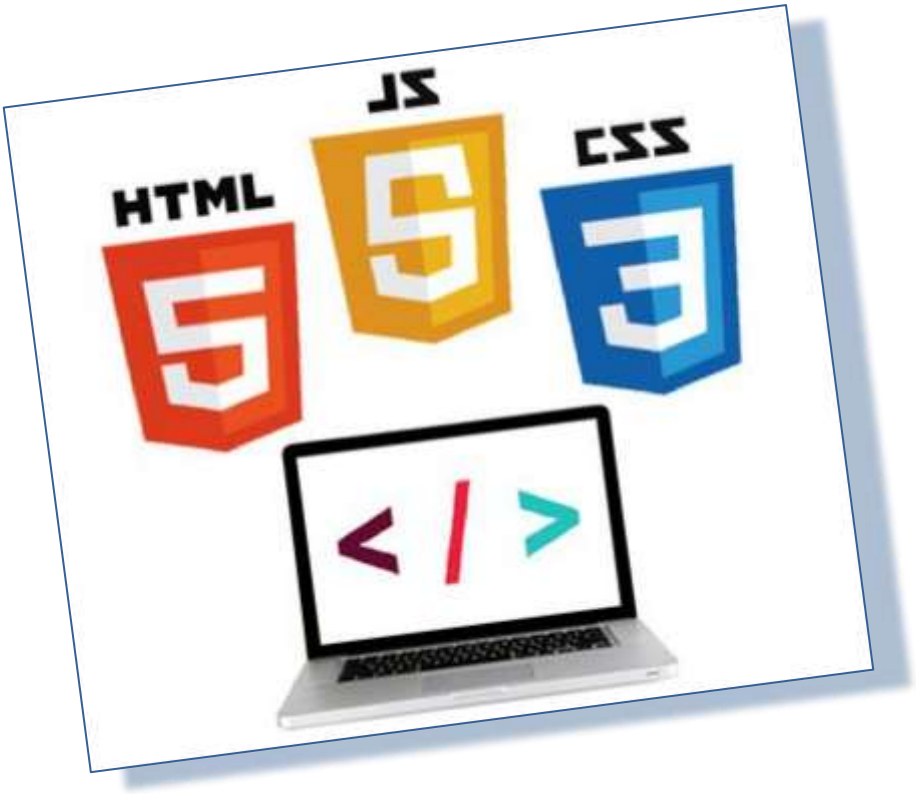
Desarrollo web en entorno cliente



Tema 3: Arrays

1. Introducción

Un array es un tipo especial de Object. Se utiliza para almacenar más de un valor en una sola variable. La diferencia con los objetos radica en que en los arrays se usan índices numéricos mientras que en los objetos son nombres. Un array no tiene tamaño fijo, por lo que se le pueden añadir valores nuevos en cualquier momento.

Se puede declaran un array como instancia del objeto Array o, más recomendado, de forma directa. Además, se puede inicializar con un número determinado de valores (aunque se podrán añadir más después), o declararlo vacío. Las siguientes dos formas de declaración hacen exactamente lo mismo, aunque es más simple y rápida en ejecución la primera de ellas:

```
let variables1 = ["var1","var2","var3","var4"];
let variables2 = [];
.....
let variables1 = new Array("var1","var2","var3","var4");
let variables2 = new Array();
```

Se accede a un elemento del array indicando su posición dentro del mismo. Hay que tener en cuenta que las posiciones empiezan en 0 y no en 1. Si se intenta recuperar el valor de una posición a la que no se haya asignado, el valor devuelto será undefined:

```
let variables = ["var1","var2","var3","var4"];
let variable = variables[2]; //variable contendrá "var3"
let variable2 =variables[4]; //variable2 contendrá "undefined"
```

Y se puede acceder al array completo simplemente por su nombre. Las siguientes sentencias muestran (siguiendo el ejemplo anterior): la primera de ellas, el valor de la posición tercera; la segunda, los valores de la primera y la sexta; y la última, todos los valores que almacena el array:

```
alert(variable);
alert(variables[0] +"---"+ variables[5]);
alert(variables);
```

Los elementos de un array pueden ser de cualquier tipo, incluso es posible almacenar valores de distintos tipos en un mismo array. Y pueden almacenar objetos, funciones u otros arrays además de los tipos primitivos:

```
var array1 = [1,2,3,4,5];
array1[1] = ["cadena1","cadena2", "cadena3"];
array1[2] = ["pepito", 5.6];
```

1.1. Deconstrucción o desestructuración de arrays

La desestructuración de arrays permite almacenar, en variables independientes entre sí, los valores que se estén almacenando en las distintas posiciones de un array:

```
const pueblos = ["Calzada", "Santa Marta", "Mogarraz"];
const [pueblo1, pueblo2, pueblo3] = pueblos;
console.log(pueblo1);
console.log(pueblo2);
console.log(pueblo3);
```

Esta deconstrucción de arrays suele encontrarse de la mano del uso de expresiones regulares. Cuando la función exec(), o match(), de expresiones regulares encuentra una coincidencia con la expresión regular indicada, devuelve un array que contiene, en su primera posición, toda la parte coincidente de la cadena y, a continuación en las consiguientes posiciones, las partes de la cadena que coinciden con cada grupo entre paréntesis en la expresión regular. El uso de deconstrucción permite desechar la primera posición y asignar en cada variable que se indique en las siguientes posiciones cada uno de los valores:

```
const direccion = "One Infinite Loop, Cupertino 95014";
const expresionRegularCodigoPostalCiudad =
/^[^,\\]+[,\\\s]+(.+?)\s*(\d{5})?$/;
const [, ciudad, codigoPostal] =
  direccion.match(expresionRegularCodigoPostalCiudad) || [];
guardarCP(ciudad, codigoPostal);
```

Esta misma figura se puede aplicar a objetos, con el fin de asignar a diferentes variables los valores que almacenen las propiedades de un objeto.

1.2. Operador ...

Este operador permite incluir como elementos de un array los elementos de otro, incrementando el número de posiciones del primero:

```
const valoresIniciales = [1,2,3];
const valoresFinales = [...valoresIniciales, 4,5,6];
console.log(valoresFinales); //[ 1, 2, 3, 4, 5, 6]
```

Por otro lado, permite también recuperar un array sin saber, a priori, el número de elementos que va a tener:

```
function calcular(operador, ...valores) {
    console.log(operador); // +
    console.log(valores); // [1, 2, 3]
}
calcular('+', 1, 2, 3);
```

2. Añadir y eliminar elementos

Se puede añadir un elemento al array utilizando la función ***push()***, que lo incluirá al final del mismo, o haciendo uso de la función ***unshift()***, que lo meterá como primer elemento del array:

```
let deportes = ["futbol", "baloncesto", "balonmano", "tenis"];
deportes.push("padel");
deportes.unshift("atletismo");
```

También se puede incluir un nuevo elemento al final del array utilizando la propiedad ***length***:

```
deportes[deportes.length] = "tiro con arco ";
```

Se puede eliminar el último elemento de un array con la función ***pop()*** o el del principio con ***shift().*** El valor que devuelven ambos métodos es el elemento borrado y el array pasará a tener una posición menos:

```
deportes.pop();
deportes.shift();
```

EJERCICIO 1:

- Implementa una función donde:
  - Primero se creará un array vacío.
  - Después se pedirá al usuario, repetidamente, que introduzca un número (que podrá ser positivo, negativo o 0) y un nombre de persona.
  - Cada vez se preguntará al usuario si se va a añadir el valor indicado:
    - Si acepta que se añada:
      - Si ha introducido un número mayor o igual que 0 se incluye el nombre en el array en la posición indicada.
      - Si el número es negativo, se añadirá al final del array sin sobrescribir ninguno.
    - Si no lo acepta:
      - Si el número es mayor o igual que 0, se eliminará el valor del elemento que se encuentre en la posición indicada ignorando el introducido.
      - Si es negativo, se eliminará el primero.
  - La ejecución terminará cuando, en lugar de un número, se introduzca una cadena. Antes de finalizar se mostrará el contenido final del array por consola.

2.1. slice

El método ***slice()*** devuelve, en un nuevo array, los elementos contenidos en el array de acuerdo a los parámetros que se le pasan, que serán uno o dos valores numéricos. El array original no se verá modificado.

array.slice(posiciónInicio, posiciónFinal);

El primer valor indicará la posición a partir de la cual empieza la extracción. Si es negativo, indica un desplazamiento desde el final del array. Si, por el contrario, es mayor a la longitud del array, se devuelve un array vacío. Si es omitido el valor por defecto es 0.

```
let deportes2 = deportes.slice(1, 3);
let deportes3 = deportes.slice(-2);
```

El segundo valor indicará hasta donde se extrae, pero sin incluir ese último elemento. Si es negativo, indicará un desplazamiento desde el final de la secuencia. Si no se incluye o es mayor a la longitud del array, slice extrae hasta el final del array (array.length):

```
let deportes4 = deportes.slice(0,-2);
```

Si no se pasa ningún valor, se hará una copia completa del array.

2.2. splice

Mediante el método ***splice()*** se puede cambiar el contenido de un array eliminando elementos existentes y/o agregando nuevos. Devolverá un array con los elementos eliminados.

Supresión

Se puede eliminar elementos de un array indicando la posición del primer elemento a eliminar y el número de elementos que se eliminarán a partir del mismo:

array(posición, numElementosEliminar);

```
deportes.splice(1, 2);
```

Inserción o reemplazo

Se puede insertar elementos en un array indicando la posición donde se van a agregar, el número de elementos a eliminar y el elemento a insertar. Opcionalmente, se puede especificar una cuarta, quinta, o cualquier número de otros parámetros a insertar:

array(posición, numElementosEliminar, elemento1, elemento2, …, elementoN);

```
deportes.splice(3, 0, "frontón", "bailes de salón");
deportes.splice(2, 2, "halterofilia");
deportes.splice(2, 1, "natación", "waterpolo");
```

EJERCICIO 2:

- Prueba, en la consola del navegador, los ejemplos anteriores de las funciones slice y splice.
- Implementa una función que incluya en un array la siguiente lista: tela, bies, hilo, tijeras, máquina de coser, botón. Después, utilizando las funciones vistas:
  - Elimina el bies. Muestra el array por consola.
  - Añade, entre hilo y tijeras, hilo torzal. Muéstralo de nuevo por consola.
  - Quita tijeras y sustitúyelas por cúter rotatorio y mesa de corte. Una vez mas, muestra el array por consola.
  - Crea, a partir del primer array, dos nuevos donde el primero incluya los tres primeros elementos del original y el segundo del cuarto al sexto.
  - En el segundo de los nuevos arrays creados:
    - Incluye, en la primera posición, remalladora.
    - Incluye, en la penúltima posición, cinta métrica.
  - Muestra por consola el contenido de los tres arrays.

3. Métodos de uso habitual

3.1. length, sort

La propiedad ***length*** devuelve el número de elementos que tiene el array. Se puede modificar el tamaño del array modificando el valor de esta propiedad:

```
let arrayNumeros = ["uno", "dos", "tres"];
arrayNumeros.length; //Devolverá 3
arrayNumeros.length = 2; //Desparecerá el tercer elemento
```

Mediante la función ***sort()***, podemos ordenar un array alfabéticamente siguiendo el criterio Unicode (por ejemplo mayúsculas van antes que minúsculas), por lo tanto, no servirá para ordenar arrays con valores numéricos:

```
arrayNumeros.sort();//arrayNumeros quedará [ "dos", "tres", "uno"]
```

Si se quiere ordenar según un criterio diferente, se le puede pasar una función que indique cómo hacerlo. Recibirá como parámetros dos valores a comparar y:

- Si devuelve un valor negativo, sort() entiende que el primer elemento se colocará delante del segundo.
- Si devuelve un valor positivo, sort() colocará el segundo elemento por delante del primero.
- Si devuelve 0, sort() no modificará el orden de esos dos elementos.

Por ejemplo, para ordenar un array de string sin tener en cuenta mayúsculas o minúsculas:

```
let arrayNumeros = ["uno", "dos", "Tres", "cuatro", "Cinco"];
    arrayNumeros.sort(function (elem1, elem2) {
        if (elem1.toLocaleLowerCase() > elem2.toLocaleLowerCase()) {
            return 1;
        }
        if (elem1.toLocaleLowerCase() < elem2.toLocaleLowerCase()) {
            return -1;
        }
        return 0;
    });
```

O para ordenar un array de objetos, pudiéndolo ordenar por un campo concreto del objeto (por ejemplo, el DNI para ordenar personas):

```
    personas.sort(function (persona1, persona2) {
        return persona1.dni - persona2.dni;
    });
```

EJERCICIO 3:

- Actualiza la función del segundo punto del ejercicio 2 para que, tras realizar todos los cambios pedidos en el mismo, se ordene el array original resultante según el número de caracteres que tenga cada valor, de mayor a menor.

3.2. toString, join y split

Todos los objetos pueden hacer uso del método ***toString()*** que los convierte en una cadena. Es el método llamado cuando se muestra un array por consola (cuando se utiliza únicamente el nombre) y, en el caso concreto de los arrays, devuelve todos sus valores separados por comas:

```
    let ciudades = ["Salamanca", "Dublín", "Praga", "Florencia"];
    console.log(ciudades);
    //Se mostrará [ "Salamanca", "Dublín", "Praga", "Florencia" ]
```

Para convertir los elementos de un array en una cadena, existe la función ***join()***. Esta función admite como parámetro el carácter que se quiera utilizar como separador de los distintos valores del array. Si no se indica nada, los separará con comas:

```
   console.log(ciudades.join());
//Mostrará Salamanca,Dublín,Praga,Florencia
   console.log(ciudades.join("*"));
//Mostrará Salamanca*Dublín*Praga*Florencia
```

Por el contrario, el método ***split()*** convierte cualquier cadena en un array de cadenas. Si no se indica el carácter que marcará el fin de un elemento y el comienzo del siguiente, toda la cadena será el primer y único elemento del array. Dicho carácter no formará parte de los valores resultantes:

```
    let cadena = "pedro juan jose";
    console.log(cadena.split());
//el array resultante será [ "pedro juan jose"]
    console.log(cadena.split("j"));
//el array resultante será [ "pedro ", "uan ", "ose"]
```

EJERCICIO 4:

- Implementa una función que:
  - Pida por pantalla un número indeterminado de valores numéricos al usuario separados por el carácter “/”.
  - Meta todos esos valores en un array.
  - Ordene el array de menor a mayor.
  - Muestre por pantalla al usuario una única cadena con todos los valores del array, separados por el carácter “–“.

3.3. concat, reverse, indexOf, lastIndexOf

Es posible concatenar arrays con el método ***concat()***. Se unirán los valores de ambos arrays en un tercero sin sufrir modificaciones los dos primeros:

```
let ciudades = ["Salamanca", "Dublín", "Praga", "Florencia"];
let ciudades2 = ["Bilbao", "Burdeos", "Londres", "Budapest"];
let todasLasCiudades = ciudades.concat(ciudades2);
```

Por su parte, el método **reverse()** invierte el orden de los elementos del array:

```
ciudades.reverse();
```

Por último, ***indexOf()*** devuelve la primera posición del elemento pasado como parámetro o -1 si no se encuentra en el array, mientras que ***lastIndexOf()*** devuelve la última posición del elemento pasado como parámetro o -1 si no se encuentra en el array.

4. Programación funcional en JavaScript

La programación funcional es un paradigma de programación declarativa. Este paradigma promulga que el código debe centrarse en describir qué hacer, y no en cómo hacerlo. Una herramienta que se usa mucho en este tipo de programación son las funciones flecha.

4.1. Funciones flecha (arrow function)

Una función flecha es una función anónima declarada de forma más corta, concreta y clara. Las bases de una función flecha son:

- Desaparece la palabra reservada function.
- Los parámetros de entrada a la función van entre paréntesis separados por comas, salvo que haya solo uno que entonces se puede obviar los paréntesis.
- Se utiliza => para separar parámetros de entrada del cuerpo de la función.
- Si la función tiene una única sentencia, se pueden eliminar los paréntesis y la palabra reservada return.

La declaración básica de una arrow function es la siguiente:

const funcion = (arg1, arg2, ...argN) => expresión

Por ejemplo:

```
// Con retorno explícito
const funcion1 = (var1) => { return var1 * 2; };
// Con retorno implícito y sin paréntesis opcionales en argumentos
const funcion2 = var1 => var1 * 2;
```

En el siguiente ejemplo se va a definir una función anónima que calcule el cuadrado de un número:

```
let numero = Number(prompt("Introduce un número "));
const cuadrado = function (base) {
    return base * base;
};
alert("El cuadrado de " + numero + " es " + cuadrado(numero));
```

Con arrow function de retorno explícito quedaría como sigue:

```
let numero = Number(prompt("Introduce un número "));
const cuadrado = (base) => {
    return base * base;
};
alert("El cuadrado de " + numero + " es " + cuadrado(numero));
```

Y por último, con arrow function de retorno implícito:

```
let numero = Number(prompt("Introduce un número "));
const cuadrado = base => base * base;
alert("El cuadrado de " + numero + " es " + cuadrado(numero));
```

Antes de usar las funciones flecha hay que tener en cuenta que:

- El valor de this no puede ser modificado dentro de una función flecha.
- Las funciones flecha no pueden ser utilizadas como constructores del mismo modo que el resto de funciones.
- Estas funciones no disponen de la variable arguments[] como sí tienen otras.

4.2. Manipulación de arrays

Desde la versión 5.1 javascript incorpora métodos que permiten la programación funcional. Los métodos que se van a ver a continuación permiten, en combinación con las funciones flecha, una más sencilla manipulación de arrays.

Dichos métodos (salvo para los que se indique lo contrario) tienen todos la misma entrada:

- Como primer parámetro se le pasa una función(callback) que es la que especifica la condición. Esta función recibe, como primer parámetro, el elemento actual del array. El segundo y tercer parámetro son opcionales y, de existir, serán el índice del elemento y el array completo respectivamente.
- El segundo parámetro es opcional y será un valor que se podrá usar como this cuando se ejecute el callback.

El callback se ejecutará una vez por cada elemento del array.

filter

El método ***filter()*** devuelve un nuevo array formado por los elementos del array original que cumplan la condición especificada en el callback. El callback devuelve true en caso de que haya que incluir el elemento en el array resultante y false en caso contrario.

En el siguiente ejemplo se recuperarán del array los elementos que empiecen por “a”:

```
    let arrayPalabras = ["abedul", "casa", "coche", "rio", "alameda", "a
legría", "portón"];
    let empiezanPorA = arrayPalabras.filter(function (palabra) {
        if (palabra[0] == "a") {
            return true;
        } else {
            return false;
        }
    });
```

El mismo ejemplo anterior, usando funciones flecha:

```
let arrayPalabras = ["abedul", "casa", "coche", "rio", "alameda", "alegr
ía", "portón"];
let empiezanPorA = arrayPalabras.filter(palabra => palabra[0] == "a");
/**La función flecha recibe un elemento del array(se ejecutará tantas ve
ces como elementos haya) y comprueba su primera letra con la a. Dada la
```

```
naturaleza del operador usado, palabra[0] == "a" devuelve true si se
cumple y false en caso contrario*/
```

find, findIndex

find() es un método que devuelve el primer elemento del array que cumpla con la condición especificada o, en caso de no cumplirla ninguno, undefined. Ejecutará el callback por cada elemento del array solo hasta que devuelva true para uno de ellos.

Siguiendo el ejemplo dado para filter():

```
let arrayPalabras = ["casa", "abedul", "coche", "rio", "alameda", "aleg
  ría", "portón"];
let empiezanPorA = arrayPalabras.find(palabra => palabra[0] == "a");
```

Si lo que se quiere recuperar es la posición en la que se encuentra un elemento concreto, se utilizará el método ***findIndex()*** que devolverá el índice del primer elemento que cumpla con lo especificado en el callback. En caso de que este último no devuelva true en ningún caso, findIndex() devolverá -1:

```
let arrayPalabras = ["casa","abedul","coche","rio","alameda","alegría",
"portón"];
let posicion = arrayPalabras.findIndex(palabra => palabra[0] == "a");
```

Ambos métodos ofrecen más posibilidades cuando se utilizan arrays de objetos.

every , some

El método ***every()*** devolverá true si todos los elementos del array cumplen la condición especificada, mientras que al método ***some()*** le basta con que un solo elemento cumpla la condición para devolver true.

En caso de que el array sobre el que se opere esté vacío, every() devolverá true mientras que some() devolverá false.

```
    let arrayPalabras = ["casa", "abedul", "coche", "rio", "alameda", "ale
gría", "portón"];
    let todasEmpiezanPorA = arrayPalabras.every(palabra => palabra[0] == "
a");
    let algunaEmpiezaPorA = arrayPalabras.some(palabra => palabra[0] == "a
");
```

EJERCICIO 5:

- A partir de un array que contenga 15 notas entre 1 y 10, obtén otro en el que solo se incluyan los aprobados.
- Dados los arrays del punto anterior, busca la posición, en ambos, de la nota con valor 5.5. Comprueba el valor devuelto en caso de que no exista dicha nota.

- Dado el array original del primer punto, comprueba si todos son aprobados.
- A partir de un array que contenga 12 palabras, obtén otro en el que solo se incluyan las que tengan menos de 6 caracteres.
- Dado el array original del punto anterior, recupera el primer elemento cuya última letra sea una o.
- Dado un array con 8 números de teléfono, busca si alguno termina en 3.

map

Este método crea un nuevo array permitiendo modificar cada elemento del original (para meterlo en uno nuevo) dentro de la función callback. El array original no se verá modificado.

En el siguiente fragmento pone en mayúsculas la letra inicial de cada valor del array:

```
let arrayPalabras = ["casa", "abedul", "coche", "rio", "alameda", "alegr
ía", "portón"];
let arrayMayus = arrayPalabras.map(palabra => palabra[0].toLocaleUpperCa
se() + palabra.slice(1));
```

reduce

reduce() devuelve un único valor tras ejecutar una función que lo calculará a partir de los elementos del array. Como primer parámetro recibirá el callback y como segundo, de manera opcional, un valor inicial para usar como primer argumento en la primera llamada al callback. Si este segundo argumento no se proveyera, se utilizará el primer elemento del array.

En este caso la función callback recibe los siguientes parámetros:

- Valor Anterior: es el valor resultante de la última llamada al callback o, en caso de haberse incluido en la llamada, el valor inicial.
- Valor Actual: elemento del array a procesar.
- Índice Actual: opcional. Es el índice del elemento a procesar. Empieza en 0 si se incluyó un valor inicial, en 1 en caso contrario.
- Array: también opcional. Es el array completo sobre el que se está trabajando.

En el ejemplo se van a concatenar todos los elementos del array en una sola cadena:

```
let arrayPalabras = ["casa", "abedul", "coche", "rio", "alameda"
let cadenaUnica = arrayPalabras.reduce((valorAnt, valorAct) => valorAnt
+ valorAct);
let cadenaUnica2 = arrayPalabras.reduce((valorAnt, valorAct) => valorAnt
 + valorAct, "Concatenados: ");
```

EJERCICIO 6:

- Prueba los ejemplos incluidos de map y reduce, intenta comprender el resultado de cada uno.
- A partir de un array que contenga 15 notas entre 1 y 10, obtén la media de la clase.
- A partir de un array que contenga 15 notas entre 1 y 10, obtén la nota más alta.

- A partir de un array que contenga 12 precios de una lista de la compra, obtén el gasto total de la misma.
- A partir del array del punto anterior, obtén otro array con los precios con IVA incluido. Aplica a todos un 10% de IVA.

forEach

El método ***forEach()*** ejecutará la función callback por cada elemento del array, pudiéndose cambiar el contenido del mismo. Este método devuelve undefined.

```
    let arrayPalabras = ["casa","abedul","coche","rio","alameda","alegría"
,"portón"];
    arrayPalabras.forEach((elemento,i, arrayPalabras) => {
    arrayPalabras[i] = elemento + "!!";
    console.log("Valor en la posición " + i + ":" + elemento); })
```

includes

includes() devolverá true si el array contiene el valor pasado como parámetro. El segundo parámetro(opcional) indicará la posición del array a partir de la cual buscar.

Hay que tener en cuenta que includes() distingue entre mayúsculas y minúsculas. Además, este es un método bastante genérico por lo que se puede aplicar a diferentes tipos de objetos.

```
    let arrayPalabras = ["casa", "abedul", "coche", "rio", "alameda", "a
legría", "portón"];
    let palabra = "Estetoscopio";
    let existe = arrayPalabras.includes("casa");
    let existe2 = palabra.includes("u");
```

Array.from

Método que crea una nueva instancia de Array a partir de algún objeto iterable (por ejemplo, colecciones). Se podrá crear un array a partir de otro, objetos como Map o Set y objetos con propiedad length.

```
    let palabra = "Estetoscopio";
    let arrayNuevo = Array.from(palabra);
```

Se puede incluir como segundo parámetro una función a ejecutar sobre cada elemento del array que se haya creado. Hay un tercer parámetro, también opcional, que será un valor para usar como this durante la ejecución de la función pasada como segundo parámetro.

```
    let titulares = Array.from(document.getElementsByClassName("titularN
oticia"));
```

EJERCICIO 7:

- Prueba los ejemplos de forEach e includes, intenta comprender el resultado obtenido.
- Crea una página web que permita probar el ejemplo de Array.from. Incluirá:
  - 3 secciones con su atributo class= "titularNoticia". Estas secciones

contendrán una cabecera de una noticia cada una.

- 2 secciones con otro valor en su atributo class. El contenido será un texto cualquiera.
- Un botón que, al ser pulsado, desencadenará la ejecución de una función.
- Esta función mostrará un alert para cada elemento con class="titularNoticia" con el texto que contenga, salvo que dicho texto incluya la cadena "tr" en alguna posición.

5. Copia de arrays

Cuando se copia una variable de tipo primitivo o se pasa una como parámetro a una función, en ambos casos se está haciendo una copia independiente de la misma y, en caso de modificarla después, la variable original no se verá afectada.

Pero, si se copia un objeto o un array (hay que recordar que es un tipo de objeto) lo que se está haciendo es una referencia al objeto original, es decir, tanto la variable nueva como la original apuntan a la misma posición de memoria y los cambios que se hagan en una afectan a la otra ya que no es que sean iguales si no que son las mismas.

Para hacer una copia real del objeto original, se puede hacer uso de métodos como slice. Si lo que se quiere es duplicar un objeto, se hará uso de métodos o propiedades propios de objetos.

```
    let palabra = "Estetoscopio";
    let arrayPalabras = ["casa", "abedul", "coche", "rio", "alameda"
, "alegría", "portón"];
    let palabraCopia = palabra;
    palabra = "Sauce";
    let arrayCopia1 = arrayPalabras;
    let arrayCopia2 = arrayPalabras.slice();
    arrayCopia1[2] = "autobús";
    arrayCopia2[3] = "bahía";
```

EJERCICIO 8:

- Prueba el ejemplo incluido anteriormente sobre la copia de arrays, y comprueba el contenido final de cada variable o array.
- Partiendo de los arrays de notas y precios del ejercicio 7:
  - Haz copias de los mismos tanto directamente como a través de cualquier método de los vistos en apartados anteriores.
  - Modifica las copias y comprueba los casos en que los arrays son copias independientes y los casos en que son referencias al mismo array.