Immutable.js es una librería de JavaScript que cada día este cogiendo más empuje. La librería aporta un conjunto de tipos de abstractos de datos que son Immutables. ¿Qué quiere decir esto exactamente?. Quiere decir que una vez creado cada uno de los objetos , estos jamás son modificados. En un principio esto parece algo poco útil, sin embargo nada más lejos de la realidad. Vamos a explicarlo poco a poco. Cuando un objeto es immutable funciona de una forma diferente. Imagenemonos una lista de elementos

Si la lista es inmutable quiere decir que no la podemos modificar.

Es imposible añadir un nuevo elemento a la lista , ¿Entonces cómo funcionan? . Bueno la realidad es que cada vez que intentemos añadir un elemento a la lista lo que ocurrira es que se creará una nueva lista con los elementos anteriores y el nuevo.

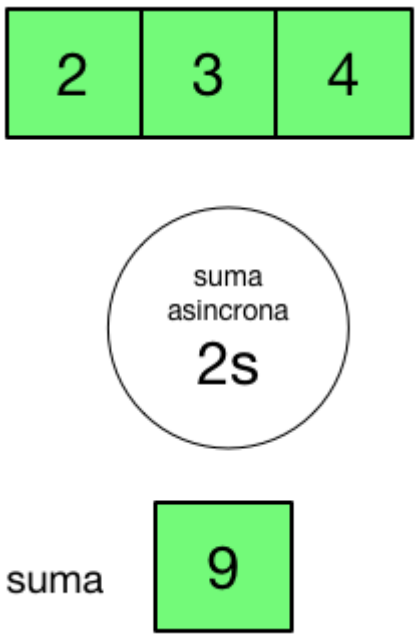
En un primer momento nos puede parecer que esto carece de sentido , pero tiene una gran

ventaja. Todos los objetos inmutables son Thread Safe. Eso quiere decir que no les afectan los cambios que se realicen por varios hilos o por programación asíncrona, algo que en JavaScript es muy habitual .Vamos a abordar un ejemplo que ayuda a clarificar.

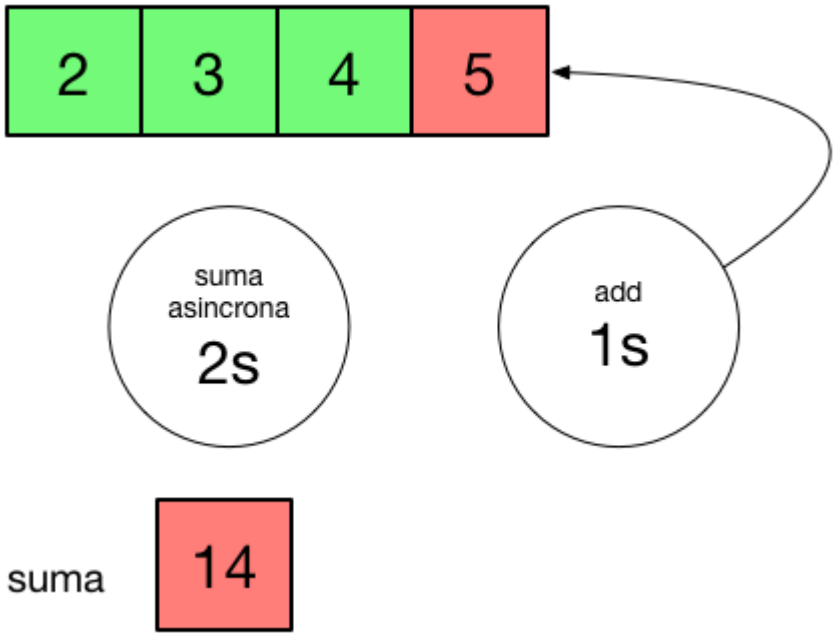
```
<!DOCTYPE html>
<html>

<head>
<title></title>

<script type="text/javascript">
var lista = [2, 3, 4];
function sumar(lista) {

return lista.reduce(function(resultado, numero) {
return resultado + numero;
});

}

setTimeout(function sumaAsincrona() {
console.log(sumar(lista));
}, 2000);

setTimeout(function add() {

lista.push(5);
console.log("hemos añadido uno")
```

```
}, 1000);
</script>
</head>

<body>
</body>

</html>
```

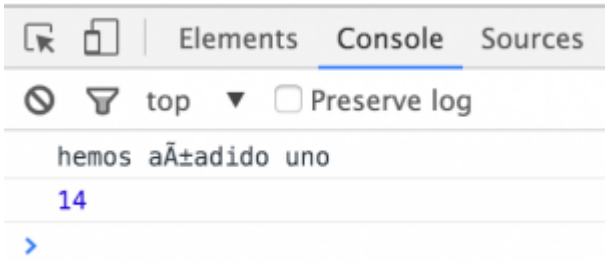
Disponemos de una lista de elementos que vamos a sumar. La peculiaridad es que lo vamos a sumar de forma asíncrona. Es decir vamos a esperar 2 segundos antes de realizar la suma.



El resultado debería ser 9 , pero lo vamos a complicar ya que vamos a añadir otro proceso asíncrono que se encargue de añadir un elemento a la lista al cabo de 1 segundo.



El resultado que aparecerá por consola será 14 y no 9 que era la suma de los elementos originales de la lista. Sin embargo cuando lanzamos el proceso asíncrono solo había 3 elementos en el array y no 4.

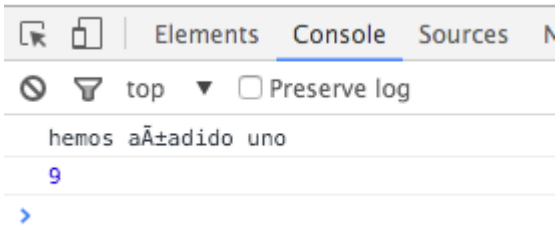


JavaScript es eminentemente asíncrono y este tipo de situaciones se puede producir de forma habitual. Si queremos evitarlas podemos usar Immutable.js . La librería se encargará de asegurarnos que ninguna operación asíncrona nos de problemas .El nuevo código a construir es :

```
<!DOCTYPE html>
<html>

<head>
<title></title>
<script type="text/javascript" src="immutable.min.js">
</script>
<script type="text/javascript">
var lista = Immutable.List([2,3,4]);

function sumar(lista) {

return lista.reduce(function(resultado, numero) {
return resultado+numero;
});

}

setTimeout (function sumaAsincrona() {
console.log(sumar(lista));
},2000);

setTimeout(function add() {

lista.push(5);
console.log("hemos añadido uno")

},1000);
</script>
</head>
```

```
<body>
</body>

</html>
```

Volvemos a ejecutar el código;



La suma se imprime de forma correcta y no le afecta para nada que hayamos añadido un nuevo término. Ya que este termino se añade a una nueva lista no a la lista que ya existía, eso sí la nueva lista no la hemos usado para nada.

Otros artículos relacionados: JavaScript Reduce , JavaScript Pure Functions