ARRAYS. Introduccion

Los Arrays en JavaScript son **estructuras de datos importantes** y tienen muchos usos dentro del desarrollo de software. Además, poseen métodos con los cuales puedes hacer que tu código sea más legible y con menos líneas.

Conceptos

Son un tipo de objeto y no tienen tamaño fijo sino que podemos añadirle elementos en cualquier momento. Podemos crearlos como instancias del objeto Array:

```
let a=new Array();          // a = []
let b=new Array(2,4,6);     // b = [2, 4, 6]
```

Pero lo recomendado es crearlos usando notación JSON (recomendado):

```
let a=[];

let b=[2,4,6];
```

Sus elementos pueden ser de cualquier tipo, incluso podemos tener elementos de tipos distintos en un mismo array. Si no está definido un elemento su valor será *undefined*. Ej.:

```
let a=['Lunes', 'Martes', 2, 4, 6];
console.log(a[0]);        // imprime
'Lunes'
console.log(a[4]);        // imprime 6
a[7]='Juan';          // ahora a=['Lunes',
'Martes', 2, 4, 6, , , 'Juan']
console.log(a[7]);  // imprime 'Juan'
console.log(a[6]);  // imprime undefined
```

Propiedades de un array

Length → Esta propiedad devuelve la longitud de un array:

```
let a=['Lunes', 'Martes', 2, 4, 6];

console.log(a.length);    // imprime
5
```

Podemos reducir el tamaño de un array cambiando esta propiedad:

```
a.length=3;  // ahora a=['Lunes',
             'Martes', 2]
```

Añadir un elemento

Añadir/Eliminar Elementos	
. push(*elemento***)**	***Añade uno o varios elementos al final del array.***
. pop()	***Elimina y devuelve el último elemento del array.***
. unshift(*elemento***)**	***Añade uno o varios elementos al inicio del array.***
. shift()	***Elimina y devuelve el primer elemento del array.***
. concat(*elemento***)**	***Concatena los elementos (o elementos de los arrays) pasados por parámetro.***

Podemos añadir elementos al final de un array con **`push`** o al principio con **`unshift`**:

```
let a=['Lunes', 'Martes', 2, 4, 6];
a.push('Juan');    // a=['Lunes', 'Martes', 2, 4, 6, 'Juan']
a.unshift(7);      // a=[7, 'Lunes', 'Martes', 2, 4, 6, 'Juan']
```

Podemos borrar el elemento del final de un array con **`pop`** o el del principio con **`shift`**. Ambos métodos devuelven el elemento que hemos borrado:

```
let a=['Lunes', 'Martes', 2, 4, 6];
let ultimo=a.pop();      // a=['Lunes', 'Martes', 2, 4] y ultimo=6 let primero=a.shift();       // a=['Martes', 2, 4] y primero='Lunes'
```

Crear un array derivado

Crear array derivado	
. slice(*inicio, num_elem***)**	Devuelve los elementos desde la posición "inicio".
. join(*separador***)**	Construye una cadena, uniendo los elementos del array mediante el separador
. split(*separador***)**	Construye un array, a partir de una cadena y un separador.

Por ejemplo, **Slice**, Devuelve un subarray con los elementos indicados pero sin modificar el array original

```
let a=['Lunes', 'Martes', 2, 4, 6];
```

```
let subArray=a.slice(1, 3);    // a=['Lunes', 'Martes', 2, 4, 6]  // subArray=['Martes', 2, 4];
```

Podemos convertir los elementos de un array a una cadena con **`.join()`** especificando el carácter separador de los elementos.

```
let a=['Lunes', 'Martes', 2, 4, 6];
let cadena=a.join('-');       // cadena='Lunes-Martes-2-4-6'
```

Búsqueda y comprobación

Búsqueda y comprobación	
Array.**isArray**(*obj*)	*Comprueba si obj es un array. Devuelve true o false.*
includes(*obj, from*)	*Comprueba si obj es uno de los elementos incluidos en el array.*
.**indexOf**(*obj, from*)	*Devuelve la posición de la primera aparición de obj desde from.*

Includes → Devuelve **true** si el array incluye el elemento pasado como parámetro. Ejemplo:

```
let arrayNotas = [5.2, 3.9, 6, 9.75, 7.5, 3];

arrayNotas.includes(7.5);      // true
```

Ordenación

Ordenación	
. **reverse**()	*Invierte el orden de elementos del array.*
. **sort**()	*Ordena los elementos del array, ordenación alfabética.*

Sort → Ordena **alfabéticamente** los elementos del array

```
let a=['hola','adios','Bien','Mal',2,5,13,45]

let b=a.sort();      // b=[13, 2, 45, 5, "Bien",
"Mal", "adios", "hola"]
```

Array Functions

Son métodos propios de arrays, que permiten operar sobre todos los elementos del array para alcanzar un objetivo concreto.

- *Se les pasa una función de callback que se ejecutará en cada uno de los elementos del array*

Arrays Functions	
.**forEach***(cb, arg)*	*Realiza la operación definida en cb* ***por cada elemento*** *del array.*
.**every***(cb, arg)*	*Comprueba si* ***todos los elementos*** *del array cumplen la condición de cb.*
.**some***(cb, arg)*	*Comprueba si* ***al menos un elemto*** *del array cumple la condición de cb.*
.**map***(cb, arg)*	*Construye un array* ***con lo que devuelve*** *cb por cada elemento del array.*
.**findIndex***(cb, arg)*	*Devuelve la* ***posición del elemento que cumple*** *la condición de cb.*
.**find***(cb, arg)*	*Devuelve el* ***elemento que cumple*** *la condición de cb.*
. **sort**(*func*)	***Ordena los elementos*** *del array bajo un criterio de ordenación func.*

Resumiendo, las funciones anteriores pueden ser utilizadas para:

```
var array = ['a', 'bb', 'bc', 'd'];

    array.forEach(  function(e,i) { alert('Elemento.' + e
                    + ' en la posición' + i);
                });


    array.every(  e => e.length == 1 );                 // false
    array.some(  e => e.length == 2 );                  // true
    var nuevoArr = array.map( e => e.length );          // [1 , 2 , 2 , 1]
    var nuevoArr = array.filter(  e => e[0] == 'b' );   // ['bb','bc']
    var valor  = array.find( e => e[0] == 'b');         //'bb'
```

Creacion Array .Ejemplos

El método inverso llamado pop extrae el último elemento del Array y decrementa en uno el atributo length:

```
let vec=[];
vec.push(10,20,30,40);
document.write(vec.length+'<br>');  //imprime 4
vec.pop();
document.write(vec.length+'<br>');  //imprime 3
document.write(vec.pop()+'<br>');
document.write(vec.length+'<br>');
```

El método pop() además de eliminar el último elemento del vector retorna el valor almacenado en dicha componente.

```
if (valor < 100) {
            vec.unshift(valor);
        } else {
            vec.push(valor);
        }

<script>
    let vec = [];
    for (let f = 0; f < 10; f++) {
        let valor = parseInt(Math.random() * 1000);
        vec.push(valor);
    }
    document.write('Vector antes de borrar<br>');
    for (let f = 0; f < 10; f++) {
        document.write(vec[f] + '<br>');
    }
    for (let f = 0; f < 10; f = f + 2) {
        delete vec[f];
    }
    document.write('Vector luego de borrar las posiciones pares<br>');
    for (let f = 0; f < 10; f++) {
        document.write(vec[f] + '<br>');
    }
  </script>
```

Metodo Sort:

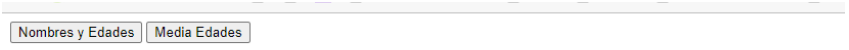
```
<!DOCTYPE html>
<html>

<head>
    <title>Ejemplo de JavaScript</title>
```

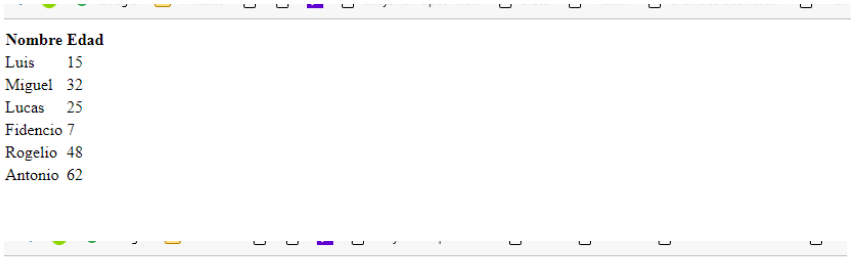
```
    <meta charset="UTF-8">
</head>

<body>

    <script>
        let nombres = ['marcos', 'ana', 'luis', 'jorge', 'carlos'];
        document.write('Vector antes de ordenarlo<br>');
        for (let f = 0; f < nombres.length; f++) {
            document.write(nombres[f] + '<br>');
        }
        nombres.sort();
        document.write('Vector después de ordenarlo<br>');
        for (let f = 0; f < nombres.length; f++) {
            document.write(nombres[f] + '<br>');
        }
    </script>

</body>

</html>
```

Para ordenar una lista de enteros se complica el algoritmo ya que debemos pasar al método sort una función anónima indicando como implementar la comparación entre elementos:

```
<!DOCTYPE html>
<html>

<head>
    <title>Ejemplo de JavaScript</title>
    <meta charset="UTF-8">
</head>

<body>

    <script>
        let vec = [100, 5, 60, 3, 90];
        document.write('Vector antes de ordenarlo<br>');
        for (let f = 0; f < vec.length; f++) {
            document.write(vec[f] + '<br>');
        }
        vec.sort(function(v1, v2) {
            if (v1 < v2)
                return -1; // v1 es menor que v2
            else
                return 1; // v1 es mayor que v2
        });
        document.write('Vector después de ordenarlo<br>');
        for (let f = 0; f < vec.length; f++) {
            document.write(vec[f] + '<br>');
        }
    </script>

</body>

</html>
```

Ejercicio A:






```
let aNombres = []; //Array para almacenar nombres
let aEdades = []; //Array para almacenar edades

//Introducir en el elemento 0 del primer array un nombre
aNombres.push('Luis');
aNombres.push('Miguel');
aNombres.push('Lucas');
aNombres.push('Fidencio');
aNombres.push('Rogelio');
aNombres.push('Antonio');
//En el elemento 0 del segundo array su edad
aEdades.push(15);
aEdades.push(32);
aEdades.push(25);
aEdades.push(7);
aEdades.push(48);
aEdades.push(62);
//Calcula la media de las edades
nMediaEdades= 0;
aEdades.forEach(function(nEdad){//Suma de todas las edades
   nMediaEdades += nEdad;
});
nMediaEdades = nMediaEdades/aEdades.length;//dividido entre la cant de edades

//Muestra pantalla cada nombre con su edad
function MuestraPersonas(){
   let vPersonas = window.open('Personas.html');//abre nueva pantalla
   //al cargar la pantalla colocar la lista en el elemento con id "Lista"
   vPersonas.onload = function(){
      vPersonas.document.getElementById("Lista").innerHTML=ListaPersonas();
   }
}


//Muestra patalla para ver la media de todas las edades.
function MuestraMedia() {
   let vMediaEdad = window.open('Media.html');//abre nueva pantalla
   //al cargar la pantalla colocar el mensaje con la media de las edades en el elemento con id
"Lista"
   vMediaEdad.onload = function(){
      vMediaEdad.document.getElementById("Lista").innerHTML='La media de las edades
es:'+nMediaEdades;
   }
}
//Genera la lista por cada persona
function ListaPersonas() {
   let sTablaPersonas ='<table>';//creara un string con la tabla que sera colocada en la nueva
pantalla
   sTablaPersonas += '<tr><th>Nombre</th><th>Edad</td></tr>';
   aNombres.forEach(function(sNombre, index){//crea fila por cada nombre y edad
      sTablaPersonas += '<tr><td>'+sNombre+'</td><td>'+aEdades[index]+'</td></tr>';
   });
   sTablaPersonas += '</table>';
   return sTablaPersonas;//regresa la lista creada
```

```
}
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div id="Lista"></div>
</body>
</html>
```

OBJETOS DEFINIDOS POR EL USUARIO. FUNCIONES Y CLASES

Javascript es un lenguaje que permite el uso de objetos. Muchas veces el término clase y objeto se confunden. Una definición podría ser que una clase define "***como es un objeto***" y que un objeto es la plasmación efectiva de ese objeto. A partir de una clase se pueden crear si se desea muchos objetos.

Para entenderlo ***un ejemplo***:
Supongamos tenemos la clase "casa". Esa clase define que atributos tiene una casa. Un ejemplo de esos atributos podría ser: dirección, numero habitaciones, metros cuadrados.

Ahora bien, cada objeto es una casa existente. Podemos tener por ejemplo dos objetos que surgen a partir de la clase "casa". Uno seria una casa con dirección "Avenida del puerto 1, Valencia", 3 habitaciones y 100m2 y otro una casa con dirección "Calle Colón 1, Valencia", 5 habitaciones y 200m2.
La clase definía como podían ser los objetos y los objetos en si son clases contextualizadas en algo concreto.

FUNCIONES

Hay varias formas de crear funciones en Javascript:

- Por **declaración** o por **expresión(***anónima, lambda***) o flecha(***Ecmas6***)**:

Definición	Descripción

function nombre (p_1,p_2....) { }	Crea una función mediante **declaración**
nombre = function (p_1,p_2....) { }	Crea una función mediante **expresión**
Nombre = () => { };	Crea una función mediante **flecha**

return

******Funciones flecha*** *es igual a funciones por expresión pero sin usar la palabra reservada function.*

DEFINIR CLASES Y OBJETOS

Funciones constructoras

La forma de definir clases en Javascript es ligeramente distinta a la utilizada en otros lenguajes de programación. Aquí la forma de definir una clase y su constructor asociado es simplemente definir una función.

```
                        / ***** Definir la Clase **********/
function PersonaF(nombre, edad){
    this.nombre = nombre;      // Propiedades
    this.edad = edad;
                                                    // Métodos
    this.cumplirAnos = function (incremento){
                    this.edad=this.edad + incremento;
            };
    }
                    / ***** Instanciar la Clase **********/
    let Manuel = new PersonaF("Manuel",36);
    Manuel.cumplirAnos(1);
    console.log(`</br> Nombre: ${Manuel.nombre} y edad : ${Manuel.edad}`);
```

Objeto Json

Un objeto JSON está rodeado de llaves {}. Siendo su estructura especificada como pares clave / valor.

```
ana = {
      nombre: "Ana María",                          // Propiedades
      edad: 40 ,
      cumplirAnos: function(incremento){            // Métodos
               this.edad = this.edad + incremento;
       }
      };

ana.cumplirAnos(1);
console.log(`</br> Nombre: ${Manuel.nombre} y edad : ${Manuel.edad}`);
```

Ejemplo:

Crear un objeto llamado tvSamsung con lsa propiedades, nombre (TV Samsung 42”), categoria (Televisores), unidades (4), precio(345.95) y con un metodo llamado importe, que devuelve el valor total de las unidades (nª de unidades * precio).

```
1  //Objeto TV Samsung usando programación orientada a objetos
2
3  const tvSamsung = {
4    nombre: 'TV Samsung 42"',
5    categoria: "Televisores",
6    unidades: 4,
7    precio: 345.95,
8    importe: function() {
9      return this.unidades * this.precio
10   }
11
12 }
13
14 console.log(tvSamsung);
15 console.log("El importe total es: ", tvSamsung.importe());
```

CLASES EN ESMAC 6

ES6 (entre otras novedades) incorpora una nueva forma de definir clases. Esto hará que los programadores que vienen de otros lenguajes se sientan mas cómodos y que podamos aplicar conceptos de la POO como la herencia.

La sintaxis de clases es muy similar a lenguajes como JAVA o C#:

```
class personaClass {

    constructor(nombre, edad) {

            this._nombre = nombre;

            this._edad = edad;

    }


    get edad() {

            return this._edad;

    }


    set edad(valor) {
            this._edad = valor;

    }


    cumplirAnos(incremento){

            this.edad = this.edad + incremento;

    }
    imprimirInfo() {
            document.write("Nombre" + this.nombre);

            document.write(", Edad " + this.edad);

    }


    static confirmacion() {

            alert("Enhorabuena esta funcionando la clase Persona");
    }
}
```

```
var mama = new personaClass("Ana", 67);
mama.cumplirAnos(1); mama.imprimirInfo();
mama.edad;
personaClass.confirmacion();
```

Constructor

El constructor es un método especial que inicializa una instancia de la clase. En JavaScript solo puede haber un constructor, no se admite la sobrecarga. Hemos visto en los ejemplos anteriores que usaremos **this** para asignar los valores contextuales y que podemos usar valores por defecto.

```
constructor(nombre, edad) {
        this._nombre = nombre;
        this._edad = edad;
 }
```

Getter y Setter

Estos métodos de acceso nos permiten obtener y asignar valores a los atributos de los objetos. En JavaScript estos métodos se corresponden con atajos sintácticos sin funcionalidad adicional. Deberemos tener en cuenta que podemos caer en una referencia circular lo que daría lugar a un desbordamiento. El uso del guion bajo evitará estos errores.

```
get edad() {
        return this._edad;
}

set edad(valor) {
        this._edad = valor;
}
```

Herencia

Una vez que JS tiene clases, podemos esperar crear clases hijas a partir de otras, o como también se suele expresar: crear clases que extiendan a otras.

```
class nieto extends personaClass {
   constructor(nombre, edad, cuidador) {
        super(nombre, edad);
        this.cuidador = cuidador;
   }

   imprimirInfo() {
        super.imprimirInfo();
        document.write(', Cuidador :' + this.cuidador);
   }
```

```
var miguel = new nieto("Miguelito",6 , "Saly");
miguel.imprimirInfo();
miguel.cumplirAnos(1);

document.write(", Edad:" + miguel.edad);
```

TRABAJANDO CON CLASES

ToString()

Al convertir un objeto a string (por ejemplo al concatenarlo con un String) se llama al método *.toString()* del mismo, que devuelve [object object]. Podemos sobrecargar este método para que devuelva lo que queramos:

```
class personaClass {
    ...
    toString() {
        return this.nombre
    }
}
let cpo = new personaClass('Carlos', 19);
console.log('Persona:' + cpo)   // imprime 'Persona: Carlos'
```

valueOf()

Al comparar objetos (con >, <, …) se usa el valor devuelto por el método *.toString()* pero si sobrecargamos el método ***.valueOf()*** será este el que se usará en comparaciones:

```
class personaClass {
    ...
    valueOf() {
     return this.edad
     }
}

let cpo = new personaClass('Carlos', 19)
let aat = new personaClass('Ana', 23)
console.log(cpo < aat)          // imprime true ya que 19<23
```

Prototipos

Cada objeto tiene un prototipo del que hereda sus propiedades y métodos (es el equivalente a su clase, pero en realidad es un objeto que está instanciado). Si añadimos una propiedad o método al prototipo se añade a todos los objetos creados a partir de él lo que ahorra mucha memoria.

```
personaClass.prototype.getInfo2 = function() { return
        'Estoy añadiendo una nueva función'
    }
}
let cpo = new personaClass('Carlos', 19)
console.log(cpo.getInfo2())     // imprime: 'Estoy añadiendo una nueva función'
```