CLASS(E)

Módulo 2: Estructuras de datos I

Listas

Listas

```
const array = [1, "dos", true] // pueden incluir cualquier valor
const array = [[1, 2], [3, 4]] // también otras listas
```

Las listas nos permiten compilar diferentes datos en una única variable.

Listas

Podemos utilizar listas de listas para representar matrices.

```
const butacasCine = [
    ["A1", "A2", "A3"],
    ["B1", "B2", "B3"],
    ["C1", "C2", "C3"],
]
```

Listas

Matrices de cualquier dimensión.

```
const cuboRubik = [
    [
        ["blue", "blue", "blue"],
        ["blue", "blue", "blue"],
        ["blue", "blue", "blue"],
    ],
    [
        ["red", "red", "red"],
        // ...
    [
]
```

Recorrer listas

Disponemos de varios métodos para recorrer arrays.

Si queremos tener acceso al valor en cada iteración:

```
const array = ["a", "b", "c"]

for(let item of array) {
    console.log(item)
} // a, b, c

array.forEach(item => console.log(item)) // a, b, c
```

Recorrer listas

Si queremos tener acceso al índice:

```
const array = ["a", "b", "c"]

array.forEach((item, index) => console.log(index, item)) // 1 a, 2 b, 3 c

for(let i = 0; i < array.length; i++){
   console.log(i, array[i])
} // 1 a, 2 b, 3 c
```

Añadiendo elementos

Si tenemos una lista y queremos añadir elementos, podemos usar el metodo .push

```
const array = ["a", "b", "c"]

array.push("d")

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

Ejercicio listas I

Implementa una función que reciba un array e imprima el primer y último elemento.

- En caso de tener un solo elemento, imprime este una vez.
- En caso de no tener elementos, no imprimas nada.

Ejercicio listas II

```
cut(["a", "b", "c", "d"], 1, 2) // ["b", "c"]
cut(["a", "b", "c", "d"], 1) // ["b", "c", "d"]
```

Implementa una función cut que reciba una lista, y dos índices. La función debe devolver una nueva lista con los valores comprendidos entre los dos índices.

- Incluye los extremos.
- Si la función no recibe un segundo índice, incluye el resto de valores.
- No utilices la función nativa slice.

Ejercicio listas III

Implementa una función `flatten` que reciba una lista de listas de strings y devuelva una única lista con todos los valores.

```
flatten([
    ["A1", "A2", "A3"],
    ["B1", "B2", "B3"],
    ["C1", "C2", "C3"],
]) // => ["A1", "A2", "A3", "B1", "B2", ...]
```

Ejercicio listas IV 🔥

```
// [1, 2, 3, 4, 5...]
flattenDeep([1, [2, 3], [[4, 5], [6]], [[[8]]]])
```

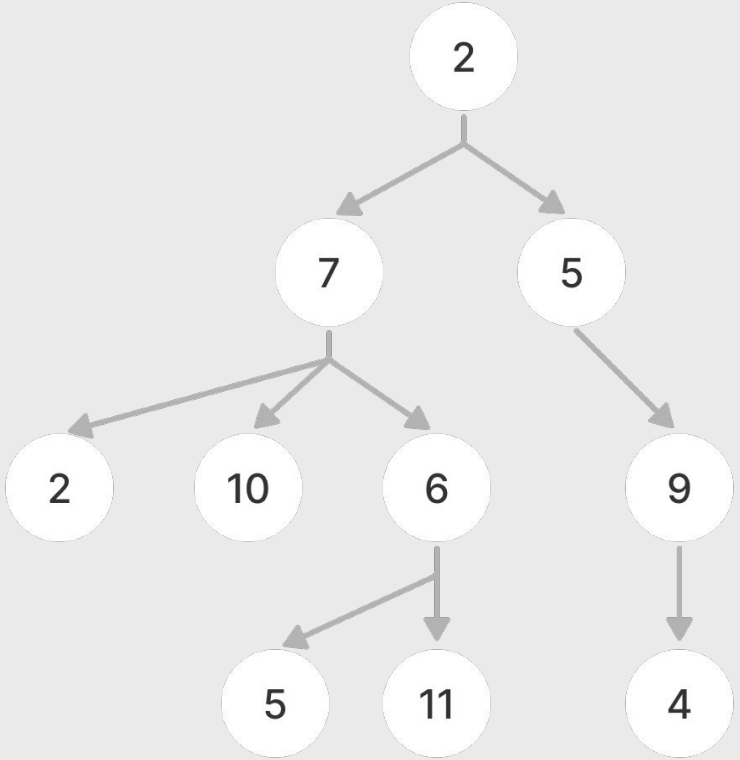
Implementa una función `flattenDeep` que reciba una lista de listas y devuelva una única lista con todos los valores.

Esta vez, cualquiera de las listas puede contener dentro `cualquier número de listas`.

¿Por qué el ejercicio anterior requiere el uso de recursividad?

Recursividad

La lista presentada tiene una estructura de árbol.



Ejercicio recursividad 🔥

```
sumDeep([1, [2, 3], [[4, 5]]]) // 15
```

Implementa una función `sumDeep` que devuelva la suma total de todos los números de un árbol cualquiera.

Destructuring de listas

Destructuring

```
const [a, b] = [10, 20]
console.log(a) // 10
console.log(b) // 20
```

El destructuring nos permite descomponer un array en varias variables.

Destructuring

El operador rest nos permite capturar el resto de variables en una nueva lista.

```
const [a, b, ...c] = [10, 20, 30, 40, 50]
console.log(a) // 10
console.log(b) // 20
console.log(c) // [30, 40, 50]
```

Destructuring

Podemos descomponer cualquier número de variables…

```
const [a] = [10, 20, 30, 40, 50]     // a = 10
const [a, b] = [10, 20, 30, 40, 50]  // a = 10, b = 20
```

en cualquier posición.

```
const [, , a] = [10, 20, 30, 40, 50]     // a = 30
const [, , a, b] = [10, 20, 30, 40, 50]  // a = 30, b = 40
```

Ejercicio destructuring

¿Cuánto vale la variable tail en cada caso?

```
const [head, tail] = [1, 2, 3]
const [head, ...tail] = [1, 2]
const [head, ...tail] = [1]
const [head, , ...tail] = [1, 2, 3]
```

Destructuring

El destructuring se puede utilizar en los parámetros de una función.

```
function printCoords([lat, lon]){
    console.log(lat, lon)
}
printCoords([45.0, 90.0])
```

Destructuring

Podemos utilizar el operador rest para capturar todos los argumentos de una función.

```
function printArgs(...args){
    args.forEach(arg => {
        console.log(arg)
    })
}
printArgs("lorem", "ipsum", "dolor")
```

Operador spread

```
const nums = [1, 2, 3]
Math.max(...nums) // equivale a Math.max(1, 2, 3)
```

El operador contrario a rest es spread y utiliza la misma sintaxis.

Permite aplicar los elementos de una lista como parámetros posicionales.

Funciones sobre listas

Funciones sobre listas

Hay tres operaciones fundamentales relacionadas con listas:

- Mapeo
- Filtrado
- Acumulación

Mapeo de listas

```
const array = [1, 2, 3]
const result = array.map(x => x + 1) // [2, 3, 4]
```

La función map nos permite transformar todos los valores de una lista aplicándoles una función.

Ejercicio map I

```
// [8, 12, 20] -> [4, 6, 10]
// [1, 7, 50] -> ["1", "7", "50"]
// [-2, 5, 15, -7, -8] -> ["-", "+", "+", "-", "-"]
```

Realiza las siguientes transformaciones:

Ejercicio map II

```
// ["lorem ipsum dolor", "hello world"] -> ["lid", "hw"]
// [{name: “Alberto”}, {name: “Fran”}] -> [“Alberto”, ”Fran”]
```

Realiza las siguientes transformaciones:

Ejercicio map III

```
// [[1, 2], [34, 20, 5], [11], [2, 4]] -> [3, 59, 11, 6]
```

Realiza la siguiente transformación:

Filtrado de listas

```
const array = [1, 2, 3, 4, 5]
const result = array.filter(x => x > 3 ) // [4, 5]
```

La función filter nos permite filtrar los valores de una lista aplicando un predicado.

Ejercicio filter

Realiza los siguientes ejercicios utilizando `filter`:

- Dada una lista de números, conserva los números impares.
- Dada una lista de objetos, conserva los objetos que tengan una propiedad important: true.

```
[{name: 'lorem', important: false}, {name: 'ipsum', important: true}]
```

Acumulación de listas

La función reduce nos permite realizar acumulaciones.

```
const array = [1, 2, 3, 4, 5]
const result = array.reduce((acc, x) => acc + x, 0) // 15
```

Acumulación de listas

Reduce recibe dos parámetros:

- Un callback que se ejecuta por cada valor de la lista y devuelve el nuevo valor acumulado.
- Un valor inicial (en este caso 0).

```
const array = [1, 2, 3, 4, 5]
const result = array.reduce((acc, x) => acc + x, 0) // 15
```

Acumulación de listas

A su vez, el callback recibe dos parámetros:

- El valor acumulado hasta el momento (acc).
- El valor actual del array en esta iteración (x).

```
const array = [1, 2, 3, 4, 5]
const result = array.reduce((acc, x) => acc + x, 0) // 15
```

Ejercicio reduce I

Realiza las siguientes acumulaciones utilizando reduce:

- Calcular la suma de todos los números de un array.
- Concatenar todas las strings de un array.

Ejercicio reduce II

Realiza las siguientes acumulaciones utilizando reduce:

- Calcular la suma de todos los números de un array (excepto los negativos).
- Encontrar el máximo de un array de números.

Ejercicio reduce III 🔥

Realiza la siguiente transformación utilizando `reduce`:

```
// [[1, 2], [34, 20, 5], [11], [2, 4]] -> [3, 59, 11, 6]
```

Funciones sobre listas

`includes` nos permite saber si un elemento está presente en una lista.

```
const lista = [1, 2, 3]
lista.includes(2) // true
lista.includes(4) // false
```

Funciones sobre listas

find nos permite buscar el primer valor en una lista que cumpla un predicado.

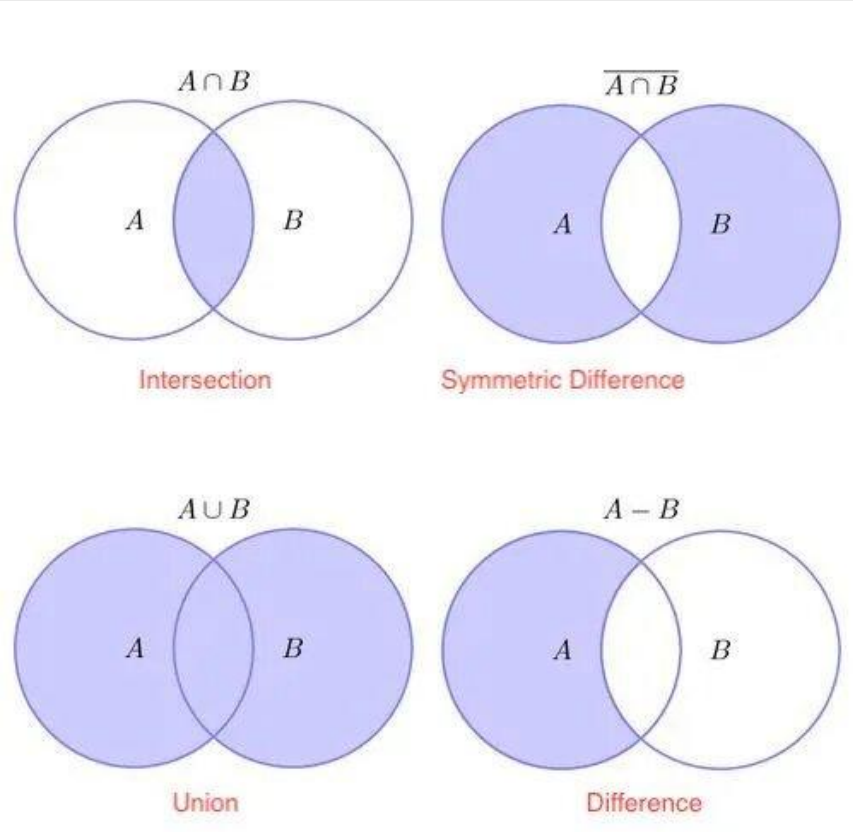
```
const lista = [1, 2, 3]
const result = lista.find(num => num > 1)   // 2
const result = lista.find(num => num === 0) // undefined
```

Operaciones de set

Operaciones de set o conjuntos

Las listas se pueden interpretar como sets o conjuntos matemáticos.

Algunas operaciones sobre sets son muy útiles a la hora de tratar datos.



Ejercicio sets I

Implementa una función `difference` que reciba dos listas y devuelva una nueva lista con los elementos de la primera que no tienen en común.

Ejercicio sets II

Implementa una función `intersection` que reciba dos listas y devuelva una nueva lista con elementos que tienen en común.

Bonus: Lodash

Lodash

Lodash es una de las librerías de utilidades más usadas del ecosistema Javascript.

Es extremadamente útil para acelerar el desarrollo.

Acceder a la documentación de Lodash
(https://lodash.com/docs/4.17.15)

Lodash y listas

Lodash incluye todas las funciones que hemos implementado en este módulo.

- flatten, flattenDeep, slice, union, difference.

Otras funciones útiles relacionadas con listas:

- chunk, last, dropwhile, takeWhile, isEmpty…