

Arrays

Tema 4.1

Arrays

- Un **array** o matriz es una estructura de datos que almacena una **serie de elementos** (que no tienen porque ser del mismo tipo,) a los que **se accede** por medio de un número que indica su **posición** en la matriz que se conoce con el nombre de **índice**.

Arrays

- **Declaración de arrays:**

Al igual que ocurre con las variables, es necesario declarar un array antes de poder usarlo.

Las matrices en Javascript son objetos de tipo Array, por lo tanto se crean usando el **constructor Array()**:

```
//creamos una matriz vacía
```

```
var a = new Array(); var fruta = [];
```

```
//también podemos crear una matriz vacía pero reservar espacio para n elementos
```

```
var b = new Array(10);
```

```
//o especificar sus elementos a la hora de crear el array
```

```
var personajes = new Array("Minnie", "Pluto", "Donald");
```

```
var pais = ['Mexico', 'España', 'Argentina', 'Chile',  
            'Colombia', 'Venezuela', 'Perú', 'Costa Rica'];
```

Arrays

- Las Arrays no son estáticas, una vez creadas podemos cambiar los elementos que contiene así como acceder a los elementos de cualquier posición, ambas cosas gracias al operador [].

Arrays

- **¿Como accedo a un elemento?**
- Si quedemos acceder al primer elemento del vector escribiremos el nombre del vector y entre parentesis cuadrados pondremos el indice del elemento.
- ```
<script>
dias = new Array ('lunes', 'martes', 'miercoles', 'jueves'
, 'viernes', 'sabado', 'domingo');
alert(dias[0]); _____comienzan desde
0.
</script>
```

# Recurrer arrays

- For

```
var f;
```

```
for(f=0;f<vec.length;f++) {
 document.write(vec[f]+'-');
}
```

# Recorrer arrays

- For in

```
for (nombreIndice in nombreObjeto) {
 ... ejecución de sentencias ...
}
```

```
for(var indice in vec) {
 document.write(indice);
 document.write(vec[indice]+'-');
}
```

# Recorrer Arrays

## Bucle for-of

**for (variable of objeto) sentencia** *Para cualquier tipo Iterable (Arrays)*

```
var miArray = [2, 4, 6, "hola", "mundo"];
for (var valor of miArray) {
 console.log("Valor: " + valor);
}
```



# Funciones callback

- las **funciones callback** no son más que un tipo de funciones que se pasan por parámetro a otras funciones.

Forma tradicional

```
const list = ["A", "B", "C"];

for (let i = 0; i < list.length; i++) {
 console.log("i=", i, " list=", list[i]);
}
```

# foreach

- `arr.forEach(function callback(currentValue, index, array) { // tu iterador }, thisArg);`

```
var a = ['a', 'b', 'c'];
```

```
a.forEach(function(element) {
 console.log(element);
});
```

```
// a
// b
// c
```

```
var miArray = [2, 4, 6, 8, 10];
miArray.forEach(function(valor, indice, array) {
 console.log("En el indice " + indice + " hay este valor: " + valor);
});
```

# foreach

- *arr.forEach(function callback(currentValue, index, array) { // tu iterador }, thisArg);*

```
var a = ['a', 'b', 'c'];
```

```
a.forEach(function(element) {
 console.log(element);
});
```

```
// a
// b
// c
```

```
var miArray = [2, 4, 6, 8, 10];
miArray.forEach(function(valor, indice, array) {
 console.log("En el índice " + indice + " hay este valor: " + valor);
});
```

## Bucle for-of

**for (variable of objeto) sentencia** Para cualquier tipo Iterable (Arrays)

```
var miArray = [2, 4, 6, "hola", "mundo"];
for (var valor of miArray) {
 console.log("Valor: " + valor);
}
```

# Arrays

- **Propiedades de los arrays:**

El objeto array tiene dos propiedades:

- **1.length:** Esta propiedad nos dice en cada momento la longitud del array, es decir, cuántos elementos tiene.

```
var lista = new Array(50);
elementos = lista.length;
```

- **2.prototype:** Nos permite asignar nuevas propiedades al objeto-`Array.prototype.nueva_propiedad= valor;`

```
Array.prototype.nuevo_metodo= nombre_de_la_funcion;
```

Ejemplo de nueva propiedad

```
Array.prototype.descriptor = null;
```

```
dias = new Array ('lunes', 'Martes', 'Miercoles', 'Jueves',
 'Viernes');
```

```
dias.descriptor = "Dias laborables de la semana";
```

# Arrays

## Métodos

| Métodos                |                       |
|------------------------|-----------------------|
| <code>push()</code>    | <code>shift()</code>  |
| <code>concat()</code>  | <code>pop()</code>    |
| <code>join()</code>    | <code>slice()</code>  |
| <code>reverse()</code> | <code>sort()</code>   |
| <code>unshift()</code> | <code>splice()</code> |

# Arrays

- Métodos de los arrays—push():
- Añade nuevos elementos al array y devuelve la nueva longitud del array.

- ejemplo



# Arrays

- Métodos de los arrays—concat():
- Selecciona un array y lo concatena con otros elementos en l

- Ejemplo



```
var equipos_copa_del_rey =
 equipos_a.concat(equipos_b);
```

# Arrays

- Métodos de los arrays—join():
- Concatena los elementos de un array en una sola cadena separada por un carácter opcional.
- ejemplo



`pizzas.join(" - ")`



# Arrays

- Métodos de los arrays—reverse():
- Invierte el orden de los elementos de un array.



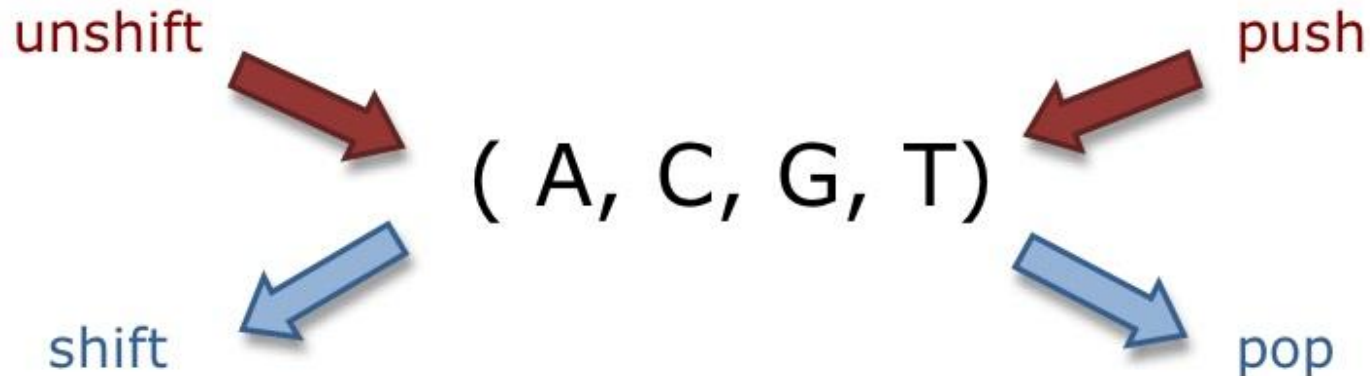
# Arrays

- Métodos de los arrays—unshift():
- Añade nuevos elementos al inicio de un array y devuelve el número de elementos del nuevo array modificado.



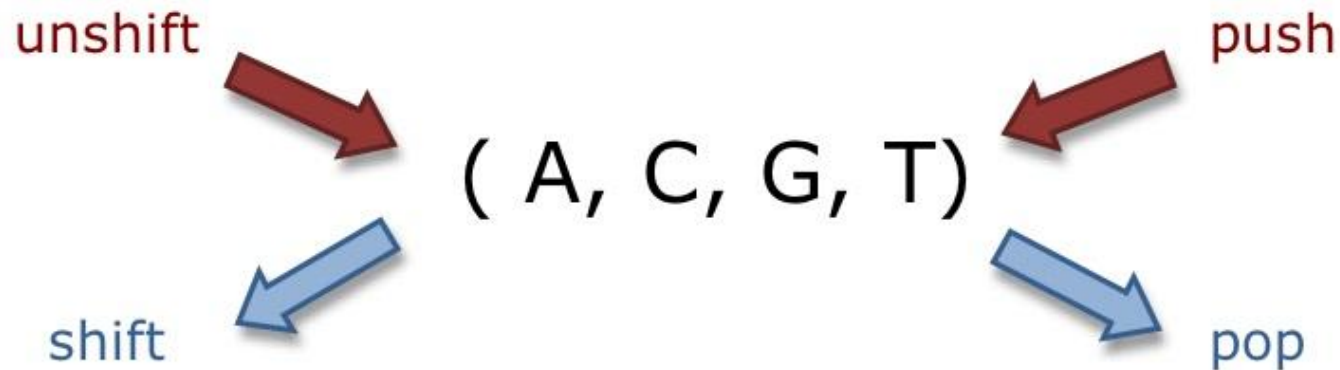
# Arrays

- Métodos de los arrays—shift():
- Elimina el primer elemento de un array.
- Devuelve el elemento eliminado



# Arrays

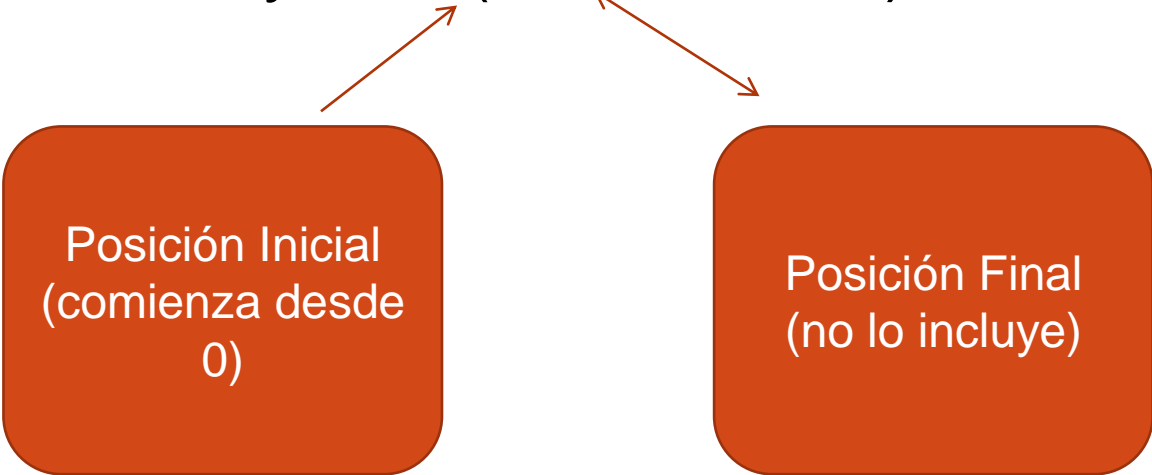
- Métodos de los arrays—pop():
- Elimina el último elemento de un array.
- Devuelve el elemento eliminado



# Arrays

- Métodos de los arrays—slice():
- Devuelve un nuevo array con un subconjunto de los elementos del array que ha usado el método.

`miArray.slice(num1,num2)`



Posición Inicial  
(comienza desde  
0)

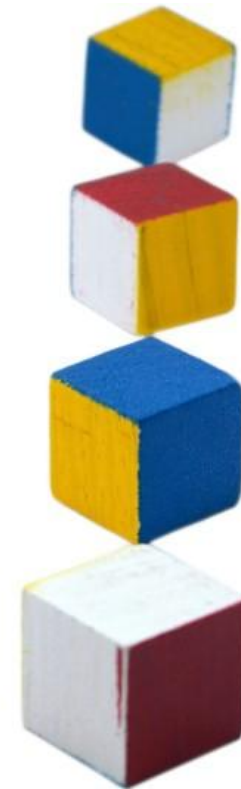
Posición Final  
(no lo incluye)

# Arrays

- Métodos de los arrays—sort():
- Ordena alfabéticamente los elementos de un array. Podemos definir una nueva función para ordenarlos con otro criterio.

```
var arr = ['a', 'b', 'Z', 'Aa', 'AA'];
arr.sort(); //['AA', 'Aa', 'Z', 'a', 'b']
```

```
var arr = [40, 1, 5, 200];
arr.sort(); //[1, 200, 40, 5]
```



# Parámetro del método sort()

- El método `.sort()` tiene **un único parámetro opcional que permite ayudar a este método para realizar la ordenación del contenido**. Esta es la clave para que este método se comporte como nos interesa en cada caso.

# Trabajo 1

Averiguar como utilizando el sort de Array puedo cambiar los criterios anteriores para hacer una ordenación más ajustada.

Hacer un powerpoint con ejemplos y explicaciones.



# Arrays



- Métodos de los arrays—splice():
- Elimina, sustituye o añade elementos del array dependiendo de los argumentos del método.
- **Borrar elementos.** Para borrar elementos debemos indicar dos parámetros al método splice, **el primero indica a partir de que posición** procedemos a **borrar** componentes y **el segundo** parámetro indica la **cantidad** de componentes a borrar.
- Crear un vector con un for que contenga los números del 1 al 10. Visualizarlo. Introducir por teclado la a partir de que posición elimino y cuantos. Visualizar una vez aplicado el método.

# Arrays

- Teniendo en cuenta que splice **genera un vector con los elementos extraídos**, manipular el ejercicio anterior para que nos visualice también los elementos extraídos.
- Podemos usar en el argumento elementos negativos en el primer parámetro, ej:

**Vector.splice(-2,2)**

**Borra los dos últimos**

# Arrays

- Insertar valores:
- Debemos pasar como mínimo 3 o más parámetros. Los dos primeros cumplen la misma función que cuando borramos (pero en este caso pasamos un cero en el segundo parámetro) y del tercer parámetro en adelante se indican los valores a insertar en el vector.
- Ejemplo: Insertar en el vector del ejercicio anterior :20,40.50 al principio.Visualizar.

# Arrays

- Splice permite borrar e insertar a la vez mezclando los conceptos anteriores.
- Probar en el vector a borrar 3 datos a partir de la posición 1 , y después insertar 5 nuevos.

# Arrays

```
var array = [2, 5, 9];
var index = array.indexOf(2);
// index es 0
index = array.indexOf(7);
// index es -1
```

Nos sirve para saber la posición de un elemento, si no está devuelve -1-

# Copias en profundidad

En los valores primitivos, a cada copia se le asigna su propio espacio en memoria; pero los que son del tipo objeto apuntan al mismo sitio. Por lo tanto, lo que hagamos en la copia afecta al original. En código:

```
var miArray = [1, 2, 3];
var otroArray = miArray;
otroArray[1] = "0";
console.log(miArray[1]); // 0
```

# Copias en profundidad

Por lo tanto, para poder sacar una copia de verdad de un array tenemos que hacer un apañó o, como dicen los expertos, una copia en profundidad. La forma más sencilla y rápida de hacerlo es utilizando el método `slice()`.

```
var miArray = [1, 2, 3];
var otroArray = miArray.slice();
otroArray[1] = "0"
console.log(miArray[1]); // 2
```

# isArray

- Si existe algún riesgo de que el array nos llegue null o undefined conviene mirar primero si es un array con el método **isArray()** antes de tratar de acceder a su propiedad length, pues de lo contrario nos daría un error. Dicho de otra forma, esto no casca:

```
miArray = undefined;
```

```
if (Array.isArray(miArray)) {
```

```
for (var i=0, len=miArray.length; i<len; i++) {
```

```
 console.log(miArray[i]);
```

```
}
```



# Filter , map y reduce

- Trabajo métodos de Array.
- Investigar los métodos arriba indicados.
- Crear un pp
- Por cada método, explicación de para que sirve, sintaxis general y ejemplos.
- Sería interesante la comparación entre el código tradicional sin el método y el código con el método.
- Explicación de los ejemplos.