Programación Funcional

Esto es algo que se comentó en el documento pasado, la diferencia enorme entre programación funcional y programación orientada a objetos.

Francamente quise ver un video sobre como sirve la programación orientada a objetos en Java y lo odie, no lo comprendí (Esta sección es mi típico momento de odio a Java)

Lo que no logro comprender no es el lenguaje, en realidad me agrada el lenguaje, el tipiado es bastante bonito y la sintaxis estricta; Algo que, aparentemente, los lenguajes funcionales dejaron atrás. Hasta que hace un tiempito crearon lenguajes tipiados como TypeScript… que sigue siendo Javascript pero tipiado. LO QUE NO LOGRO COMPRENDER es como funciona realmente la programación orientada a objetos. Public voids dentro de clases, pero no podés accederlos remotamente? ¿Entonces para que existir si solo funcionan en un contexto? Y varias cosas así.. quizás les parezca entendible esto que acabo de decir, pero en la programación funcional las cosas funcionan de manera más suelta y “natural”:

Lo más importante que hay que notar de la programación funcional:

LAS FUNCIONES SON VARIABLES

¿Cómo así?

function triple(x) {

return x \* 3;

}

Esta es una función llamada triple, que recibe una variable, la multiplica por 3 y la retorna. Nada nuevo.

Pero también se le puede asignar una función a una variable…

var triple = function(x) {

return x \* 3;

}

(Aquí es donde los lenguajes orientados a objetos se quedan de “:000”)

La facilidad de esto es que se pueden invocar desde cualquier lugar, donde sea cuando sea.

Incluso puedes tomar un segmento de una función, crear una nueva y mandarla a llamar en varias otras sin ningún problema, cosa que en la POO suele ser.. ehm… ya lo dejo ahí.

Pero hagamos ese mismo código más bonito, usemos ES6 ☺

const triple = (x) => {

return x \* 3;

}

Se ve más bonito… pero incluso se puede simplificar más

const triple = x => x \* 3;

WHAT?

Esta sintaxis es mucho más corta y fácil de entender una vez se comprende, pero inicialmente es algo difícil acostumbrarse a ella, aquí están las reglas para ello:

const triple =

const es el tipo de variable, habitualmente siempre usamos const porque al alargar el código nos puede servir al evitar cambiar algo que no debemos, solo usamos let cuando sea realmente necesario. Triple es el nombre de la constante.

const triple = x =>

¿Por que el x => ?

Si la función solo recibe un parámetro, so puede escribir de esa forma.

function(x) es lo mismo que

(x) => o

x =>

Function(x,y) es lo mismo que

(x,y) => PERO

x, y => … ESO ES ERROR DE SINTAXIS.

Al no recibir ningún parámetro es necesario utilizar paréntesis:

function() {} es igual a

() => {}

Solo estas dos formas se pueden usar para declarar la función.

Pero ahora, ¿que hay dentro de la función?

const triple = x => {

console.log("este es el primero comando");

const y = x \* 3;

console.log("este es el tercer comando");

return y // Y este es el cuarto comando

}

En este ejemplo, se declara entre llaves lo que hará la función, y se realizan 4 comandos.

En este:

const triple = x => {

return x \* 3;

}

Solo se realiza 1 comando. Al darse esta característica, se puede eliminar las llaves :0

const triple = x => x \* 3;

automáticamente retornará el valor, sin que nosotros tengamos que escribir el “return”. Otra cosa importante es que hay que usar obligatoriamente el “;” al final de la función para que entienda que ahí acaba. Esta es una de las pocas veces donde es necesario usar el punto y coma, pero es muy útil y se ve mucho más legible.

Espero hayan entendido, sino, [aquí hay una buena referencia para estudiarlo mejor](https://www.youtube.com/watch?v=5jr-XSLOqp8), pues usaremos mucho ES6 en el transcurso de esta aplicación.

Bueno, ahora que entendimos que las funciones se pueden pasar a variables, igual que los string, ints, floats o arrays; estas también se pueden pasar a otras funciones, funciones de orden alto o HIGH ORDER FUNCTIONS(HOF).

¿Para que sirven las HOF? COMPOSICION. Metemos varias funciones pequeñas en otras más grandes.

Higher Order Functions

Antes que nada, mencionar que amo mencionarlas porque pareciera que sos inteligente, algo así como decir “Quintaesencia” o “Clusters”. Simple pero suena astral :^).

Antes que nada, esto es una conversión al español ligera de uno de mis youtubers favoritos, [MPJ de Fun Fun Function](https://www.youtube.com/channel/UCO1cgjhGzsSYb1rsB4bFe4Q) y devTips. Para una explicación más profunda y bonita, pueden ver sus videos.

Filter:

Filter sirve para filtrar datos de un array. Al retornar true no guarda el true, lo que hace es guardar el valor que está siendo evaluado, a diferencia de otras funciones como map. Filter es muy útil:

Supongamos que queremos hacer una aplicación que obtenga los nombres de todos los perros de un array:

let animales = [

{nombre: 'Pelusa', especie: "gato"},

{nombre: 'Maik', especie: "perro"},

{nombre: 'Jimmy', especie: "perico"},

{nombre: 'Firulais', especie: "perro"},

{nombre: 'Nidalee', especie: "gato"},

{nombre: 'Renekton', especie: "lagarto"},

];

Normalmnte lo haríamos así:

var perros = [];

for (var i = 0; i < animales.length; i++) {

if (animales[i].especie === "perro") {

perros.push(animales[i]);

}

}

PERO ESO ES MUY ABURRIDO Y LO HAREMOS A COMO SAN GUGUL NOS DICE QUE LO HAGAMOS (Porque, ya saben, literalmente esto lo hizo google).

Realizandolo en ES5 quedaría algo así:

var perros = animales.filter(function(animal) {

if (animal.especie === "perro") return animal;

});

Y el console.log(perros) daría:



Mudándolo a una versión un poco burda en ES6:

const perros = animales.filter((animal) => {

if(animal.especie === "perro") {

return animal;

}

});

Esto se ve algo mejor… pero simplifiquémoslo a como estaba antes, lo normal:

const perros = animales.filter(animal => {

if (animal.especie === "perro") return animal;

})

Pero para simplificarlo en toda la belleza de ES6:

const perros = animales.filter(animal => animal.especie === "perro");

console.log(perros);

Esa fue en la función filter. Filter recibe una función, osea que es la función más alta en rango, es la “HOF”, otro ejemplo más claro:

const esPerro = (animal) => animal.especie === "perro";

const perros = animales.filter(esPerro);

console.log(perros);

Lo que hacemos entonces, es filtrar a todos los animales que sean perros, de esta forma la pequeña función “esPerro” puede ser reutilizada en diferentes funciones HOF. En vez de filtrar a todos los perros, podríamos eliminar a todos los perros del array, o quizás cambiarles el nombre a otro, o lo que se nos ocurra, no solo usar “filter”. Esa es la idea de las HOF.

MAP

Supongamos que ahora queremos obtener los nombrs de todos los animales. Normalmente sería:

var nombres = [];

for (var i = 0; i < animales.length; i++) {

nombres.push(animales[i].nombre);

}

console.log(nombres);

y para hacerlo con una HOF, tenemos map. Map hace lo mismo que un for de 0 al ultimo miembro del array, y retorna lo que queramos:

const nombres = animales.map(function(animal) {

return animal.nombre;

});

console.log(nombres);

y su versión compacta:

const nombres = animales.map(animal => animal.nombre);

console.log(nombres);

map espera un callback que retorna lo que quieras como callback (Más tarde hablaremos de callbacks y promesas).

Podemos decir por ejemplo:

const nombres = animales.map(animal => {

return animal.nombre + " es un " + animal.especie;

});

Que retornaría:



Y en una sintaxis más corta con apostrofes en vez del típico string:

const nombres = animales.map(animal => `${animal.nombre} es un

${animal.especie}`);

Bonito verdad? Habitualmente usaremos más map que algún for.

Reduce:

Reduce es mágina, es novedad, es poder. Reduce es tan grande que podría ser su propio lenguaje!

Bueno no… realmente no… pero casi casi su propio Framework!

Uno de los Frameworks más famosos para visualización de data es Redux. Funciona en Vue, React, Angular y quien sabe que más. Está basado en Reduce. Reduce, Redux, bonito verdad?

Si map o filter o forEach o cualquier otra función no te da lo que necesitas que haces? Bueno, normalmente lloras en bolita un rato y luego te vas a StackOverflow. Pero eventualmente descubres que es Reduce y lo usas. Para todo. Hasta cuando no deberías de usarlo.

Supongamos que tenemos unos valores y queremos contar la cantidad total:

var pedidos = [

{cantidad: 250},

{cantidad: 400},

{cantidad: 100},

{cantidad: 325},

];

Resolvamoslo a la antigüita:

var cantidadTotal = 0;

for(var i = 0; i < pedidos.length; i++) {

cantidadTotal += pedidos[i].cantidad;

}

console.log(cantidadTotal);

Ahora, como lo haríamos con Reduce?

const cantidadTotal = pedidos.reduce(function(suma, pedidoActual) {

return suma + pedidoActual.cantidad;

},0);

console.log(cantidadTotal);

Como funciona esto?

Reduce recibe como primer parámetro una función y como segundo el valor inicial. La estructura es algo así:

algo.reduce(función,parámetro);

y como salida lo que hace es sumarlo todo en un solo espagueti ordenado.

AAAAAH QUE BONITO QUE ES.

En realidad, podrías saltarte el map, filter, todo lo que quieras y solo usar reduce… **pero no lo hagas. Las funciones existen por algo.**

Ahora, visto más compacto:

cantidadTotal = pedidos.reduce((suma, pedidoActual) => suma + pedidoActual.cantidad,0);

console.log(cantidadTotal);

Bonito verdad? No, PRECIOSO, DIVINO.

Para entender mejor los parámetros:

const cantidadTotal = pedidos.reduce((suma, pedidoActual) => {

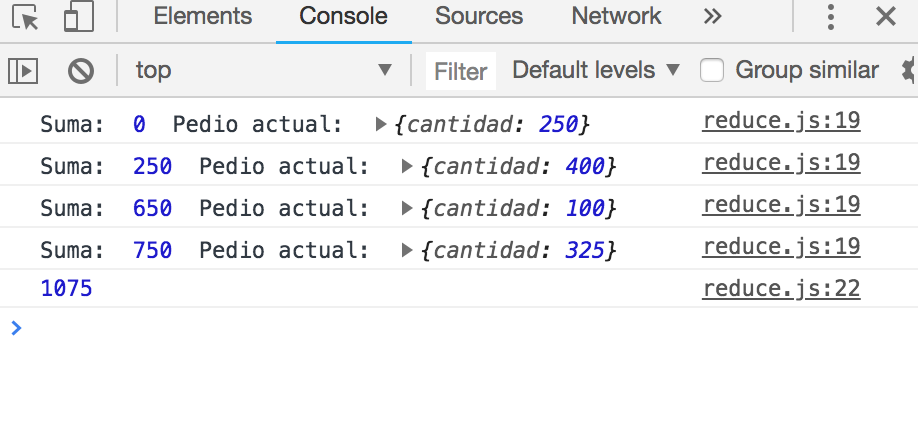
console.log("Suma: ", suma, " Pedio actual: ",pedidoActual);

return suma + pedidoActual.cantidad;

},0);

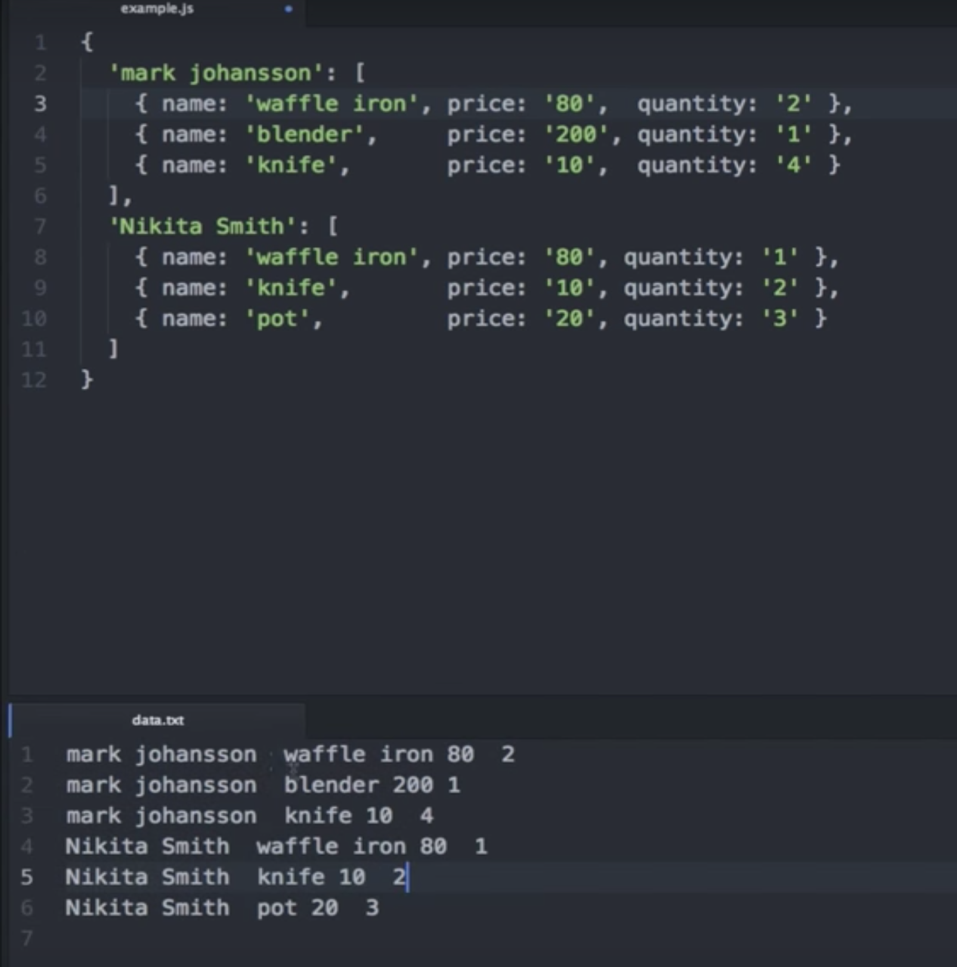
console.log(cantidadTotal);

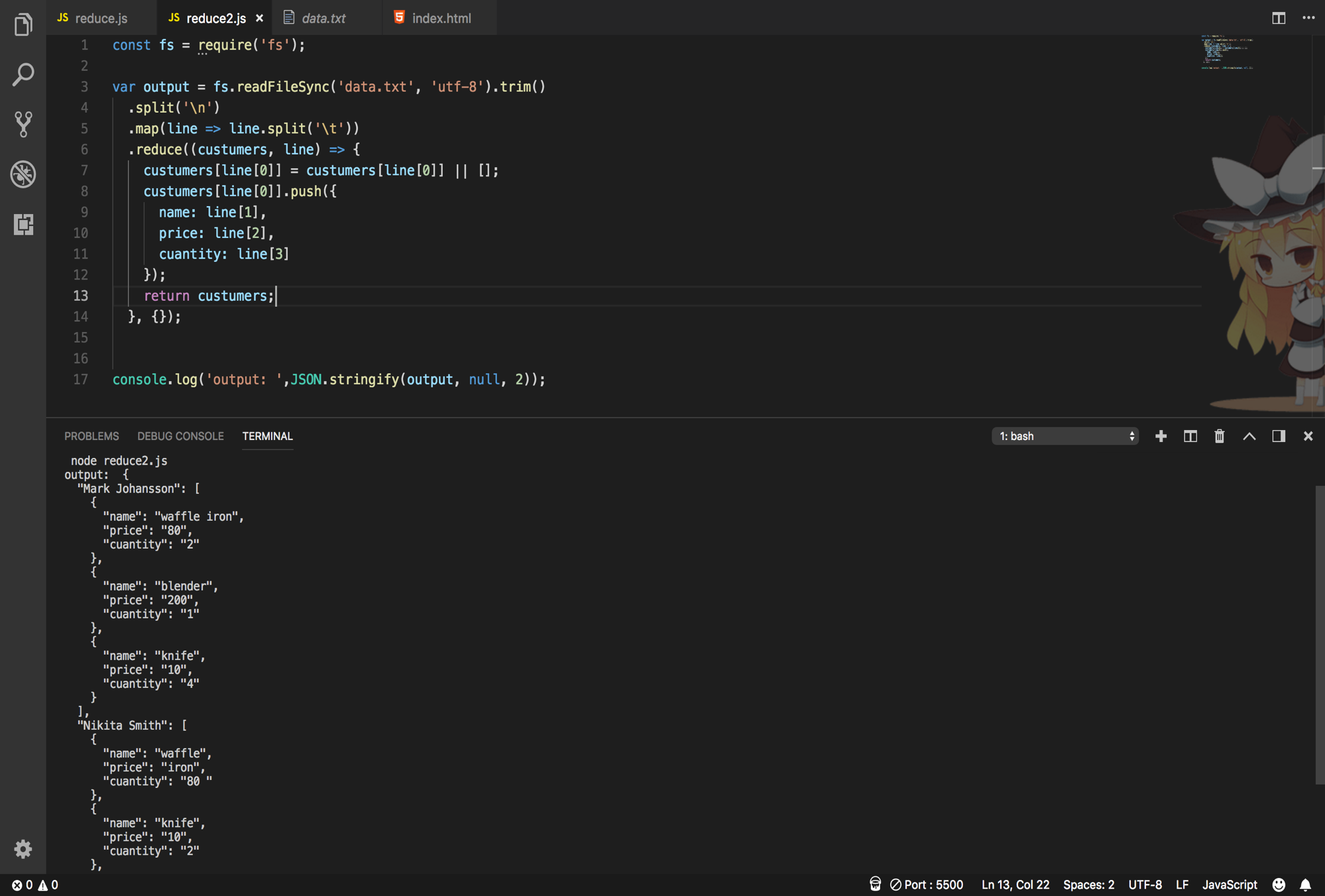
daría como resultado:



Es básicamente un for. Uno muy pero muy útil.

Un ejemplo del poder de Reduce es esto: Con NodeJS se ha leído un archivo txt separado por espacios, tabs y saltos de línea. Se ha convertido con Reduce en un Objeto. (Por su salud menta, no intenten comprender el código siguiente xd)





El código aparenta ser realmente complejo, cuando en realidad es bastante simple una vez se comprender. Esto se hizo fácil con la ayuda de Reduce y NodeJS, que puede abarcar muchas cosas, pero es un día más en la vida de un back-end developer. MPJ de fun fun Function hizo esto, si quieren ver más ejemplos interesantes hasta redes neorunales, su canal tiene mucho contenido relativamente avanzado.

Lo dejaremos hasta aquí con las clases, unos ejercicios para que intenten resolver ahora:

Ejercicios:

Los siguientes ejercicios son Katas de dificultad 8 a 7 de CodeWars. Si tardas más de una hora y no lo terminas, saltalo, lo importante es intentar.

1. Crear una función que reciba un array de números y los ordene de menor a mayor

Ejemplo:

sortArray([5, 3, 2, 8, 1, 4]) // retorna [1, 2, 3, 4, 5, 8]

1. Realizar una función que reciba un string de números separados por espacio y retorne un string que muestre el numero mayor y menor:

Sugerencias: [Split](https://www.w3schools.com/jsref/jsref_split.asp) y [sort](https://www.w3schools.com/jsref/jsref_sort.asp)

Ejemplos:

mayorYMenor("1 2 3 4 5"); // retorna "5 1"

mayorYMenor("1 2 -3 4 5"); // retorna "5 -3"

mayorYMenor("1 9 3 4 -5"); // retorna "9 -5"

Suerte :^)