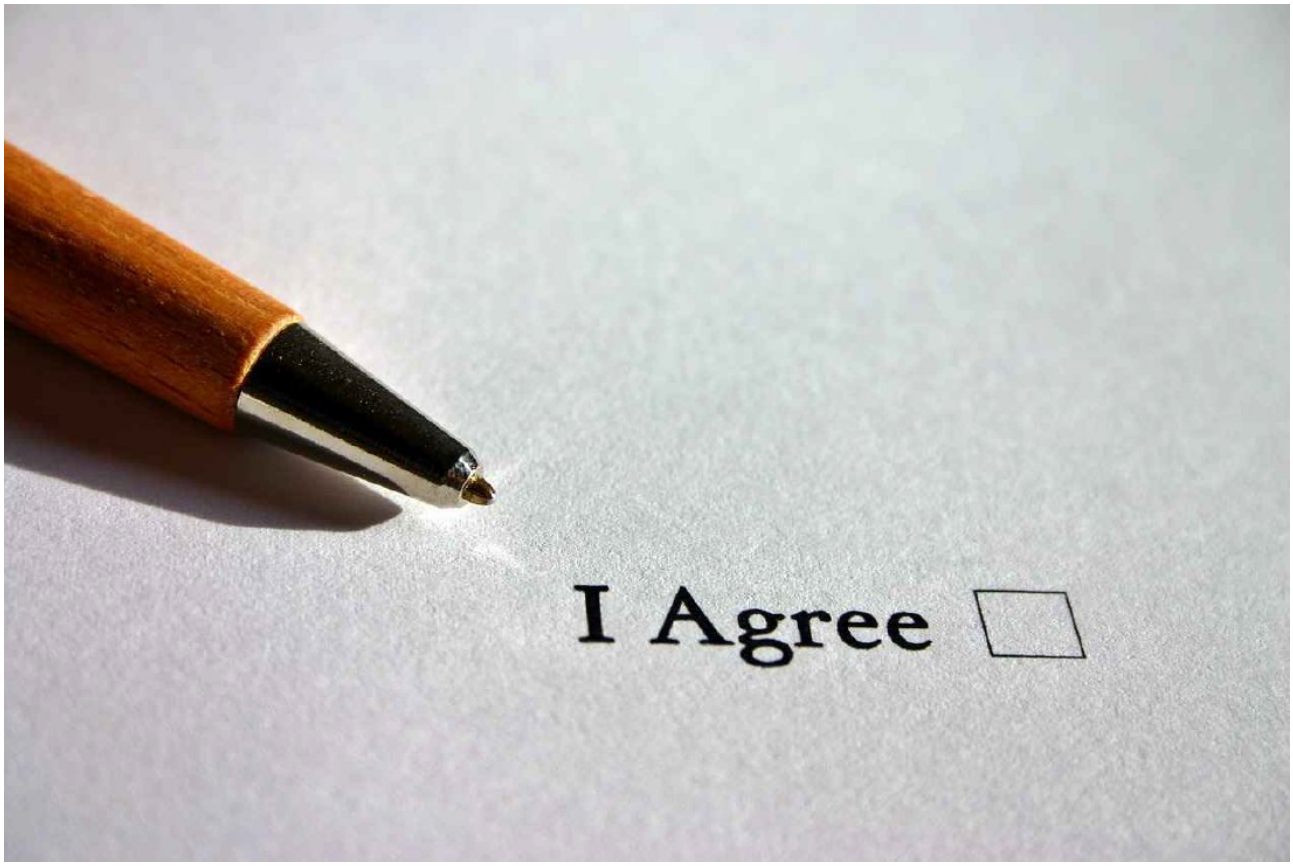
DESARROLLO EN ENTORNO CLIENTE/SERVIDOR

UD4: ESTRUCTURAS DE DATOS Y FORMULARIOS

Arrays en JavaScript



UD4 – Estructuras de datos y formularios – ARRAYS EN JAVASCRIPT

1. EVALUACIÓN

El presente documento, junto con sus actividades, cubren los siguientes criterios de evaluación:

RESULTADOS DE APRENDIZAJE	CRITERIOS DE EVALUACIÓN
RA4. Programa código para clientes Web analizando y utilizando estructuras definidas por el usuario.	a) Se han clasificado y utilizado las funciones predefinidas del lenguaje. b) Se han creado y utilizado funciones definidas por el usuario. c) Se han reconocido las características del lenguaje relativas a la creación y uso de arrays. d) Se han creado y utilizado arrays. e) Se han reconocido las características de orientación a objetos del lenguaje. f) Se ha creado código para definir la estructura de objetos. g) Se han creado métodos y propiedades. h) Se ha creado código que haga uso de objetos definidos por el usuario. i) Se ha depurado y documentado el código.

2. ARRAYS

En JavaScript, los arrays son un tipo de objetos. Podemos crear un array con la instancia de un objeto de clase Array. Estos no tienen un tamaño fijo, por tanto, podemos inicializarlo con un tamaño y luego añadirle más elementos.

El constructor puede recibir 0 parámetros (array vacío), 1 número (el tamaño del array), o en cualquier otro caso, se creará un array con los elementos recibidos. Debemos tener en cuenta que en JavaScript un array puede contener al mismo tiempo diferentes tipos de datos: number, string, boolean, object, etc.

```
let a = new Array(); // Crea un array vacío

a[0] = 13;

console.log(a.length); // Imprime 1

console.log(a[0]); // Imprime 13

console.log(a[1]); // Imprime undefined
```

Fíjate que cuando accedes a una posición del array que no ha sido definida, devuelve undefined. La longitud de un array depende de las posiciones que han sido asignadas. Vamos a ver un ejemplo de lo que ocurre cuando asignas una posición mayor que la longitud y que no es consecutiva al último valor asignado.

```
let a = new Array(12); // Crea un array de tamaño 12
console.log(a.length); // Imprime 12
a[20] = "Hello";
```

console.log(a.length); // Ahora imprime 21 (0-20). Las posiciones 0-19 tendrán el valor undefined

Podemos reducir la longitud del array modifcando directamente la propiedad de la longitud del array (length). Si reducimos la longitud de un array, las posiciones mayores a la nueva longitud serán consideradas como undefined (borradas).

```
let a = new Array("a", "b", "c", "d", "e"); // Array con 5 valores
console.log(a[3]); // Imprime "d"
a.length = 2; // Posiciones 2-4 serán destruidas
console.log(a[3]); // Imprime undefined
```

Puedes crear un array usando corchetes en lugar de usar new Array(). Los elementos que pongamos dentro, separados por coma serán los elementos que inicialmente tendrá el array.

```
let a = ["a", "b", "c", "d", "e"]; // Array con 5 valores inicialmente
console.log(typeof a); // Imprime object
console.log(a instanceof Array); // Imprime true, a es instancia de array
a[a.length] = "f"; // Insertamos in nuevo elemento al final
console.log(a); // Imprime ["a", "b", "c", "d", "e", "f"]
```

2.1. Recorriendo arrays

Podemos recorrer un array con los clásicos bucles while y for, creando un contador para el índice que iremos incrementando en cada iteración. Pero tenemos dos versiones alternativas del bucle for, el bucle **for..in** y el bucle **for..of**.

2.1.1. Bucle for..in

Con este bucle podemos iterar los índices de un array o las propiedades de un objeto (similar al bucle foreach de otros lenguajes, pero recorriendo los índices en lugar de los valores).

```
let ar = [4, 21, 33, 24, 8]; //Este es el Array que vamos a recorrer
```

```
let i = 0;
// Imprime los elementos del array con un while
while(i < ar.length) {
      console.log(ar[i]);
      i++;
}
// Imprime los elementos del array con un for
for(let i = 0; i < ar.length; i++) {
      console.log(ar[i]);
}
// Imprime los elementos del array con un for..in
for (let i in ar) {
      console.log(ar[i]);
}
```

2.1.2. Bucle for..of

También podemos iterar por los elementos de un array sin utilizar el índice. Para ello se utiliza el bucle **for..of** (que se comporta como un bucle foreach de otros lenguajes):

```
let ar = [4, 21, 33, 24, 8];
// Imprime los elementos del array, pero sin necesidad del índice
for(let item of a) {
      console.log(item);
}
```

Iterando otros objetos

En realidad, con **for..in** y **for..of** podemos iterar otros tipos de objetos

```
let persona = {
      nombre : "Juan",
      edad : 45,
```

```
      telefono: "987654321"
};
//Ejemplo de bucle for..in para recorrer un objeto
for (let campo in persona) {
      console.log(campo + ": " + persona[campo]);
}
// Imprimirá:
// nombre: Juan
// edad: 45
// telefono: 987654321
//Ejemplo de bucle for..of para recorrer una cadena de caracteres
let cadena = "abcdefg";
for(let letra of cadena) {
      if(letra.match(/^[aeiou]$/)) { //Usamos una expresión regular
            console.log(letra+ " es una vocal");
      } else {
            console.log(letra + " es una consonante");
      }
}
```

2.2. Métodos de arrays

2.2.1. Insertar valores

Insertar valores al principio de un array **unshift**() y al fnal **push**().

```
let a = []; //Creamos un array vacío
// Inserta el valor al final del array
a.push("a");
// Inserta estos nuevos valores al final
a.push("b", "c", "d");
```

```
console.log(a); // Imprime ["a", "b", "c", "d"]

// Inserta nuevos valores al principio del array

a.unshift("A", "B", "C");

console.log(a); // Imprime ["A", "B", "C", "a", "b", "c", "d"]
```

2.2.2. Eliminar valores

Ahora, vamos a ver la operación opuesta. Eliminar del principio **shift()** y del final **pop()** del array. Estas funciones, además de hacer la operación, devolverán el valor que ha sido eliminado:

```
//Seguimos con el array anterior

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"]
```

2.2.3. Convertir a string

Podemos convertir un array a string usando **join**() en lugar de **toString**(). Por defecto, devuelve un string con todos los elementos separados por coma. Sin embargo, podemos especificar el separador a imprimir:

```
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"
```

2.2.4. Concatenar

Podemos concatenar dos arrays usando **concat**(), es importante reseñar que los arrays originales no se modifican:

```
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"], no ha sido modificado

console.log(b);//tampoco se ha modificado b
```

2.2.5. Dividir o trocear arrays

El método **slice**() nos devuelve un nuevo sub-array.

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

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

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

2.2.6. Invertir elementos de un array

Podemos invertir el orden del array usando el método **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"]
```

2.2.7. Ordenar arrays

Podemos ordenar los elementos de un array usando el método **sort**().

```
let a = ["Pedro", "Ana", "Tomás", "Jenaro", "Roberto", "Alicia"];
// Ordena el array original
a.sort();
// Imprime ["Alicia", "Ana", "Jenaro", "Pedro", "Roberto", "Tomás"]
console.log(a);
```

El orden por defecto es alfabético, pero ¿qué ocurre si intentamos ordenar elementos que no son string?. Por defecto, los ordenará como si su valor fuese un string, teniendo en cuenta que si son objetos, se intentará llamar al método toString() para ordenarlo.

Para evitar esto, tendremos que pasar una función de ordenación, que comparará 2 valores del array y devolverá un valor numérico indicando cual es menor de la siguiente forma:

- Un valor positivo si el primero elemento es mayor
- 0 si son iguales
- Un valor negativo si el primero es menor

```
let a = [20, 6, 100, 51, 28, 9];
a.sort(); // Ordena el array original sin pasarle función
console.log(a); // Imprime [100, 20, 28, 51, 6, 9]
a.sort((n1, n2) => n1 - n2); //se pasa una función flecha de ordenación
console.log(a); // Imprime [6, 9, 20, 28, 51, 100]
```

Veamos un ejemplo con objetos (se verán más adelante), en este caso son personas y las ordenaremos según la edad:

```
// Constructor de la clase persona
function Persona(nombre, edad) {
    this.nombre = nombre;
```

```
        this.edad = edad;
        this.toString = function() { // Método toString()
            return this.nombre + " (" + this.edad + ")";
        }
}
let personas = [];
personas[0] = new Persona("Tomás", 24);
personas[1] = new Persona("María", 15);
personas[2] = new Persona("Juan", 51);
personas[3] = new Persona("Ana", 9);
// Ordenamos mediante una función de comparación específica
personas.sort((p1, p2) => p1.edad - p2.edad));
// Imprime: "Ana (9),María (15),Tomás (24),Juan (51)"
console.log(personas.toString());
```

2.2.8. Encontrar elementos en un 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
```

2.2.9. Condiciones que cumplen los elementos del array

El método **every**() devolverá un boolean indicando si todos los elementos del array cumplen cierta condición. A **every**() se le pasará una función, que a su vez recibirá cada elemento, lo testeará, y devolverá cierto o falso dependiendo de si cumple la condición o no. Si todos los cumplen, **every**() devolverá **true**, si no, devolverá **false**:

```
let a = [3, 21, 15, 61, 9, 54];
// Comprueba si cada número es menor a 100. En este caso, imprime true
console.log(a.every(num => num < 100));
// Comprueba si cada número es par. En este caso, imprime false
console.log(a.every(num => num % 2 == 0));
```

Por otro lado, 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];
// Comprueba si algún elemento es par. En este caso imprime true
console.log(a.some(num => num % 2 == 0));
```

2.2.10. Iterar por el array

Podemos iterar por los elementos de un array usando el método **forEach**(). A este método se le pasa una función que recibirá como primer parámetro el elemento del array que esté interando. De forma opcional, podemos llevar un seguimiento del índice al que está accediendo en cada momento, como segundo parámetro de la función, e incluso recibir el array como tercer parámetro:

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

2.2.11. Funciones map(), filter() y reduce()

Estas tres funciones las vamos a utilizar mucho en la programación en JavaScript. Nos van a permitir transformar los arrays a partir de aplicar una serie de transformaciones sobre los arrays. En ningún caso estas funciones modifican el array original, si no que nos devuelven otro array (o valor) a partir de la transformación realizada.

map()

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 transformados:

```
let a = [4, 21, 33, 12, 9, 54];
// En este caso, vamos a duplicar cada elemento del array
console.log(a.map(num => num*2)); // Imprime [8, 42, 66, 24, 18, 108]
```

filter()

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**(). A filter le vamos a pasar una función, que deberá devolver **true** o false**.** Los elementos del array que hagan que la función devuelva **true**, estarán en el nuevo array, los que den **false**, no estarán:

```
let a = [4, 21, 33, 12, 9, 54];
//Nos vamos a quedar sólo con los elementos pares
console.log(a.flter(num => num % 2 == 0)); // Imprime [4, 12, 54]
```

reduce()

El método **reduce**() se le debe pasar una función que acumula un valor, procesando cada elemento (segundo parámetro) con el valor acumulado (primer parámetro). Como segundo parámetro de **reduce**(), se debe pasar un valor inicial. Si no se pasa 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];
```

```
// Suma todos los elementos del array.
console.log(a.reduce(
                        (total, num) => total + num
                        , 0
                        )
); // Imprime 133

// Número máximo del array.
console.log(a.reduce(
                        (max, num) => num > max? num : max
                        ,0
                        )
); //Imprime 54
```

Para hacer lo mismo que reduce hace pero al revés, usaremos **reduceRight**().

```
let a = [4, 21, 33, 12, 9, 154];
// Comienza con el último número y resta todos los otros números
console.log(a.reduceRight((total, num) => total - num)); // Imprime 75
```

Si no le damos un valor inicial, empezará con el valor de la última posición del array

2.3. Otras funciones de arrays

2.4. Array.of(value)

Si queremos instanciar un array con un sólo valor, y éste es un número, con new Array() no podemos hacerlo, ya que crea vacío con ese número de posiciones.

```
let array = new Array(10); // Array vacío (longitud 10)
let array = Array(10); // Mismo que arriba: array vacío ( longitud 10)
let array = Array.of(10); // Array con longitud 1 -> [10]
let array = [10]; // // Array con longitud 1 -> [10]
```

2.5. Array.from(array, func)

Funciona de forma similar al método map, crea un array desde otro array. Se aplica una operación de transformación (función lambda o anónima) para cada ítem.

```
let array = [4, 5, 12, 21, 33];
let array2 = Array.from(array, n => n * 2);
console.log(array2); // [8, 10, 24, 42, 66]
let array3 = array.map(n => n * 2); // Igual que Array.from
console.log(array3); // [8, 10, 24, 42, 66]
```

2.6. Array.fill(value)

Este método sobrescribe todas las posiciones de un array con un nuevo valor. Es una buena opción para inicializar un array que se ha creado con N posiciones.

```
let sums = new Array(6); // Array con 6 posiciones
sums.fill(0); // Todas las posiciones se inicializan a 0
console.log(sums); // [0, 0, 0, 0, 0, 0]
let numbers = [2, 4, 6, 9];
numbers.fill(10); // Inicializamos las posiciones al valor 10
console.log(numbers); // [10, 10, 10, 10]
```

2.7. Array.fill(value, start, end)

Este método hace lo mismo que antes pero rellenando el array desde una posición inicial (incluida) hasta una final (excluida). Si no se especifica la última posición, se rellenará hasta el final

```
let numbers = [2, 4, 6, 9, 14, 16];
numbers.fill(10, 2, 5); // Las posiciones 2,3,4 se ponen a 10
console.log(numbers); // [2, 4, 10, 10, 10, 16]
let numbers2 = [2, 4, 6, 9, 14, 16];
numbers2.fill(10, -2); // Las dos últimas posiciones se ponen a 10
console.log(numbers2); // [2, 4, 6, 9, 10, 10]
```

2.8. Array.find(condition)

Encuentra y devuelve el primer valor que encuentre que cumple la condición que se establece. Con findIndex, devolvemos la posición que ocupa ese valor en el array.

```
let numbers = [2, 4, 6, 9, 14, 16];
// Imprime 14 (primer valor encontrado >= 10)
console.log(numbers.find(num => num >= 10));
// Imprime 4 (numbers[4] -> 14)
console.log(numbers.findIndex(num => num >= 10));
```

2.9. Array.copyWithin(target, startwith)

Copia los valores del array empezando desde la posición startWith, hasta la posición target en el resto de posiciones del array (en orden). Por ejemplo:

```
let numbers = [2, 4, 6, 9, 14, 16];
numbers.copyWithin(3, 0); // [0] -> [3], [1] -> [4], [2] -> [5]
console.log(numbers); // [2, 4, 6, 2, 4, 6]
```

3. Rest y spread

Rest es la acción de transformar un grupo de parámetros en un array, y spread es justo lo opuesto, extraer los elementos de un array (o de un string) a variables. Para usar rest en los parámetros de una función, se declara siempre como último parámetro (no puede haber parámetros después del rest) y se le ponen tres puntos'...' delante del mismo. Este parámetro se transformará automáticamente en un array conteniendo todos los parámetros restantes que se le pasan a la función. Si por ejemplo, el parámetro rest está en la tercera posición, contendrá todos los parámetros que se le pasen a excepción del primero y del segundo (a partir del tercero).

```
function getMedia(...notas) {
      console.log(notas); // Imprime el array
      let total = notas.reduce((total,notas) => total + notas, 0);
      return total/notas.length; //Devuelve la media
```

```
}
console.log(getMedia(5, 7, 8.5, 6.75, 9)); // Imprime 7.25
function imprimirUsuario(nombre, ...lenguajes) {
      console.log(nombre + " sabe " + lenguajes.length + " lenguajes: " +
      lenguajes.join(" - "));
}
// Imprime "Pedro sabe 3 lenguajes: Java - C# - Python"
imprimirUsuario("Pedro", "Java", "C#", "Python");
// Imprime "María sabe 5 lenguajes: JavaScript - VUE - PHP - HTML - CSS"
imprimirusuario("María", "JavaScript", "VUE", "PHP", "HTML", "CSS");
```

Spread es lo “opuesto” de rest. Si tenemos una variable que contiene un array, y ponemos los tres puntos ‘...’ delante de este, extraerá todos sus valores. Podemos usar la propiedad por ejemplo con el método **Math.max**(), el cual recibe un número indeterminado de parámetros y devuelve el mayor de todos.

```
let nums = [12, 32, 6, 8, 23];
// Imprime NaN (array no es válido)
console.log(Math.max(nums));
// Imprime 32 -> equivalente a Math.max(12, 32, 6 ,8 ,23)
console.log(Math.max(...nums));
```

Podemos usar también esta propiedad si necesitamos clonar un array:

```
let a = [1, 2, 3, 4];
let b = a; // Referencia el mismo array que 'a' (las modifcaciones
afectan a ambos).
let c = [...a]; // Nuevo array -> contiene [1, 2, 3, 4]
```

4. Desestructuración de arrays

Desestructurar es la acción de extraer elementos individuales de un array (o propiedades de un objeto) directamente en variables individuales. Podemos también desestructurar un string en

caracteres.

Vamos a ver un ejemplo donde asignamos los tres primeros elementos de un array, en tres variables diferentes, usando una única asignació:

```
let array = [150, 400, 780, 1500, 200];
// Asigna los tres primeros elementos del array
let [frst, second, third] = array;
console.log(third); // Imprime 780
```

¿Qué pasa si queremos saltarnos algún valor? Se deja vacío (sin nombre) dentro de los corchetes y no será asignado:

```
let array = [150, 400, 780, 1500, 200];
let [frst, , third] = array; // Asigna el primer y tercer elemento
console.log(third); // Imprime 780
```

Podemos asignar el resto del array a la última variable que pongamos entre corchetes usando rest (como en el punto anterior del tema):

```
let array = [150, 400, 780, 1500, 200];
let [frst, second, ...rest] = array; // rest -> array
console.log(rest); // Imprime [780, 1500 ,200]
```

Si queremos asignar más valores de los que contiene el array y no queremos obtener undefined, podemos usar valores por defecto:

```
let array = ["Peter", "John"];
let [first, second = "Mary", third = "Ann"] = array; // rest -> array
console.log(second); // Imprime "John"
console.log(third); // Imprime "Ann" -> valor por defecto
```

También podemos desestructurar arrays anidados:

```
let sueldos = [["Pedro", "Maria"], [24000, 35400]];

let [[nombre1, nombre2], [sueldo1, sueldo2]] = sueldos;

// Imprime "Pedro gana 24000€"

console.log(nombre1 + " gana " + sueldo1 + "€");
```

También se puede desestructurar un array enviado como parámetro a una función en valores individuales:

```
function imprimirUsuario([id, nombre, email], otraInfo = "Nada") {

      console.log("ID: " + id);

      console.log("Nombre: " + nombre);

      console.log("Email: " + email );

      console.log("Otra info: " + otraInfo );

}

let infoUsu = [3, "Pedro", "peter@gmail.com"];

imprimirUsuario(infoUsu, "No es muy listo");
```