Clase 6: Call Backs

Js tiene First class functions: es decir que sus funciones son iguales a cualquier otro objeto. Una función se puede pasar por argumento, se puede retornar desde otra función, se puede guardar como variable, dentro de un objeto o array.

En otros lenguajes esto no ocurre.

Una funcion pasada como argumento recibe el nombre de Call Back (CB)

**function** decirHolaAlUsuario(usuario) {

**return** 'Hola ' **+** usuario **+** '!';

}

**function** decirAdiosAlUsuario(usuario) {

**return** 'Adiós ' **+** usuario **+** '!';

}

**function** crearSaludo(usuario, cb) {

**return** cb(usuario);

}

crearSaludo('Dan', decirHolaAlUsuario); *// 'Hello Dan!'*

crearSaludo('Dan', decirAdiosAlUsuario); *// 'Goodbye Dan!'*

es una funcion ya declarada con anterioridad. Citada en otra funcion

hay que pasarle la funcion sin invocarla es decir sin () en el final. Porque si no lo que l eestamos dando es lo que retornaria esa funcion de cb y no la funcion en si

call backs en arreglos:

funciones del prototype de array:

for each: (no devuelve nada)

tipo de programación declarativa (no decimos con detalle todo lo que hay que hacer)

alumnos.forEach(function (elemento, índice)){

console.log(elemento) }

tipo de programación imperativa (ya decis todo lo que queres decir):

tmb se hace sin callback

for (i = 0; i > alumnos.length; i++); {

console.log(alumnos(i)) }

maps: (devuelve un arreglo con los mismo componentes que en el arreglo donde se lo invoca)

A Map object iterates its elements in insertion order — a [for...of](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...of) loop returns an array of [key, value] for each iteration.

Reduce: ( devuelve un único valor) – recibe 4argumentos –

Acumulador, valor actual, índice actual, array

Cuando el reduce termina de iterar los elementos, retorna lo acumulado en acc

Var sumaReduce = nums.Reduce (function (, acumulador acc, elemento el) {

Return acumulador mas elemento.}

Los nombres de las variables no están reservadas pero si las posiciones (1ro acc, 2do el)

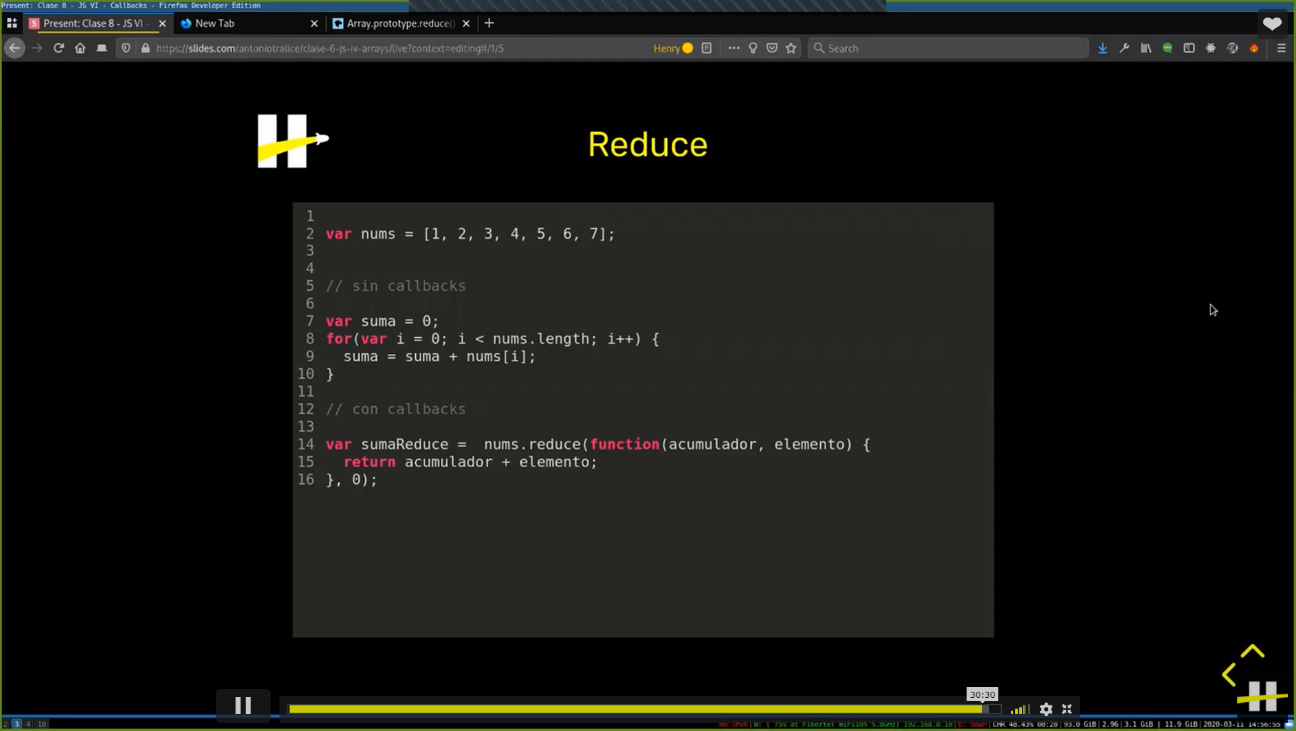
Acumulador : 0

Return 0 + 1

Return 0 + 1

La función va retornando cada variable dentro del objeto y sumándola al acumulador

En el código anterior tenemos un array de nombres. Para lograr contarlos, llamo a la función **reduce()**y dentro de ella paso una función como parámetro que a su vez recibirásus propios parámetros**(contadorNombre, nombre)**, **nombre** representa cada elemento del array y **contadorNombre** será una objeto donde guardaré la cantidad de veces que un nombre se repita. A diferencia del ejemplo anterior donde retornábamos un array, aquí el resultado será un objeto, por lo tanto a la función **reduce**, debo pasarle un parámetro más, que en este caso es un objeto vacío y representa el **estado inicial** de mi contador de nombres.



.toUpperCase() = cambia a mayúsculas las letras del string. O la letra que indiques. Va a la derecha del string

.slice() = retorna el string desde la posición del índice dado

4) para sumar con for los elementos del array primero crear la variante suma = 0, no dsp

  var suma = 0

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

    suma = suma + numeros[i];

  }

  cb(suma);

o con .reduce

var suma = numeros.reduce(function (acc, curr) {

 return (acc + curr);

} 0 );

cb(suma);

}

Números.reduce (function (acc = acumulador, curr = current)

5) con for:

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

cb(array[i]);

}

}

Asi call back se va a ejecutar tantas veces como array.length

Con for each: function call back recibe elemento(current value) e índice (opcional, que índice del arreglo ocurre en el momento), arreglo (manda todo el arreglo), this (opcional cual será el valor de this en el callback)

array.forEach(function (elemento) {

  cb(elemento);

});

}

Recibe una funcion con parámetros (elemento, índice, etc) por cada elemento del array va a tomar el elemento en cuestión y una vez iterado todo ejecuta la funcion.

6)

var newarray = [];

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

  newarray.push(cb(array[i]));

}

return newarray;

}

En la funcion del for se debe pushear al newarray los índices[i] de la func cb

El return no devuelve el mismo array, sino que devuelve el array pasado por la funcion callback

 var nuevoarray = array.map (function(element) {

return cb(element)

});

return nuevoarray;

}

El .map ahorra el for y el push

Pasa los elementos del array a la funcion CB y retorna (en este caso) el resultado

7) el .filter devuelve un array

Hay que pasarle una funcion (function (elemento)) {

Return elemento[0] === ‘a’;

})

A la funcion le das elemento y dsp le escribis lo que queiras… para que busque lo que necesitas

También hay una forma de hacerlo con =>

Newarray = array.filter(elemento => elemento[0] === ‘a’);

newArray = array.filter(function(elemento) {

 return elemento[0] === 'a';

function mayuscula(nombre) {

  //La función recibe un nombre y debe devolver el mismo que recibe pero con su primer letra en mayúscula

  //ej: Recibe "mario" ----> Devuelve "Mario"

  //Tu código:

  return nombre[0].toUpperCase() + nombre.slice(1);

también puede ser asi

return nombre.charAt(0).toUpperCase() + nombre.slice(1);

}

También se puede hacer asi. Usando el método replace en el cual damos como argumentos la variable que apuntamos a reemplazar y como segundo arg la variable por la cual queremos hacer el reemplazo.

   var inicial = nombre[0];

   var mayusc = inicial.toUpperCase();

   return nombre.replace(nombre[0], mayusc);

  }

function invocarCallback(cb) {

  // Invoca al callback `cb`

  //Tu código:

  return cb();

}

function operacionMatematica(n1, n2, cb) {

  //Vamos a recibir una función que realiza una operación matemática como callback junto con dos números.

  //Devolver el callback pasándole como argumentos los números recibidos.

  //Tu código:

  return cb(n1, n2);

}

function sumarArray(numeros, cb) {

  // Suma todos los números enteros (int/integers) de un array ("numeros")

  // Pasa el resultado a `cb`

  // No es necesario devolver nada

  //Tu código:

var suma = numeros.reduce(function (acc, curr) {

 return (acc + curr);

}, 0);

cb(suma);

}

Números.reduce recibe dos argumentos: una función, que a su vez recibe una acc y un curr; y un valor inicial para el acumulador, el 0 del final.

Números.reduce(function(acc, curr){la funcion}, 0’valor inicial del acc’… si yo quisiera qu earranqu een 10 el numero para la suma debería poner 10 en este argumento) si fuera una concatenación de strings ese valor inicial podría ser ‘’.

Una vez que hicimos la funcion reduce… la pasamos al cb que esta enunciado en la funcion principal

… la funcion reduce recibe un afuncion call back… que no tiene nada qu ver con la funcio call back de la que habla la funcion principal

Reduce devuelve un solo numero, no un array

function forEach(array, cb) {

  // Itera sobre la matriz "array" y pasa los valores al callback uno por uno

  // Pista: Estarás invocando a `cb` varias veces (una por cada valor en la matriz)

  //Tu código:

array.forEach(function (curr) {

  cb(curr);

});

}

Foreach le ejecuta a cada elemento del array una funcion call back que en este caso tiene como argumento curr, osea el elemento actual del array… entonces esa funcion pasa los valores current del array a la funcion cb invocada en la funcion anterior.

La funcion termina dentro de los argumentos de la funcion call back que le damos por argumento a foreach. Por eso cuando termina el código el final antes del } final es un ) que cierra esos argumentos del foreach

El foreach devuelve un arreglo igual al que oficia de método pero luego de pasar por al funcion call back indicada.

function map(array, cb) {

  // Crea un nuevo array

  // Itera sobre cada valor en "array", pásalo a `cb` y luego ubicar el valor devuelto por `cb` en un nuevo array

  // El nuevo array debe tener la misma longitud que el array del argumento

  //Tu código:

// var nuevoarray = array.map (function(element) {

//return cb(element)

//});

//return nuevoarray;

//}

Pudede ser asi sin.map, con for

var newarray = [];

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

  newarray.push(cb(array[i]));

}

return newarray;

}

O puede ser asi

 var nuevoarray = array.map (function(element) {

return cb(element)

});

return nuevoarray;

}

O asi

 var nuevoarray = [];

array.map (function(element) {

return nuevoarray.push(cb(element));

});

return nuevoarray;

}

Cada elemento que le pasamos a la funcion call back de maps, se los terminamos pasando a otra funcon call back, que no conocoemos… lo que esa fucnio desconocida retorne se la agregamos al new array y lo devolvemos

Osea la fucnon call backk de .map le pasa elementos a otra CB desconocida.

function filter(array) {

  //Filtrar todos los elementos del array que comiencen con la letra "a".

  //Devolver un nuevo array con los elementos que cumplen la condición

  //Tu código:

Puede ser con for

var newarray = [];

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

  if (array[i][0] === 'a') {

newarray.push(array[i]);

}

}

return newarray;

}

También puede ser con filter:

newArray = array.filter(function(elemento) {

 return elemento[0] === 'a';

})

return newArray;

}

También se puede asi, con if:

newArray = array.filter(function(elemento) {

if (elemento[0] === 'a')

return elemento;});

return newArray;

}

Array filter, le pasamos una funcion call back que lo que hace es retornar en un array nuevo los elementos que en su índice 0 tengan la letra ‘a’… todo eso lo guardamos en la variable new array y luego lo retornamos