JavaScript Programación Funcional

...

Resumen en base a https://developer.mozilla.org/es/docs/Web/JavaScript

Programación funcional

La programación funcional es un paradigma de programación que se enfoca en el uso de funciones como elementos básicos para resolver problemas y construir programas.

La programación funcional se basa en el concepto matemático de funciones, y se enfoca en la evaluación de expresiones y en la evitación de efectos secundarios. En este enfoque, una función toma una o más entradas y devuelve una salida sin cambiar los valores de las entradas o cualquier otro estado fuera de la función.

Programación funcional

En JavaScript, la programación funcional se puede lograr utilizando funciones de orden superior, como map(), filter() y reduce(), así como con el uso de técnicas como la inmutabilidad y la recursión.

Funciones aisladas: sin dependencia alguna del estado del programa, el cual incluye variables globales sujetas a cambios.

Funciones puras: una misma entrada siempre da la misma salida.

Funciones con efectos secundarios limitados: cualquier cambio o mutación en el estado del programa fuera de la función son cuidadosamente controlados

Recorrer una lista de números con For Each

Ejemplo:

```
const lista = [4,6,-2,3,8];
 /*
 for(let i=0; i<lista.length; i++) {
     console.log(lista[i]);
 }
 */
 lista.forEach( numero => console.log(numero) );
```

Recorrer una lista de objetos con foreach

```
<script>
    const productos = [
        {id:100,desc:'Tornillo'},{id:101,desc:'Martillo'},{id:102,desc:'Pinza'}]
    productos.forEach(
        p => console.log(`id ${p.id} descripcion: ${p.desc}`) )
 </script>
```

Map

El método map() crea un nuevo array con los resultados de la llamada a la función indicada aplicados a cada uno de sus elementos.

Sirve para evitar mutar el array original, evitando realizar lo siguiente:

```
const numeros = [3,4,-3,10,12,-3];
 for(let i = 0; i < numeros.length; i++){
     numeros[i] = numeros[i] + 2;
 }
 console.log(numeros);
```

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/map

Map

En programación funcional, evitamos mutar los arreglos:

```
const numeros = [3,4,-3,10,12,-3];

const copiaNumeros = [];

for(let i = 0; i < numeros.length; i++){

    copiaNumeros.push(numeros[i] + 2);

}

console.log(copiaNumeros);
```

Utilizando Map

Utilizando el método map.

```
const numeros = [3,4,-3,10,12,-3];

 const copiaNumeros = numeros.map(e => e+2);

 console.log(copiaNumeros);
```

Map con lista de objetos

Ejemplo:

```
const personas = [
    {nombre: "Juan", edad: 30},
    {nombre: "Ana", edad: 28}
 ]
 const personasMasDosAnios = personas.map(e => { return {
    nombre:e.nombre,edad:e.edad+2}});
 console.log(personasMasDosAnios);
```

Map. Ejercicio 1

Dada una lista de números, crear otra lista utilizando map, multiplicando cada elemento por 5.

Map. Ejercicio 2

Dada esta lista de precios: [100,120,300,230,320]

Obtener una nueva lista de precios con impuestos, utilizando map, agregando la tasa de iva del 21%

Map. Ejercicio 3

Dada esta lista de comprobantes de notas de crédito
[{id:100,descripcion:”prov1”,precio:1000},{id:102,descripcion:”prov2”,precio:1300}]

Devolver una nueva lista de comprobantes, con el precio con impuesto de iva del 21%

Filter

El método filter() crea un nuevo array con todos los elementos que cumplan la condición implementada por la función dada.

```
const numeros = [23,12,34,8,10];

const resultado = numeros.filter(e => e > 12);

console.log(resultado);
```

Filter con lista de objetos

Ejemplo:

```
const ncs = [{id:100,descripcion:"prov1",precio:1000},
{id:102,descripcion:"prov2",precio:1300},
{id:103,descripcion:"prov3",precio:600}];
// notas de crédito con importe mayor a 1000
const rta = ncs.filter(n => n.precio >= 1000);
console.log(rta);
```

Filter. Ejercicio 1

Dada una lista de números, realizar una función que tome dos parámetros, la lista , un numero y que devuelva otra lista con numeros menores al parámetro.

Filter. Ejercicio 2

Data esta lista de notas de crédito:

```
const ncs = [{id:100,descripcion:"prov1",precio:1000},
{id:102,descripcion:"prov2",precio:1300},
{id:103,descripcion:"prov3",precio:600}];
```

Realizar una función que retorne otra lista con las notas de crédito inferiores a un número pasado por parámetro.

Reduce

Ejemplo:

```
const numeros = [3,12,34,12,17,25];

const rta = numeros.reduce((a,c) => a + c, 0);

console.log(rta);
```

Reduce con lista de objetos

Ejemplo:

```
const ncs = [{id:100,descripcion:"prov1",precio:1000},
 {id:102,descripcion:"prov2",precio:1300},
 {id:103,descripcion:"prov3",precio:600}];
 // total notas de credito
 const rta = ncs.reduce((a,c) => a+c.precio,0);
 console.log(rta);
```

Reduce. Ejercicio 1

Dada esta lista de números [4,3,-2,5,8,-12,10]

Obtener la suma total de los elementos, iniciando en 100

Reduce. Ejercicio 2

Dada esta lista de recibos:

```
const recibos = [
        {numero: 1000, importe: 1500},
        {numero: 1001, importe: 2300},
        {numero: 1002, importe: 2100}
    ]
```

Obtener la suma total de los importes.

Sort

Para ordenar una lista, usamos el método sort, que ordena la misma lista (mutación)

```
const numeros = [4,6,8,9,10,12];
console.log(numeros.sort((a,b) => a-b));
```

Para ordenar String

```
const nombres = ['Juan','Alex','Maria', 'Marta'];

console.log(nombres.sort((a,b) => a.localeCompare(b)));
```

Para obtener una nueva lista ordenada, primero hacemos una copia y luego ordenamos

```
const numerosCopia = [...numeros];
```

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Spread_syntax

Combinando map, filter y reduce

Ejemplo, dada una lista de numeros, obtener la suma de los elementos multiplicados por 2 , que sean mayores a 10.

```
const numeros = [4,6,8,9,10,12];

 console.log(numeros.map(e => e*2).filter(e => e >=20).reduce((a,b)=> a+b,0 ));
```

Ejercicio combinado, map, filter y reduce

Dada esta lista de precios sin iva [2300,2500,5000,10000,1300]

Devolver la suma total de precios con iva que superen el importe de 3000.

Ejercicio integrador

const recibos = [{numero: 1000, importe: 1500, fecha: 20230320},
{numero: 1001, importe: 2300, fecha: 20230319},
{numero: 1002, importe: 2100, fecha: 20230318}]

De la anterior lista de recibos, obtener en la consola del navegador:

- Nueva lista ordenada por fecha de forma ascendete.
- Nueva lista de recibos con importe mayor a 2100.
- Nueva lista con un aumento del importe del 30%.
- Suma total de importes de la lista de recibos.
- Suma total de importes menores a 2500 de la lista de recibos, restando un 20% al importe