

FUNCIONES Y ARRAYS EN JAVASCRIPT

**UT4.- PROGRAMACIÓN CON FUNCIONES,
ARRAYS Y OBJETOS DE USUARIO**



Susana López Luengo

Objetivos

- Conocer en detalle las principales funciones del lenguaje JavaScript
 - Crear funciones personalizadas para realizar tareas específicas
 - Comprender el objeto Array y familiarizarse con sus propiedades y métodos
-

Funciones predefinidas del lenguaje

- JavaScript tiene funciones integradas en el lenguaje.
 - Las siguientes funciones son algunas de las principales predefinidas de JavaScript:
 - `encodeURIComponent()`
 - `Number()`
 - `eval()`
 - `String()`
 - `isFinite()`
 - `parseInt()`
 - `isNaN()`
 - `parseFloat()`
-

Funciones predefinidas del lenguaje

encodeURIComponent()

- La función **encodeURIComponent()** es usada para codificar una [URI](#)
- Esta función codifica caracteres especiales a excepción de , / ? : @ & = + \$ #
- La función **decodeURI()** se utiliza para decodificar una URI
- Ejemplo

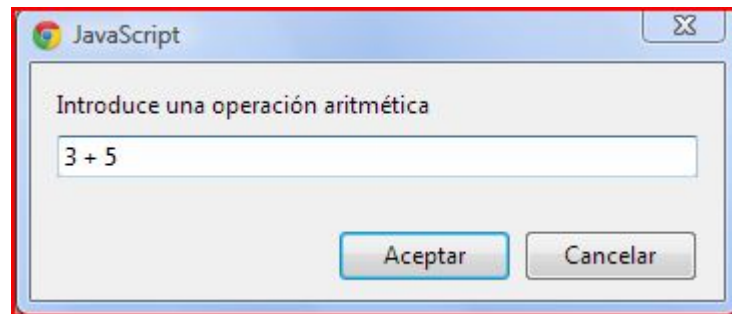
```
var uri = "my test.asp?name=ståle&car=saab";  
var enc = encodeURIComponent(uri);  
// Codificada: my%20test.asp?name=st%C3%A5le&car=saab  
var dec = decodeURI(enc);  
// Decodificada my test.asp?name=ståle&car=saab
```

Funciones predefinidas del lenguaje

eval()

- Convierte una cadena que pasamos como argumento en código JavaScript ejecutable.
- Ejemplo

```
var x = 10;
var y = 20;
var a = eval("x * y")+ <br> // 200 <br>
```



```
var input = prompt("Introduce una operación numérica");
var resultado = eval(input);
```

Funciones predefinidas del lenguaje

isFinite()

- verifica si el número que pasamos como argumento es o no un número finito, un número válido.

<code>isFinite(123)</code>	<code>true</code>
<code>isFinite(-1.23)</code>	<code>true</code>
<code>isFinite(5-2)</code>	<code>true</code>
<code>isFinite(0)</code>	<code>true</code>
<code>isFinite("123")</code>	<code>true</code>
<code>isFinite("Hello")</code>	<code>false</code>
<code>isFinite("2005/12/12")</code>	<code>false</code>
<code>isFinite(NaN)</code>	<code>false</code>
<code>isFinite(Infinity)</code>	<code>false</code>

Funciones predefinidas del lenguaje

isNaN()

- Comprueba si el valor que pasamos como argumento es un de tipo numérico. Es el acrónimo de is Not a Number (no es un número).
 - Esta función es diferente al método Number.isNaN()
 - isNaN() primero lo convierte a un número y después devuelve true si es un número y false en caso contrario
 - Number.isNaN() no lo convierte a un número, y no devuelve true para ningún valor que no sea del tipo Number
-

Funciones predefinidas del lenguaje

isNaN()

<code>isNaN(123)</code>	<code>false</code>
<code>isNaN(0)</code>	<code>false</code>
<code>isNaN("123")</code>	<code>false</code>
<code>isNaN("")</code>	<code>false</code>
<code>isNaN(undefined)</code>	<code>true</code>
<code>isNaN("Hello")</code>	<code>true</code>
<code>isNaN("2005/12/12")</code>	<code>true</code>
<code>isNaN(NaN)</code>	<code>true</code>
<code>isNaN(Infinity)</code>	<code>false</code>
<code>isNaN(true)</code>	<code>false</code>

Funciones predefinidas del lenguaje

String()

- convierte el objeto pasado como argumento en una cadena que represente el valor de dicho objeto
- Ejemplos

```
String(Boolean(0)) // "false"
```

```
String(Boolean(1)) // "false"
```

```
String(12345) // "12345"
```

```
String(new Date())
```

```
// "Sun Oct 15 2017 20:04:47 GMT+0200 (Romance Summer Time)"
```

Funciones predefinidas del lenguaje

Number()

- Convierte el objeto pasado como argumento en un número que representa el valor de dicho objeto.
- Si la conversión falla, la función devuelve NaN
- Ejemplos

```
Number(false) // 0
```

```
Number(true) // 1
```

```
Number("999") // 999
```

```
Number(new Date()) // 1382704503079
```

```
Number("999 888") // NaN
```

Funciones predefinidas del lenguaje

`parseInt(String, [base])`

- Convierte el string pasado como argumento en un valor numérico de tipo entero.
 - Si no especificamos la base como segundo argumento se utiliza automáticamente la base decimal (10).
 - La base puede ser de 2 a 32
 - Si el String comienza por "0x" se pasa a hexadecimal los que empezaban "0" ya no se pasan a octal
 - Si la función encuentra en la cadena a convertir, algún carácter que no sea numérico, devuelve el valor encontrado hasta ese punto.
 - Si el primer valor no es numérico, la función devuelve NaN.
-

Funciones predefinidas del lenguaje

`parseInt(String, [base])`

<code>parseInt(10)</code>	10
<code>parseInt(10.33)</code>	10
<code>parseInt("34 45 66")</code>	34
<code>parseInt(" 60 ")</code>	60
<code>parseInt("40 años")</code>	40
<code>parseInt("tenía 40 años")</code>	NaN
<code>parseInt("12", 8)</code>	10
<code>parseInt("010")</code>	8
<code>parseInt("0x10")</code>	16
<code>parseInt("F", 16)</code>	15

Funciones predefinidas del lenguaje

parseFloat(String)

- Convierte la cadena que pasamos como argumento en un valor numérico de tipo flotante.
- La diferencia es que en lugar de convertir el argumento a un número entero, intenta convertirlo a un número de punto flotante.

<code>parseInt("10")</code>	10
<code>parseFloat("10.33")</code>	10.33
<code>parseInt("34 45 66")</code>	34
<code>parseInt(" 60 ")</code>	60
<code>parseInt("40 años")</code>	40
<code>parseFloat("tenía 40 años")</code>	NaN

Funciones del usuario

- Las funciones son uno de los pilares fundamentales en JavaScript
 - Una función es un procedimiento en JavaScript—un conjunto de sentencias que realizan una tarea o calculan un valor
 - Para usar una función, debe definirla en algún lugar del ámbito desde el cual desea llamarla.
-

Funciones del usuario

Declaraciones de funciones

- **SINTAXIS**

```
function n_funcion ([arg1,.. argn]) {  
    instrucciones...;  
    [return valor];  
}
```

- **n_funcion**: El nombre de la función (opcional).
 - **arg**: Lista de **argumentos** para la función, separados por comas (,) - opcional
 - Las sentencias JavaScript que definen la función, encerradas por llaves, { }
 - **return valor**: valor que devuelve - opcional
-

Funciones del usuario

Ejemplo función

- La función alCuadrado toma un argumento, llamado numero
- La función consiste de una sentencia que expresa el retorno del argumento de la función (numero) multiplicado por él mismo
- La sentencia return especifica el valor retornado por la función.

```
function alCuadrado (numero) {  
    return numero * numero;  
}
```

Funciones del usuario

Nombre de la función

- **Deben usarse sólo letras, números o los caracteres \$ _**
 - Debe ser único en el código JavaScript de la página web.
 - **No pueden empezar por un número.**
 - No puede ser una de las palabras clave del lenguaje.
 - No puede ser una de las palabras reservadas del lenguaje.
 - El nombre debería ser representativo de la tarea realizada por el grupo de instrucciones que ejecute la función.
-

Funciones del usuario

Parámetros

- Los parámetros **primitivos** (como puede ser un número) son pasados a las funciones por **valor**
 - El valor es pasado a la función, pero si la función cambia el valor del parámetro, este cambio no es reflejado globalmente o en otra llamada a la función.
 - Si pasa un **objeto** (p. ej. un valor no primitivo, como un Array o un objeto definido por el usuario) como parámetro, este es pasada por **referencia**
 - Si la función cambia las propiedades del objeto, este cambio es visible desde afuera de la función
-

Funciones del usuario

Funciones como objetos

- Las **funciones** son objetos de pleno derecho
 - Pueden **asignarse a variables, a propiedades**, pasarse como **parámetros...**
 - Literal de función: `function (..) {..}`
 - **Función sin nombre**, que suele asignarse a una variable, que es la que le da nombre
 - Se puede invocar a través del nombre de la variable
 - El **operador (...)** invoca una función ejecutando su código
 - Este operador sólo es aplicable a funciones (objetos de la clase `Function`), sino da error
 - El operador puede incluir parámetros separados por coma, accesibles en el código de la función
-

Funciones del usuario

Expresiones de función

- Las funciones pueden también ser creadas por una expresión de función
- **Tal función puede ser anónima; es decir, no tener un nombre.**
- Una expresión de función puede ser guardada en una variable
- Por ejemplo, la función alCuadrado podría haber sido definida como:

```
var alCuadrado = function(numero) {return numero *  
numero}
```

```
var x = alCuadrado(4) //x obtiene el valor 16
```

Funciones del usuario

Expresiones de función

- Se puede proporcionar un nombre a una expresión de función
- Esto puede ser utilizado dentro de la función para hacer una llamada recursiva, o en un depurador para identificar la función en el trazado de pila:

```
var factorial = function fact(n) {return n<2 ? 1 :  
n*fact(n-1)};
```

```
console.log(factorial(3));
```

Funciones del usuario

Expresiones de función

- Las expresiones de función se utilizan sobre todo para **pasar una función como argumento a otra función**
- Ejemplo

```
function simularmap(f,a) { //pasamos por par. función y array
  var result = []; // Crea un nuevo Array
  var i;
  for (i = 0; i != a.length; i++)
    result[i] = f(a[i]);
  return result;
}
```

//Expresión de función:

```
var multiplicar= function(x) { return x * x;}
simularmap(multiplicar, [0, 1, 2, 5, 10]);
//resultado --- [0, 1, 4, 25, 100]
```

Funciones del usuario. Funciones autoinvocables

- Las funciones pueden autoinvocarse , esto es, la función se ejecuta automáticamente sin llamarla
- Para que una función se ejecute automáticamente sin ser llamada al final de la función se añade ()
- Para que la función se ejecute como una expresión se debe meter entre paréntesis

```
(function (){
    alert("Hola");
})();
```

```
var saludo = "hola";
(function(nombre){
    var saludo = "Hi";
    alert(saludo + " " +nombre);
})("Juan"); //>> Hi Juan
```

Permiten crear variables locales sin que haya conflicto con las globales (Ej.saludo)

Funciones del usuario

Parámetros por defecto

- En JavaScript, los parámetros de funciones están establecidos por defecto a **undefined**
- A partir de ECMAScript 6, es posible establecerlos a un valor suministrado por defecto diferente.

```
function multiplicar(a, b = 1) {  
    return a*b;  
}
```

```
multiplicar(5); // 5
```

Funciones del usuario

Funciones flecha

- En ECMAScript 2015 se introdujo una nueva forma de declarar funciones
- Una expresión de función flecha (fat arrow function) tiene una sintaxis más corta comparada con las expresiones de función

```
const greet = function (greeting, person) {           // defined with function literal
  return `${greeting} ${person}, how are you?` ;
};

// The same function defined with arrow notation
const greet = (greeting, person) => {
  return `${greeting} ${person}, how are you?` ;
};

// Parenthesis may be omitted with only one parameter
let square = x => x*x; // Blocks with one instruction may omit curly brackets and return

const say_hi = () => "Hi, how are you?";              // function without parameters
```

Funciones del usuario

Funciones flecha

- Las funciones flecha son siempre funciones anónimas.
- Dos factores influenciaron la introducción de las funciones flecha:
 - funciones más cortas
 - el léxico this. Tiene vinculación léxica (y no dinámica) con su entorno, como las variables

Ejemplo:

Formato tradicional:

```
function duplicar(numero)
  {return numero * 2 }
```

Formato flecha.

Varias opciones:

```
(numero)=> {return numero * 2; }
(numero) => numero * 2;
numero => numero * 2;
```

Arrays

- Un array es un conjunto ordenado de valores relacionados
 - Cada uno de estos valores se denomina elemento.
 - Cada elemento tiene un índice que indica su posición numérica en el array, empezando por índice 0
 - Además de los valores y los índices de estos valores, un array debe tener un nombre
-

Arrays

- Un objeto Array de JavaScript es un objeto global que es usado en la construcción de arrays
- Podemos crear un array de dos formas

```
var frutas = ['Manzana', 'Plátano'];
```

```
var frutas = new Array('Manzana', 'Plátano');
```

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

- Para saber si una variable es un array, podemos usar el método `Array.isArray()` o `instanceof`:

```
Array.isArray(frutas); // true
```

```
frutas instanceof Array; // true
```

Arrays

Inicialización de arrays

- Sintaxis

```
nombre_del_array[indice] = valor_del elemento;  
var ultimo = frutas[frutas.length - 1];// Plátano'
```

- Ejemplo

```
var verduras = new Array();  
verduras[0] = "puerro";  
verduras[1] = "berenjena";  
verduras[2] = "apio";
```

- También se pueden inicializar cuando se crea

```
var verduras = new Array("puerro", "berenjena", "apio");
```

- Un array puede contener elementos de tipos diferentes

Arrays

Acceso a arrays

- Se Accede a un array por su índice
`var primero = frutas[0];// Manzana`
`var ultimo = frutas[frutas.length - 1];// Plátano'`
 - En JavaScript podemos acceder a todo el array con su nombre
`console.log(frutas)`
-

Arrays. Acceso mediante bucles

```
var miarray = [7, 'hi', 'adios'];  
for (let i= 0; i< miarray.length; i++){  
    document.writeln("Elemento: " + miarray[i] + "<br>");
```

```
miarray.forEach(p => document.writeln("Elemento: " + p +  
"<br>"));
```

[Ejemplo](#)

```
for (var p in miarray) {  
    document.writeln("Elemento: " + p + " = " + miarray[p] +  
"<br>");
```

[Ejemplo](#)

```
for (var p of miarray) {  
    document.writeln("Elemento [" + i + "] = " + p + "</br>");  
    i++;
```

[Ejemplo](#)

Arrays

Acceso a arrays mediante bucles. Ejemplos

- Estos 2 ejemplos suman los elementos del array:

```
let n = [7, 4, 1, 23];
let add = 0;

for (let i=0; i < n.length; ++i){
  add += n[i];
}
```

add // => 35 (7+4+2+23)

```
let n = [7, 4, 1, 23];
let add = 0;

n.forEach(elem => add += elem)
```

add // => 35

Arrays

Destructuring en arrays

- Es un modo de extraer valores de los arrays y asignarlos a un grupo de variables
- Ejemplo:

```
var myArray = ["Hey", "I" , "am", "an", "array"];  
var [greeting, one, another, anotherOne, type] = myArray;
```

```
console.log(greeting); // Hey  
console.log(type); // array
```

Ver vídeo [Destructuring de arrays y objetos](https://www.youtube.com/watch?v=...) (Didacticode.com)
arrays: hasta el minuto 3:50

Bucles for...in y for...of

◆ JavaScript incluye el bucle for...in que itera en las propiedades de un objeto

- El bucle **for...of** de ES6 itera con una función generadora en los elementos de un objeto iterable
 - <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...of>
 - <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...in>
- Los arrays son objetos y son iterables por lo que pueden procesarse con ambos bucles

◆ La sentencia for (prop in object) {..bloque..} (object es un array u objeto)

- Ejecuta el bloque para cada **propiedad** (accesible en la variable **prop**) del objeto o array
 - Los **índices** de los elementos de un array son **propiedades** especiales (su nombre es el número)

◆ La sentencia for (elem of object) {..bloque..} (object debe ser un obj. o array iterable)

- Ejecuta el bloque para cada **elemento** (accesible en la variable **elem**) del objeto o array
 - object** debe ser un iterable que define el orden del recorrido
 - Por ejemplo, en un array el recorrido empieza en el elemento de índice **0** y termina en el de **length-1**
- Estos 2 ejemplos de suma de elementos de Array con los nuevos bucles equivalen a los 3 ya vistos

```
let n = [7, 4, 1, 23];  
let add = 0;
```

```
for (let i in n){  
  add += n[i];  
}
```

```
add // => 35
```

```
let n = [7, 4, 1, 23];  
let add = 0;
```

```
for (let elem of n){  
  add += elem;  
}
```

```
add // => 35
```

```
let n = [7, 4, 1, 23];  
let add = 0;
```

```
for (let i=0; i < n.length; ++i){  
  add += n[i];  
}
```

```
add // => 35
```

```
let n = [7, 4, 1, 23];  
let add = 0;
```

```
n.forEach(elem => add += elem)
```

```
add // => 35
```

```
[7, 4, 1, 23].reduce((acc, elem) => acc += elem, 0); // => 35
```

Arrays

Métodos más utilizados

- **push**(elemento) Añade un elemento al final del array
- **pop**(elemento) Elimina un elemento al final del array
- **shift**(elemento) Elimina un elemento al principio del array
- **unshift**(elemento) Añade un elemento al principio del array
- **indexOf**(elemento) Devuelve el índice en el que aparece un elemento
- **slice**(inicio,fin) Devuelve un nuevo array con los elementos seleccionados.

inicio: posición de inicio (por def 0, valores negativos es que empieza por el final del array)

fin: posicion final, es opcional, por defecto final del array

Arrays

Métodos

- **toString()** Devuelve un string con todos los elementos separados por comas
 - **join([separador])** Une todos los elementos de un array, con la opción de seleccionar el separador
 - **concat(array1, array2...)** Crea un nuevo array con la unión de dos o más arrays
 - Más métodos:
https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array#M.C3.A9todos_en_general
-

Arrays

Método splice

- El método splice () es el más complejo del objeto Array
 - Su sintaxis es
splice(posicion,numEliminar,[nuevosEle])
 - Permite añadir o eliminar objetos de un array
 - **frutas.splice(1, 1)** elimina el segundo elemento del array
 - **frutas.splice(2, 0, "Limón", "Kiwi");** añade en la posición dos, dos nuevas frutas, limón y kiwi
-

Arrays

Ordenar Arrays

- El método **sort()** ordena alfabéticamente de la A a la Z un Array

```
var nombres = ["Carlos", "Zoe", "Ana", "Manuel"];  
nombres.sort(); // =["Ana","Carlos","Manuel","Zoe"]
```

- El método **reverse()** cambia el orden alfabético de un Array ordenado de la Z a la A

```
var nombres = ["Carlos", "Zoe", "Ana", "Manuel"];  
nombres.reverse(); // =["Zoe","Manuel","Carlos","Ana"]
```

Métodos para ordenar, invertir, concatenar o buscar

◆ **sort()**

Estos métodos no modifican el array original, solo devuelven el resultado como parámetro retorno.

- devuelve el array ordenado

```
[1, 5, 3].sort() // => [1, 3, 5]
```

◆ **reverse()**

- devuelve el array invertido

```
[1, 5, 3].reverse() // => [3, 5, 1]
```

◆ **concat(e1, ..., en)**

- devuelve un nuevo array con **e1, ..., en** añadidos al final

```
[1, 5, 3].concat(9) // => [1, 5, 3, 9]  
[1, 5, 3].concat(9, 3) // => [1, 5, 3, 9, 3]
```

◆ **join(<separador>)**

- concatena elementos en un string
 - ♦ introduce <separador> entre elementos

```
[1, 5, 3, 7].join(';') // => '1;5;3;7'  
[1, 5, 3, 7].join('') // => '1537'
```

◆ **indexOf(elem, offset)**

- devuelve índice de primer **elem**
 - ♦ **offset**: comienza búsqueda (por defecto 0)

```
[1, 5, 3, 5, 7].indexOf(5) // => 1  
[1, 5, 3, 5, 7].indexOf(5, 2) // => 3
```

```
[1, 5, 3].concat(2).sort().reverse() // => [5, 3, 2, 1]
```

Los métodos encadenados aplican el segundo método sobre retorno del primero.

Más métodos: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Extraer, modificar o añadir elementos al array

- ◆ **slice(i,j)**: devuelve la rodaja entre i y j
 - Índice negativo (j) es relativo al final
 - ◆ índice "-1" es igual a a.length-2
 - No modifica el array original
- ◆ **splice(i, j, e1, e2, ..., en)**
 - sustituye j elementos desde i en array
 - ◆ por e1, e2, ..., en
 - Devuelve rodaja eliminada
- ◆ **push(e1, ..., en)**
 - añade e1, ..., en al final del array
 - ◆ devuelve el tamaño del array (a.length)
- ◆ **pop()**
 - elimina último elemento y lo devuelve

```
[1, 5, 3, 7].slice(1, 2) => [5]
[1, 5, 3, 7].slice(1, 3) => [5, 3]
[1, 5, 3, 7].slice(1, -1) => [5, 3]
```

```
let a = [1, 5, 3, 7];

a.splice(1, 2, 9) => [5, 3]
a                 => [1, 9, 7]

a.splice(1,0,4,6) => []
a                 => [1, 4, 6, 9, 7]
```

```
let b = [1, 5, 3];

b.push(6, 7)    => 5
b               => [1, 5, 3, 6, 7]

b.pop()         => 7
b               => [1, 5, 3, 6]
```

Más métodos: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Operador spread/rest: ...x

◆ ES6 añade el operador spread/rest (...x)

- Tiene semántica spread (esparcir) o rest (resto) dependiendo del contexto

◆ Operador spread (...x) esparce los elementos del **array x** en otro array

- Actúa así cuando se aplica al constructor de array o en la invocación de una función
 - ♦ Por ejemplo, `[x, ...y, z, ...t]` o `mi_funcion(x, ...y, z, ...t)`

◆ Operador rest (...x) agrupa un conjunto de valores en el **array x**

- Agrupa en un array el resto de los elementos asignados de una lista
 - ♦ Por ejemplo, `[x, y, ...resto] = [1, 2, 3, 4, 5]` o `function f(x, y, ...resto) {...}`
 - La variable agrupadora debe ir al final y agrupa los últimos elementos de la lista

```
....  
const a = [2, 3];  
const b = [0, 1, ...a];  
b => [0, 1, 2, 3]  
  
f(0, 1, ...a) => f(0, 1, 2, 3)
```

```
let [x, y, ...rest] = [0, 1, 2, 3, 4];  
x      => 0  
y      => 1  
rest   => [2, 3, 4]  
  
function f(x, y, ...z) {...}  
f(0, 1, 2, 3) => f(0, 1, [2, 3])
```

```
let x, y, z;  
[y, z, ...x] = [2, 3, 4, 5];  
x      => [4, 5];  
y      => 2  
z      => 3
```

Buen tutorial sobre destructuring assignment y spread/rest: <https://javascript.info/destructuring-assignment>
Destructuring assignment: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment
Spread/rest syntax: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_operator

Arrays Multidimensionales

- Se define como un array que contiene como elementos arrays
- Para poder utilizar este tipo de estructuras de datos debemos definir un array, en el que en cada una de sus posiciones debemos crear a su vez otro array.

Arrays Multidimensionales

Ejemplo: Array con las 3 palabras más buscadas en Google de España y Portugal

```
var p_espana = new Array ('instagram',  
    'youtube', 'twitter');
```

```
var p_portugal = new Array  
    ('jogos',download', 'youtube');
```

```
var palabras = new Array(p_espana, p_portugal);
```

Método map()

- El método map() crea un nuevo array con el resultado de llamar a una función por cada elemento del array
 - Sintaxis:
`array.map(function(currentValue, index, arr), thisValue)`
 - Ejemplo, multiplicar por dos todos los elementos de un array:

```
var numeros= [1, 5, 10, 15];  
var numerosDobles = numeros.map(x => x * 2);
```
-

Programación Funcional en JavaScript

filter()

- El método filter crea un nuevo array con todos los elementos que pasan un test (ofrecido como una función)
 - Sintaxis:
`array.filter(function(currentValue, index, arr), thisValue)`
 - Ejemplo, filtrar todos los mayores de edad de un array:

```
var aEdad= [32, 33, 16, 40];  
var aMayorEdad= aEdad.filter((edad) => edad  
>= 18);
```
-

Programación Funcional en JavaScript

reduce()

- El método reduce aglutina el array en un solo valor.
- Este ejecuta una función que se le pasa a cada valor del array (de izquierda a derecha) y el valor de retorno es guardado en un acumulador

- Sintaxis:

```
array.reduce(function(total, currentValue,  
currentIndex, arr), initialValue)
```

- Ejemplo, sumar todos los números de un array:

```
var aNumeros= [65, 44, 12, 4];  
var suma = aNumeros.reduce((a, b) => a + b,  
0);
```

Otros métodos iteradores de Array

◆ Estos métodos invocan la función también con los mismos 3 parámetros

- **elem**: elemento del array accesible en la invocación en curso
- **i**: índice al elemento del array accesible en la invocación en curso
- **a**: array completo sobre el que se invoca el método

◆ **find**(function(elem, i, a){...})

```
[7, 4, 1, 23].find(elem => elem < 3); // => 1
```

- devuelve el 1^{er} elemento donde la función retorna true

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find

◆ **findIndex**(function(elem, i, a){...})

```
[7, 4, 1, 23].findIndex(elem => elem < 3); // => 2
```

- devuelve el índice del 1^{er} elem. donde la función retorna true

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/findIndex

◆ **filter**(function(elem, i, a){...})

```
[7, 4, 1, 23].filter(elem => elem > 5); // => [7, 23]
```

- elimina los elementos del array donde la función retorna false

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter

◆ **map**(function(elem, i, a){...})

```
[7, 4, 1, 23].map(elem => -elem); // => [-7, -4, -1, -23]
```

- sustituye cada elemento del array por el resultado de invocar la función

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map

Método reduce

- ◆ El método **reduce** añade el parámetro **accumulator** a element, index y array
 - **accumulator**: variable con valor retornado por invocación anterior de la función
 - ♦ además están los 3 parámetros típicos de los métodos iteradores: **element**, **index** y **array**
- ◆ **reduce**(function(accumulator, element, index, array){...}), initial_value)
 - Inicializa accumulator con **initial_value** e itera de 0 a **array.length-1**
 - ♦ **accumulator** recibe en cada nueva iteración el valor de retorno de la función
 - https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array/reduce
 - si **initial_value** se omite inicia accumulator con **array[0]** e itera de 1 a **array.length-1**

```
// Example of addition of numbers with reduce
[7, 4, 1, 23].reduce((acc, elem) => acc += elem, 0); // => 35
```

```
// Example which orders first the array and eliminates then duplicated numbers
[4, 1, 4, 1, 4].sort().reduce((ac, el, i, a) => el !== a[i-1] ? ac.concat(el) : ac, []); // => [1, 4]
```

```
// sort(..) and reduce(..) are composed in series, where each one performs the following
[4, 1, 4, 1, 4].sort(); // => [1, 1, 4, 4, 4]
[1, 1, 4, 4, 4].reduce((ac, el, i, a) => el !== a[i-1] ? ac.concat(el) : ac, []); // => [1, 4]
```


Webgrafía. Enlaces

- <https://www.w3schools.com/js/>
 - Curso Desarrollo de aplicaciones con HTML,node.js y Javascript. UPM. Miriadax
 - <https://es.wikipedia.org/>
 - www.openclassrooms.com
 - <https://developer.mozilla.org/es/docs/Web/JavaScript>
-

