JavaScript pure functions es uno de los conceptos que más cuesta manejar cuando trabajamos con JavaScript. ¿Qué es y para que sirve una función pura? . Son funciones que no dependen ni modifican variables que estén fuera de su propio scope (ámbito), en una invocación con unos valores x siempre devuelve un resultado y . Vamos a construir un ejemplo con jQuery para entender mejor esto.

```
<!DOCTYPE html>
<html>

<head>
 <title></title>
 <script type="text/javascript" src="jquery-1.11.3.min.js"></script>
 <script type="text/javascript">


 $(document).ready(function() {


 $("#boton").click(function() {


 sumarParrafos();

 });

 });

 function sumarParrafos() {
 var total = 0;
```

```
 $("#parrafos p").each(function(item) {


 total += parseInt($(this).text());
 });

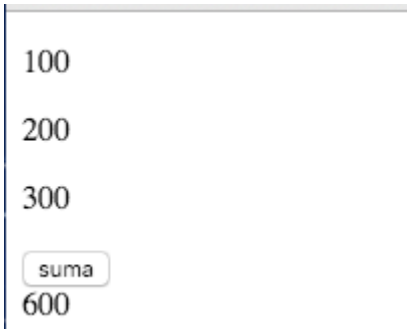
 $("#resultado").text(total);


 }
 </script>
</head>

<body>
 <div id="parrafos">
 <p>100</p>
 <p>200</p>
 <p>300</p>
 </div>
 <input type="button" id="boton" value="suma"/>
 <div id="resultado">
 </div>
</body>

</html>
```

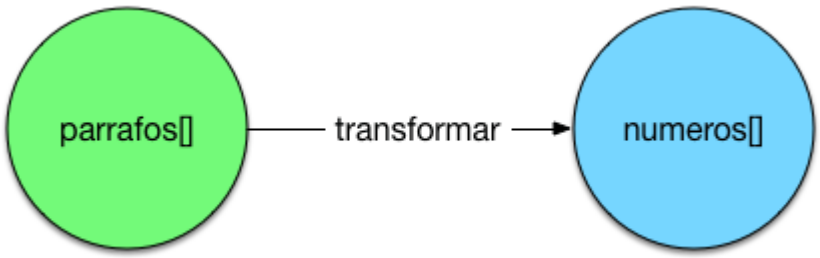
En este caso tenemos un botón que cuando le pulsamos suma varios párrafos utilizando jQuery y la función each.



¿Tenemos algún problema con el código? , en principio no lo parece la funcionalidad a implementar es sencilla. Sin embargo si que tenemos un problema importante, este código no es reutilizable ni testeable y necesitamos variables globales para trabajar. Vamos a ver como un enfoque orientado a JavaScript pure functions nos puede ayudar a construir un código más solido y modular.

JavaScript pure functions

En nuestro caso existen dos operaciones a realizar en un array de párrafos, la primera es de transformación que permita convertir el array de párrafos en un array de números.



Este primer paso se puede realizar a través de una función pura de JavaScript que use el método map.

```
function obtenerNumeros(nodos) {
```

```
 return nodos.map(function(objeto) {

return parseInt(objeto.innerHTML);

});

}
```

Recibimos un conjunto de nodos y los convertimos a un array de números. Hemos usado una función pura. El siguiente paso es sumar los diferentes elementos. Para ello usaremos otra función que se apoya en en el método reduce de Javascript

```
function sumar(numeros) {

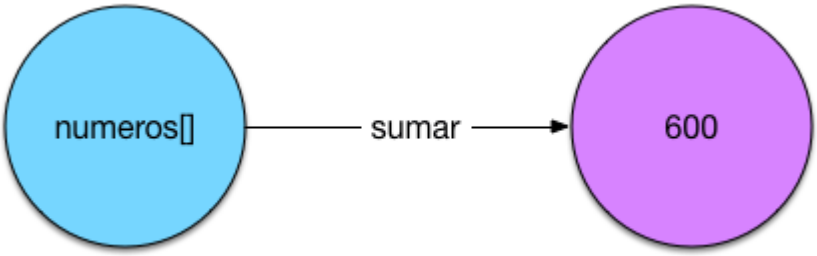
return numeros.reduce(function(suma,siguiente) {

return suma+siguiente;

},0);

}
```

Esta función también es una función pura y nos suma los diferentes item del array



Nos queda por ver algunas transformaciones adicionales del código para que se puedan apoyar en estas funciones:

```
<!DOCTYPE html>
<html>
<head>
 <title></title>
 <script type="text/javascript"
 src="jquery-1.11.3.min.js"></script>
 <script type="text/javascript">
 $(document).ready(function() {


 $("#boton").click(function() {


 actualizarVista();

 });



 });
```

```
function obtenerNumeros(nodos) {

return nodos.map(function(objeto) {

return parseInt(objeto.innerHTML);

});

}


function sumar(numeros) {

return numeros.reduce(function(suma,siguiente) {

return suma+siguiente;

},0);

}


function actualizarVista() {


$("#resultado").text(sumar(obtenerNumeros($("#parrafos
p").toArray())))

}

</script>
```

```
</head>
<body>
<div id="parrafos">
<p>100</p>
<p>200</p>
<p>300</p>
</div>
 <input type="button" id="boton" value="suma"/>
<div id="resultado">



</div>

</body>
</html>
```

Hemos usado el método toArray() de jQuery para quedarnos inicialmente con el grupo de párrafos y comenzar las transformaciones. Ahora nuestro código es mucho más reutilizable, modular y testeable que antes . Poco a poco tenemos que acercarnos a la programación funcional ya que sus ventajas son enormes.

Otros artículos relacionados: Java Reduce , JavaScript Map y JSON