_] ] ]
  ▼ 0: Array(4) [ (3) [_, (3) [_, (3) [_, - ] ] ]
    ▶ 0: Array(3) [ 1, 2, 3 ]
    ▶ 1: Array(3) [ 4, 5, 6 ]
    ▶ 2: Array(3) [ 7, 8, 9 ]
    ▶ 3: Array(3) [ -1, -2, -3 ]
    length: 4
    <prototype>: Array []
  ▼ 1: Array(5) [ (3) [_, (3) [_, (3) [_, - ] ] ]
    ▶ 0: Array(3) [ 11, 12, 13 ]
    ▶ 1: Array(3) [ 14, 15, 16 ]
    ▶ 2: Array(3) [ 17, 18, 19 ]
    ▶ 3: Array(3) [ -4, -5, -6 ]
    ▶ 4: Array(3) [ -7, -8, -9 ]
    length: 5
    <prototype>: Array []
  ▼ 2: Array(3) [ (3) [_, (3) [_, (3) [_] ] ]
    ▶ 0: Array(3) [ 21, 22, 23 ]
    ▶ 1: Array(3) [ 24, 25, 26 ]
    ▶ 2: Array(3) [ 27, 28, 29 ]
    length: 3
    <prototype>: Array []
    length: 3
    <prototype>: Array []

▼ Array(9) [ (3) [_, (3) [_, (3) [_, (3) [_, (3) [_, (3) [_, (3) [_, (3) [_] ] ] ] ] ] ] ]
  ▶ 0: Array(3) [ 1, 2, 3 ]
  ▶ 1: Array(3) [ 4, 5, 6 ]
  ▶ 2: Array(3) [ 7, 8, 9 ]
  ▶ 3: Array(3) [ 11, 12, 13 ]
  ▶ 4: Array(3) [ 14, 15, 16 ]
  ▶ 5: Array(3) [ 17, 18, 19 ]
  ▶ 6: Array(3) [ 21, 22, 23 ]
  ▶ 7: Array(3) [ 24, 25, 26 ]
  ▶ 8: Array(3) [ 27, 28, 29 ]
  length: 9
  <prototype>: Array []

▶ Array(27) [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, - ]
▶ Array(36) [ 1, 2, 3, 4, 5, 6, 7, 8, 9, -1, - ]

```

- ♦ **reduce(*nombre-función* [, *valor-inicial*])**: devuelve un valor correspondiente a realizar las operaciones que se indican en la función, esta función tiene dos parámetros, que son un acumulador y el valor de un elemento del array, tiene otros dos parámetros opcionales que son el índice del elemento y el nombre del array, a esta función se la llama una vez por cada elemento que tenga el array. El valor inicial es el valor que se le asigna de forma inicial al acumulador, si no se le asigna un valor inicial va a tomar el valor del primer elemento del array como valor inicial y se ejecuta la función a partir del segundo elemento.
- ♦ **reduceRight(*nombre-función* [, *valor-inicial*])**: devuelve un valor correspondiente a realizar las operaciones que se indican en la función, esta función tiene dos parámetros, que son un acumulador y el valor de un elemento del array, a esta función se la llama una vez por cada elemento que tenga el array, empezando por el último hacia el primero. El valor inicial es el valor que se le asigna de forma inicial al acumulador, si no se le asigna un valor inicial va a tomar el valor del último elemento del array como valor inicial y se ejecuta la función a partir del elemento anterior.

ejemplo-04-376-reduce.js

001	<code>var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanessa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</code>
002	<code>var VitSumalon=VstNombres.reduce(sumaLongitud,"");</code>
003	<code>function sumaLongitud(acumula, elemento){</code>
004	<code> return acumula+elemento;</code>
005	<code>}</code>
006	<code>VitSumalon2=VstNombres.reduceRight(sumaLongitud,"");</code>
007	<code>document.writeln(`Elementos del array \${VstNombres}
`);</code>
008	<code>document.writeln(`Concatenación Elementos del array \${VitSumalon}
`);</code>
009	<code>document.writeln(`Concatenación Elementos del array \${VitSumalon2}

`);</code>

Resultado

001	Elementos del array Juan,Lucas,María,Angel,Fuenciasla,Yolanda,Julia,Almudena,Félix,María, Isabel,Pedro,Inés,Julian,Beatriz,Angel,Consuelo,María,Carlos,Elena,Fernando,Ursula,Marino, Vanessa,Guillermo,Rosa,Ismael,María,Soledad,Javier,Olga,Luis,Paz
002	Concatenación Elementos del array JuanLucasMaríaAngelFuenciaslaYolandaJuliaAlmudenaFélix MaríaIsabelPedroInésJulianBeatrizAngelConsueloMaríaCarlosElenaFernandoUrsulaMarinoVanessaGu illermoRosaIsmaelMaríaSoledadJavierOlgaLuisPaz
003	Concatenación Elementos del array PazLuisOlgaJavierSoledadMaríaIsmaelRosaGuillermoVanessa MarinoUrsulaFernandoElenaCarlosMaríaConsueloAngelBeatrizJulianInésPedroIsabelMaríaFélixAlm udenaJuliaYolandaFuenciaslaAngelMaríaLucasJuan

- ♦ **toString()**: devuelve una cadena con los valores de los elementos del array separados por comas.
- ♦ **toLocaleString([*código-país*])**: devuelve los elementos del array con el formato del país indicado, o el establecido por defecto, separados por comas. (cuidado con el separador de los números reales, ya que en español es la coma y coincide con el separador de elementos).

ejemplo-04-382-tostring.js

001	<code>var VstNombres = new Array("Juan", "Lucas", "María", "Angel", "Fuenciasla", "Yolanda", "Julia", "Almudena", "Félix", "María", "Isabel", "Pedro", "Inés", "Julian", "Beatriz", "Angel", "Consuelo", "María", "Carlos", "Elena", "Fernando", "Ursula", "Marino", "Vanessa", "Guillermo", "Rosa", "Ismael", "María", "Soledad", "Javier", "Olga", "Luis", "Paz");</code>
-----	--

002	<code>var VstNono=VstNombres.toLocaleString();</code>
003	<code>document.writeln(`Elementos del array \${VstNono}

`);</code>
004	<code>VstNono=VstNombres.toString();</code>
005	<code>document.writeln(`Elementos del array \${VstNono}

`);</code>
<i>Resultado</i>	
001	Elementos del array Juan, Lucas, María, Ángel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Ángel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz
002	Elementos del array Juan, Lucas, María, Ángel, Fuenciasla, Yolanda, Julia, Almudena, Félix, María, Isabel, Pedro, Inés, Julian, Beatriz, Ángel, Consuelo, María, Carlos, Elena, Fernando, Ursula, Marino, Vanesa, Guillermo, Rosa, Ismael, María, Soledad, Javier, Olga, Luis, Paz

Métodos aplicados a Array.

- ♦ **of(lista-valores)**: crea un nuevo array con tantos valores como se indican, y cada uno de los elementos del array es uno de los valores indicados, los valores son números. Se diferencia de crear un array con `new Array` en que cuando utilizamos un único parámetro numérico, en este caso se crea un array con el número de elementos indicados y con `of` se crea un array con un elemento que tiene ese valor.

ejemplo-04-381-of.js

001	<code>var VstNono=Array.of("Carmen","Carlos","Esther","Elias","Gema","Guillermo");</code>
002	<code>document.writeln(`Elementos del array \${VstNono}

`);</code>
<i>Resultado</i>	
001	Elementos del array Carmen, Carlos, Esther, Elias, Gema, Guillermo

- ♦ **from(objeto-map|objeto-set)**: convierto los objetos indicados en un array.

001	<code>var numeros=new Array(12,23,25,14);</code>
002	<code>var cadena="";</code>
003	<code>numeros.forEach(function(valor,indice, arreglo){</code>
004	<code> cadena+="valor: "+valor.toString()+" Indice: "+indice.toString()+" \n";</code>
005	<code>});</code>
006	<code>alert(cadena);</code>

001	<code>var numeros=new Array(12,23,25,14);</code>
002	<code>var cadena="";</code>
003	<code>numeros.forEach(manejo);</code>
004	<code>function manejo(valor,indice, arreglo){</code>
005	<code> cadena+="valor: "+valor.toString()+" Indice: "+indice.toString()+" \n";</code>
006	<code>}</code>
007	<code>alert(cadena);</code>

Dentro de las cadenas tenemos el siguiente método relacionado con los arrays.

- ♦ **split(caracter)**: devuelve un array cuyos elementos están constituidos por los caracteres de la cadena que están separados por el carácter indicado.

ejemplo-04-377-split.js

001	<code>var VstTudela="Marcos\$Marta\$Almudena\$Angel\$Fatima\$Félix";</code>
002	<code>var VstNovedad=VstTudela.split("\$");</code>
003	<code>document.writeln(`Cadena \${VstTudela}
`);</code>
004	<code>document.writeln(`Elementos del array \${VstNovedad}

`);</code>
<i>Resultado</i>	
001	Cadena Marcos\$Marta\$Almudena\$Angel\$Fatima\$Félix
002	Elementos del array Marcos, Marta, Almudena, Angel, Fatima, Félix

Arrays multidimensional

Si queremos tener arrays con más de una dimensión (lo normal es tener arrays bidimensionales) vamos a tener varias posibilidades que vamos a ir viendo una a una.

En la **primera posibilidad** vamos a inicializar el array incluyendo otros arrays dentro, los valores de un array se incluyen entre corchetes.

```
001 var nuevo = [ ["Juan", "Pedro"], ["Antonio", "Felix"] ];
```

Para acceder a los elementos del array vamos a poner nombre del array entre corchetes la fila y entre otros corchetes la columna.

nombre-array[filas][columna]

```
001 document.writeln(nuevo[0][0]+"<br />");
```

En la **segunda posibilidad** vamos a inicializar el array incluyendo otros arrays vacíos y luego asignamos valores a los elementos del array

```
001 var nuevo=[[],[]]
002 nuevo[0][0]="Juan"
003 nuevo[0][1]="Pedro"
004 nuevo[1][0]="Antonio"
005 nuevo[1][1]="Felix"
```

En la **tercera posibilidad** vamos a declarar un array y luego cada uno de los elementos del mismo va a ser un nuevo array.

```
001 var mitabla=new Array();
002 mitabla[0]=new Array("Juan","Pedro","Antonio");
003 mitabla[1]=new Array("Felix","Luis","Ana");
004 mitabla[2]=new Array("Rosa","Laura","Rocio");
005 for(let i=0; i < mitabla.length;i++){
006     for(let j=0; j < mitabla[i].length; j++){
007         document.write(mitabla[i][j]+"<br />")
008     }
009 }
```

Bucles para arrays

Para obtener todos los valores del array

for (nombre of nombre-array) instrucción;

Se va a ejecutar una vez por cada uno de los elementos del array y en cada ejecución de la instrucción nombre va a ir tomando cada uno de los valores del array.

Para obtener todos los índices del array

for (nombre in nombre-array) instrucción;

Se va a ejecutar una vez por cada uno de los elementos del array y en cada ejecución de la instrucción nombre va a ir tomando cada uno de los índices del array

El **objeto Map** nos va a permitir tener un array cuyo índice es un valor de tipo alfanumérico.

Método constructor

◆ **new()**

var novedad = new Map();

Propiedades

◆ **size**: nos indica el número de elementos que tiene el map.

Métodos

- ◆ **get(*clave*)**: devuelve el elemento que tiene esa clave.
- ◆ **set(*clave* , *valor*)**: incluye el valor en el map asociado a la clave indicada.
- ◆ **has(*clave*)**: devuelve un valor lógico que nos indica si existe un elemento con esa clave.
- ◆ **delete(*clave*)**: borra el elemento del map que tiene la clave indicada.
- ◆ **clear()**: borra todos los elementos del map.
- ◆ **entries()**: devuelve un objeto iterator que va a tener en cada posición un array con la clave y el valor del elemento del objeto Map inicial.
- ◆ **keys()**: devuelve un objeto iterator con todas las claves del objeto Map.
- ◆ **values()**: devuelve un objeto iterator con todos los valores del objeto Map.
- ◆ **toString()**: devuelve el elemento como una cadena. "[Object Map]"
- ◆ **valueOf()**: devuelve el objeto Map.
- ◆ **forEach(función (*valor*, *clave*, *objeto*) { *cuerpo-función* })**: realiza la acción indicada para cada elemento del Map.

ejemplo-04-020.js

001	var nuevo=new Map();
002	function anadir(){
003	let valor=prompt("Introduce un valor");
004	let clave=prompt("Introduce su clave");
005	if(nuevo.has(clave))
006	alert("Ya existe esa clave en el array");
007	else
008	nuevo.set(clave, valor);
009	}
010	function consulta(){
011	let clave=prompt("Introduce su clave");
012	if(nuevo.has(clave))
013	alert("El valor correspondiente a la clave "+ clave +" es "+nuevo.get(clave));
014	else
015	alert("NO existe esa clave en el array");
016	}
017	function borrar(){
018	let clave=prompt("Introduce su clave");
019	if(nuevo.has(clave)){
020	nuevo.delete(clave);
021	alert("Valor borrado del array");
022	}else
023	alert("NO existe esa clave en el array");
024	}
025	function numero(){

026	<code>alert("El número de elementos del array es "+ nuevo.size.toString());</code>
027	<code>}</code>
028	<code>function todos(){</code>
029	<code>nuevo.clear()</code>
030	<code>alert("Todos los elementos han sido borrados ");</code>
031	<code>}</code>
032	<code>function valores(){</code>
033	<code>let todosValores="";</code>
034	<code>let todasClaves="";</code>
035	<code>let conjunto="";</code>
036	<code>nuevo.forEach(function(valor, clave , mismo){</code>
037	<code>todosValores+=valor + " \n";</code>
038	<code>todasClaves+=clave + "\n";</code>
039	<code>conjunto+= " clave: " + clave + " valor; " + valor + "\n";</code>
040	<code>});</code>
041	<code>alert("Valores \n"+ todosValores);</code>
042	<code>alert("Claves \n "+ todasClaves);</code>
043	<code>alert("todos \n "+ conjunto);</code>
044	<code>}</code>
045	<code>function valor(){</code>
046	<code>alert("valueOf() \n"+ nuevo.valueOf());</code>
047	<code>}</code>
048	<code>function cadena(){</code>
049	<code>alert("toString() \n "+ nuevo.toString());</code>
050	<code>}</code>

Bucle for para el Objeto Map

for (*nombre* of *objeto-map*) *instrucción*;

Se ejecuta la instrucción tantas veces como elementos tiene el objeto map, en cada una de las ejecuciones nombre toma la dupla clave, valor en un array.

for ([*clave*, *valor*] of *objeto-map*) *instrucción*;

Se ejecuta la instrucción tantas veces como elementos tiene el objeto map, en cada una de las ejecuciones clave y valor toman los valores del elemento del objeto map.

Esto que hemos visto con el objeto Map también se puede hacer con un array normal, como se muestra en el siguiente ejemplo, no podremos acceder al array mediante un índice, sino mediante un valor alfanumérico. En este caso si consultamos la propiedad length siempre nos va a devolver el valor 0, aunque tenga elementos. Si accedemos al array con una clave que no existe vamos a obtener el valor null, no da error de ejecución.

001	<code>var nombres =new Array()</code>
002	<code>nombres["primero"]="Juan"</code>
003	<code>nombres["segundo"]="Pedro"</code>
004	<code>nombres["tercero"]="Antonio"</code>
005	<code>nombres["cuarto"]="Felix"</code>
006	<code>document.writeln(nombres['primero']+"
")</code>
007	<code>document.writeln(nombres.segundo+"
")</code>

En este caso el bucle **for .. of** no funciona.

d) Creación de objetos. Definición de métodos y propiedades.

Vamos a ver diferentes formas de crear objetos, en concreto vamos a ver cuatro formas diferentes de crear objetos.

Primera Forma

Para la creación de objetos vamos a utilizar el objeto **Object** y su método constructor. El objeto Object es un objeto genérico de datos.

```
var nombre-variable = new Object()
```

Crea un objeto genérico con el nombre indicado.

001	var personal=new Object();
-----	----------------------------

La forma de declarar las propiedades es asignando valor a las mismas a continuación de la creación del objeto, poniendo

```
nombre-objeto.nombre-propiedad=valor
```

También podemos utilizar:

```
nombre-objeto[nombre-propiedad]= valor
```

En este caso el nombre de la propiedad puede venir representada por una variable o una constante de tipo cadena.

La declaración de los métodos se realiza:

```
nombre-objeto.nombre-método= function([parámetros]) {  
    cuerpo-método  
}
```

y también podemos utilizar la siguiente forma

```
nombre-objeto[nombre-método]=function ([parámetros]) {  
    cuerpo-método  
}
```

en este caso como en el caso anterior el nombre del método puede venir expresado como una variable o una constante de tipo cadena.

Para acceder desde los métodos a las propiedades deberemos poner:

```
nombre-objeto.nombre-propiedad
```

ejemplo-4-030.js

001	var coche =new Object();
002	coche.marca=vmarca;
003	coche.modelo=vmodelo;
004	coche.precio=parseFloat(vprecio);

005	coche.potencia=parseInt(vpotencia);
006	coche.cilindrada=parseInt(vcilindrada);
007	coche.consumo=parseFloat(vconsumo);
008	coche.precioKm=function(precioCombustible){
009	let elprecio= coche.consumo * precioCombustible /100;
010	return elprecio;
011	}
012	coche.precioCil=function(){
013	let valor= coche.precio / coche.cilindrada;
014	return valor;
015	}
016	coche.incrementoPrecio=function(incremento){
017	let incre=(coche.precio * incremento /100);
018	coche.precio +=incre;
019	}

Segunda Forma

También podemos declarar un objeto a través de un método constructor que es una función, de la siguiente forma:

```
function nombre-pseudoclase(lista-parámetros) {  
    cuerpo  
}
```

Luego nos declaramos un objeto de esa clase a través de:

```
var nombre-objeto= new nombre-pseudoclase(valores-parámetros)
```

Para definir propiedades usaremos dentro del cuerpo:

```
this.nombre-propiedad=valor
```

Para declara propiedades de solo lectura desde dentro usaremos

```
this.__defineGetter__(nombre-propiedad,  
function([parámetro]) { cuerpo}
```

Para declara propiedades de solo escritura desde dentro usaremos

```
this.__defineSetter__(nombre-propiedad,  
function(parámetro) { cuerpo}
```

Para declarar métodos usaremos dentro del cuerpo:

```
this.nombre-método=function ([parámetros]) {  
    cuerpo-método  
}
```

Dentro de los métodos para poder acceder a las propiedades deberemos poner:

```
this.nombre-propiedad
```

Si deseamos añadir alguna propiedad desde fuera deberemos usar:

nombre-pseudoclase.prototype.nombre-propiedad=valor

Para declara propiedades de solo lectura desde fuera usaremos

***nombre-objeto.__defineGetter__(nombre-propiedad,
function([parámetro]) { cuerpo }***

Para declara propiedades de solo lectura desde fuera usaremos

***nombre-pseudoclase.prototype.__defineGetter__(nombre-propiedad,
function([parámetro]) { cuerpo }***

Para declara propiedades de solo escritura desde fuera usaremos

***nombre-objeto.__defineSetter__(nombre-propiedad,
function(parámetro) { cuerpo }***

Para declara propiedades de solo escritura desde fuera usaremos

***nombre-pseudoclase.prototype.__defineSetter__(nombre-propiedad,
function(parámetro) { cuerpo }***

Si deseamos añadir algún método desde fuera pondremos

***nombre-pseudoclase.prototype.nombre-método= function(
[parámetros]) {
cuerpo
}***

ejemplo-04-031.js

001	function tipoVehiculo(pmarca, pmodelo, pprecio, pcilindrada, ppotencia, pconsumo, pfechaCompra){
002	var tipoCombustible="Gasolina";
003	this .marca=pmarca;
004	this .modelo=pmodelo;
005	this .precio=pprecio;
006	this .cilindrada=pcilindrada;
007	this .potencia=ppotencia;
008	this .consumo=pconsumo;
009	this .fechaCompra=pfechaCompra;
010	this .precioKm= function (precioCombustible){
011	let elprecio= this .consumo * precioCombustible /100;
012	return elprecio;
013	}
014	this .precioCil= function () {
015	let valor= this .precio / this .cilindrada;
016	return valor;
017	}
018	this .incrementoPrecio= function (incremento){
019	let incre=(this .precio * incremento /100);
020	this .precio +=incre;
021	}
022	this .__defineGetter__ ("añoCompra", function (){
023	return this .fechaCompra.getFullYear();
024	});

025	<code>this. defineSetter ("añoCompra",function(anyo){</code>
026	<code> this.fechaCompra.setFullYear(anyo);</code>
027	<code>});</code>
028	<code>this. defineGetter ("Combustible",function(){</code>
029	<code> return tipoCombustible;</code>
030	<code>});</code>
031	<code>this. defineSetter ("Combustible",function(combus){</code>
032	<code> tipoCombustible=combus;</code>
033	<code>});</code>
034	<code>}</code>
035	<code>tipoVehiculo.prototype.precioMetalizado=1000;</code>
036	<code>tipoVehiculo.prototype.precioCompleto=function(complemento){</code>
037	<code> return this.precio + complemento;</code>
038	<code>}</code>
039	<code>tipoVehiculo.prototype. defineGetter ("nombreCompleto",function(){</code>
040	<code> return this.marca + " "+this.modelo;</code>
041	<code>});</code>
042	<code>tipoVehiculo.prototype. defineSetter ("mesCompra",function(vmes){</code>
043	<code> this.fechaCompra.setMonth(vmes -1);</code>
044	<code>});</code>

Si queremos aplicar herencia utilizando esta segunda forma, deberemos crearnos la clase padre y luego dentro de la clase hija para heredar el comportamiento de la clase padre deberemos poner.

nombre-clase-padre.call(this, parámetros)

ejemplo-04-032.js

001	<code>function tipoCoche(pmar, pmod){</code>
002	<code> console.log(pmar + " "+ pmod)</code>
003	<code> this.marca=pmar;</code>
004	<code> this.modelo=pmod;</code>
005	<code>}</code>
006	<code>function tipoVehiculo(pmarca, pmodelo, pprecio, pcilindrada, ppotencia, pconsumo, pfechaCompra){</code>
007	<code> let tipoCombustible="Gasolina";</code>
008	<code> console.log(pmarca+ " "+ pmodelo);</code>
009	<code> tipoCoche.call(this, pmarca, pmodelo);</code>
010	<code> this.precio=pprecio;</code>
011	<code> this.cilindrada=pcilindrada;</code>
012	<code> this.potencia=ppotencia;</code>
013	<code> this.consumo=pconsumo;</code>
014	<code> this.fechaCompra=pfechaCompra;</code>
015	<code> this.precioKm=function(precioCombustible){</code>
016	<code> let elprecio=this.consumo * precioCombustible /100;</code>
017	<code> return elprecio;</code>
018	<code> }</code>
019	<code> this.precioCil=function(){</code>
020	<code> let valor=this.precio /this.cilindrada;</code>
021	<code> return valor;</code>
022	<code> }</code>
023	<code> this.incrementoPrecio=function(incremento){</code>
024	<code> let incre=(this.precio * incremento /100);</code>
025	<code> this.precio +=incre;</code>
026	<code> }</code>
027	<code> this. defineGetter ("añoCompra",function(){</code>
028	<code> return this.fechaCompra.getFullYear();</code>
029	<code>});</code>
030	<code> this. defineSetter ("añoCompra",function(anyo){</code>
031	<code> this.fechaCompra.setFullYear(anyo);</code>
032	<code>});</code>
033	<code> this. defineGetter ("Combustible",function(){</code>
034	<code> return tipoCombustible;</code>
035	<code>});</code>
036	<code> this. defineSetter ("Combustible",function(combus){</code>
037	<code> tipoCombustible=combus;</code>

038	});
039	}

Una clase puede tener herencia múltiple, es decir que herede el comportamiento de varias clases, para lo cual deberemos poner la instrucción anterior tantas veces como veces herede el comportamiento de otras clases.

ejemplo-04-033.js

001	function tipoCoche(pmar, pmod){
002	this.marca=pmar;
003	this.modelo=pmod;
004	}
005	function tecnicos(pcilin,ppoten,pcons){
006	this.cilindrada=pcilin;
007	this.potencia=ppoten;
008	this.consumo=pcons;
009	}
010	function tipoVehiculo(pmarca, pmodelo, pprecio, pcilindrada, ppotencia, pconsumo, pfechaCompra){
011	let tipoCombustible="Gasolina";
012	console.log(pmarca+" "+ pmodelo);
013	tipoCoche.call(this, pmarca, pmodelo);
014	this.precio=pprecio;
015	tecnicos.call(this, pcilindrada,ppotencia, pconsumo);
016	this.fechaCompra=pfechaCompra;
017	this.precioKm=function(precioCombustible){
018	let elprecio=this.consumo * precioCombustible /100;
019	return elprecio;
020	}
021	this.precioCil=function(){
022	let valor=this.precio /this.cilindrada;
023	return valor;
024	}
025	this.incrementoPrecio=function(incremento){
026	let incre=(this.precio * incremento /100);
027	this.precio +=incre;
028	}
029	this.__defineGetter__("añoCompra",function(){
030	return this.fechaCompra.getFullYear();
031	});
032	this.__defineSetter__("añoCompra",function(anyo){
033	this.fechaCompra.setFullYear(anyo);
034	});
035	this.__defineGetter__("Combustible",function(){
036	return tipoCombustible;
037	});
038	this.__defineSetter__("Combustible",function(combus){
039	tipoCombustible=combus;
040	});
041	}

Para acceder desde la clase hija a un elemento de la clase padre, deberemos poner **this.elemento-padre**.

Tercera forma

Nos declaramos una clase

```
class nombre-clase{
    [ constructor ([parámetros]) {
        instrucciones
    } ]
    [ [static] nombre-método(parámetros) {
        instrucciones} ]
}
```


Mediante la palabra constructor nos estamos declarando el método constructor de la clase y en el cual vamos a inicializar todas las propiedades de la clase, que van a llevar siempre el prefijo **this**. También se pueden declarar variables cuyo ámbito será el constructor y se pueden declarar así mismo, métodos.

Mediante **static** nos estamos declarando un método llamado estático, método que puede ser llamado sin ser estanciado, esto es, que se puede llamar a ese método utilizando la clase y no el objeto de la clase.

ejemplo-04-053.js

001	class coches {
002	constructor (pmarca,pmodelo,pprecio){
003	this .marca=pmarca;
004	this .modelo=pmodelo;
005	this .precio=pprecio;
006	}
007	cuotamensual(meses){
008	let valor=(this .precio *1.20)/ meses;
009	return valor;
010	}
011	}
012	var mio=new coches("seat","arosa",12450);
013	document .writeln(mio.marca + " ");
014	document .writeln(mio.modelo + " ");
015	document .writeln(mio.precio + " ");
016	document .writeln(mio.cuotamensual(12)+" ");

ejemplo-04-055.js

001	class coches {
002	constructor (pmarca,pmodelo,pprecio){
003	this .marca=pmarca;
004	this .modelo=pmodelo;
005	this .precio=pprecio;
006	let dolar=0;
007	this .valor_dolar=function(pvalor){
008	dolar=pvalor;
009	}
010	this .precio_dolar=function(){
011	return this .precio / dolar;
012	}
013	};
014	cuotamensual(meses){
015	let valor=(this .precio *1.20)/ meses;
016	return valor;
017	}
018	
019	}
020	var mio=new coches("seat","arosa",12450);
021	document .writeln(mio.marca + " ");
022	document .writeln(mio.modelo + " ");
023	document .writeln(mio.precio + " ");
024	document .writeln(mio.cuotamensual(12)+" ");
025	mio.valor_dolar(0.87);
026	document .writeln(mio.precio_dolar()+" ");

Dentro de la clase y fuera del constructor nos podemos declarar propiedades de solo lectura a través de:

```
get nombre-propiedad(){
    cuerpo
    return expresión;
}
```

También dentro de la clase y fuera del constructor nos podemos declarar propiedades de solo escritura mediante:

```
set nombre-propiedad(parámetro){  
    cuerpo  
}
```

001	class coches {
002	constructor(pmarca, pmodelo, pprecio){
003	this.marca=pmarca;
004	this.modelo=pmodelo;
005	this.precio=pprecio;
006	this.dolar=0;
007	};
008	cuotamensual(meses){
009	let valor=(this.precio *1.20)/ meses;
010	return valor;
011	}
012	set valor_dolar(pvalor){
013	this.dolar=pvalor;
014	}
015	get precio_dolar(){
016	return this.precio /this.dolar;
017	}
018	}
019	var mio=new coches("seat", "arosa", 12450);
020	document.writeln(mio.marca + " ");
021	document.writeln(mio.modelo + " ");
022	document.writeln(mio.precio + " ");
023	document.writeln(mio.cuotamensual(12)+ " ");
024	mio.valor_dolar=0.87;
025	document.writeln(mio.precio_dolar + " ");
026	mio.dolar=0.93;
027	document.writeln(mio.precio_dolar + " ");

Dentro de la declaración de una clase también nos podemos declarar variables y métodos privados, para ello bastara con anteponer el símbolo de la almohadilla “#” delante del nombre de la variable o del método. Las variables las podemos declarar delante del constructor. Cuando queramos hacer referencia a estos elementos privados deberemos poner **this.#nombre_variable** o **this.#nombre_método()**. De momento no funciona en Firefox, Internet Explorer ni Microsoft Edge.

001	class persona {
002	#nom="";
003	constructor(pnombre, papellidos){
004	this.#nom=pnombre;
005	this.apellidos=papellidos;
006	};
007	set nombre(pvalor){
008	this.#nom=pvalor;
009	}
010	get nombre(){
011	return this.#nom;
012	}
013	#completo(){
014	return this.#nom + " "+this.apellidos;
015	}
016	total(){
017	return this.#completo();
018	}

019	}
020	if(document.addEventListener)
021	window.addEventListener("load", inicio)
022	else if(document.attachEvent)
023	window.attachEvent("onload", inicio);
024	function inicio(){
025	let boton=document.getElementById("resultado");
026	if(document.addEventListener)
027	boton.addEventListener("click", tratar)
028	else if(document.attachEvent)
029	boton.attachEvent("onclick", tratar);
030	}
031	function tratar(){
032	let ape=document.getElementById("apellidos").value;
033	let nom=document.getElementById("nombre").value;
034	let nuevo =new persona(nom,ape);
035	console.log(nuevo.nombre);
036	console.log(nuevo.apellidos);
037	document.getElementById("completo").value=nuevo.total();
038	}

Para aplicar herencia utilizaremos

```
class nombre-clase extends clase-padre{
    [ constructor ([parámetros]) {
        super([parámetros]);
        instrucciones
    }]
    [ [static] nombre-método([parámetros]) {
        instrucciones}]
}
```

Para llamar al método constructor de la clase padre utilizamos dentro del constructor de la clase hija **super** con sus correspondientes parámetros.

Si desde la clase hija queremos llamar a algún método de la clase padre deberemos poner **super.nombre-método**

ejemplo-04-056.js

001	class coches {
002	constructor(pmarca,pmodelo,pprecio){
003	this.marca=pmarca;
004	this.modelo=pmodelo;
005	this.precio=pprecio;
006	this.dolar=0;
007	};
008	cuotamensual(meses){
009	let valor=(this.precio *1.20)/ meses;
010	return valor;
011	}
012	set valor_dolar(pvalor){
013	this.dolar=pvalor;
014	}
015	get precio_dolar(){
016	return this.precio /this.dolar;
017	}
018	completo(){
019	return this.marca + " " +this.modelo;
020	}
021	
022	}
023	class vehiculos extends coches {
024	constructor(pmarca,pmodelo,pacabado,pprecio,pcilin,ppoten){
025	super(pmarca,pmodelo,pprecio);

026	<code>this.acabado=pacabado;</code>
027	<code>this.cilindrada=pcilin;</code>
028	<code>this.potencia=ppoten;</code>
029	<code>}</code>
030	<code>completo(){</code>
031	<code>return super.completo()+" "+this.acabado;</code>
032	<code>}</code>
033	<code>}</code>
034	<code>var mio=new coches("seat", "arosa", 12450);</code>
035	<code>document.writeln(mio.marca + "
");</code>
036	<code>document.writeln(mio.modelo + "
");</code>
037	<code>document.writeln(mio.precio + "
");</code>
038	<code>document.writeln(mio.cuotamensual(12)+"
");</code>
039	<code>document.writeln(mio.completo()+"
");</code>
040	<code>mio.valor_dolar=0.87;</code>
041	<code>document.writeln(mio.precio_dolar + "
");</code>
042	<code>mio.dolar=0.93;</code>
043	<code>document.writeln(mio.precio_dolar + "
");</code>
044	<code>var nuestro =new vehiculos("opel", "vectra", "alto", 19850, 2000, 150);</code>
045	<code>document.writeln(nuestro.marca + "
");</code>
046	<code>document.writeln(nuestro.modelo + "
");</code>
047	<code>document.writeln(nuestro.acabado + "
");</code>
048	<code>document.writeln(nuestro.potencia + "
");</code>
049	<code>document.writeln(nuestro.cilindrada + "
");</code>
050	<code>document.writeln(nuestro.precio + "
");</code>
051	<code>document.writeln(nuestro.cuotamensual(24)+"
");</code>
052	<code>document.writeln(nuestro.completo()+"
");</code>
053	<code>document.writeln(mio.cuotamensual(36)+"
");</code>

Cuarta Forma

Declararnos un objeto de forma implícita

```
var nombre-objeto = {  
    cuerpo  
}
```

Para declarar propiedades pondremos

```
nombre-propiedad : valor,
```

Para declarar propiedades de solo lectura pondremos

```
get nombre-propiedad() { cuerpo },
```

En el cuerpo va a actuar como una función, con lo cual debe devolver un valor.

Para declarar propiedades de solo escritura pondremos

```
set nombre-propiedad(parámetro) { cuerpo },
```

También podemos declararnos propiedades a través del set y de get y que no dependan de ninguna otra propiedad, en este caso se necesita una variable auxiliar que se debe declarar dentro de la función donde se crea el objeto y que se puede utilizar en el set y en el get.

Para declarar métodos según esta cuarta forma usaremos

```
nombre-método : function ([parámetros]) {  
    cuerpo  
}
```

Dentro de los métodos, del set y del get para poder acceder a las propiedades deberemos poner:

this.nombre-propiedad

001	var coche={
002	marca:vmarca,
003	modelo:vmodelo,
004	precio:parseFloat(vprecio),
005	potencia:parseInt(vpotencia),
006	cilindrada:parseInt(vcilindrada),
007	consumo:parseFloat(vconsumo),
008	fechaCompra:vfecha,
009	precioKm:function(precioCombustible){
010	let elprecio=this.consumo*precioCombustible/100;
011	return elprecio;
012	},
013	precioCil:function(){
014	let valor=this.precio/this.cilindrada;
015	return valor;
016	},
017	incrementoPrecio:function(incremento){
018	let incre=(this.precio*incremento/100);
019	this.precio+=incre;
020	},
021	get añoCompra(){
022	return this.fechaCompra.getFullYear();
023	},
024	set añoCompra(anyo){
025	this.fechaCompra.setFullYear(anyo);
026	}
027	}

Con los objetos podemos utilizar las siguientes instrucciones:

for (variable in objeto) {
 cuerpo
}

Se va a ejecutar una vez por cada elemento del objeto ya bien sea propiedad o método y en donde variable va a tomar el nombre de los elementos del objeto.

Ejecutas el cuerpo de las instrucciones por cada uno de los valores del objeto, solo para objeto iterables, los objetos que nos creamos no lo son.

for (variable of objeto) {
 cuerpo
}

Se puede hacer referencia a las propiedades y métodos del objeto sin hacer referencia al mismo ya que se indica al principio.

with (objeto) {
 instrucciones
}

001	class coches {
002	constructor(pmarca,pmodelo,pprecio){
003	this.marca=pmarca;

004	<code>this.modelo=pmodelo;</code>
005	<code>this.precio=pprecio;</code>
006	<code>this.dolar=0;</code>
007	<code>};</code>
008	<code>cuotamensual(meses){</code>
009	<code>let valor=(this.precio *1.20)/ meses;</code>
010	<code>return valor;</code>
011	<code>}</code>
012	<code>set valor_dolar(pvalor){</code>
013	<code>this.dolar=pvalor;</code>
014	<code>}</code>
015	<code>get precio_dolar(){</code>
016	<code>return this.precio /this.dolar;</code>
017	<code>}</code>
018	<code>}</code>
019	<code>var mio=new coches("seat","arosa",12450);</code>
020	<code>with(mio){</code>
021	<code>marca="Volkswagen";</code>
022	<code>modelo="Golf";</code>
023	<code>precio=35000;</code>
024	<code>dolar=0.98;</code>
025	<code>}</code>
026	<code>for(let dato in mio){</code>
027	<code>document.writeln(dato+" "+eval("mio."+dato)+"
");</code>
028	<code>}</code>

Para saber si un objeto es de una clase tenemos:

`nombre-objeto.constructor.name === "nombre-clase"`

`nombre-objeto.constructor === nombre-clase`

`Object.getPrototypeOf(nombre-objeto) === nombre-clase.prototype`

También podemos utilizar **instanceof**, pero deberemos tener en cuenta que si la clase es una clase hija, nos va a decir que es de la clase propia y de la clase padre.

`nombre-objeto instanceof nombre-clase`

Quinta Forma

En esta forma va a ser a través del objeto **Object**, sus métodos y propiedades.

El Objeto Object.

Características de Object y de los objetos.

Propiedad **constructor**

`Nombre-objeto.constructor` → tiene una referencia al constructor del objeto.

001	<code>var misDatos =newObject();</code>
002	<code>if(misDatos.constructor===Object){</code>
003	<code>console.log("El constructor de misDatos es Object");</code>
004	<code>}</code>

001	<code>function coches (){</code>
002	<code>this.marca = "";</code>
003	<code>this.modelo="";</code>
004	<code>this.precio =0;</code>
005	<code>this.precioComplementos=0;</code>
006	<code>this.nombreCompleto=function(){return(this.marca + " "+this.modelo);}</code>
007	<code>this.incrementoPrecio=function(porcentaje){this.precio *=(1+(porcentaje/100));}</code>
008	<code>this.incrementoComplementos=function(){this.precioComplementos *=1.05;}</code>
009	<code>}</code>
010	<code>var miCoche =new coches;</code>

```
0011 if(miCoche.constructor== coches){
0012     alert("El constructor de miCoche es coches");
0013 }
```

```
0001 var coches ={
0002     marca : "",
0003     modelo : "",
0004     precio :0,
0005     precioComplementos :0,
0006     nombreCompleto :function(){return(this.marca + " "+this.modelo)},
0007     incrementoPrecio :function(porcentaje){this.precio *=(1+(porcentaje/100));},
0008     incrementoComplementos :function(){this.precioComplementos *=1.05;},
0009 }
0010 if(coches.constructor==Object){
0011     alert("El constructor de coches es Object");
0012 }
```

create crear un objeto a partir de un prototipo con unas propiedades. El prototipo puede ser **null** bien **Object.prototype** o bien otro objeto o bien una clase o bien **class.prototype**.

Object.create(*nombre-objeto*, {*definición propiedades*})

```
0001 var misDatos =new Object();
0002 misDatos.nombre="pedro";
0003 var miObjeto=Object.create(misDatos,{
0004     apellidos:{
0005         value:"Garcia",
0006         writable:true,
0007         enumerable:true,
0008         configurable:true
0009     },
0010 });
0011 if(miObjeto.constructor==Object){
0012     alert("El constructor de miObjeto es Object");
0013 }
```

Para definir una propiedad vamos a poner:

```
nombre-propiedad : {
    value:valor,
    writable:true|false ,
    enumerable:true|false,
    configurable:true|false
}
```

En este caso ponemos **value** para asignar un valor. El resto de opciones se pueden poner o bien omitir y tienen el siguiente significado. Con **writable** nos indica si en la propiedad se puede escribir (true) o bien no se puede (false). Con **enumerable** nos indica si la propiedad la podemos utilizar en un bucle for in, si se puede (true) y si no se puede (false). Con **configurable** nos indica si la propiedad se puede configurar mediante otros métodos de la clase Object, con true se puede y con false no se puede.

También se pueden declarar propiedades utilizando el **set** si es de solo escritura, utilizando el **get** si es de solo lectura y también se puede declarar utilizando el **set** y **get**. Si la propiedad no depende de ninguna otra propiedad se puede poner

una variable auxiliar que se declara en la función que crea el objeto y que se puede utilizar. La forma de declarar las propiedades de esta forma es:

```
nombre-propiedad : {
    get : function([parametros]) { cuerpo-función },
    set : function(parámetro [, parametros]) { cuerpo-función },
    enumerable:true|false ,
    configurable:true|false
}
```

Se puede poner todo o bien solo el **set** y/o el **get** el resto de las opciones se pueden poner o bien omitir.

```
001 var coche =Object.create(null,{
002     marca:{value:"", writable:true, configurable:true, enumerable:true},
003     modelo:{value:"", writable:true, configurable:true, enumerable:true},
004     precio:{value:1.0, writable:true, configurable:true, enumerable:true},
005     potencia:{value:1, writable:true, configurable:true, enumerable:true},
006     cilindrada:{value:1, writable:true, configurable:true, enumerable:true},
007     consumo:{value:1.0, writable:true, configurable:true, enumerable:true},
008     fechaCompra:{value:new Date(), writable:true, configurable:true, enumerable:true},
009     añoCompra:{
010         get:function(){
011             return this.fechaCompra.getFullYear()
012         },
013         set:function(anyo){
014             this.fechaCompra.setFullYear(anyo)
015         }
016     },
017     precioCilindrada:{
018         get:function(){
019             return this.precio /this.cilindrada;
020         }
021     },
022 });
```

```
001 function tipoCoche(pmarca,ppre){
002     this.marca=pmarca;
003     this.precio=ppre;
004 }
005 var nuevo =new tipoCoche(vmarca,vpre);
006 var coche =Object.create(tipoCoche.prototype,{
007     modelo:{value:"", writable:true, configurable:true, enumerable:true},
008     potencia:{value:1, writable:true, configurable:true, enumerable:true},
009     cilindrada:{value:1, writable:true, configurable:true, enumerable:true},
010     consumo:{value:1.0, writable:true, configurable:true, enumerable:true},
011     fechaCompra:{value:new Date(), writable:true, configurable:true, enumerable:true},
012     añoCompra:{
013         get:function(){
014             return this.fechaCompra.getFullYear()
015         },
016         set:function(anyo){
017             this.fechaCompra.setFullYear(anyo)
018         }
019     },
020     precioCilindrada:{
021         get:function(){
022             return this.precio /this.cilindrada;
023         }
024     },
025 });
001 var nuevo =new Object();
```

002	nuevo.marca=vmarca;
003	nuevo.precio=vpre;
004	var coche =Object.create(nuevo ,{
005	modelo:{value:"", writable:true, configurable:true, enumerable:true},
006	potencia:{value:1, writable:true, configurable:true, enumerable:true},
007	cilindrada:{value:1, writable:true, configurable:true, enumerable:true},
008	consumo:{value:1.0, writable:true, configurable:true, enumerable:true},
009	fechaCompra:{value:new Date(), writable:true, configurable:true, enumerable:true},
010	añoCompra:{
011	get:function(){
012	return this.fechaCompra.getFullYear()
013	},
014	set:function(anyo){
015	this.fechaCompra.setFullYear(anyo)
016	}
017	},
018	precioCilindrada:{
019	get:function(){
020	return this.precio /this.cilindrada;
021	}
022	},
023	});

Todas las declaraciones de las propiedades van a estar separadas por comas.

defineProperty → añade una propiedad a un objeto

**Object.defineProperty(*nombre-objeto*,*nombre-propiedad*,
descriptor-propiedad)**

```
001 | Object.defineProperty (miObjeto,"edad",{ value:33, writable:true});
```

```
001 Object.defineProperty(coche,"color",{value:vcolor, writable:true, configurable:true,
enumerable:true});
```

defineProperties → añade propiedades a un objeto

Objet.defineProperties(*objeto*, *descriptores-propiedades*)

001	Object.defineProperty(miObjeto,{ localidad:{value:"Madrid", writable:true},
002	estadoCivil:{value:"Soltero", writable:true}
003	});

001	<code>Object.defineProperty(coche,{ matricula:{value:vmatricula, writable:true, configurable:true, enumerable:true},</code>
002	<code>bastidor:{value:vbastidor, writable:true, configurable:true, enumerable:true}});</code>

Para ver si dos objetos son iguales

Object.is(objeto-1, objeto-2)

Devuelve un valor lógico que nos indica si los dos objetos son iguales.

Para copiar una serie de objetos a otro

Object.assign(*destino*, *lista-objetos*)

Copia la lista de objetos sobre el destino y devuelve una copia del mismo.

freeze → impide añadir propiedades, modificar propiedades o atributos.

Object.freeze(*objeto*)

```
001 Object.freeze(miObjeto);
```

isExtensible → indica si se pueden añadir nuevas propiedades al objeto

Object.isExtensible(*objeto*)

```
001 Object.isExtensible(miObjeto);
```

isFrozen → indica si NO se pueden modificar propiedad, atributos ni añadir nuevas propiedades.

Object.isFrozen(*objeto*)

```
001 Object.isFrozen(miObjeto);
```

isSealed → indica si no se pueden modificar atributos de propiedades no se pueden añadir nuevas propiedades.

Object.isSealed(*objeto*)

```
001 Object.isSealed(miObjeto);
```

seal → impide modificar atributos de propiedades y añadir nuevas propiedades.

Object.seal(*objeto*)

```
001 Object.seal(miObjeto);
```

getOwnPropertyNames → devuelve un array con el nombre de las propiedades y métodos de un objeto.

array=Object.getOwnPropertyNames(*objeto*)

```
001 var nombres =Object.getOwnPropertyNames(miObjeto);
```

getOwnPropertyDescriptor → devuelve el descriptor de una propiedad de un objeto.

Object.getOwnPropertyDescriptor(*objeto*, *nombre-propiedad*)

***nombre-objeto*.toString()** → devuelve el objeto como una cadena.

***nombre-objeto*.propertyIsEnumerable(*nombre-propiedad*)** → indica si la propiedad es enumerable (indica si puede estar en un bucle for each).

nombre-objeto-1.isPrototypeOf(objeto-2)→indica si el objeto 2 tiene objeto 1 en su cadena de prototipos.

nombre-objeto.hasOwnProperty(nombre-propiedad)→ indica si el objeto tiene la propiedad indicada.

Object.preventExtensions(objeto)→ impide que se puedan añadir más propiedades

Object.keys(objeto)→ devuelve un array con los nombres de los métodos y propiedades.

objeto.watch(propiedad, función)→ función que se ejecuta cuando se asigna valor a la propiedad.

objeto.unatch(propiedad)→deja de ejecutarse la función.

objeto.__lookupGetter__(propiedad) →referencia a la función de un getter para la propiedad.

objeto.__LookupSetter__(propiedad)→referencia a la función de un setter para la propiedad.

El método **create** también se puede utilizar para cambiar el comportamiento de una clase existente si ponemos

objeto.prototype = objeto.create(clase-padre.prototype, {declaración-propiedades});