LMSGI

2. Funciones en Javascript

Alejandro Amat Reina

Funciones

- En Javascript se utiliza la palabra reservada **function** antes del nombre de la función.
- Los argumentos se pasan entre paréntesis tras el nombre de la función (recuerda que en Javascript no hay tipos de datos)
- Posteriormente va el cuerpo de la función entre llaves.
- El nombre de la función, como las variables, debe seguir el formato CamelCase con la primera letra en minúscula.

```
function sayHello(name) {
    console.log('Hello' + name);
}
sayHello('Álex');
```

Declaración de funciones

- Puedes declarar el prototipo de la función donde quieras.
- No necesitas tener la función declarada antes de llamarla, ya que Javascript primero procesa las declaraciones de las variables y funciones y después ejecuta el resto del código.
- Para llamar a una función, simplemente, pones el nombre de la función, y entre paréntesis, el valor de los parámetros que le pasas.

Paso de parámetros

- El número de parámetros que le pases puede no coincidir con el que número de parámetros que tiene la declaración de la función:
  - Si envías de más: Los sobrantes son ignorados
  - Si envías de menos: A los que faltan se les asigna el valor undefined
- Ejemplo: Llamada a la función sayHello anterior

```
sayHello(); // Imprimirá undefined
```

Parámetros por defecto

- Si un parámetro se declara en una función y no se pasa cuando la llamamos, se establece su valor como **undefined**.
- El operador boolean **||** puede ser usado para simular valores por defecto en una función

```
function sayHello(name) {
// Si name es undefined o vacío (""), Se le asignará "Anonymous" por defecto
    var sayName = name || "Anonymous";
    console.log("Hello " + sayName);
}
sayHello("Peter"); // Imprime "Hello Peter"
sayHello(); // Imprime "Hello Anonymous"
```

Parámetros por defecto en ES6

- Desde **ES2015** tenemos la opción de establecer un valor por defecto.

```
function printNombre(nombre = "Anónimo") {
    console.log("Nombre: " + nombre);
}
printNombre(); // Imprime Nombre: Anónimo
```

- También podemos asignarle un valor por defecto basado en una expresión

```
function getPrecioTotal(precio, impuesto=precio*0.07) {
    return precio + impuesto;
}
console.log(getPrecioTotal((100))); // Imprime 107
```

Valor de retorno

- Cuando una función devuelve un valor utilizaremos la palabra reservada **return**.
- Para recibir el valor enviado por la función, la llamada deberemos asignársela a una variable o meterla dentro del parámetro de otra función.
- Si intentamos recuperar algo de una función que no devuelve nada nos dará undefined.

```
function totalPrice(priceUnit, units) {
   return priceUnit * units;
}

let total = totalPrice(5.95, 6); //  Imprime 35.7
console.log(totalPrice(5.95, 6)); //  Imprime 35.7
```

Ejercicio 1

- Crea una función que reciba 3 parámetros (nombre de producto, precio e impuesto en porcentaje sobre 100). Dicha función hará lo siguiente:
  - Los parámetros deberán tener un valor por defecto por si no los recibe que deben ser: "Producto genérico", 100 y 21.
  - Convierte el nombre de producto a string (por si acaso) y los otros 2 a número.
  - Si el precio o el impuesto no son números válidos muestra un error.
  - Si son válidos, muestra por consola el nombre del producto y el precio final contando impuestos.
  - Llama a la función varias veces, omitiendo parámetros, con todos ellos, y pasando algún valor no numérico en el precio o impuesto

Ámbito de las variables

- El ámbito de una variable puede ser:
  - Global: Podemos acceder al valor de la variable en cualquier parte del script.
  - Local: Solo podemos acceder a la variable en lugar dónde sea declarada, por ejemplo, en una función.

Modo estricto

- En javascript, cuando usamos una variable que no hemos declarado antes, se declara de forma automática como global.
- Para evitar este comportamiento podemos usar el modo estricto, escribiendo al principio del fichero la siguiente sentencia:

```
'use strict';
```

- Esto obligará a declarar las variables antes de usarlas, por lo que nos evitaremos muchos dolores de cabeza.
- **NOTA: A partir de ahora todos nuestros scripts deben incluir al principio la declaración de modo estricto**

Ámbito de variables

- Javascript utiliza una jerarquía de niveles para buscar el ámbito de una variable.
- Siempre prevalece lo local sobre lo global.
- Cuando Javascript encuentra una variable busca su declaración en el nivel donde se encuentra, si no la encuentra, la busca en el nivel superior, y va subiendo hasta el nivel global.
- Es decir vamos siempre de dentro hacía fuera, por ello las variables globales se ven en todo el programa y las locales no.

Variables globales

- Para declarar una variable usaremos la palabra reservada let.
- Si la declaramos en el bloque principal (fuera de cualquier función), la variable será global.
- Si la declaramos dentro de una función, la variable será local a la función.

```
'use strict';
var global = 'Hola';
function cambiaGlobalç() {
   global = 'Adios';
}
cambiaGlobal();
console.log(global); // Imprime 'Adios'
console.log(window.global); // Imprime 'Adios'
```

- Como se puede observar en el ejemplo, las variables globales declaradas con var se asocian al objeto window, de forma que podemos acceder a ellas como si fueran propiedades de este objeto
- Esto no ocurre declarando la variable con let

Variables locales

- Todas las variables que se declaran dentro de una función son locales

- No podremos acceder a ellas desde fuera de la fuinción

```
function setPerson() {
   let person = 'Álex';
}
setPerson();
console.log(person); // Error -> Uncaught ReferenceError: person is not defined
```

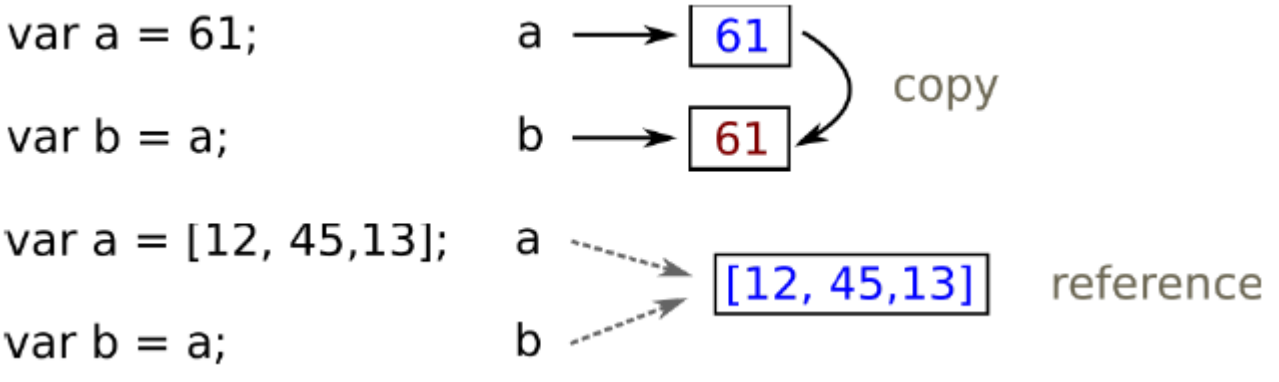
Colisión de nombres de variables

- Si tenemos una variable local con el mismo nombre que una local, dentro del ámbito de la variable local se utilizará esta, pero fuera se utilizará la variable global

```
function setPerson() {
    let person = 'Álex';
    console.log(person); // Imprime 'Álex'
}
let person = 'Pablo';
setPerson();
console.log(person); // Imprime 'Pablo'
```

Paso de parámetros por referencia

- A diferencia de los tipos primitivos como boolean, number o string, los arrays son tratados como un objeto en JavaScript, lo cual significa que tienen un puntero o referencia a la dirección de memoria que contiene el array.
- Cuando se copia una variable o se envía esa variable como parámetro a una función, estamos copiando la referencia y no el array, por tanto ambas variables apuntarán a la misma zona de memoria.
- Si, por ejemplo, cambiamos la información que teníamos almacenada en una de esas variables, estaremos cambiando la información de la otra también ya que son el mismo objeto.



Ejemplo

```
let a = 61;
let b = a; // b = 61
b = 12;
console.log(a); // Imprime 61
console.log(b); // Imprime 12

let a = [12, 45, 13];
let b = a; // b ahora referencia al mismo array que a
b[0] = 0;
console.log(a); // Imprime [0, 45, 13]
```

Ejercicio 2

- Crea una función que reciba un array y un valor, e indique si dicho valor se encuentra en el array o no. Si existe en el array, devolverá la posición del elemento, si no existe devolverá -1.
- Realiza una función que reciba un array numérico y te indique si TODOS sus valores son pares.
- Realiza una función que reciba un array numérico y te indique si ALGUNOS de sus valores son pares.

Ejercicio 3

- Crea un script (usando al menos 1 función) en el que se le pida al usuario 2 datos:
  - El primero será dos números del 1 al 10 (debes dar error en otro caso)
  - El segundo será el modo de visualización: LISTA o TABLA (debe dar error si elige otra cosa)
  - El programa mostrará la tabla de multiplicar del número más pequeño, llegando hasta el número más grande
  - Ojo, tendrás que tener en cuenta que el usuario puede ponerte el segundo número más pequeño que el primero.
  - Ejemplo, si el usuario introduce los número 2 y 4, el resultado de la página será el siguiente

2x1	=	2
2x2	=	4
2x3	=	6
2x4	=	8

Modo tabla

- 2x1=2
- 2x2=4
- 2x3=6
- 2x4=8

Modo lista

Trabajo con strings

- Para ver cómo trabajar con strings en Javascript debes consultar los siguientes enlaces de W3schools:
  - https://www.w3schools.com/js/js_strings.asp
  - https://www.w3schools.com/js/js_string_methods.asp
- Es importante destacar que un string puede ser un dato primitivo ("hola"), o un objeto (String("hola"), y que no son exactamente lo mismo.
- Como puedes ver en el segundo enlace, existen multitud de métodos para trabajar con strings, a continuación se proponen una serie de ejercicios para que puedas familiarizarte con estos métodos.

Ejercicio 4

- Crea una función que reciba un string con un path linux (directorios separados por /) y devuelva un array que contenga todos los nombres de los directorios del path en sus elementos

- Crea una función que reciba dos strings y devuelva otro string que contenga el primero con todas las apariciones del segundo string eliminadas. La búsqueda debe ser case insensitive, es decir, no se distinguirá entre mayúsculas y minúsculas.

- Crea una función que reciba tres strings y devuelva un string que contenga el primer string con todas las apariciones del segundo string sustituidas por el tercer string.

- Crea un string que reciba un nombre de fichero con extensión y devuelva el mismo nombre de fichero, pero sustituyendo la extensión por .bak

Fechas en Javascript

- En Javascript tenemos la clase Date, que encapsula información sobre fechas y métodos para operar, permitiéndonos almacenar la fecha y hora local (timezone).

```
let date = new Date(); // Crea objeto Date almacena la fecha actual
console.log(typeof date); // Imprime object
console.log(date instanceof Date); // Imprime true
console.log(date); // Imprime la fecha convertida a string
```

Creando objetos Date

- Podemos enviarle al constructor el número de milisegundos desde el 1/1/1970 a las 00:00:00 GMT (Llamado Epoch o UNIX time).
- Si pasamos más de un número, (sólo el primero y el segundo son obligatorios),el orden debería ser: 1º → año, 2º → mes (0..11), 3º → día, 4º → hora, 5º → minuto, 6º → segundo.
- Otra opción es pasar un string que contenga la fecha en un formato válido.

```
let date = new Date(1363754739620); // Nueva fecha 20/03/2013 05:45:39 (milisegundos desde Epoch)
let date2 = new Date(2015, 5, 17, 12, 30, 50); // 17/06/2015 12:30:50 (Mes empieza en 0 -> Ene, ... 11 -> Dic)
let date3 = new Date("2015-03-25"); // Formato de fecha largo sin la hora YYYY-MM-DD (00:00:00)
let date4 = new Date("2015-03-25T12:00:00"); // Formato fecha largo con la fecha
let date5 = new Date("03/25/2015"); // Formato corto MM/DD/YYYY
let date6 = new Date("25 Mar 2015"); // Formato corto con el mes en texto (March también sería válido).
let date7 = new Date("Wed Mar 25 2015 09:56:24 GMT+0100 (CET)"); // Formato completo con el timezone
```

Obtener Timestamp

- Si, en lugar de un objeto Date, queremos directamente obtener los milisegundos que han pasado desde el 1/1/1970 (Epoch), lo que tenemos que hacer es usar los métodos Date.parse(string) y Date.UTC(año, mes, día, hora, minuto, segundos).
- También podemos usar Date.now(), para la fecha y hora actual en milisegundos.

```
let nowMs = Date.now(); // Momento actual en ms
let dateMs = Date.parse("25 Mar 2015"); // 25 Marzo 2015 en ms
let dateMs2 = Date.UTC(2015, 2, 25); // 25 Marzo 2015 en ms
```

Getters y setters

- La clase Date tiene setters y getters para las propiedades: fullYear, month (0-11), date (día), hours, minutes, seconds, y milliseconds.
- Si pasamos un valor negativo, por ejemplo, mes = -1, se establece el último mes (dic.) del año anterior.

```
// Crea un objeto fecha de hace 2 horas
let twoHoursAgo = new Date(Date.now() - (1000*60*60*2));
// (Ahora - 2 horas) en ms
// Ahora hacemos lo mismo,pero usando el método setHours
let now = new Date();
now.setHours(now.getHours() - 2);
```

Convertir a string

- Cuando queremos imprimir la fecha, tenemos métodos que nos la devuelven como string en diferentes formatos:

```
let now = new Date();
console.log(now.toString());
console.log(now.toISOString()); // Imprime 2016-06-26T18:00:31.246Z
console.log(now.toUTCString()); // Imprime Sun, 26 Jun 2016 18:02:48 GMT
console.log(now.toDateString()); // Imprime Sun Jun 26 2016
console.log(now.toLocaleDateString()); // Imprime 26/6/2016
console.log(now.toTimeString()); // Imprime 20:00:31 GMT+0200 (CEST)
console.log(now.toLocaleTimeString()); // Imprime 20:00:31
```

Más información

- Puedes obtener más información sobre el objeto Date en los siguientes enlaces de w3schools:
  - https://www.w3schools.com/js/js_dates.asp
  - https://www.w3schools.com/js/js_date_formats.asp
  - https://www.w3schools.com/js/js_date_methods.asp
  - https://www.w3schools.com/js/js_date_methods_set.asp

Ejercicio 5

- Crea un script que muestre la fecha actual en los siguientes formatos:
  - Formato español (dd/mm/aaaa hh:mm:ss)
  - Formato americano (mm/dd/aaaa hh:mm:ss)
- No puedes usar el método toLocaleString

- Repite el ejercicio utilizando el método toLocaleString
- Puedes encontrar información del método en el siguiente enlace:
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/toLocaleString

Métodos de arrays

- Al igual que ocurre con el resto de objetos existentes en Javascript (String, Date, etc.), el objeto Array también dispone de multitud de métodos que nos harán más sencillo el trabajo con arrays.

- A continuación veremos los más destacados

Insertar y mostrar elementos

- Podemos insertar elementos al principio de un array (**unshift**) o al final (**push**).

```
let a = [];
a.push("a"); // Inserta el valor al final del array
a.push("b", "c", "d"); // Inserta estos nuevos valores al final
console.log(a.valueOf()); // Imprime ["a", "b", "c", "d"]. Puedes omitir valueOf(), aun así será invocado
a.unshift("A", "B", "C"); // Inserta nuevos valores al principio del array
console.log(a.toString()); // Imprime A,B,C,a,b,c,d. toString() es un poco diferente de valueOf()
```

- Como ves se puede convertir un array a string con el método toString y con el método valueOf

Eliminar elementos del array

- Podemos eliminar elementos del principio (shift) y también del final (pop)
- Estas operaciones nos devolverán el valor que ha sido eliminado

```
console.log(a.pop()); // Imprime y elimina la última posición → "d"
console.log(a.shift()); // Imprime y elimina la primera posición → "A"
console.log(a); // Imprime ["B", "C", "a", "b", "c"]
```

Convertir a string

- Podemos convertir un array a string usando join() en lugar de toString().
- De esta forma, podremos elegir el separador a utilizar.
- Por defecto, devuelve un string con todos los elementos separados por coma.

```
let a = [3, 21, 15, 61, 9];
console.log(a.join()); // Imprime "3,21,15,61,9""
console.log(a.join(" -#- ")); // Imprime "3 -#- 21 -#- 15 -#- 61 -#- 9"
```

Concatenar arrays

- Usamos el método concat

```
let a = ["a", "b", "c"];
let b = ["d", "e", "f"];
let c = a.concat(b);
console.log(c); // Imprime ["a", "b", "c", "d", "e", "f"]
console.log(a); // Imprime ["a", "b", "c"] . El array a no ha sido modificado
```

Obtener partes de un array

- El método slice nos devuelve un nuevo array a partir de posiciones intermedias de otro

```
let a = ["a", "b", "c", "d", "e", "f"];
let b = a.slice(1, 3); // (posición de inicio → incluida, posición final → excluida)
console.log(b); // Imprime ["b", "c"]
console.log(a); // Imprime ["a", "b", "c", "d", "e", "f"]. El array original no es modificado
console.log(a.slice(3)); // Un parámetro. Devuelve desde la posición 3 al final → ["d", "e", "f"]
```

Eliminar e insertar elementos

- splice elimina elementos del array original y devuelve los elementos eliminados.
- También permite insertar nuevos valores

```
let a = ["a", "b", "c", "d", "e", "f"];
a.splice(1, 3); // Elimina 3 elementos desde la posición 1 ("b", "c", "d")
console.log(a); // Imprime ["a", "e", "f"]
a.splice(1,1, "g", "h"); // Elimina 1 elemento en la posición 1 ("e"), e inserta "g", "h" en esa posición
console.log(a); // Imprime ["a", "g", "h", "f"]
a.splice(3, 0, "i"); // En la posición 3, no elimina nada, e inserta "i"
console.log(a); // Imprime ["a", "g", "h", "i", "f"]
```

Invertir los elementos del array

- Podemos invertir el orden del array usando el metodo **reverse**

```
let a = ["a", "b", "c", "d", "e", "f"];
a.reverse(); // Hace el reverse del array original
console.log(a); // Imprime ["f", "e", "d", "c", "b", "a"]
```

Ordenar un array

- Podemos ordenar los elementos de un array usando el método **sort**

```
let a = ["Peter", "Anne", "Thomas", "Jen", "Rob", "Alison"];
a.sort(); // Ordena el array original
console.log(a); // Imprime ["Alison", "Anne", "Jen", "Peter", "Rob", "Thomas"]
```

Ejercicio 6

- Realiza los siguientes pasos (muestra por consola el resultado después de aplicar cada uno, pero con los elementos separados por "==>" (Join)):
  - Crea un array con 4 elementos numéricos
  - Concatena 2 elementos más al final y 2 al principio (unshift y push)
  - Elimina las posiciones de la 3 a la 5 (incluida) (splice)
  - Inserta 2 elementos más entre el penúltimo y el último (splice)
  - Invierte el array (reverse)
  - Muestra el array ordenado (sort)

Funciones anónimas

- Como su nombre indica, son funciones sin nombre.
- Podemos asignar dicha función a una variable.
- Se utiliza igual que una función clásica.

```
let totalPrice = function (priceUnit, units) {
    return priceUnit * units;
}
// imprime "function" (tipo de la variable totalPrice)
console.log(typeof totalPrice);
console.log(totalPrice(5.95, 6)); //  Imprime 35.7
//  Referenciamos a la misma función desde la variable getTotal
let getTotal = totalPrice;
console.log(getTotal(5.95, 6)); //  Imprime 35.7
```

Funciones anónimas & Funciones con nombre

- No tienen diferencia en cuanto a rendimiento u operatividad
- Con las funciones anónimas ahorras código y se suelen usar si no van a ser llamadas repetidas veces, también son el primer paso para las funciones flecha que veremos a continuación.
- Si vamos a usar mucho la función o hacer recursividad es conveniente funciones con nombre.
- La principal diferencia es el hoisting antes mencionado. La declaración de variables y funciones con nombre se van SIEMPRE al principio del ámbito.

```
function miFunc()
{
        console.log(v()); // Dará error
        let v = function () {
            return 5;
        }
}
```

```
function miFunc()
{
    console.log(v()); // Imprimirá 5
    function v() {
        return 5;
    }
}
```

Uso incorrecto de funciones anónimas

- Al usar funciones anónimas no debemos caer en errores de este tipo:

```
for (let i = 0;i < colElementos.length;i++) {
    colElementos[i].onclick = function () { ... };
}
```

- En el ejemplo anterior estamos creando una función nueva en cada iteración del bucle.
- Es mucho más eficiente declarar la función fuera, ya sea anónima o con nombre, y dentro del bucle simplemente usarla.

```
let manejador = function() { ... };

for (let i = 0;i < colElementos.length;i++) {
    colElementos[i].onclick = manejador;
}
```

Ordenar array con función de ordenación

- Podemos controlar la ordenación de los elementos de un array pasándole una función de ordenación al método sort:

```
let a = [20, 6, 100, 51, 28, 9];
a.sort(); // Ordena el array original
console.log(a); // Imprime [100, 20, 28, 51, 6, 9]
a.sort(function(n1, n2) {
   return n1 - n2;
});
console.log(a); // Imprime [6, 9, 20, 28, 51, 100]
```

Ejercicio 7

- Crea una función que reciba un array de strings y lo devuelva ordenado alfabéticamente, de forma ascendente o descendente, en función de un segundo parámetro que recibirá la función para indicar el tipo de ordenación.

Buscar elementos en array

- Usando indexOf, podemos conocer si el valor que le pasamos se encuentra en el array o no.
- Si lo encuentra nos devuelve la primera posición donde está, y si no, nos devuelve -1.
- Usando el método lastIndexOf nos devuelve la primera ocurrencia encontrada empezando desde el final

```
let a = [3, 21, 15, 61, 9, 15];
console.log(a.indexOf(15)); // Imprime 2
console.log(a.indexOf(56)); // Imprime -1. No encontrado
console.log(a.lastIndexOf(15)); // Imprime 5
```

Verificar una condición en todos los elementos de un array

- El método every devolverá un boolean indicando si todos los elementos del array cumplen cierta condición.
- Esta función recibirá cualquier elemento, lo testeará, y devolverá cierto o falso dependiendo de si cumple la condición o no.

```
let a = [3, 21, 15, 61, 9, 54];
console.log(a.every(function(num) { // Comprueba si cada número es menor a 100
   return num < 100;
})); // Imprime true
console.log(a.every(function(num) { // Comprueba si cada número es par
   return num % 2 == 0;
})); // Imprime false
```

Verificar una condición en alguno de los elementos de un array

- El método some es similar a every, pero devuelve cierto en el momento en el que uno de los elementos del array cumple la condición.

```
let a = [3, 21, 15, 61, 9, 54];
console.log(a.some(function(num) { // Comprueba si algún elemento del array es par
   return num % 2 == 0;
})); // Imprime true
```

Ejercicio 8

- Crea una función que reciba un array como parámetro y devuelva cierto si todos los elementos son pares y falso en caso contrario.
- Además, la función recibirá un segundo parámetro booleano para indicar si quieres verificar números pares o impares. El valor por defecto de este parámetro será que verifique números pares.
- Debes usar every

Recorrer un array

- Podemos iterar por los elementos de un array usando el método forEach.
- De forma opcional, podemos llevar un seguimiento del índice al que está accediendo en cada momento, e incluso recibir el array como tercer parámetro.

```
let a = [3, 21, 15, 61, 9, 54];
let sum = 0;
a.forEach(function(num) { //
   sum += num;
});
console.log(sum); // Imprime 163
a.forEach(function(num, indice, array) { // índice y array son parámetros opcionales
   console.log("Índice " + indice + " en [" + array + "] es " + num);
}); // Imprime -> Índice 0 en [3,21,15,61,9,54] es 3, Índice 1 en [3,21,15,61,9,54] es
21, ...
```

Ejercicio 9

- Crea una función que reciba un array con nombres de alumnos y muestre por pantalla los nombres ordenados alfabéticamente y numerados.
- Utiliza forEach para recorrer el array

Modificar todos los elementos de un array

- Para modificar todos los elementos de un array, el método map recibe una función que transforma cada elemento y lo devuelve.
- Este método devolverá al final un nuevo array del mismo tamaño conteniendo todos los elementos resultantes.

```
let a = [4, 21, 33, 12, 9, 54];
console.log(a.map(function(num) {
    return num*2;
})); // Imprime [8, 42, 66, 24, 18, 108]
```

Ejercicio 10

- Usando map, crea una función que reciba un array y devuelva otro donde si los números son pares los divida entre 2 y si son impares los multiplique por 2.

Filtrar elementos de un array

- Para filtrar los elementos de un array, y obtener como resultado un array que contenga sólo los elementos que cumplan cierta condición, usamos el método filter.

```
let a = [4, 21, 33, 12, 9, 54];
console.log(a.filter(function(num) {
   return num % 2 == 0; // Si devuelve true, el elemento se queda en el array devuelto
})); // Imprime [4, 12, 54]
```

Ejercicio 11

- Crea una función que reciba un array y devuelva otro solo con sus elementos pares.
- Usa filter

Realizar cálculos con todos los elementos de un array

- El método reduce usa una función que acumula un valor, procesando cada elemento (segundo parámetro) con el valor acumulado (primer parámetro).
- Como segundo parámetro de reduce, deberías pasar un valor inicial.
- Si no pasas un valor inicial, el primer elemento de un array será usado como tal (si el array está vacío devolvería undefined).

```
let a = [4, 21, 33, 12, 9, 54];
console.log(a.reduce(function(total, num) { // Suma todos los elementos del array
    return total + num;
}, 0)); // Imprime 133
console.log(a.reduce(function(max, num) { // Obtiene el número máximo del array
    return num > max? num : max;
}, 0)); // Imprime 54
```

Ejercicio 12

- Usando reduce.
- Crea una función que se le pase un array de números como argumento y muestre el número más pequeño, el número más grande y la suma de todos sus elementos.

Lo mismo pero al contrario

- Si prefieres hacer lo mismo que reduce hace pero al revés, usaremos reduceRight

```
let a = [4, 21, 33, 12, 9, 154];
console.log(a.reduceRight(function(total, num) { // Comienza con el último
número y resta todos los otros
   números
   return total - num;
}));
// Imprime 75 (Si no queremos enviarle un valor inicial, empezará con el
valor de la última posición del array
```

Paso de arrays por referencia

- Como hemos comentado anteriormente los arrays se pasan a las funciones por referencia, por lo tanto si modificamos sus elementos, se modificarán en el array original pasado a la función

```
function changeNumber(num) {
    let localNum = num; // Variable local. Copia el valor
    localNum = 100; // Cambia solo localNum. El valor fue copiado
}
let num = 17;
changeNumber(num);
console.log(num); // Imprime 17. No ha cambiado el valor original

function changeArray(array) {
    let localA = array; // Variable local. Hace referencia al original
    localA.splice(1,1,101, 102); // Elimina 1 item en la posición1 e inserta 101 y
102 ahí
}
let a = [12, 45, 13];
changeArray(a);
console.log(a); // Imprime [12, 101, 102, 13]. Ha sido cambiado dentro de la
función
```

Funciones lambda o arrow functions

- Una de las funcionalidades que se añadió en ES2015 fue la posibilidad de usar las funciones lambda (o funciones flecha).
- Otros lenguajes como C# o Java también las soportan.
- Ofrecen la posibilidad de crear funciones anónimas pero con algunas ventajas.
- Veamos una función hecha como anónima y justo debajo como arrow:

```
let sum = function (num1, num2) {
    return num1+ num2;
};
console.log(sum(12, 5)); // imprime 17

let sum = (num1, num2) => num1 + num2;
console.log(sum(12, 5)); // imprime 17
```

Funciones lambda simples

- Cuando declaramos una función lambda, la palabra reservada **function** no se usa.
- Si solo se recibe un parámetro, los paréntesis pueden ser omitidos.
- Después de los parámetros debe ir una fecha (**=>**), y a continuación, el contenido de la función.
- Si solo hay una instrucción dentro de la función lambda, podemos omitir las llaves '{}', y **debemos omitir** la palabra reservada **return** ya que lo hace de forma implicita (devuelve el resultado de esa instrucción).

```
let square = num => num * num;
console.log(square(3)); // Imprime 9
```

Funciones lambda complejas

- Si hay mas de una instrucción, usamos las llaves, y si devuelve algo, debemos usar la palabra reservada return.

```
let sumInterest = (price, percentage) => {
    let interest = price * percentage / 100;
    return price + interest;
};
console.log(sumInterest(200, 15)); // Imprime 230
```

Ventajas de las funciones lambda

- Conseguimos un código más compacto, y conseguimos expresar las funciones de un modo resumido.
- Es ideal para el uso de las propiedades asíncronas del lenguaje JavaScript que usa mucho la parte del servidor Node.js con el uso de promesas.
- Al ser una función anónima se puede usar en el mismo lugar que se precisan, sin necesidad de declararlas antes.

Ejercicio 13

- Repite los ejercicios 10, 11 y 12 utilizando funciones lambda

IIFEs

- Un IIFE (Immediately Invoked Function Expression), es una función autoejecutable (no hay que llamarla) cuyo contexto existe de forma separada del contexto global.

- Ejemplo:

```
'use strict';
(function() {
    console.log("I'm automatically called");
})(); // Imprime "I'm automatically called"
```

Separación de ámbito

- Al estar el código dentro de la función, está separado del ámbito global (variables, funciones).

```
'use strict';
(function() {
    let num = 14;
})();
console.log(num); // Uncaught ReferenceError: num is not defined
```

Objetos globales en los IIFE

- Podemos pasar objetos globales a un IIFE dentro de los paréntesis de cierre (y declararlos como parámetros al principio)
- Esto se suele usar para crear objetos globales como una librería (jQuery por ejemplo), y dentro asignarle sus métodos propiedades, etc.
- Las variables y funciones declaradas dentro de un IIFE no son accesibles desde fuera, por lo que no crean posibles conflictos con otras variables o funciones creadas por el usuario (u otras librerías) que se llamen igual.

```
'use strict';
let Library = {}; // Objeto que contiene métodos de la librería y propiedades
(function(library) {
    let localName = "Peter"
    library.sayHello = function() { // Creando un nuevo método dentro de IIFE
        console.log("Hello " + localName); // Desde aquí podemos acceder a localName
    }
})(Library); // Pasamos el objeto global a la función del IIFE
Library.sayHello(); // Imprime "Hello Peter"
```

Recomendación de uso

- Muchas veces es recomendable, cuando nuestro código está separado en ficheros, meter el código de cada fichero (lo que no tenga que ser global) dentro de un IIFE.

- Así nos aseguramos que no entra en conflicto con código de otros ficheros y el riesgo que ello supone (machacar valores de variables o funciones de otros archivos, etc.).

Ejercicio 14

- Repite el ejercicio 9 llamando a la función que ordena el array desde un IIFE.