releevant.

Full Stack Bootcamp

Programación & Algorítmica & Proyectos Software
Introducción a BBDD
Bases de Datos avanzadas
Javascript avanzado & EcmaScript 6
MongoDB

releevant.

Un DOM para dominarlos a todos

- Abreviatura de Document Object Model

```
<!DOCTYPE html>
<html>
    <body>
        <p>Esto es una p&aacute;gina web</p>
    </body>
</html>
```

- La importancia de validar el HTML (DOCTYPE)
- Representación del árbol de etiquetas
- Podemos manipular el DOM desde JavaScript

Ejercicio: Sacar el DOM de una URL que nos guste

releevant.

Document Object Library - Eventos

- Occuren muchos eventos en el DOM , por ejemplo onClick u onLoad
  - onClick sucede cuando el usuario hace “click” en un objeto del DOM
  - onLoad sucede cuando se carga el DOM al completo
- Se definen dentro del HTML del DOM
- Cuando ocurre un evento puedo ejecutar código JavaScript

```
<!DOCTYPE html>
<html>
    <body onLoad=“miFuncion()”>
        <p>Esto es una p&aacute;gina web</p>
    </body>
</html>
```

Ejercicio: Incluir evento onLoad a vuestro HTML y que llame a una función que muestre un mensaje por consola

releevant.

Document Object Library - Eventos

- Occuren muchos eventos en el DOM , por ejemplo onClick u onLoad
  - onClick sucede cuando el usuario hace “click” en un objeto del DOM
  - onLoad sucede cuando se carga el DOM al completo
- Se definen dentro del HTML del DOM
- Cuando ocurre un evento puedo ejecutar código JavaScript

```
<!DOCTYPE html>
<html>
    <body onLoad=“miFuncion()”>
        <p>Esto es una p&aacute;gina web</p>
    </body>
</html>
```

Ejercicio: Incluir evento onLoad a vuestro HTML y que llame a una función que muestre un mensaje por consola

releevant.

Document Object Library - Identificadores

- Les da un nombre identificativo a cada objeto del DOM

```
<!DOCTYPE html>
<html>
    <body onLoad="miFuncion()">
        <div id="container">
            <p id="text">Esto es una p&aacute;gina web</p>
        </div>
    </body>
</html>
```

Document Object Library – Incluir datos

- Acceder a elementos del DOM y manipular su contenido

var container = Document.getElementByID(“container”);

```
<!DOCTYPE html>
<html>
    <body onLoad=“miFuncion()”>
        <div id=“container”>
            <p id=“text”>Esto es una p&aacute;gina web</p>
        </div>
    </body>
</html>
```

Ejercicio: Modificar desde JavaScript el texto en el elemento <p>

NodeJS – MySQL

releevant.

- Instalar la libraría de MySQL

```
npm install mysql
```

- Incluir en nuestro código

```
let mysql = require('mysql’);
```

- Configurar la conexión con MySQL

```
let connection = mysql.createConnection({
        host: 'localhost’,
        user: 'root’,
        password: ‘’,
        database: 'todoapp’});
```

- Conectar a la base de datos

```
connection.connect(function(err) {
    if (err) {
        return console.error('error: ' + err.message);
    }
    console.log('Connected to the MySQL server.’);
});
```

releevant.

NodeJS – MySQL (II)

- Cerrar la conexión

```
connection.end(function(err) {
    if (err) {
        return console.log('error:' + err.message);
    }
    console.log('Close the database connection.');
});
```

releevant.

NodeJS – MySQL (III)

- Ejecutar una sentencia

```
connection.query('SELECT * FROM `Books` WHERE `title`= ?',
                 ['book_title'], function(error, results, fields) {
    // Gestionar error y resultados
});
```

Ejercicio: Escribir el código para obtener el listado de productos de nuestra base de datos de Ecommerce.

NodeJS – MySQL (IV)

- Ejercicios

1. *Sacar por consola el pago mas usado por los usuarios y cuantos usuarios usan dicho método*
2. *Sacar por consola lista de productos con la cantidad total vendida.*
3. *Sacar por consola la lista de productos ordenados de mas caro a mas barato y otra lista de mas barato a mas caro*
4. *Sacar por consola Lista de productos con el average del rating*
5. *Sacar por consola cual es el producto mas vendido*
6. *Sacar por consola el producto que mayor ganancia ha generado*
7. *Sacar por consola el usuario que mas artículos ha comprado*
8. *Sacar por consola el usuario que se ha gastado mas dinero. Incluir el símbolo €.*

Notas:

- Usar funciones para cada punto
- Usar bucles para iterar sobre la lista de los resultados de base de datos

releevant.

Alcance, Scope y Closure

- Alcance global

```
let mysql = require('mysql');
```

- Alcance local

```
function conecta() {
    var con = mysql.connect(…);

    return con;
}
```

releevant.

Alcance, Scope y Closure (II)

- Closure

```
function miFun() {
    var nombre = “minombre”;
    function mostrar() {
        console.log(nombre);
    }
    mostrar();
}
miFun();
```

Alcance, Scope y Closure (II)

- Ejercicios: Analiza que sale por consola

```
var a = 12;
(function() {
  console.log(a);
})();

var a = 5;
(function() {
  var a = 12;
  console.log(a);
})();
```

```
var a = 10;
var x = (function() {
  var a = 12;
  return (function() {
    console.log(a);
  });
})();
x();
```

Alcance, Scope y Closure (IV)

- Ejercicios: Analiza que sale por consola

```
var a = 10;
var x = (function() {
  var y = function() {
    var a = 12;
  };
  return function() {
    console.log(a);
  }
})();
x();
```

```
var a = 10;
var x = (function() {
  (function() {
    a = 12;
  })();
  return (function() {
    console.log(a);
  });
})();
x();
```

releevant.

Alcance, Scope y Closure (V)

- Ejercicios: Analiza que sale por consola

```
var a = 10;
(function() {
  var a = 15;
  window.x = function() {
    console.log(a);
  }
})();
x();
```

releevant.

AJAX – Asynchronous Javascript and XML

- Permite desde JavaScript poder hacer llamadas asíncronas

  Por ejemplo, http://localhost:8080/productos

- Lee los datos de la URL y hace que los datos estén disponibles en JavaScript
- También permite enviar datos a una URL

releevant.

AJAX – Como se usa

```
let xhttp = new XMLHttpRequest();

xhttp.onreadystatechange = function() {

    if (this.readyState == 4 && this.status == 200) {
        document.getElementById("demo").innerHTML = this.responseText;
    }
};

xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

Ejercicio: incluir AJAX a nuestra solución de Ecommerce.

Objetos y clases en JavaScript

- Recordemos el paradigma programación orientada a objetos
- Clase motor en JavaScript

```
function motor(potencia, valvulas) {
        this.potencia = potencia;
        this.valvulas = valvulas;
        this.encendido = false;
        this.encender = function() {
                this.encendido = true;
        }
        this.apagar = function() {
                this.encendido = false;
        }
}
```

```
let motorCuatroV = new motor(1500, 4);

let motorSeisV = new motor(3000, 6);


motorCuatroV.encender();

motroSeisV.apagar();
```

releevant.

Objetos y clases - This

- En JavaScript *This* se refiere al objeto al que pertenece
  - En un metodo, this ser refiere al objeto que lo contiene

```
let obj = {
        param: “estoesunavariable”,
        doSomething: function() {
                this.param = “otro valor”;
        }
}
```

  - Solo, this se refiere al objeto global (objeto window en web)
  - En una function this se refiere al objeto global
  - En una function en ‘strict mode’ this es undefined
  - En un evento this se refiere al element que recibe el evento

```
<button onClick=“function(e) {
                this.innerText = “pulsado”;
                this.state = false;
        }
</button>
```

releevant.

Objetos y clases - Ejercicio

- Crear un script que defina un objeto llamado Producto_alimenticio.
- Este objeto debe presentar las propiedades código, nombre y precio, además del método imprimeDatos, el cual escribe por pantalla los valores de sus propiedades.
- Posteriormente, cree tres instancias de este objeto y guárdelas en un array.
- Posteriormente, utilice el método imprimeDatos para mostrar por pantalla los valores de los tres objetos instanciados.

Objetos y clases – Ejercicio (II)

- Crear una clasepp llamada Factura que permita instanciar objetos de ese tipo de la siguiente forma:
- Factura(cliente,elementos). Cliente es un objeto que guarda los datos del cliente (nombre, dirección, teléfono y nif) y elementos es un array que contiene la siguiente información por cada uno de los ítems que puede tener la factura: descripción, cantidad y precio.
- Además sobre cada factura se desea guardar los siguientes datos: base imponible,iva, total y forma de pago. Por defecto, tendrán como valores 0,21,0,"contado" respectivamente.
- Añade con posterioridad a la pseudoclase Factura:
  - Propiedad empresa que guardará información sobre la empresa que emite la factura (nombre, dirección, teléfono y cif).
  - Método que calcule el total de la factura (con el IVA aplicado);
  - Método que muestre el total.
- Realiza varias instanciaciones de Factura y muestra el total usando un for.

releevant.

Objetos y clases – Ejercicio (III)

- Crea una clase que llamaremos Bus. Sus atributos serán:
  - capacidad: número máximo de pasajeros
  - pasajeros: número de pasajeros (inicialmente 0)
  - conductor: objeto conductor.
- Sus métodos
  - subir(pasajeros): aumenta el numero de pasajeros
  - bajar(pasajaeros): disminuye el número de pasajeros
  - conductor: asigna un objeto conductor.
- El ojeto conductor es de una clase (Conductor) cuyos atributos son:
  - nombre: nombre del conductor
  - licencia: un número que identifica al condcutor.
- Al crear el objeto se asigna también el conductor
- No pueden subir más pasajeros que los máximos admitidos y no pueden bajar más de los que hay.

Objetos y clases – Ejercicio (IV)

- Un artículo tiene un nombre, un proveedor y un precio.
- Y un proveedor tiene un nombre, email y teléfono.
- Se pide definir una clase (Proveedor) para implementar el objeto proveedor y otra (Articulo) para el objeto artículo.
- El objeto Articulo tiene los siguientes atributos o propiedades:
  - proveedor: un objeto proveedor
  - nombre: una cadena
  - precio: un número
- Y métodos:
  - telefono(): muestra por consola el nombre y telefono del proveedor
- Por su parte el objeto proveedor tiene como propiedades
  - nombre: cadena de texto
  - email: un email
  - teléfono: una cadena de dígitos

releevant.

Prototype en JavaScript

- Todos los objetos en JavaScript heredan propiedades y métodos de un prototipo (clase)
- Prototype entonces define la clase y no el objeto

```
function Person(first, last, age, eye) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eye;
}

Person.nationality = "English";
Person.prototype.nationality = "English";
```

releevant.

Prototype y Call

- Call nos permite llamar a una función del Prototype de una clase
- Ayudan a definir el concepto de constructor en JavaScript

```
function Product(name, price) {
    this.name = name;
    this.price = price;
}

function Food(name, price) {
    Product.call(this, name, price);
    this.category = 'food';
}

console.log(new Food('cheese', 5).name); // muestra cheese
```

releevant.

Prototype y Call - Ejercicio

- Queremos implementar la siguiente estructura:
  - Un Empleado se define con las propiedades nombre (cuyo valor por defecto es una cadena vacía), y un departamento (cuyo valor por defecto es "General").
  - Un Director está basado en Empleado. Añade la propiedad informes (cuyo valor por defecto es un array vacío).
  - Un Trabajador está basado también en Empleado. Añade la propiedad proyectos (cuyo valor por defecto es un array vacío).
  - Un Ingeniero está basado en Trabajador. Añade la propiedad máquina (cuyo valor por defecto es una cadena vacía) y sobreescribe la propiedad departamento con el valor "Ingeniería".
- Crear los objetos y casos de prueba necesarios para comprobar el correcto funcionamiento de la jerarquía

Prototype y Call – Ejercicio (II)

- *Define la siguiente jerarquía de objetos:*
  - *Objeto Persona con las propiedades nombre, edad y género. Además incorpora el método obtDetalles(), el cual mostrará las propiedades de la persona.*
  - *Objeto Estudiante, que hereda de Persona, e incluye las propiedades curso y grupo. Además incorpora el método registrar().*
  - *Objeto Profesor, que hereda de Persona, e incluye las propiedades asignatura y nivel. Además incorpora el método asignar().*
- *Crear los objetos y casos de prueba necesarios para comprobar el correcto funcionamiento de la jerarquía.*

releevant.

Tipo de datos Map

- Nos sirve para almacenar datos en pares clave-valor (Key-Value)

```
const map1 = new Map();
map1.set('a', 1);   // clave ‘a’, valor 1
map1.set('b', 2);  // clave ‘b’, valor 2
map1.set('c', 3);  // clave ‘c’, valor 3

map1.keys();     // devuelve array [‘a’, ‘b’, ‘c’]
map1.values();  // devuelve array [1, 2, 3]
map1.get(‘a’);   // devuelve 1
```

Tipo de datos Map - Ejercicio

- *Utiliza un map almacenar información sobre módulos impartidos en este Bootcamp*
- *Añade la información con posterioridad a la creación de la estructura :*
  - *Muestra cuántos módulos hay almacenados*
  - *Muestra el contenido de la estructura*
  - *Devuelve las abreviaturas de todos los módulos guardados*
  - *Devuelve el nombre completo de todos los módulos*
  - *Consulta si está el módulo “JavaScript Avanzado“*
  - *Si está, elimínalo.*

releevant.

Tipo de datos Set

- Nos sirve para almacenar una colección de valores

```
const mySet1 = new Set()

mySet1.add(1)                  // Set [ 1 ]
mySet1.add(5)                  // Set [ 1, 5 ]
mySet1.add(5)                  // Set [ 1, 5 ]
mySet1.add('some text') // Set [ 1, 5, 'some text' ]

const o = {a: 1, b: 2}
mySet1.add(o)                  // Set [1, 5, 'some text', {a: 1, b: 2}]
```

releevant.

Tipo de datos Set - Ejercicio

- A partir del ejercicio de Map
- Ahora de cada módulo se desea guardar su nombre, duración y alumnos matriculados (módulo, numAlumnos).
- Utiliza la estructura Set

releevant.

Ejercicio Extra

- Haciendo uso de la API https://dummyapi.io/docs
- Hacer paginas web que muestren:
  - Listado de usuarios
  - Incluir paginado al listado
  - Incluir nuevos usuarios
  - Actualizar usuario existente
  - Visualizar todos los datos de un usuario

JavaScript - Iterables

- Los *Iterables* son objetos que se pueden iterar
- Los objetos Iterables pueden iterarse usando for … of
- Un ejemplo:

```
let letras = [‘a’, ‘b’, ‘c’];
for (const letra of letras) {
        console.log(letra);        //’a’, ‘b’, ‘c’
}

let numeros = [1, 2, 3, 4, 5];
for (const n of numeros) {
        console.log(n);            // n = 1, n = 2, n = 3, n = 4, n = 5
}
```

releevant.

Iterables - Ejercicios

- Implementar el HTML y JavaScript para incluir dentro de un elemento <ol> las letras “hola”, el resultado debería ser:

```
<ol>
    <li>h</li>
    <li>o</li>
    <li>l</li>
    <li>a</li>
</ol>
```

Inicialmente el elemento <ol> debe estar vacío.

- Implementar el HTML y JavaScript para incluir en un elemento <p> las palabras en el array *let palabras = [‘Hola’, ‘soy’, ‘un’, ‘párrafo’]*. Inicialmente el elemento <p> debe estar vacío.

Iterables – Symbol.iterator

- Es un objeto de Iterable y define de que forma se itera el iterable
- Debe implementar un método next() que devuelve un objeto con las propiedades siguientes:
  - done: boolean // indica si ya se ha terminado de iterar la lista
  - value: any // contiene el siguiente valor si *done* es *true*

releevant.

Symbol.iterator - Ejemplo

```
let range = {
  from: 1,
  to: 5
};

range[Symbol.iterator] = function() {
return {
    current: this.from,
    last: this.to,
next() {
      if (this.current <= this.last) {
        return { done: false, value: this.current++ };
      } else {
        return { done: true };
      }
    }
  };
};

for (let num of range) {
  alert(num); // 1, then 2, 3, 4, 5
}
```

Symbol.iterator - Ejercicios

- Implementar un Symbol.iterator que solo devuelva los elementos pares de la lista.

let a = [1, 2, 3, 4, 5, 6] // debiera dejar iterar 2, 4, 6

- Implementar un Symbol.iterator que solo devuelva los elementos impares de la lista.

let a = [1, 2, 3, 4, 5, 6] // debiera dejar iterar 1, 3, 5

releevant.

ECMAScript 6

- Es el estándar oficial del lenguaje JavaScript
- TC39 es el comité que define y desarrolla la especificaciones del estándar
- ECMAScript 3 se aprobó en 1999, época de la burbuja .com
- ECMAScript 5 se aprobó en 2009, la versión anterior
- ECMAScript 6 se aprobó en 2014
- ECMAScript 7? Veamos este link
- ECMAScript 6 specs

releevant.

ES6 – Constantes y tipos de datos

- Uso de let y const
- Tipos primitivos
  - Undefined
  - Boolean
  - Number
  - String
  - BigInt
  - Symbol
  - Null
  - Object
  - Function

releevant.

ES6 – Templates Literals

- El ejemplo que todos conocemos

```
const strB1 = "Template literals";
const strB2 =  "make them simple";
console.log(`${strB1} ${strB2}.`);
```

- También podemos hacer

```
console.log(`There will be a tab
space    after this end of string.`);
console.log(`First Line
Second Line`);
```

- Pero tambien podemos hacer:

```
console.log(
`The current background color
is ${darkMode ? "#000000" : "#FFFFFF" }`
);
```

- Y esto:

```
console.log(
`The price of product is ${price} and
after 16% discount it would cost
${price-price*16*0.01}`
);
```

ES6 – Tagged templates

- Veámoslo con un ejemplo

```
function useless(strings, ...values) {
      return 'I render everything useless.';
}

let name = 'Benedict';
let occupation = 'being awesome';

let sentence = useless`Hi! I'm ${ name } and I'm busy at ${ occupation }.`;

console.log(sentence); // ‘I render everything useless.’
```

releevant.

ES6 – Tagged templates - Ejercicio

- Implementar el HTML y el JavaScript que coja el valor de dos edit text, uno con el nombre y otro con la profesión
- Cuando un usuario indica su nombre y profesión en los edit text y pulsa el botón Obtener, se debe mostrar debajo del botón el texto ‘Hola! Soy [nombre] y me dedico a [profesión]’
- Haced uso de Tagged Literals para generar el texto final
- Opcional: Los datos de nombre y profesión deben mostrarse en mayusculas

releevant.

ES6 – Parámetros por defecto

- Se puede especificar parámetros por defecto en funciones

```
function miFuncion(a, b = 10) {
    return a + b;
}

miFuncion(1); // devuelve 11
miFuncion(1, 10);    // devuelve 11
miFuncion(1, 2);  // devuelve 3
```

ES6 – Parámetros indefinidos

- Se puede especificar un array indeterminado

```
function miFuncion(a, …numeros) {
        let sum = a;
        numeros.forEach(function(n) {
                sum += n;
        });

        return sum;
}

const a = [1, 2, 3, 4, 5];
miFuncion(10, …a);                  // devuelve 25
miFuncion(10, …[1, 2, 3, 4, 5, 6]);     // devuelve 31
```

ES6 – Funciones flecha

- Una forma de escribir funciones de una manera mas legible

```
// ES5
var x = function(x, y) {
   return x * y;
}

// ES6
const x = (x, y) => x * y;
```

ES6 – Desestructuración

- Expresión de JavaScript que permite desempacar valores de arreglos o propiedades de objetos en distintas variables.
- Vamos a ver varios ejemplos

```
let a, b, rest;
[a, b] = [10, 20];
console.log(a); // muestra 10
console.log(b); // muestra 20

[a, b, ...rest] = [10, 20, 30, 40, 50];
console.log(rest);  // muestra [30, 40, 50]
```

releevant.

ES6 – Desestructuración (II)

- Hacer la prueba con ({ a, b }= { a: 10, b: 10 });

```
({ a, b } = { a: 10, b: 20 });
console.log(a);
console.log(b);
```

```
({a, b, ...rest} = {a: 10, b: 20, c: 30, d: 40});
console.log(a);
console.log(b);
console.log(rest);
```

releevant.

ES6 – Desestructuración (III)

- Otro ejemplo

```
const foo = ['one', 'two', 'three'];
const [red, yellow, green] = foo;

console.log(red);
console.log(yellow);
console.log(green);
```

releevant.

ES6 – Desestructuración (IV)

- A ver otro mas:

```
let a, b;

[a, b] = [1, 2];
console.log(a);
console.log(b);
```

releevant.

ES6 – Desestructuración (V)

- Espera que queda otro ejemplo

```
let a, b;

[a=5, b=7] = [1];
console.log(a);
console.log(b);
```

releevant.

ES6 – Desestructuración (VI)

- Y también vamos a ver con otro ejemplo:

```
let a = 1;
let b = 3;

[a, b] = [b, a];
console.log(a);
console.log(b);

const arr = [1,2,3];
[arr[2], arr[1]] = [arr[1], arr[2]];
console.log(arr);
```

ES6 – Desestructuración (VII)

- A por otro ejemplo monguer:

```
function f() {
  return [1, 2];
}

let a, b;
[a, b] = f();
console.log(a);
console.log(b);
```

releevant.

ES6 – Desestructuración (VIII)

- Un ultimo ejemplo

```
const [a, ...b] = [1, 2, 3];

console.log(a);
console.log(b);
```

releevant.

ES6 – Symbols

- Es un tipo de dato primitivo como Boolean, String o Number
- No se puede tener dos symbols iguales
- Nos sirve para asegurar que no hay ambigüedades

```
let producto = {
      nombre: “Pepe”,
      apellidos: “Contreras Carretero”
      direccion: “C/ Malaga 12, 4D”
}
producto.id = Symbol(“id”);     // valor unico

Symbol(“id) == Symbol(“id”)   // es siempre false
```

releevant.

ES6 – Promesas

- Nos da control sobre operaciones asíncronas
- Llamadas a API suelen ser asíncronas
- Tratamiento de arrays o listas largas mejor asincronas

```
let p = new Promise(function(resolve, reject){
        // hacer un trabajo largo
        if (ok) {
                resolve('Trabajo completado');
        }
        else{
                reject('ERROR , no se pudo realizar la tarea');
        }
})
```

ES6 – Promesas (II)

- Veamos un ejemplo

```
function sumaPositivosAsync(a, b) {
      let p = new Promise(function (resolve, reject) {
        if (a >= 0 && b >= 0) {
        resolve(a + b);
        } else {
           reject('Los numeros indicados no son positivos')
;
        }
        return p;
  })

   sumaPositivosAsync(10, 20)
     .then(ok)
     .catch(error);
```

```
sumaPositivosAsync(-10, -20)
     .then(ok)
     .catch(error);

  function error(err) {
     console.log('Error: ', err);
  }
  function ok(result) {
     console.log('OK: ', result);
  }

  console.log('end');
```

releevant.

ES6 – Promesas (III)

- Ejercicio: Modificar el anterior para sumar 10000 números
- Ejercicio: Ahora con 1 millón de números
- Ejercicio: En nuestra de solución Ecommerce usar Promesas para las llamadas a las URLs