¡Les damos la bienvenida!

¿Comenzamos?

Esta clase va a ser

- grabada

Clase 02. PROGRAMACIÓN BACKEND

Nuevas funcionaalidades de los lenguajes ECMAScrip

Temario

01

Principios básicos de JavaScript

- ✓ Tipos de datos y variables en Javascript
- ✓ Javascript y ES6
- ✓ Funciones en Javascript

02

Funcionalidades de los lenguajes ECMAScript

- ✓ Nuevas funciones de ECMAScript
- ✓ Funciones más utilizadas en backend

03

Programación sincrónica y asincrónica

- ✓ Funciones en Javascript
- ✓ Callbacks
- ✓ Promesas
- ✓ Sincronismo vs. Asincronismo

Objetivos de la clase

- Conocer las características de ECMAScript
- Aplicar los conceptos incorporados en el desarrollo backend

MAPA DE CONCEPTOS

Javascript y ECMAScript

Repaso

- ✓ En la clase 1 se abordó el concepto de ECMAScript, y cómo éste influye en el lenguaje de javascript que nosotros aplicamos en los navegadores y servidores en el día a día.
- ✓ ECMAScript definirá los estándares y funcionalidades que tendrá el lenguaje de programación javascript, con el fin de que los navegadores se ajusten a ello y pueda existir una compatibilidad más controlada.

Proceso de un cambio por ECMAScript

1. Se definen las nuevas funciones que habrá en el lenguaje y se eliminan las obsoletas
2. Dichos cambios pasan a una etapa de prueba para corroborar las funcionalidades
3. Se hace “release” de la nueva versión, los navegadores deberán ajustarse a ese nuevo versionado

Punto de partida: ES6

CODERHOUSE

ECMAScript 6

Desde ES5 en 2009, hubo un largo período de tiempo en el cual no hubo grandes actualizaciones en los estándares del lenguaje. Hasta ECMAScript 6 en 2015, el cual supuso una revolución en el lenguaje debido a sus grandes cambios.

La salida de ES6 marcó un antes y un después en la historia del lenguaje, ya que a partir de éste se lo comenzó a considerar una implementación “moderna”.

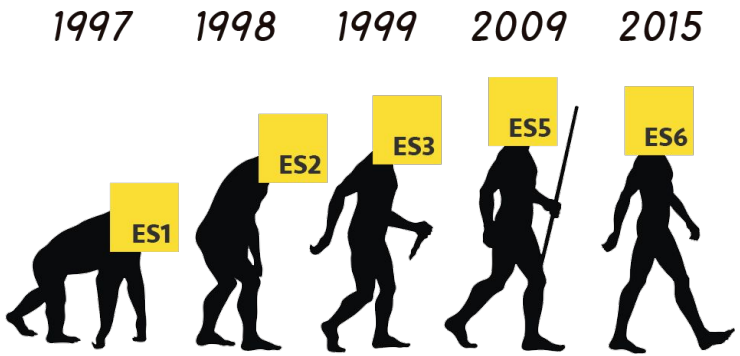
Javascript y ES7

CODERHOUSE

¿Qué hay de nuevo?

Las principales funcionalidades de este release son:

- ✓ Se introduce el operador **exponencial** **, independizándose poco a poco de la librería Math.
- ✓ Manejo de **array includes.** Éste nos permitirá saber si algún elemento existe dentro del arreglo.



CODERHOUSE

Ejemplo en vivo

- ✓ Utilización del operador exponencial y manejo de array con includes.

Ejemplo de uso de ** y Array.includes

exponential_includes.js

```
//Exponential  ** permite hacer el equivalente de la operación Math.pow(base,exp), para elevar un valor base a un exponente dado.
let valoresBase = [1,2,3,4,5,6] //Tenemos un conjunto de valores base.
let nuevosValores = valoresBase.map((numero,indice)=>numero**indice);
console.log(nuevosValores); // resultado: [ 1, 2, 9, 64, 625, 7776 ]
/**
 * El código mostrado arriba toma un arreglo de valores base y, con ayuda del operador map, utiliza el operador exponencial para elevar el
 * valor base, por su índice: (1**0,2**1,3**2,4**3,5**4,6**5)
 */

//Includes: Corrobora si algún elemento existe dentro del arreglo.
let nombres = ['Juan','Camilo','María','Ana','Humberto'];
if(nombres.includes('Camilo')){//includes devolverá sólo true o false según sea el caso, por lo cual podemos usarlo dentro del if
    console.log("Camilo sí aparece dentro del arreglo")
}else{
    console.log("Nombre no encontrado en la base de datos")
}
```

CODERHOUSE

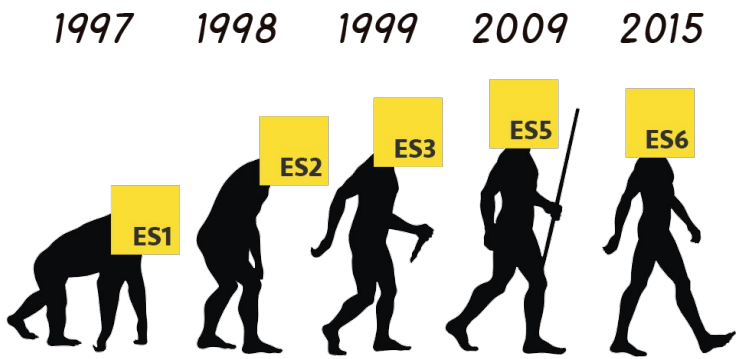
Javascript y ES8

CODERHOUSE

¿Qué hay de nuevo?

Las principales funcionalidades de este release son:

- ✓ Async await para mejor control asíncrono, sobre este ahondaremos más en futuras clases.
- ✓ Object.entries, Object.values, Object.keys para un mejor control interno sobre las propiedades de un objeto.



Ejemplo en vivo

- ✓ Utilización de operadores nullish
- ✓ Utilización de variables privadas

Ejemplo de uso de Object.entries, Object.keys, Object.values

entries_values_keys.js

```
}
let parLlaveValor = Object.entries(impuestos)
console.log(parLlaveValor) //resultado: [ [impuesto1,2341], [impuesto2,341], [impuesto3,4611], [impuesto4,111] ]
/**
 * Notamos cómo Object.keys obtiene en arreglos individuales la propiedad con su valor, en caso de que necesitemos utilizarlas por separado.
 */
let soloPropiedades = Object.keys(impuestos)
console.log(soloPropiedades)//resultado: [impuesto1,impuesto2,impuesto3,impuesto4]
/**
 * Ahora podemos obtener sólo las propiedades del objeto, sin necesidad de su valor, este método es MUY ÚTIL en códigos profesionales,
 * sin embargo, por cuestiones de complejidad se abordará en elementos prácticos más adelante.
 */
let soloValores = Object.values(impuestos)
console.log(soloValores)//resultado : [2341,341,4611,111]
/**
 * Teniendo sólo los valores del objeto, podemos utilizarlos para hacer un total (En este ejemplo nos apoyamos de un método ya existente
 * conocido como .reduce)
 */
let impuestosTotales = soloValores.reduce((valorInicial,valorAcumulado)=>valorAcumulado+valorInicial);
console.log(impuestosTotales);// 7404, total de todos los impuestos.
```

Nota: reduce es un método complejo al momento de utilizarlo, puede aprovecharse este ejemplo para ahondar en él.

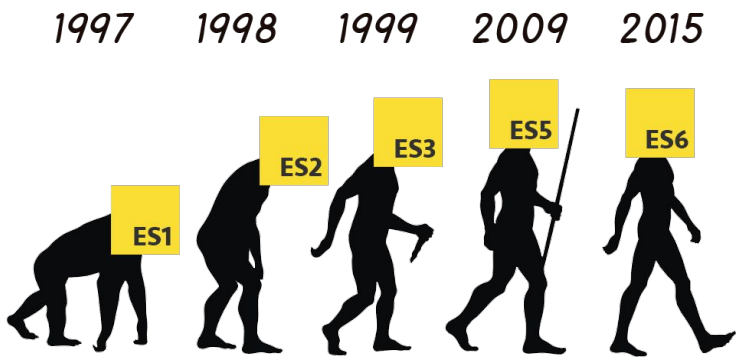
Javascript y ES9

CODERHOUSE

¿Qué podemos destacar?

Las principales funcionalidades de este release son:

- ✓ Resolvedores de promesas con .finally(), para atender una promesa, se cumpla o no se cumpla.
- ✓ Mayor control a los objetos con operadores **rest** y **spread** (aplicable también a arrays)



Ejemplo en vivo

- ✓ Utilización básica de operador rest y operador spread en los objetos.

Ejemplo de uso de spread operator y rest operator

spread_rest.js

```js
//Dados los siguientes objetos
let objeto1 = {
    propiedad1:2,
    propiedad2:"b",
    propiedad3:true
}
let objeto2 = {
    propiedad1:"c",
    propiedad2:[2,3,5,6,7]
}
//SPREAD OPERATOR Nos sirve para hacer una destructuración del objeto, para poder utilizar sólo las propiedades que queremos.
let  {propiedad1,propiedad2} = objeto1; //Tomamos el objeto1 y lo "rompemos" para obtener sólo las primeras dos propiedades.
let objeto3 = {...objeto1,...objeto2} //Spread operator también se puede utilizar para vaciar propiedades de un objeto en otro objeto nuevo.

console.log(objeto3); // resultado :{ propiedad1: 'c', propiedad2: [ 2, 3, 5, 6, 7 ], propiedad3: true }
//Notamos cómo, si dos elementos comparten el mismo nombre de propiedad, se superponen, por lo que propiedad1 y propiedad2 del objeto uno ya
//no existen dentro del objeto 3, sino que fueron "superpuestos" por propiedad1 y propiedad2 del objeto 2.

//REST OPERATOR nos servirá para obtener un objeto SÓLO con las propiedades RESTANTES del objeto que hayamos destructurado, por ejemplo:
let objeto4 = {
    a : 1,
    b : 2,
    c : 3
}
let  {a,...rest} = objeto4; //Indicamos que queremos trabajar con la propiedad a, y guardar en un objeto el resto de las propiedades de ese
//objeto, en caso de que los necesitemos más adelante.
console.log(rest); //resultado: { b: 2, c: 3 }
```

Utilización ES6-ES9

Pongamos en práctica algunos de los módulos vistos en los ejemplos

Duración: **15-20 min**

CODERHOUSE

ACTIVIDAD EN CLASE

Utilización ES6-ES9

Descripción de la actividad.
Dados los objetos indicados en la siguiente diapositiva:

- ✓ Realizar una lista nueva (array) que contenga todos los tipos de productos (no cantidades), consejo: utilizar **Object.keys** y **Array.includes.** Mostrar el array por consola.
- ✓ Posteriormente, obtener el total de productos vendidos por todos los objetos (utilizar **Object.values)**

```
const objetos =  [
    {
        manzanas:3,
        peras:2,
        carne:1,
        jugos:5,
        dulces:2
    },
    {
        manzanas:1,
        sandias:1,
        huevos:6,
        jugos:1,
        panes:4
    }
]
```

Break

¡10 minutos y volvemos!

Para pensar

Hemos notado muchas cosas que se han implementado a lo largo de ECMAScript.

- ✓ ¿Es obligatorio aprender cada nueva implementación para mantenerse en el mundo laboral?
- ✓ ¿El uso de una nueva implementación implica el reemplazo de lo viejo?

Nuevas implementaciones con ECMAScript 10 y 11

¿Qué considerar aquí?

A partir de estas implementaciones, tenemos que hacer una aclaración importante. Muchos de los cambios en estas versiones están estrechamente ligados con otros conceptos que irás aprendiendo a lo largo del curso. Por lo tanto, sólo se hará énfasis en los elementos con los que podemos trabajar a esta altura del curso.

CODERHOUSE

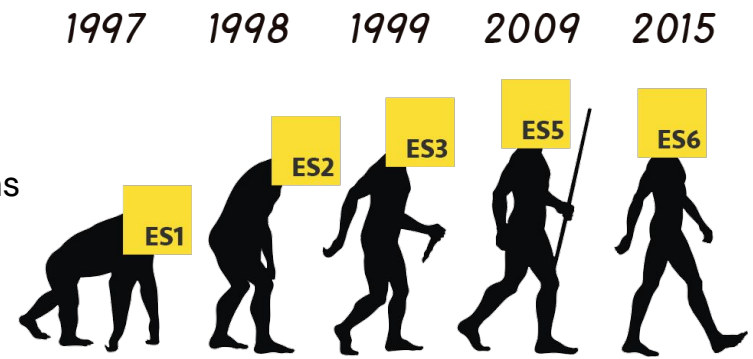
Javascript y ES10

CODERHOUSE

¿Qué podemos destacar?

Las principales funcionalidades de este release son:

- ✓ String.trim(): Remueve los espacios innecesarios en una cadena. Sirve para validar cadenas enviadas vacías o eliminar los espacios iniciales y finales.
- ✓ Array.flat():Remueve anidaciones internas en arrays para dejar un arreglo plano.
- ✓ Dynamic import: Permite importar **sólo los módulos necesarios**, ahorrando espacio y memoria.



CODERHOUSE

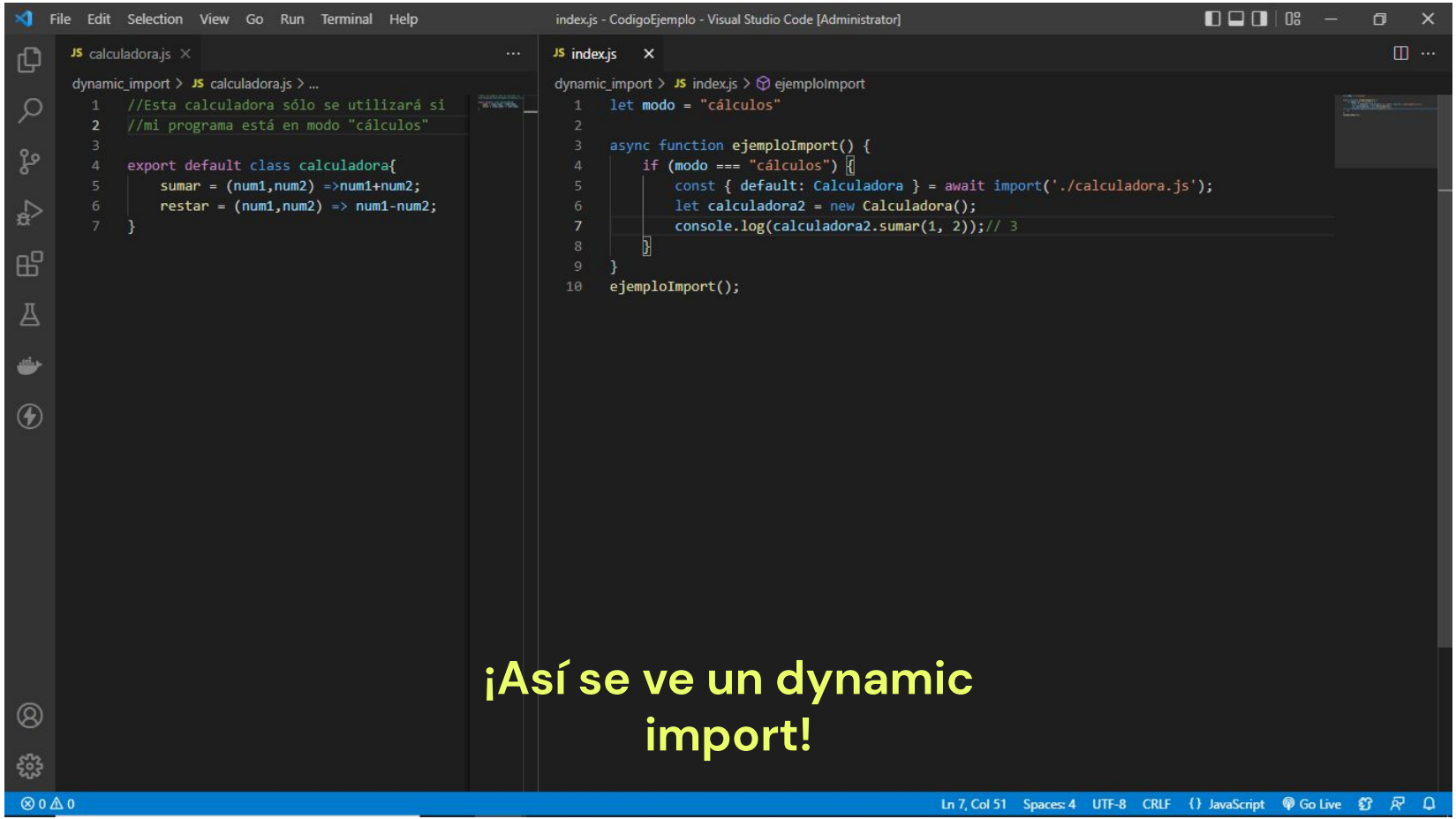
Dynamic import

Uno de los problemas principales de los imports tradicionales, es que terminamos importando TODOS LOS MÓDULOS, aun cuando no estamos utilizando todos al mismo tiempo.

Con dynamic import, esto cambia.

Dynamic import permite importar **sólo los módulos** que necesito según una situación particular, lo cual permite optimizar la utilización de recursos, al pedir a la computadora sólo lo que estaré utilizando.

Es utilizado principalmente en códigos que utilizan el patrón de diseño Factory (se abordará más adelante).


File Edit Selection View Go Run Terminal Help
index.js - CodigoEjemplo - Visual Studio Code [Administrator]
calculadora.js
dynamic_import > calculadora.js > ...
1 //Esta calculadora sólo se utilizará si
2 //mi programa está en modo "cálculos"
3
4 export default class calculadora{
5 sumar = (num1,num2) =>num1+num2;
6 restar = (num1,num2) => num1-num2;
7 }
index.js
dynamic_import > index.js > ejemploImport
1 let modo = "cálculos"
2
3 async function ejemploImport() {
4 if (modo === "cálculos") {
5 const { default: Calculadora } = await import('./calculadora.js');
6 let calculadora2 = new Calculadora();
7 console.log(calculadora2.sumar(1, 2));// 3
8 }
9 }
10 ejemploImport();
¡Así se ve un dynamic import!
Ln 7, Col 51 Spaces: 4 UTF-8 CRLF {} JavaScript Go Live

CODERHOUSE

Ejemplo en vivo

- ✓ Validación de cadena con trim
- ✓ Aplanado de Array con múltiple anidación

Ejemplo de uso de los métodos trim y flat

trim_flat.js

```
let cadena1 = `              hola`//El mensaje no debería enviarse de esa manera, ya que consume espacio innecesario a almacenar.
console.log(cadena1.trim());//resultado: "hola"
//Podemos validar también si me enviaron un mensaje vacío, para saber si almacenarlo o no enviar el mensaje (simulando un chat)
let mensajes=[];
let intentoDeMensaje=`                          `
if(intentoDeMensaje.trim().length>0){//Por lo menos hay un caracter (no espacio) para enviar al usuario, entonces es un mensaje válido.
    mensajes.push(intentoDeMensaje.trim());
}else{//Si la condición entra a else, es porque el mensaje venía vacío y por lo tanto no deberíamos guardarlo ni enviarlo al otro usuario
    console.log("Mensaje vacío, para poder enviar un mensaje, favor de escribir algo")
}
/**
 * Sin el método trim, permitimos muchas brechas de seguridad al momento de querer procesar cadenas de texto, de modo que es bueno limitarlas
 * a un formato que dominemos (sin espacios extras ni cadenas aparentemente vacías)
 */

//Uso de flat
let arrayAnidado = [1,32,4,5,6,[1,4,5,1],[3411,3,4]]//Anteriormente, para poder obtener los valores internos de cada arreglo, usábamos
// una técnica llamada "Recursividad", esta vez es más sencillo.
console.log(arrayAnidado.flat());//resultado: [1,32,4,5,6,1,4,5,1,3411,3,4], notamos que ya no hay más anidación y podemos manejarlos mejor.
```

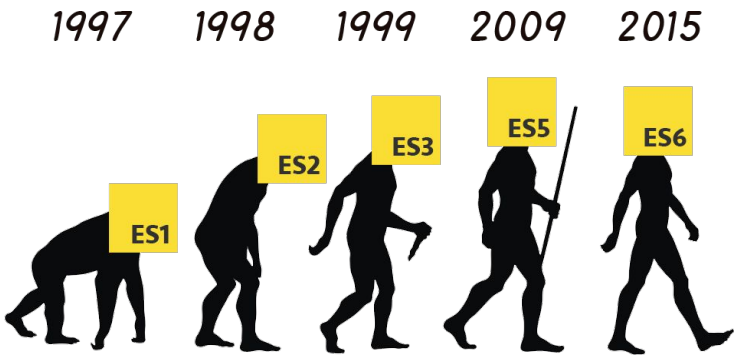
Javascript y ES11

CODERHOUSE

¿Qué podemos destacar?

Las principales funcionalidades de este release son:

- ✓ Operador nullish. Sirve para poder colocar un valor por default a variables que podrían ser nulas o indefinidas, a diferencia del operador ||, éstas filtran “falseys”
- ✓ Variables privadas dentro de las clases, esto permite que algunas variables no sean accesibles desde el entorno de instancia de una clase y sólo sea utilizada de manera interna.



CODERHOUSE

Ejemplo en vivo

- ✓ Explicación de asignación de variable a partir de un nullish, para entender su diferencia con el operador OR ||
- ✓ Explicación de una variable privada en una clase.

Ejemplo de operador nullish y variables privadas

nullish_privateVariables.js

```
/**
 * El operador nullish difiere del operador ||, ya que || también ignora valores falseys.
 */
let variableDePrueba = 0 //Reemplazar esta variable de prueba por diferentes valores null, undefined, y falseys

let variableAsignable = variableDePrueba || "Sin valor";
console.log(variableAsignable); // Tal vez yo necesitaba el valor 0, pero al utilizar ||, se toma el valor por default
let variableConNullish = variableDePrueba??"Sin valor";
console.log(variableConNullish);//Notamos que a este punto sí podemos tomar el valor 0 que necesitamos.
/**
 * Notamos que con Nullish realmente sólo nos interesa que el valor NO SEA undefined o null. Todo lo demás podemos asignarlo,
 */

// Sobre variable privada
class Persona {
    #fullname;//Primero declaramos la variable antes del constructor para poder construirla con los valores del constructor
    constructor(nombre,apellido){
        this.nombre=nombre;
        this.apellido=apellido;
        this.#fullname=`${this.nombre} ${this.apellido}` //Asignamos el valor de la variable privada.
    }
    //Esta variable la podemos utilizar de manera interna. No se puede acceder a ella por fuera.
    getFullName = () =>{
        return this.#fullname; //La única forma en la cual podemos obtener el valor de esa variable privada es pidiéndola desde un método
        //Permite utilizar variables, pero evitando que se puedan modificar desde fuera (por seguridad).
    }
    #metodoPrivado = () =>{//También podemos declarar métodos privados, para que sólo sean usados dentro de la clase, y no llamados por fuera
        //Aquí realizamos tareas que queremos que se ejecuten sólo de manera interna en la clase, no podemos mandar a llamar este método
    }
}
let instancia1 = new Persona('Mauricio','Espinosa')

console.log(instancia1.getFullName())// Mauricio Espinosa
```

CODERHOUSE

Hands on lab

En esta instancia de la clase **repasaremos** algunos de los conceptos vistos en clase con una aplicación

¿De qué manera?
El profesor demostrará cómo hacerlo y tú lo puedes ir replicando en tu computadora. Si surgen dudas las puedes compartir para resolverlas en conjunto de la mano de los tutores.

Tiempo estimado: **20 minutos**

HANDS ON LAB

Registrador de tickets de eventos

¿Cómo lo hacemos? **Se creará una clase que permitirá llevar una gestión completa de usuarios que deseen acceder a dichos eventos.**

- ✓ Definir clase TicketManager, el cual tendrá un arreglo de eventos que iniciará vacío
- ✓ La clase debe contar con una variable privada “precioBaseDeGanancia”, la cual añadirá un costo adicional al precio de cada evento.
- ✓ Debe contar con el método “getEventos” El cual mostrará los eventos guardados.
- ✓ Debe contar con el método “agregarEvento” El cual recibirá los siguientes parámetros:
  - nombre
  - lugar
  - precio (deberá agregarse un 0.15 del valor original)
  - capacidad (50 por defecto)
  - fecha (hoy por defecto)

  El método deberá crear además el campo id autoincrementable y el campo “participantes” que siempre iniciará con un arreglo vacío.

HANDS ON LAB

Registrador de tickets de eventos

- ✓ Debe contar con un método “agregarUsuario” El cual recibirá:
  - id del evento (debe existir, agregar validaciones)
  - id del usuario

  El método debe evaluar que el evento exista y que el usuario no haya estado registrado previamente (validación de fecha y capacidad se evitará para no alargar el reto)
  Si todo está en orden, debe agregar el id del usuario en el arreglo “participantes” de ese evento.
- ✓ Debe contar con un método “ponerEventoEnGira” El cual recibirá:
  - id del evento
  - nueva localidad
  - nueva fecha

  El método debe copiar el evento existente, con una nueva localidad, nueva fecha, nuevo id y sus participantes vacíos (Usar spread operator para el resto de las propiedades)

Clases con ECMAScript y ECMAScript avanzado

CODERHOUSE

DESAFÍO ENTREGABLE

Clases con ECMAScript y ECMAScript avanzado

Consigna

- ✓ Realizar una clase “ProductManager” que gestione un conjunto de productos.

Te acercamos esta ayuda 👉

Hands on lab sobre creación de clases (clase 1)

Aspectos a incluir

- ✓ Debe crearse desde su constructor con el elemento products, el cual será un arreglo vacío.
- ✓ Cada producto que gestione debe contar con las propiedades:
  - title (nombre del producto)
  - description (descripción del producto)
  - price (precio)
  - thumbnail (ruta de imagen)
  - code (código identificador)
  - stock (número de piezas disponibles)

CODERHOUSE

DESAFÍO ENTREGABLE

Aspectos a incluir

- ✓ Debe contar con un método “addProduct” el cual agregará un producto al arreglo de productos inicial.
  - Validar que no se repita el campo “code” y que todos los campos sean obligatorios
  - Al agregarlo, debe crearse con un id autoincrementable
- ✓ Debe contar con un método “getProducts” el cual debe devolver el arreglo con todos los productos creados hasta ese momento
- ✓ Debe contar con un método “getProductById” el cual debe buscar en el arreglo el producto que coincida con el id
  - En caso de no coincidir ningún id, mostrar en consola un error “Not found”
  -

Formato del entregable

- ✓ Archivo de Javascript listo para ejecutarse desde node.

Proceso de testing de este entregable ✅

CODERHOUSE

¿Preguntas?

Opina y valora esta clase

Resumen
de la clase hoy

- ✓ Repaso sobre tipos de datos y variables en Javascript
- ✓ Javascript y ES6
- ✓ Funciones en Javascript
- ✓ Uso de clases en Javascript

Muchas gracias.

#DemocratizandoLaEducación