JavaScript

GUÍA DE EJERCICIOS COMPLEMENTARIOS

CODERHOUSE

Sobre los ejercicios

Podemos identificar los elementos en:

Ejercicios: son propuestas de ejercitación práctica complementaria, basadas en problemáticas comunes del desarrollo de aplicaciones web. Algunas consignas son específicas, pero otras son un poco más abiertas, con la intención de familiarizarnos con formatos narrativos en la solicitud de funcionalidades a programar (como son las historias de usuario).

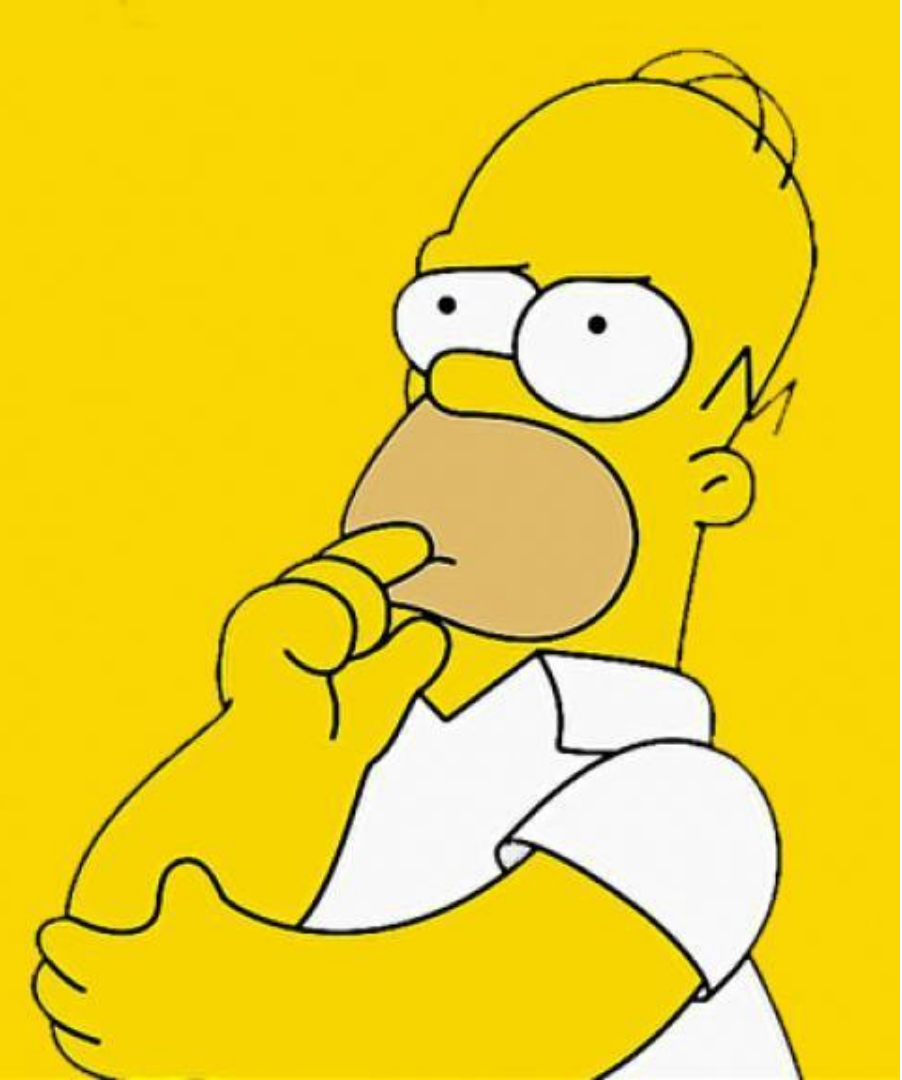
Notas: cada ejercicio presenta una nota, la cual nos brinda más información o tips para desarrollarlo (nota: modo hardcore hacerlo sin leer las notas).

Resoluciones propuestas: encontrarás las resoluciones para que las compares con tus propias soluciones. Es propuesta por que **no es perfecta**, en desarrollo de software ninguna solución es infalible, y mucho depende del contexto, por lo cual si tu resolución presenta un funcionalidad similar o mejor a la propuesta ¡felicidades, resolviste el ejercicio!.

Scripts: por cada ejercitación, contamos con un script de referencia, con el cual puedes comparar si lo que codificaste es similar a la funcionalidad esperada.

JAVASCRIPT

EJERCICIO 1



Actividad 1
Homero

Crear tres (3) variables para asignar el nombre, apellido y edad de Homero, respectivamente. Es indistinto el empleo de var o let en la declaración.

Notas actividad 1
Es posible que los valores de nombre, apellido y edad sean de otra persona a elección.

CODERHOUSE

Actividad 2
Ciudades

Crear cinco (5) variables constantes, asignando a cada una de ellas diferentes nombres de ciudades. Generalmente, dichas variables se declaran con const.

Notas actividad 2

Recordemos que el nombre de las variables es a elección, aunque es preferible escoger aquellos que permitan identificar fácilmente el significado del dato asignado a la variable

CODERHOUSE

Actividad 3
Carnet

Declarar variables para representar la información de un carnet de conducir. Luego, concatenarlas y asignar el resultado de la operación a una variable denominada carnet.

Notas actividad 3

Definir la cantidad de variables y el valor de cada una, según se prefiera. La cadena asignada a carnet debería contener al menos el nombre, la dirección, y el país de la persona dueña del carnet.

CODERHOUSE

Actividad 4 Registro de personas

Solicitar al usuario cinco (5) nombres pertenecientes a integrantes de una familia. Luego, concatenar dichas entradas y efectuar una salida por alerta.

Notas actividad 4

Usamos prompt() para solicitar datos al usuario. Es posible solicitar y asignar valores a distintas variables de forma consecutiva.

CODERHOUSE

Actividad 5
Descuentos

Solicitar al usuario uno o más precios.
Luego, efectuar un descuento en porcentajes (20%, 30%, etcétera) sobre los valores ingresados, y realizar una salida.

Notas actividad 5

Un descuento es la resta de un monto sobre el precio inicial. Buscamos calcular el porcentaje sobre dicho precio y restarlo, por ejemplo: si el precio es 100, el 20% es 20, entonces el descuento es 100-20 = 80

CODERHOUSE

Resoluciones

Solución propuesta Actividad 1

```
let nombre   = "Homero";
let apellido = "Simpsons";
let edad     =  39;
console.log(nombre);
console.log(apellido);
console.log(edad);
```

Solución propuesta Actividad 2

```
const ciudad1 = "Springfield";
const ciudad2 = "Shelbyville";
const ciudad3 = "Capital City";
const ciudad4 = "Cypress Creek";
const ciudad5 = "Ogdenville";
console.log(ciudad1);
console.log(ciudad2);
console.log(ciudad3);
console.log(ciudad4);
console.log(ciudad5);
```

Solución propuesta Actividad 3

```
let persona    = "BART SIMPSON";
let domicilio  = "742 EVERGREEN";
let pais       = "USA";
let nacimiento = "02-11-70";
const codigo   = "B47U89RE243";
let carnet = "Codigo: "+codigo+" "+
             "Nombre: "+persona+" "+
             "Pais: "+pais+" "+
             "Nacimiento: "+nacimiento+" "+
             "Domicilio: "+domicilio;
console.log(carnet);
```

Solución propuesta Actividad 4

```
let integrante1 = prompt('INGRESAR 1er INTREGANTE ');
let integrante2 = prompt('INGRESAR 2do INTREGANTE ');
let integrante3 = prompt('INGRESAR 3er INTREGANTE ');
let integrante4 = prompt('INGRESAR 4to INTREGANTE ');
let integrante5 = prompt('INGRESAR 5to INTREGANTE ');
let familia =   "1) "+integrante1+" "+
                "2) "+integrante2+" "+
                "3) "+integrante3+" "+
                "4) "+integrante4+" "+
                "5) "+integrante5;
console.log(familia);
```

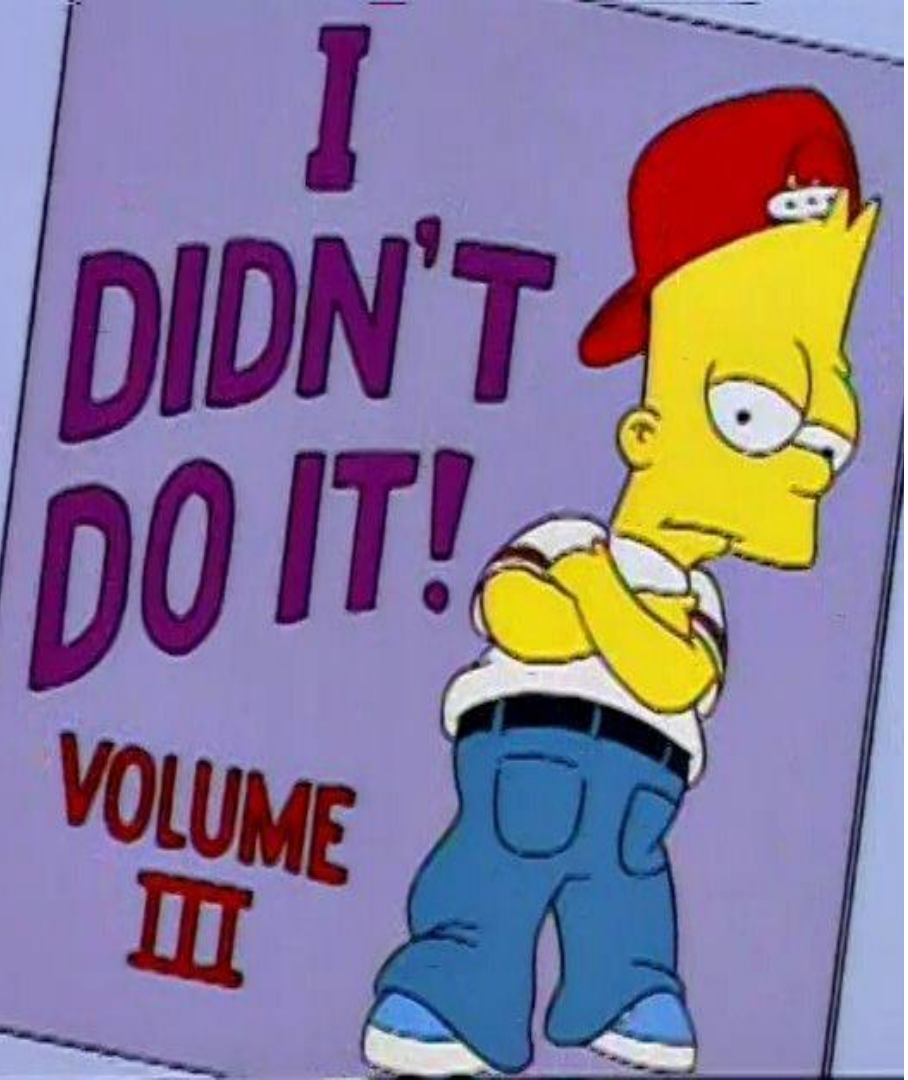
Solución propuesta Actividad 5

```
let precio = parseFloat(prompt('INGRESAR PRECIO'));
let descuento20 = precio - (precio  * 0.2);
let descuento30 = precio - (precio  * 0.3);
console.log(descuento20);
console.log(descuento30);
```

JAVASCRIPT

EJERCICIO 2

Actividades



Actividad 1
Yo no fui

Solicitar al usuario un (1) nombre.
Si el mismo es igual a otro nombre almacenado en una variable, realizar una salida por alerta con el mensaje “Fui yo”.
Caso contrario, la salida será “Yo no fui”.

Notas actividad 1
El valor del nombre a comparar es a elección. Recordemos que con la sentencia if podemos determinar qué bloque se ejecuta en el programa.

CODERHOUSE

Actividad 2
Presionar Y

Solicitar al usuario una (1) tecla.
Si se presiona “y” (minúscula), o “Y” (mayúscula), realizar una salida por alerta con el mensaje “Correcto”. Caso contrario, la salida será “Error”.

Notas actividad 2
En JS 'y' minúscula es distinto de 'Y' mayúscula. Si queremos verificar ambas comparaciones en un if, es necesario emplear el operador booleano && (and)

CODERHOUSE

Actividad 3
Escoger personaje

Solicitar al usuario un (1) número.

Si el valor está entre 1 y 4, efectuar una de las siguientes salidas según corresponda:

- ✓ “Elegiste a Homero” si es 1.
- ✓ “Elegiste a Marge” si es 2.
- ✓ “Elegiste a Bart” si es 3.
- ✓ “Elegiste a Lisa” si es 4.

Notas actividad 3

Todo valor que ingresa por prompt es un string. Si queremos verificar si es un número (obviando el tipo de dato), se recomienda siempre comparar por igualdad parcial (==).

CODERHOUSE

Actividad 4
Presupuesto

Solicitar al usuario un (1) número.

Si el valor está entre dos números, efectuar una de las siguientes salidas, según corresponda:

- ✓ “Presupuesto bajo” si está entre 0 y 1000.
- ✓ “Presupuesto medio” si está entre 1001 y 3000.
- ✓ “Presupuesto alto” si es mayor que 3000.

Notas actividad 4

Para determinar si un número está entre otros dos (intervalo), hay que usar las comparaciones menor que (<) y mayor que (>).

CODERHOUSE

Actividad 5
Vacío

Solicitar al usuario cuatro (4) productos de almacén. Si todos los elementos fueron cargados, realizar una única salida compuesta por el listado de productos. Caso contrario, la salida será “Error: Es necesario cargar todos los productos”.

Notas actividad 5

Para asegurarnos de que una variable string no esté vacía, podemos comparar (variable != ""). Luego, agrupar estas validaciones en un if, con el operador booleano && (and)

CODERHOUSE

Resoluciones

Solución propuesta Actividad 1

```
let entrada = prompt("INGRESAR UN  NOMBRE");
let nombre = "BART";
if (entrada == nombre) {
    alert("FUI YO");
} else {
    alert("YO NO FUI");
}
```

Solución propuesta Actividad 2

```
let entrada2 = prompt("INGRESAR UN  TECLA");
if ((entrada2 == "y") || (entrada2 == "Y")) {
    alert("CORRECTO");
} else {
    alert("ERROR");
}
```

Solución propuesta Actividad 3

```
let entrada3 = prompt("INGRESAR UN NUMERO");
if (entrada3 == 1) {
    alert("ELEGISTE A HOMERO");
}else if (entrada3 == 2) {
    alert("ELEGISTE A MARGE");
}else if (entrada3 == 3) {
    alert("ELEGISTE A BART");
}else if (entrada3 == 4) {
    alert("ELEGISTE A LISA");
}
else {
    alert("ERROR");
}
```

Solución propuesta Actividad 4

```
let entrada4 = parseFloat(prompt("INGRESAR UN NUMERO"));
if ((entrada4 >= 0) && (entrada4 <= 1000)) {
    alert("PRESUPUESTO BAJO");
} else if ((entrada4 >= 1001) && (entrada4 <= 3000)) {
    alert("PRESUPUESTO MEDIO");
} else if (entrada4 > 3000) {
    alert("PRESUPUESTO ALTO");
}
else {
    alert("ERROR");
}
```

Solución propuesta Actividad 5

```
let producto1 = prompt('INGRESAR 1er PRODUCTO');
let producto2 = prompt('INGRESAR 2do PRODUCTO');
let producto3 = prompt('INGRESAR 3er PRODUCTO');
let producto4 = prompt('INGRESAR 4to PRODUCTO');

if ((producto1 != "") && (producto2 != "") && (producto3 != "") && (producto4 != "")) {
    let heladera = "1) " + producto1 + " " +
        "2) " + producto2 + " " +
        "3) " + producto3 + " " +
        "4) " + producto4;
    console.log(heladera);
} else {
    console.log("ERROR: ES NECESARIO CARGAR TODOS LOS PRODUCTOS");
}
```

JAVASCRIPT

3

Actividades

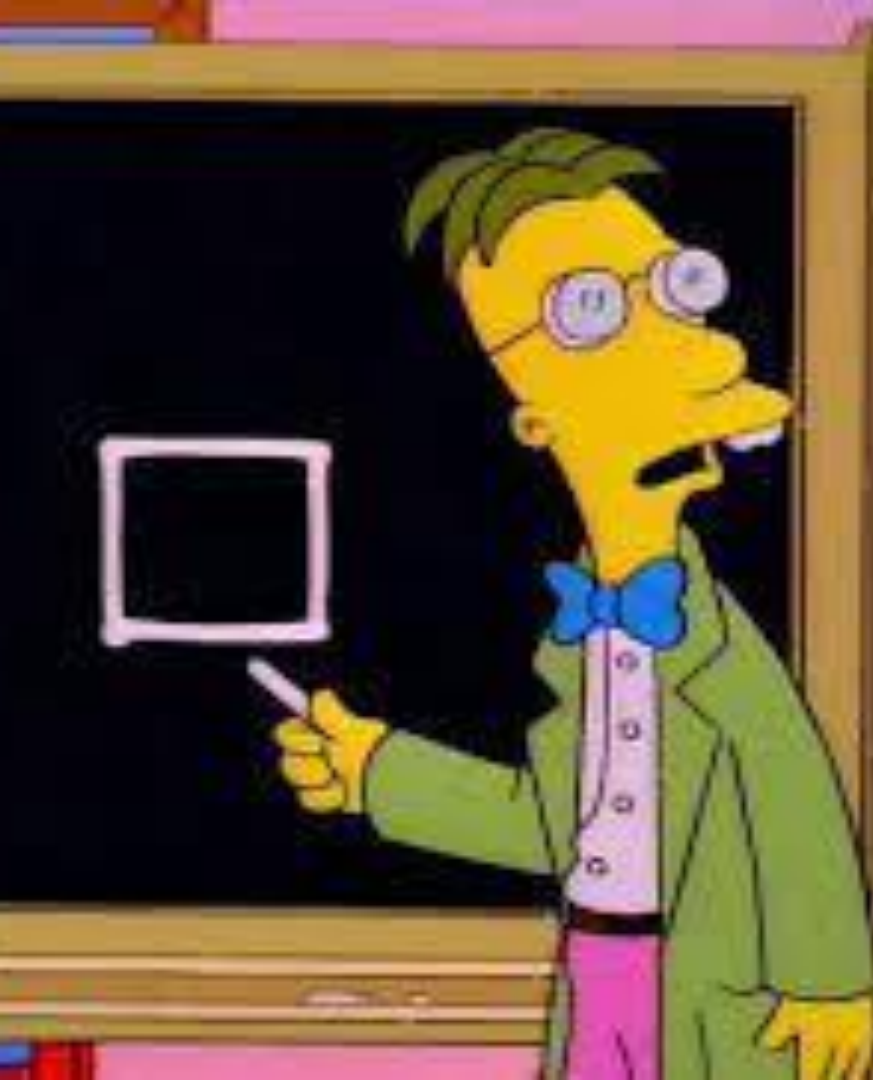
Actividad 1
El pizarrón

Solicitar al usuario un (1) un número y un (1) texto. Efectuar una salida por alerta con el mensaje ingresado por cada repetición, hasta alcanzar el valor señalado por el usuario.

Notas actividad 1

Cuando decimos “por cada repetición”, estamos señalando la necesidad de emplear ciclos. Si el ciclo está condicionado por un número de repeticiones, se usa for.

CODERHOUSE



Actividad 2
El cuadrado ordinario

Solicitar al usuario un (1) número. Emplear este valor para determinar la cantidad de repeticiones, y efectuar una salida por alerta con el mensaje “lado” en cada repetición. Pero si se alcanza un número de iteraciones mayor que cuatro (4), cancelar el ciclo.

Notas actividad 2

Es importante distinguir entre el número máximo de repeticiones de un for. y un condicional cuyo cumplimiento puede provocar la interrupción del bucle (con break).

CODERHOUSE

Actividad 3 Registro de alumnos

Solicitar al usuario la carga de diez (10) alumnos de forma consecutiva.
Luego realizar un única salida por alerta, con el listado de alumnos registrados.

Notas actividad 3

La “carga consecutiva” implica utilizar un bucle, para solicitar entradas al usuario una detrás de otra. Podemos concatenar la entrada obtenida en cada ciclo a otra variable.

CODERHOUSE

Actividad 4
El innombrable

Solicitar al usuario la carga de nombres de forma consecutiva, hasta que se ingrese “Voldemort”. Luego realizar un única salida por alerta, con todos los nombres ingresados.

Notas actividad 4

La repetición consecutiva “hasta que se ingrese un valor”, implica el uso de un bucle condicional (while o do...while), el cual se interrumpe una vez el usuario ingresa dicho valor.

CODERHOUSE

Actividad 5
Comprando productos

Solicitar al usuario un (1) número de forma consecutiva, hasta que se ingrese “ESC”. Generar una única salida compuesta por los siguientes productos, según el valor ingresado en cada ciclo:

- ✓ “Tomate” si es 1.
- ✓ “Papa” si es 2.
- ✓ “Carne” si es 3.
- ✓ “Pollo” si es 4.

Notas actividad 5

Es importante distinguir entre el valor de la entrada que asegura la repetición (entrada != “ESC”), y los posibles valores de la entrada que disparan un procesamiento (1,2,3 y 4).

CODERHOUSE

Resoluciones

Solución propuesta Actividad 1

```
let cantidad = prompt('INGRESAR CANTIDAD DE REPETICIONES');
let texto    = prompt('INGRESAR TEXTO A REPETIR');
for (let index = 0; index < cantidad; index++) {
    console.log(texto);
}
```

Solución propuesta Actividad 2

```
let lados = prompt('INGRESAR CANTIDAD DE LADOS');
for (let index = 0; index < lados; index++) {
    if (index > 3) {
        break;
    }
    alert("lado");
}
```

Solución propuesta Actividad 3

```
let alumnos = '';
for (let index = 0; index < 10; index++) {
    alumnos += prompt("INGRESAR NOMBRE DE ALUMNO")+"\n";
}
alert(alumnos);
```

Solución propuesta Actividad 4

```
let entrada    = prompt("INGRESAR NOMBRE");
let ingresados = '';
while (entrada != 'Voldemort') {
    ingresados += entrada +"\n";
    entrada     = prompt("INGRESAR NOMBRE");
}
alert(ingresados);
```

Solución propuesta Actividad 5

```
let entrada =  prompt("SELECCIONAR PRODUCTO DE 1 A 4");
while (entrada != 'ESC') {
    switch (entrada) {
        case "1":
            alert("Tomate");
            break;
        case "2":
            alert("Papa");
            break;
        case "3":
            alert("Carne");
            break;
        case "4":
            alert("Pollo");
            break;
        default:
            alert("Error");
            break;
    }
    entrada = prompt("SELECCIONAR PRODUCTO DE 1 A 4");
}
```

JAVASCRIPT

4

Actividades

Actividad 1
Entradas y salidas

Codificar tres funciones:

- ✓ Una función entrada(), la cual solicite un valor al usuario y lo retorne.
- ✓ Una función procesamiento(valor), donde se transforme la entrada.
- ✓ Una función salida(valor), la cual mostrará el resultado por alerta.

Luego, invocar las tres funciones.

Notas actividad 1
La actividad propone agrupar en funciones las instrucciones de 1) entrada. 2) procesamiento, y 3) salida. La llamada de las funciones debe coincidir con dicho orden.

CODERHOUSE

Actividad 2
Redondeo

Codificar una función que reciba un número decimal por parámetro, y lo retorne redondeado al entero más cercano. Para ello, puede emplearse Math.round.
Luego invocar la función de forma iterativa cinco (5) veces, solicitando en cada ciclo un número al usuario.

Notas actividad 2

La “llamada iterativa de la función” implica emplear un bucle. Como se solicita una cantidad determinada, se recomienda emplear for.

CODERHOUSE

Actividad 3
Impuestos

Codificar una función con la siguiente cabecera: impuesto (precio, porcentaje). En ella, se retorna un valor numérico compuesto por el precio recibido, más la suma del porcentaje ingresado. Luego invocar la función de forma iterativa cinco (5) veces, solicitando en cada ciclo un precio y porcentaje al usuario.

Notas actividad 3

Un impuesto es la suma de un monto al precio inicial. Buscamos calcular en la función el porcentaje sobre dicho precio, y sumarlo; por ejemplo: si el precio es 100, el 20% es 20, entonces el precio final es 100+20 = 120.

CODERHOUSE

Actividad 4
Cotización

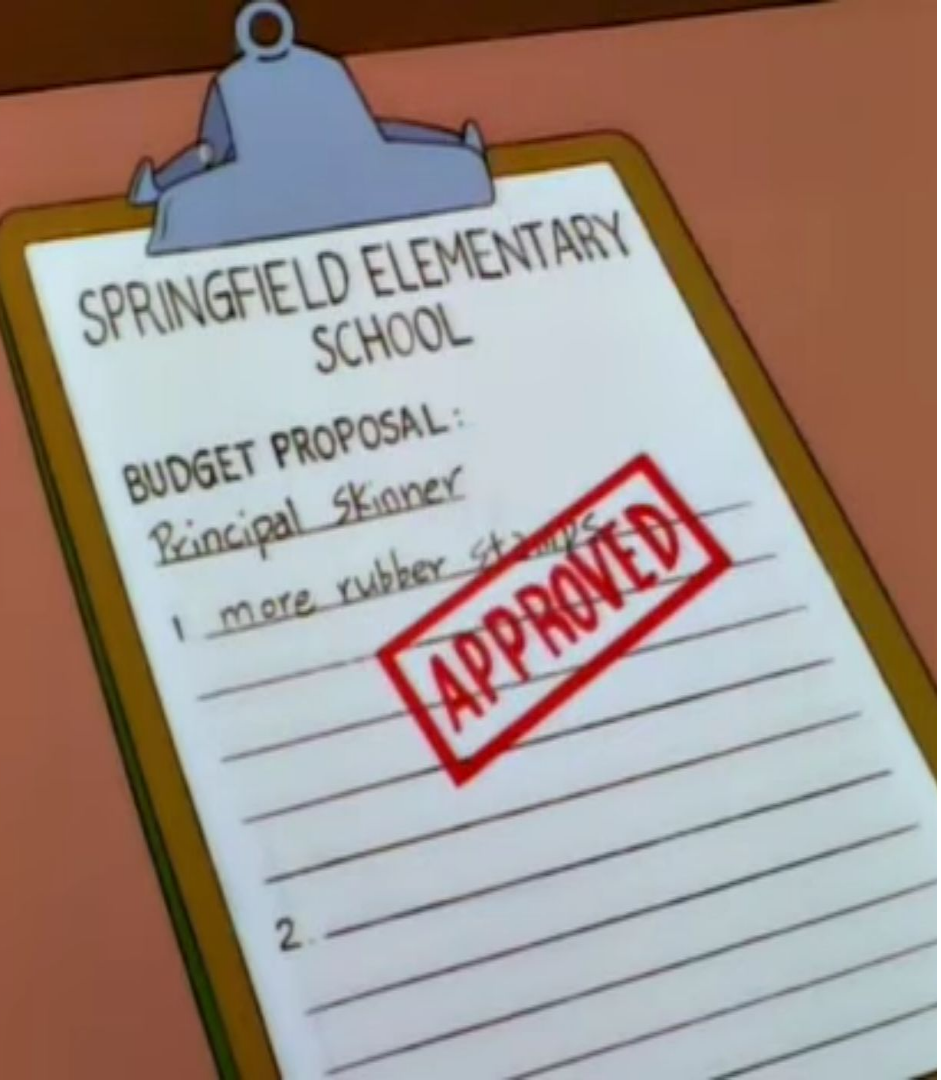
Codificar dos funciones:

- ✓ Una función cotizarDolar(pesos), la cual recibe un valor monetario en pesos argentinos, y lo retorna en su equivalente en dólares.
- ✓ Una función cotizarPesos(dolar), que recibe un valor monetario en dólares, y lo retorna en su equivalente en pesos argentinos.

Luego invocar las funciones solicitando al usuario el valor y el tipo de cotización a realizar.

Notas actividad 4
Es posible cambiar la moneda a cotizar. Lo importante es codificar dos funciones que hagan operaciones contrarias (inversas).

CODERHOUSE



Actividad 5 Validación

Codificar una función con la siguiente cabecera: validacion(cadena). En ella, se retorna un valor booleano, el cual es true si el parámetro no es un cadena vacía. Caso contrario, se retorna false. Luego invocar la función de forma iterativa, hasta que el usuario tipee “ESC”, solicitando en cada ciclo una cadena a validar.

Notas actividad 5

Es común emplear funciones para validar si un valor recibido por parámetro cumple con cierto formato. La comparación de cadena vacía es (cadena != ‘’)

CODERHOUSE

Resoluciones

Solución propuesta Actividad 1

```
function entrada(){
    return prompt("INGRESAR DATO");
}

function procesamiento(valor){
    return "LA ENTRADA ES "+valor
}

function salida(valor){
    alert(valor);
}
/SUMAR LA OTRA FORMA DE LLAMADA
salida(procesamiento(entrada()));
```

Solución propuesta Actividad 2

```
function redondeo(valor){
    return Math.round(valor)
}
for (let index = 0; index < 5; index++) {
    let entrada = prompt("INGRESAR NUMERO");
    alert(redondeo(entrada));
}
```

Solución propuesta Actividad 3

```
function impuesto(precio, porcentaje){
    return precio + ((precio * porcentaje)/100)
}
for (let index = 0; index < 5; index++) {
    let resultado = impuesto(parseFloat(prompt("INGRESAR
PRECIO"))
                        ,parseFloat(prompt("INGRESAR %")));
    alert(resultado);
}
```

CODERHOUSE

Solución propuesta Actividad 4

```
const COTIZACION_DOLAR = 150;
const cotizarDolar = (pesos) => pesos / COTIZACION_DOLAR;
const cotizarPesos = (dolar) => dolar * COTIZACION_DOLAR;
let seleccion = prompt("SELECCIONAR COTIZACION \n 1 - DOLARES A PESOS \n 2 -  PESOS A DOLAR ");
let valor = prompt("VALOR");
switch (seleccion) {
    case "1":
        alert(cotizarPesos(valor));
        break;
    case "2":
        alert(cotizarDolar(valor));
        break;
    default:
        break;
}
```

Solución propuesta Actividad 5

```
function validacion(cadena){
    return cadena != '';
}
let entrada =  prompt("INGRESAR CADENA");
while (entrada != 'ESC') {
    alert(validacion(entrada));
    entrada = prompt("INGRESAR CADENA");
}
```

JAVASCRIPT

5

Actividades

Actividad 1
La tienda

Declarar un clase Tienda que permita registrar:

- ✓ Nombre de la tienda.
- ✓ Dirección de la tienda.
- ✓ Propietario de la tienda.
- ✓ Rubro de la tienda.

Luego invocar al menos tres (3) objetos usando esta clase.

Notas actividad 1

Por cada dato a registrar en un objeto, corresponde una propiedad. Recordemos que la invocación del objeto es instanciarlo usando new y el constructor.

CODERHOUSE

Actividad 2 Registro de tiendas

Solicitar al usuario el registro de cinco (5) tiendas.
Emplear la clase Tienda de la actividad 1, para instanciar los objetos en función de las entradas capturadas. Generar una única salida compuesta por la información de los objetos instanciados.

Notas actividad 2
Es posible instanciar objetos de forma local en un bloque, y los valores pasados por parámetro al constructor pueden ser capturados desde una llamada prompt.

CODERHOUSE

Actividad 3
Abierto y cerrado

Declarar un método para la clase Tienda con la siguiente cabecera estaAbierto(hora). Este retorna true si la hora enviada está entre las 08 y 12, o entre las 15 y 19. Caso contrario, se retorna false.
Luego invocar al menos un (1) objeto usando esta clase, y solicitar al usuario tres (3) horas. Informar por alerta si la tienda está abierta, en función de la hora ingresada.

Notas actividad 3
Considerando que por prompt sólo podemos ingresar cadenas, se sugiere asumir que se trabaja sólo con horas en punto, de 0 a 22. El if debe verificar dos intervalos válidos.

CODERHOUSE

Actividad 4
Validar propietario

Declarar un método para la clase Tienda con la siguiente cabecera esPropietario(nombre). Se retorna true si el nombre enviado corresponde al propietario de la tienda.Caso contrario, se retorna false.

Luego, invocar al menos tres(3) objetos usando esta clase y solicitar al usuario cinco(5) nombres. Informar por alerta si los nombres pertenecen a algún dueño de tienda.

Notas actividad 4

Es común emplear métodos de clase para validar si un valor recibido por parámetro es igual al valor de una propiedad del objeto. Se llama el método por cada objeto creado.

CODERHOUSE

Actividad 5
El cliente

Declarar una clase **Cliente** que permita registrar:

- ✓ Registrar nombre, presupuesto, si tiene tarjeta de descuento, y número de teléfono del cliente.
- ✓ Un método transferirDinero(valor), que permita obtener y restar cierta cantidad del presupuesto, siempre que este parámetro sea menor al valor enviado, y mayor que cero.

Luego invocar al menos tres (3) objetos usando esta clase, y solicitar al usuario cinco (5) números. Informar por alerta si cada uno de los clientes cuenta con el presupuesto suficiente para realizar una transferencia de igual monto al ingresado.

Notas actividad 5

Es posible verificar en un mètòdo, usando un condicional, si la propiedad cumple con cierto valor antes de modificarla. Recordemos que una función retorna un solo valor.

CODERHOUSE

Resoluciones

Solución propuesta Actividad 1

```
class Tienda{
    constructor(nombre, direccion, propietario, rubro){
        this.nombre = nombre;
        this.direccion = direccion;
        this.propietario = propietario;
        this.rubro = rubro;
    }
}

const tienda1 = new Tienda("Kwik-E-Mart", "Park 123", "Apu","Almacen");
const tienda2 = new Tienda("Edna's Edibles", "PEREZ 323", "Edna Krabappel","Panaderia");
const tienda3 = new Tienda("Springfield Mall", "Hall 231", "Mr. Burns","Shopping");
console.log(tienda1);
console.log(tienda2);
console.log(tienda3);
```

Solución propuesta Actividad 2

```
let ingresados = '';
for (let index = 0; index < 5; index++) {
    let tienda =  new Tienda(prompt("NOMBRE"),
                             prompt("DIRECCION"),
                             prompt("PROPIETARIO"),
                             prompt("RUBRO"));

    ingresados += "Tienda: "+tienda.nombre+" "+
              "Direccion: "+tienda.direccion+" "+
              "Propitario: "+tienda.propietario+" "+
              "rubro: "+tienda.rubro+"\n";
}
alert(ingresados);
```

```
class Tienda{
    constructor(nombre, direccion, propietario, rubro){
        this.nombre = nombre;
        this.direccion = direccion;
        this.propietario = propietario;
        this.rubro = rubro;
    }
    estaAbierto(hora){
        if (((hora  >= 8) && (hora  <= 12))||((hora  >= 15) && (hora  <= 19))) {
            return true;
        }
        return false;
    }
}
const tienda4 = new Tienda("33 cents Store", "Cheap 231", "Barato SRL","Ropa");
for (let index = 0; index < 3 ; index++) {
    let entrada = parseInt(prompt("INGRESAR HORA EN PUNTO"));
    if(tienda4.estaAbierto(entrada)){
        alert("LA TIENDA ESTA ABIERTA A LAS "+entrada)
    }else{
        alert("LA TIENDA ESTA CERRADA A LAS "+entrada)
    }
}
```

Solución propuesta Actividad 3

CODERHOUSE

```
class Tienda{
    constructor(nombre, direccion, propietario, rubro){
        this.nombre = nombre;
        this.direccion = direccion;
        this.propietario = propietario;
        this.rubro = rubro;
    }
      esPropietario(nombre){
        return this.propietario == nombre;
    }
}
const tienda1 = new Tienda("Kwik-E-Mart", "Park 123", "Apu", "Almacen");
const tienda2 = new Tienda("Edna's Edibles", "PEREZ 323", "Edna Krabappel", "Panaderia");
const tienda3 = new Tienda("Springfield Mall", "Hall 231", "Mr. Burns", "Shopping");
for (let index = 0; index < 5; index++) {
    let entrada = prompt("INGRESAR NOMBRE DE PROPITARIO");
    if (tienda1.esPropietario(entrada)){
        alert(entrada + " ES PROPIETARIO DE LA TIENDA " + tienda1.nombre);
    }
    if (tienda2.esPropietario(entrada)){
        alert(entrada + " ES PROPIETARIO DE LA TIENDA " + tienda2.nombre);
    }
    if (tienda3.esPropietario(entrada)){
        alert(entrada + " ES PROPIETARIO DE LA TIENDA " + tienda3.nombre);
    }
}
```

Solución propuesta Actividad 4

CODERHOUSE

```
class Cliente{
    constructor(nombre,presupuesto,tarjeta,telefono){
        this.nombre = nombre;
        this.presupuesto = parseFloat(presupuesto);
        this.tarjeta = tarjeta;
        this.telefono = telefono;
    }

    transferirDinero(valor){
        if((this.presupuesto > 0)&&(valor < this.presupuesto)){
            this.presupuesto -= valor;
            return valor
        }else{
            return 0;
        }
    }
}
const cliente1 = new Cliente("Homero", 2000, true, "1234560");
const cliente2 = new Cliente("Carlos", "1000", false, "21234560");
const cliente3 = new Cliente("Barny", "50", false, "231234560");
for (let index = 0; index < 5; index++) {
    let entrada = parseFloat(prompt("INGRESAR MONTO"));
    if (cliente1.transferirDinero(entrada)){
        alert("EL CLIENTE " + cliente1.nombre+ " TE PUEDE PAGAR "+entrada);
    }
    if (cliente2.transferirDinero(entrada)){
        alert("EL CLIENTE " + cliente2.nombre+ " TE PUEDE PAGAR "+entrada);
    }
    if (cliente3.transferirDinero(entrada)){
        alert("EL CLIENTE " + cliente3.nombre+ " TE PUEDE PAGAR "+entrada);
    }
}
```

Solución propuesta Actividad 5

CODERHOUSE

JAVASCRIPT

6

Actividades

Actividad 1
El equipo

Declarar un array de cadenas, compuesto por los cuatro (4) nombres pertenecientes a los integrantes de un equipo. Luego recorrer el array, e informar por alerta el nombre de cada jugador, así como la posición que ocupa en la colección.

Notas actividad 1

Cuando recorremos un array, empleamos un bucle (preferentemente un for o for...of) para acceder a los elementos de la colección uno a uno.

CODERHOUSE

Actividad 2
Carga el equipo

Declarar un array vacío, y cargarlo de forma dinámica solicitando al usuario nombres de forma consecutiva, hasta que se ingrese “ESC”. Luego recorrer el array, e informar por alerta el nombre de cada jugador, así como la posición que ocupa en la colección.

Notas actividad 2

Usando while y prompt podemos cargar una colección de forma dinámica. Es decir, agregar al array en cada iteración la entrada capturada, usando el método push

CODERHOUSE

Actividad 3
Jugadores

Declarar una clase Jugador que permita registrar nombre, número de camiseta, edad, y si está lesionado. Luego instanciar al menos cinco (5) objetos usando esta clase, y asociarlos a un array.

Notas actividad 3

El método push puede recibir un objeto por parámetro, el cual se asociará al array empleado. Es posible agregar más propiedades a la clase jugador.

CODERHOUSE

Actividad 4
Buscar jugador

Codificar una función con la siguiente cabecera: buscarJugador(equipo, jugador). En ella, se recibe por parámetro un array de jugadores (objetos instanciados empleando la clase de la actividad 3), y el nombre de un jugador. La función retorna el jugador que coincide con el nombre.
Realizar al menos tres (3) búsquedas solicitando el nombre al usuario, e informar sobre el resultado de cada búsqueda.

Notas actividad 4
El método find permite encontrar un elemento en el array. Como estamos trabajando con array de objetos, usamos la notación punto para verificar el valor de las propiedades.

CODERHOUSE

Actividad 5
Filtrar jugadores

Codificar una función con la siguiente cabecera: filtroJugadores(equipo, edad). En ella, se recibe un array de jugadores (objetos instanciados empleando la clase de la actividad 3), y una edad. La función retorna los jugadores cuya edad coincide con el segundo parámetro.
Realizar al menos cinco (5) filtros solicitando la edad al usuario, e informar sobre el resultado de los jugadores filtrados.

Notas actividad 5
El método filter obtiene un array con los elementos que cumplen cierta comprobación. Como obtenemos un array al llamar la función creada, debemos recorrer el resultado.

CODERHOUSE

Resoluciones

Solución propuesta Actividad 1

```
const equipo = ['HOMERO', 'APU', 'MOE', 'MR. BURNS'];
for (let index = 0; index < equipo.length; index++) {
    alert("POSICION " + index + " JUGADOR " + equipo[index]);
}
```

Solución propuesta Actividad 2

```
let entrada = prompt("INGRESAR JUGADOR");
const equipo = [];
while (entrada != 'ESC') {
    equipo.push(entrada);
    entrada = prompt("INGRESAR JUGADOR");
}
for (let index = 0; index < equipo.length; index++) {
    alert("POSICION " + index + " JUGADOR " + equipo[index]);
}
```

Solución propuesta Actividad 3

```
class Jugador {
    constructor(nombre, camiseta, edad, lesionado) {
        this.nombre = nombre;
        this.camiseta = camiseta;
        this.edad = edad;
        this.lesionado = lesionado;
    }
}
const jugadores = [];
jugadores.push(new Jugador("BART", 15 , 11, false));
jugadores.push(new Jugador("NELSON", 2 , 18, false));
jugadores.push(new Jugador("MILHOUSE", 68 , 12, true));
jugadores.push(new Jugador("MARTIN", 0 , 11, false));
jugadores.push(new Jugador("ROD", 98 , 12, false));
console.log(jugadores);
```

Solución propuesta Actividad 4

```
function buscarJugador(equipo, jugador){
    return equipo.find(objeto => objeto.nombre === jugador.toUpperCase());
}
for (let index = 0; index < 3; index++) {
    let busqueda = buscarJugador(jugadores, prompt('INGRESAR NOMBRE DE JUGADOR'));
    if(busqueda != undefined){
        alert('JUGADOR '+busqueda.nombre+' CAMISETA '+busqueda.camiseta+' EDAD
'+busqueda.edad);
    }else{
        alert('NO EXISTE JUGADOR CON ESE NOMBRE');
    }
}
```

CODERHOUSE

Solución propuesta Actividad 5

```
function filtroJugadores(equipo, edad) {
    return equipo.filter(objeto => objeto.edad == parseInt(edad));
}
function listaJugadores(jugadores) {
    let lista = '';
    for (const jugador of jugadores) {
       lista += 'JUGADOR ' + jugador.nombre + ' CAMISETA ' + jugador.camiseta + ' EDAD ' + jugador.edad +
'\n'
    }
    return lista;
}
for (let index = 0; index < 5; index++) {
    let filtro = filtroJugadores(jugadores, prompt('INGRESAR NOMBRE DE JUGADOR'));
    if (filtro.length > 0) {
        alert(listaJugadores(filtro));
    } else {
        alert('NO EXISTE JUGADORES CON ESA EDAD');
    }
```

JAVASCRIPT

7

Actividades

Actividad 1
El visitante

Si no existe un valor con la clave “nombre” en el localStorage, solicitar un nombre al usuario, y almacenarlo con dicha clave. Si existe, mostrar el nombre obtenido desde el storage con una alerta.

Notas actividad 1
Con localStorage almacenamos datos que pueden ser útiles en el próximo uso de la aplicación. Recordemos que al refrescar la página, el script se vuelve a interpretar.

CODERHOUSE

Actividad 2
Almacenar el menú

Si no existe un valor con la clave “comidas” en el localStorage, crear un array vacío y cargarlo de forma dinámica, solicitando al usuario nombres de comida de forma consecutiva, hasta cinco (5) veces. Luego almacenarlo con dicha clave. Si existe, recorrer el array, e informar por alerta el menú cargado.

Notas actividad 2

Al almacenar un array en storage con setItem, se guarda como cadena. Por ende, es necesario usar split(“,”) al hacer getItem, para obtener lo almacenado en formato array.

CODERHOUSE

Actividad 3 Almacenar hamburguesa

Declarar una clase Hamburguesa que permita registrar nombre de la hamburguesa, el precio, los ingredientes, y el número de combo.
Luego instanciar al menos cinco (5) objetos usando esta clase, asociarlos a un array, y almacenarlos con la clave 'hamburguesas' en el localStorage.

Notas actividad 3
Para almacenar un array de objeto en el storage, es necesario emplear el método stringify. La propiedad "ingredientes" puede ser un array de ingredientes.

CODERHOUSE

Actividad 4
Comprar hamburguesa

Obtener desde el localStorage el array de objetos almacenado durante la actividad 3, con la clave ‘hamburguesas’. Recorrer el array generando un listado con el detalle de cada hamburguesa, asociando un número de selección en función de la posición.
Por último, capturar una entrada con la selección del usuario, confirmando la información del producto escogido.

Notas actividad 4
Para obtener un array de objeto desde el storage, es necesario emplear el método parse. Se recomienda emplear un for con índice para determinar el número de selección.

CODERHOUSE

Actividad 5
Aumentar precios

Obtener desde el localStorage el array de objetos almacenado durante la actividad 3, con la clave 'hamburguesas'. Recorrer el array , aumentando en un 30% el precio de cada producto. Por último, volver a almacenar el array modificado en el localStorage con la misma clave.

Notas actividad 5

Para actualizar algo almacenado en storage, se sobrescribe el valor usando la misma clave. Se recomienda añadir un método de clase para aumentar cierto porcentaje todas las hamburguesas.

CODERHOUSE

Resoluciones

Solución propuesta Actividad 1

```
let usuario = localStorage.getItem('nombre');
if (usuario == null) {
    localStorage.setItem('nombre', prompt('INGRESAR NOMBRE'));
} else {
    alert('EL NOMBRE ES ' + usuario);
}
```

CODERHOUSE

Solución propuesta Actividad 2

```
let comidas = localStorage.getItem('comidas');
if (comidas == null) {
    const menu = [];
    for (let index = 0; index < 5; index++) {
        menu.push(prompt('INGRESAR COMIDA'))
    }
    localStorage.setItem('comidas', menu);
} else {
    let menu = '';
    const arrayComidas = comidas.split(',');
    for (const nombreComida of arrayComidas) {
        menu += nombreComida + '\n';
    }
    alert(menu);
}
```

Solución propuesta Actividad 3

```
class Hamburgesa {
    constructor(nombre, precio, ingredientes, combo) {
        this.nombre = nombre;
        this.precio = parseFloat(precio);
        this.ingredientes = ingredientes;
        this.combo = parseInt(combo);
    }
}
const hamburgesas = [];
hamburgesas.push(new Hamburgesa("Krusty Burger", 150, ['Carne', 'Queso'], 1));
hamburgesas.push(new Hamburgesa("Krusty Doble", 250, ['Carne', 'Queso', 'Panceta'], 0));
hamburgesas.push(new Hamburgesa("Krusty Pollo", 150, ['Pollo', 'Queso'], 2));
hamburgesas.push(new Hamburgesa("Super Krusty", 150, ['Carne', 'Queso', 'huevo'], 5));
hamburgesas.push(new Hamburgesa("Krusty Vegan", 150, ['Espinaca', 'Soja'], 7));
localStorage.setItem('hamburgesas', JSON.stringify(hamburgesas));
```

Solución propuesta Actividad 4

```
let almacenadas = localStorage.getItem('hamburgesas');
if (almacenadas != null) {
    let array = JSON.parse(almacenadas);
    let salida = 'SELECCIONAR HAMBURGUESA \n';
    for (let index = 0; index < array.length; index++) {
       salida += index + ' -> ' + array[index].nombre + ' ' + array[index].ingredientes + ' $ ' + array[index].precio + '\n';
    }
    alert(salida);
    let eleccion = parseInt(prompt('INGRESAR HAMBURGUESA '));
    if ((eleccion >= 0) && (eleccion < array.length)) {
        alert("HAMBURGUESA SELECCIONADA " + array[eleccion].nombre)
    } else {
        alert("ERROR DE SELECCION ");
    }
}
```

Solución propuesta Actividad 5

```
let guardadas = localStorage.getItem('hamburgesas');
if (guardadas != null) {
    let array = JSON.parse(guardadas);
    array.forEach(hamburgesa => { hamburgesa.precio += (hamburgesa.precio * 0.3) });
    localStorage.setItem('hamburgesas', JSON.stringify(array));
}
```

JAVASCRIPT

8

Actividades

Actividad 1
Bienvenido a...

Declarar un array con nombres de ciudades y recorrerlo. Por cada ciudad, crear una etiqueta h2 que contenga el nombre, y agregarla al body.

Notas actividad 1

Se recomienda recorrer el array con for...of. En cada repetición se crea un elemento con createElement, se modifica su innerHTML, y se lo agrega con append.

CODERHOUSE

Actividad 2
La lista

Solicitar al usuario cinco (5) tareas de forma iterativa, cargandolas en un array. Crear una etiqueta ul, y concatenar una etiqueta li a un template de caracteres por cada tarea ingresada, asignando el resultado al interior de la etiqueta ul. Por último, agregar la lista al body.

Notas actividad 2

Si tenemos que crear una estructura html más compleja, podemos crear el nodo padre, y escribir de forma explícita el contenido del innerHTML.

CODERHOUSE

Actividad 3
Elegir destino

Declarar un array de países, y crear una etiqueta select. Por cada país, concatenar una etiqueta option a una template de caracteres, asignando el resultado al interior de la etiqueta padre. El value de cada opción es la posición de la ciudad en la colección. Por último, agregar el select al body.

Notas actividad 3

Un select es un elemento dedicado a elecciones de usuario, Aún no podemos detectar “qué selecciona el usuario”, pero al generar las opciones desde un array construimos una lista que varía en función del contenido del array.

CODERHOUSE

Actividad 4
La banda

Declarar una clase Cantante, instanciando al menos cinco (5) objetos con ella, asignándoles a un array. Luego, por cada cantante crear una etiqueta div, cuya estructura interna detalle la información del objeto, agregando cada contenedor al body.

Notas actividad 4

Se recomienda iterar el array de objetos con for...of, y por cada elemento crear un div. Usamos plantillas literales para incluir la información del objeto, y asignarla al innerHTML del contenedor.

CODERHOUSE

Actividad 5
Botón contratar

Usando de base la resolución de la actividad 4, añadir un ‘id’ como propiedad a la clase Cantante, y modificar las instancias, definiendo un identificador único por cada cantante.
Luego, por cada div generado, incluir un botón cuyo id de la etiqueta tenga el valor asociado a la propiedad ‘id’ del objeto, y el texto de cada botón sea ‘Contratar”.

Notas actividad 5
Podemos incluir botones con ids diferentes cuando generamos una estructura html dinámica, con la intención de que cuando “el usuario presione el botón” sea posible distinguir qué objeto seleccionó.

CODERHOUSE

Resoluciones

Solución propuesta Actividad 1

```
const ciudades = [  "Springfield",
                    "Shelbyville",
                    "Capital City",
                    "Cypress Creek",
                    "Ogdenville"];
for (const ciudad of ciudades) {
    let h2 = document.createElement('h2');
    h2.innerHTML = ciudad;
    document.body.appendChild(h2);
}
```

CODERHOUSE

Solución propuesta Actividad 2

```
const tareas = []
for (let index = 0; index < 5; index++) {
   tareas.push(prompt('INGRESAR TAREA'));
}
let ul = document.createElement("ul");
let inner = '';
for (const tarea of tareas) {
    inner+= `<li>${tarea}</li>`;
}
ul.innerHTML = inner;
document.body.appendChild(ul);
```

Solución propuesta Actividad 3

```
const paises = ['ARGENTINA', 'URUGUAY', 'BRASIL', 'CHILE'];
let select = document.createElement("select");
for (let index = 0; index < paises.length; index++) {
    select.innerHTML +=  `<option value='${index}'>${paises[index]}</option>`;
}
document.body.appendChild(select);
```

CODERHOUSE

```
class Cantante{
    constructor(nombre, edad, pais){
        this.nombre = nombre;
        this.edad = edad;
        this.pais = pais;
    }
}
const cantantes = [];
cantantes.push(new Cantante('HOMERO',39,'USA'));
cantantes.push(new Cantante('BARNEY',40,'USA'));
cantantes.push(new Cantante('APU',42,'INDIA'));
cantantes.push(new Cantante('SKINNER',38,'USA'));
for (const cantante of cantantes) {
    let div = document.createElement("div");
    div.innerHTML = `<h2>${cantante.nombre}</h2>
                    <p> ${cantante.edad} / ${cantante.pais}</p>
                    <hr>` ;
    document.body.appendChild(div);
}
```

Solución propuesta Actividad 4

CODERHOUSE

```
class Cantante{
    constructor(id,nombre, edad, pais){
        this.id = id;
        this.nombre = nombre;
        this.edad = edad;
        this.pais = pais;
    }
}
const cantantes = [];
cantantes.push(new Cantante(1,'HOMERO',39,'USA'));
cantantes.push(new Cantante(2,'BARNEY',40,'USA'));
cantantes.push(new Cantante(3,'APU',42,'INDIA'));
cantantes.push(new Cantante(4,'SKINNER',38,'USA'));

for (const cantante of cantantes) {
    let div = document.createElement("div");
    div.innerHTML = `<h2>${cantante.nombre}</h2>
                    <p> ${cantante.edad} / ${cantante.pais}</p>
                    <button
id='${cantante.id}'>Contratar</button>
                    <hr>` ;
    document.body.appendChild(div);
}
```

Solución propuesta Actividad 5

CODERHOUSE

JAVASCRIPT

9

Actividades

Actividad 1
Botón del núcleo

Incluir un botón con el ID ‘btnNucleo’ manipulando el DOM. Luego, escuchar el evento click sobre el botón generado, asegurando que la función manejadora dispare una salida a elección en el HTML, cuando se presione el elemento.

Notas actividad 1

Manipular el DOM implica modificar y acceder a elementos del DOM, o crear nuevos con createElement. El evento puede asociarse con addEventListener, o la propiedad onclick.

CODERHOUSE

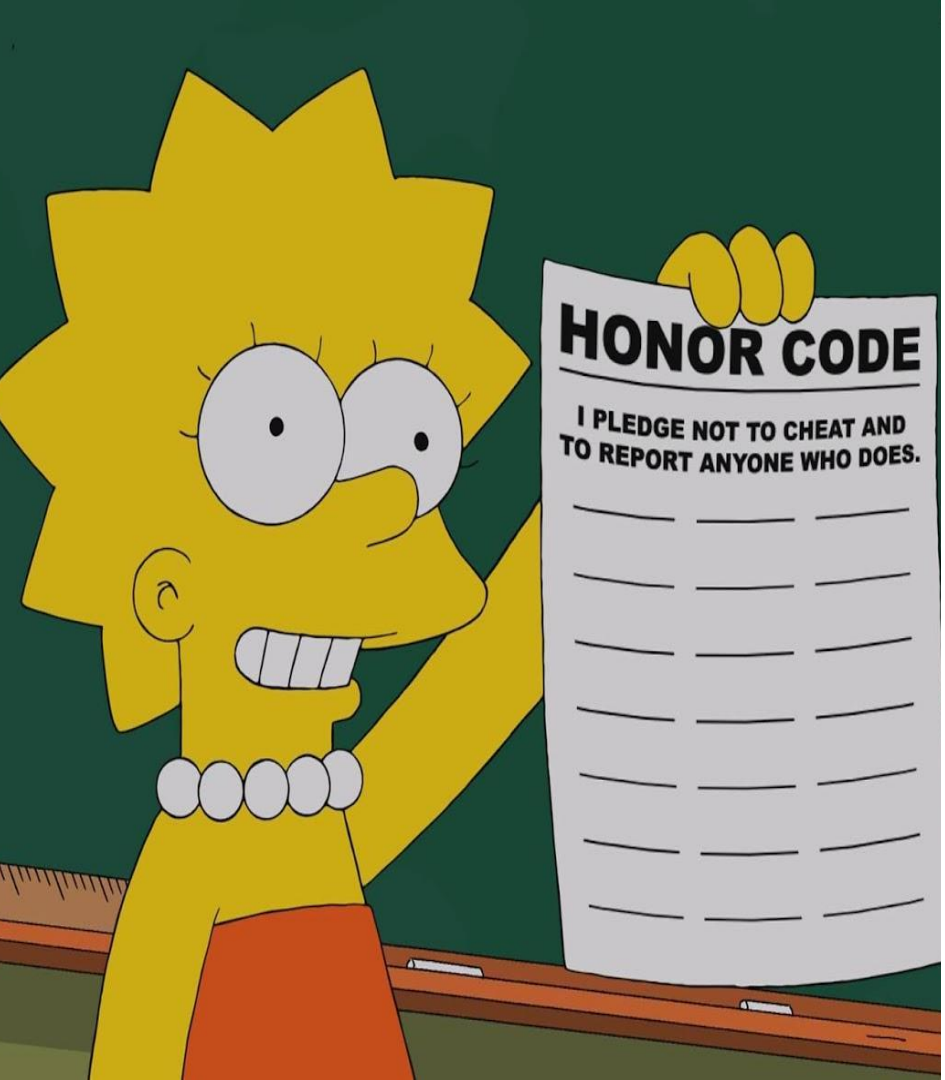
Actividad 2
Elegir asignatura

Declarar un array con nombres de asignatura, y generar un select, donde los values de las etiquetas option sean la posición de la asignatura en la colección.
Luego, escuchar el evento change sobre el select, asegurando que la función manejadora dispare una salida en el HTML, especificando el elemento escogido por el usuario.

Notas actividad 2

Generar el select implica crear toda la estructura e incluirla en la página actual. El evento change se dispara al detectar un cambio en el value del input, el cual podemos obtener a través de la información del evento, para identificar en el array la selección, ya que coincide con la posición del dato en la colección.

CODERHOUSE



Actividad 3
Alta de estudiante

Crear un formulario que permita ingresar la información de un estudiante.
Luego, escuchar el evento submit sobre el form, capturando las entradas, e invocar un objeto Estudiante usando los valores ingresados, y asociar la instancia a un array de estudiantes.
Por último, disparar una salida en el HTML, a modo de confirmación de los datos registrados.

Notas actividad 3
Se reconoce como 'alta' al proceso de registro de información en un sistema. Recordemos que en el evento submit empleamos preventDefault, para evitar el reinicio de la página.

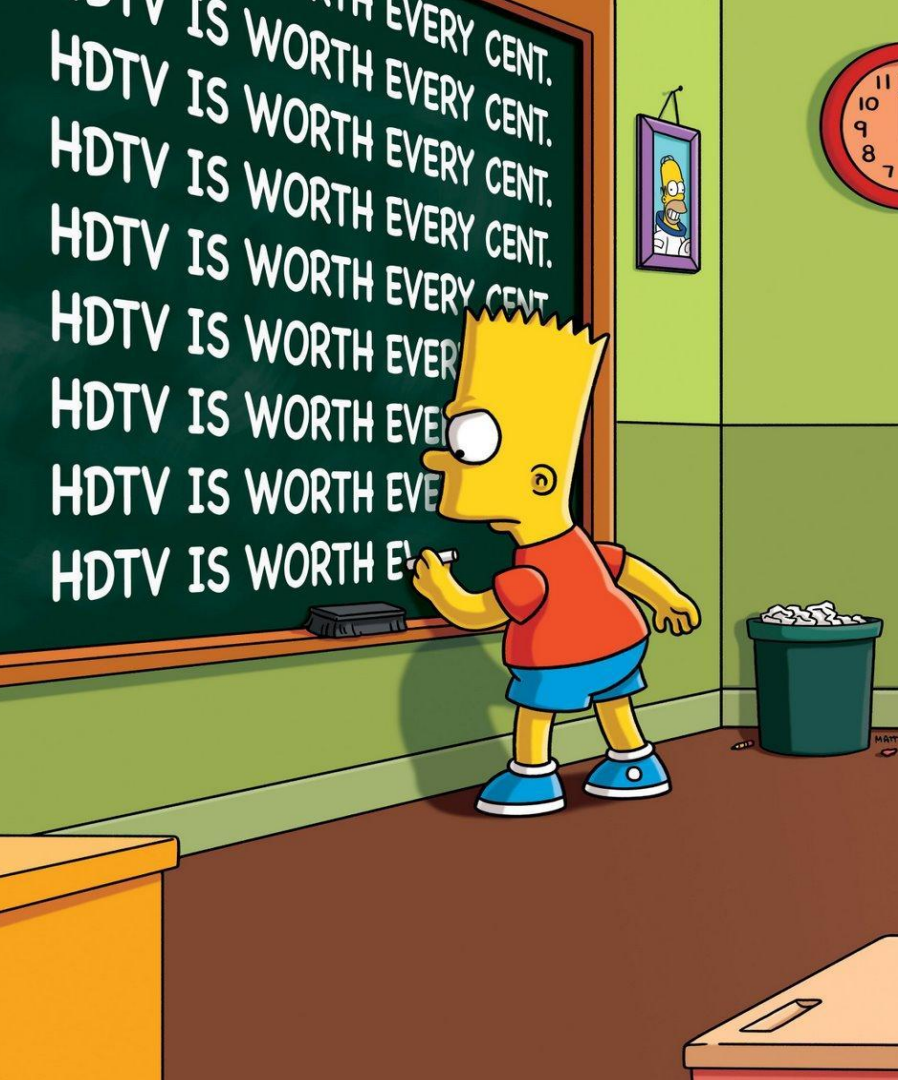
CODERHOUSE

Actividad 4
Listar estudiantes

Usando el array de estudiantes creado en la actividad 3, iterar la colección creando una etiqueta div, cuya estructura interna detalle la información del estudiante, y contenga un botón 'Seleccionar', agregando cada estructura al body.
Luego, detectar el click sobre el botón, especificando al usuario los datos del alumno seleccionado.

Notas actividad 4
Cuando queremos asociar eventos a elementos generados de forma dinámica, es recomendable realizar esta operación una vez concluida la inserción de los nuevos elementos al DOM.

CODERHOUSE



Actividad 5 Suspender estudiante

Usando de base la interfaz generada en la actividad 4, añadir un nuevo botòn ‘Suspender’ por cada estudiante.
Luego, detectar el click sobre el botón, asegurando que la función manejadora identifique el objeto estudiante seleccionado, y llame a un método suspender(), disparando una salida en el HTML confirmando la suspensión del estudiante.

Notas actividad 5

Es necesario agregar el método de clase suspender a la clase Estudiante, el cual puede modificar el valor de una propiedad booleana del objeto para determinar la suspensión.

CODERHOUSE

Resoluciones

Solución propuesta Actividad 1

```
//CREANDO EL BOTÓN DESDE JS
const btnNucleo = document.createElement("button");
//ASIGNAR ID AL BOTÓN
btnNucleo.id = 'btnNucleo';
//ASIGNAR EL INTERIOR DEL BOTÓN
btnNucleo.innerHTML = 'BOTON DEL NUCLEO';
//ESCUCHAR EL EVENTO CLICK
btnNucleo.addEventListener('click', function () {
    const h3 = document.createElement('h3');
    h3.innerHTML = 'HOLA CODER';
    document.body.appendChild(h3);
})
//AGREGAR EL BOTON AL DOM
document.body.appendChild(btnNucleo);
```

Solución propuesta Actividad 2

```
const materias = ['MATEMATICA', 'LENGUA', 'PROGRAMACION', 'HISTORIA'];
let select = document.createElement("select");
for (let index = 0; index < materias.length; index++) {
    select.innerHTML +=  `<option value='${index}'>${materias[index]}</option>`;
}
//ESCUCHAR EL EVENTO CHANGE
select.addEventListener('change', function (e) {
    const h3 = document.createElement('h3');
    h3.innerHTML = materias[e.target.value];
    document.body.appendChild(h3);
})
document.body.appendChild(select);
```

CODERHOUSE

Solución propuesta Actividad 3

```
const estudiantes = []
class Estudiante {
    constructor(literal) {
        this.id = estudiantes.length;
        this.nombre = literal.nombre;
        this.edad = literal.edad;
        this.curso = literal.curso;
     }
}
const formulario = document.createElement("form");
formulario.innerHTML = `<input type="text">
                        <input type="number">
                        <input type="text">
                        <input type="submit">;
formulario.onsubmit = (e) => {
    e.preventDefault();
    const inputs = e.target.children;
    estudiantes.push(new Estudiante({ nombre: inputs[0].value, edad: inputs[1].value, curso: inputs[2].value }));
    document.body.append(' ALUMNO REGISTRADO ');
}
```

Solución propuesta Actividad 4

```
formulario.onsubmit = (e) => {
    e.preventDefault();
    const inputs = e.target.children;
    estudiantes.push(new Estudiante({ nombre: inputs[0].value, edad: inputs[1].value, curso: inputs[2].value }));
    mostrarEstudiantes(estudiantes);
    const btnEstudiantes = document.getElementsByClassName('btnEstudiante');
    for (const boton of btnEstudiantes) {
        boton.onclick = (e) => {
            const seleccionado = estudiantes.find(obj => obj.id == e.target.id);
            let notificacion = document.createElement("h6");
            notificacion.innerHTML= `Nombre ${seleccionado.nombre} -  Edad ${seleccionado.edad}`;
            salida.prepend(notificacion);
        }
    }
}
function mostrarEstudiantes(estudiantes) {
    salida.innerHTML = '';
    for (const estudiante of estudiantes) {
        let divEstudiante = document.createElement("div");
        divEstudiante.innerHTML= `<h2>${estudiante.nombre}</h2>
                                  <p${estudiante.edad} / ${estudiante.curso} </p>
                                  <button id=${estudiante.id}' class='btnEstudiante'>Seleccionar</button>`
        salida.appendChild(divEstudiante);
    }
}
const salida = document.createElement("div");
document.body.appendChild(formulario);
document.body.appendChild(salida);
```

Solución propuesta Actividad 5

```
formulario.onsubmit = (e) => {
    e.preventDefault();
    const inputs = e.target.children;
    estudiantes.push(new Estudiante({ nombre: inputs[0].value, edad: inputs[1].value, curso: inputs[2].value }));
    mostrarEstudiantes(estudiantes);
    const btnSuspender = document.getElementsByClassName('btnSuspender');
    for (const boton of btnSuspender) {
        boton.onclick = (e) => {
            const seleccionado = estudiantes.find(obj => obj.id == e.target.id);
            seleccionado.suspender();
            console.log(seleccionado);
            let notificacion = document.createElement("h6");
            notificacion.innerHTML = `Nombre ${seleccionado.nombre} -  SUSPENDIDO`;
            salida.prepend(notificacion);
        }
    }
}

function mostrarEstudiantes(estudiantes) {
    salida.innerHTML = '';
    for (const estudiante of estudiantes) {
        let divEstudiante = document.createElement("div");
        divEstudiante.innerHTML = `<h2>${estudiante.nombre}</h2>
                                   <p>${estudiante.edad} / ${estudiante.curso} </p>
                                   <button id='${estudiante.id}' class='btnEstudiante'>Seleccionar</button>
                                   <button id='${estudiante.id}' class='btnSuspender'>Suspender</button>`
        salida.appendChild(divEstudiante);
    }
}
const salida = document.createElement("div");
document.body.appendChild(formulario);
document.body.appendChild(salida);
```

Actividades

Actividad 1
Imagen animada

Incluir una imagen manipulando el DOM. Luego, realizar al menos dos animaciones encadenadas sobre la misma. Es necesario emplear métodos de jQuery.

Notas actividad 1

Recordemos que “encadenar animaciones” consiste en disparar animaciones de forma secuencial. Dado que cada animación tiene una determinada duración en milisegundos, se recomienda emplear una proporcional entre ellas.

CODERHOUSE

Actividad 2
Banner animado

Incluir una imagen y un contenedor manipulando el DOM. Luego, realizar al menos tres animaciones encadenadas sobre la imagen, y por cada transición variar el texto del contenedor. Es necesario emplear métodos de jQuery.

Notas actividad 2

Dado que es necesario modificar el interior del contenedor cada vez que concluye una animación sobre la imagen, es necesario emplear funciones callbacks para programar este cambio todas las veces que finalice la animación.

CODERHOUSE

Actividad 3
Entrada y salida animada

Incluir un input, un contenedor, y una imagen manipulando el DOM. Luego realizar una o más animaciones sobre la imagen, al detectar un cambio de valor en el input, procesando el valor ingreso, y efectuando un salida en el contenedor con el resultado. La salida mencionada debe destacarse usando una animación. Es necesario emplear métodos de jQuery.

Notas actividad 3

Las animaciones son elementos que nos permiten distinguir entradas y salidas de forma gráfica. Si disparamos animaciones desde manejadores de eventos, los usuarios podrán divisar efectos en distintas partes a destacar de la aplicación.

CODERHOUSE

Actividad 4
Interfaz animada

Incluir una estructura HTML manipulando el DOM, la cual contenga al menos dos radio buttons. Luego, escuchar el evento click sobre los botones de la interfaz, y disparar diferentes animaciones por cada elemento.
Si las animaciones se realizan sobre la misma imagen o contenedor, se recomienda establecer un control sobre los botones, para evitar que las animaciones se superpongan. Es necesario emplear métodos de jQuery.

Notas actividad 4
Las interfaces animadas garantizan un nivel alto de interactividad; pero si existe más de una animación operando sobre el mismo elemento, hay que establecer un medio de control para obtener el efecto deseado. Dependiendo el caso, esto puede solucionarse encadenando, retrasando una animación con delay, o desactivando eventos que disparan otras animación mientras una esté en curso.

CODERHOUSE

Actividad 5
Cronómetro animado

Incluir un contenedor manipulando el DOM. El mismo debe representar un cronómetro, en el cual se modifique su contenido cada segundo, usando una animación.
Se recomienda crear una función con la cabecera cronometro(selector), dentro de la cual se codifiquen la lógica de modificación y animación cada ciertos segundos. Es necesario emplear métodos de jQuery.

Notas actividad 5
Recordemos que un (1) segundo equivale a mil (1000). El cronómetro deberá efectuar una animación cada segundo, modificando el interior del contenedor escogido. Recordamos que podemos llamar a cualquier función dentro del callback de animación.

CODERHOUSE

Resoluciones

Solución propuesta Actividad 1

```
const URLIMG =
'https://res.cloudinary.com/hdsqazxtw/image/upload/v1559681445/logo_coderhouse_2_bmqbet.png'

$('body').append(`<img id="logoCoder" src="${URLIMG}" width="150" height="150">`);
$('#logoCoder')
    .fadeOut(2000)
        .fadeIn(2000);
```

CODERHOUSE

Solución propuesta Actividad 2

```
const DURACION = 2000;
$('body').append(`<div>
                    <img id="logoBanner" src=" ${URLIMG}" width="150" height="150"  style="float:
left">
                    <h2 id="textBanner">APRENDE</h2>
                  </div>` );
$('#logoBanner')
    .fadeOut(DURACION,() => { $("#textBanner").html("PROGRAMACIÓN") })
        .fadeIn( DURACION,()  =>  { $("#textBanner").html("CON JAVASCRIPT") })
            .animate({ opacity: 0.75,  margin: "+=50"} ,
                     DURACION,
                     () => { $("#textBanner").html("EN CODERHOUSE")});
```

CODERHOUSE

Solución propuesta Actividad 3

```
$('body').append(`<div>
                    <img id="logoUI" src="${URLIMG}" width="150" height="150"  style="float: left">
                    <input id="entradaUI" type="text">
                    <h3 id="salidaUI"></h3>
                  </div>`);

$('#entradaUI').keyup(function (e) {
    $('#salidaUI')
        .hide()
            .html(e.target.value.toUpperCase())
                .slideDown(400);
    $('#logoUI')
        .animate({ opacity: .75} , 100)
            .animate({ opacity: 1 } , 100);
});
```

Solución propuesta Actividad 4

```
$('body').append(`<div>
                    <img id="logoGUI" src=" ${URLIMG}" width="150" height="150"  style="float: left">
                    <button id='btnSi'>SI</button>
                    <button id='btnNo'>NO</button>
                    <h2 id="salidaGUI"></h2>
                  </div>` );
$('#btnSi').click(function (e) {
    $('#btnNo').prop("disabled", true);
    $('#btnSi').prop("disabled", true);
    $('#salidaGUI')
            .html('ACEPTADO')
                .fadeIn(500)
                    .delay(400)
                        .fadeOut(500,()=>{
                            $('#btnNo').prop("disabled", false);
                            $('#btnSi').prop("disabled", false);
                        });
});
$('#btnNo').click(function (e) {
    $('#btnNo').prop("disabled", true);
    $('#btnSi').prop("disabled", true);
    $('#salidaGUI')
            .html('RECHAZADO')
                .fadeIn(500)
                    .delay(400)
                        .fadeOut(500,()=>{
                            $('#btnNo').prop("disabled", false);
                            $('#btnSi').prop("disabled", false);
                        });
});
```

Solución propuesta Actividad 5

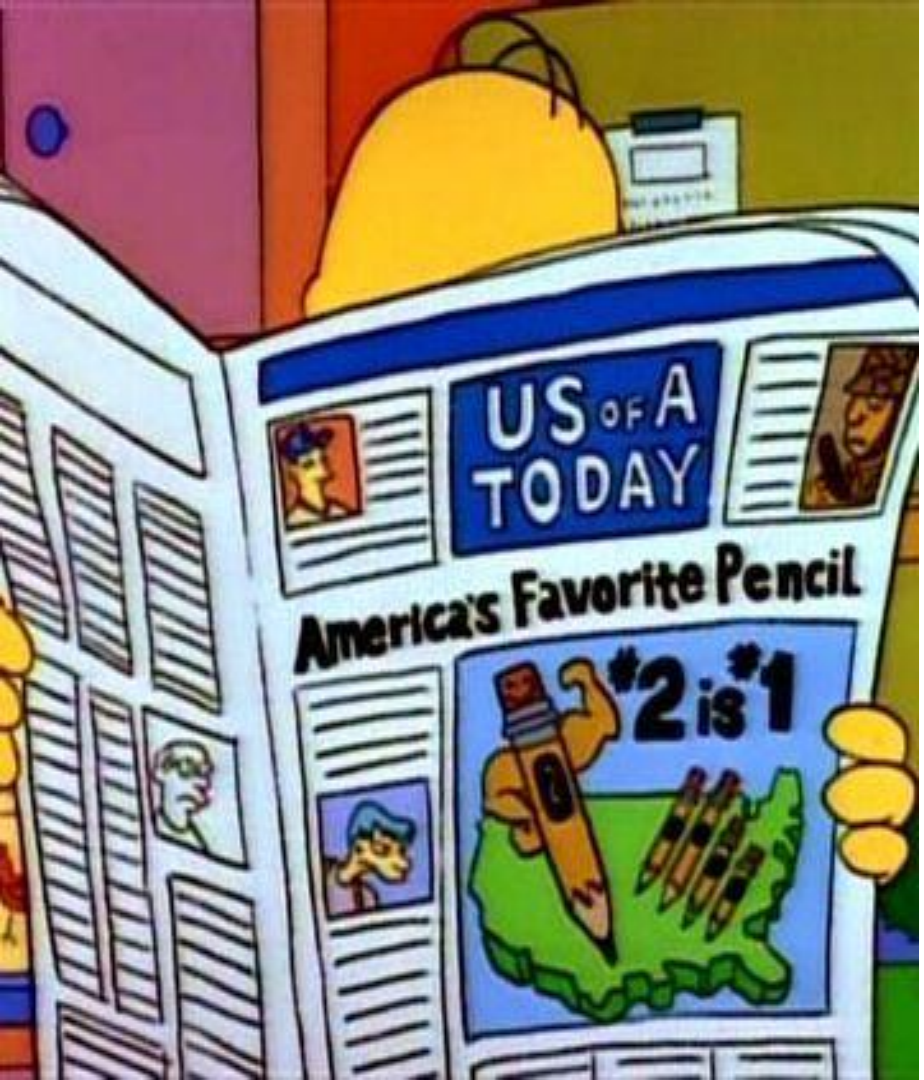
```
$('body').append("<h1 id='reloj'></h1><button id='btnStop'>DETENER</button>");
let detener = false;
let inicio = 0;
function cronometro(selector){
    $(selector)
            .delay(900)
                .fadeOut(500)
                    .html(inicio)
                        .fadeIn(500, () =>{
                            if(!detener){
                                inicio++;
                                cronometro('#reloj');
                            }
                        })
}
$('#btnStop').click(function (e) {
    detener = !detener;
    if(detener){
        $('#btnStop').html('CONTINUAR');
    }else{
        cronometro('#reloj');
        $('#btnStop').html('DETENER');
    }
});
cronometro('#reloj');
```

CODERHOUSE

JAVASCRIPT

14

Actividades



Actividad 1 Publicaciones

Solicitar el listado de publicaciones a la API JSONPlaceholder, usando una pretición AJAX (jsonplaceholder.typicode.com/posts).
Luego, por cada publicación obtenida, generar de forma iterativa una estructura HTML con la información, e incluirla en el DOM.

Notas actividad 1
Recordemos que el método AJAX a emplear para obtener datos desde la API con jQuery es $.get(). El HTML generado debe efectuarse en el callback de la petición, para visualizar la información correctamente.

CODERHOUSE

Actividad 2
Provincias Argentinas

Solicitar el listado de provincias de la Argentina a la API Georef, usando una pretición AJAX (apis.datos.gob.ar/georef/api/provincias). Luego, generar un select donde los values de las etiquetas option sean el ID de cada provincia obtenida.
Por último, escuchar el evento change sobre el select, asegurando que la función manejadora dispare una salida en el HTML, especificando la provincia seleccionada.

Notas actividad 2

Podemos emplear un API con el fin de obtener información útil para nuestra aplicación. Pero esto suele implicar un estudio de la estructura de datos que nos provee (JSON), así como una adaptación de la aplicación para trabajar con esta información.

CODERHOUSE

Actividad 3
Municipios Argentinos

Usando el select de provincias creado en la actividad 2, emplear el ID de la provincia seleccionada para solicitar al API Georef los municipios que le pertenecen. Por ejemplo, si el id de la provincia elegida tiene por valor '22' de solicitud, es apis.datos.gob.ar/georef/api/municipios?provincia=22. Luego, por cada municipio obtenido, generar de forma iterativa una estructura HTML con la información, e incluirla en el DOM.

Notas actividad 3

La totalidad de la información capaz de proveer un API puede estar dividida en diferentes rutas. En una API, podría ser necesario requerir el ID de cierto dato, para obtener información asociada a este usando una ruta. A esto se lo llama parámetro get.

CODERHOUSE

Actividad 4
Hogwarts

Solicitar el listado de estudiantes de hogwarts a la API Harry Potter, usando una pretición AJAX (hp-api.herokuapp.com/api/characters/students). Luego, por cada publicación obtenida generar de forma iterativa una estructura HTML con la información, e incluirla en el DOM.
Por cada estudiante debe existir un botón “Seleccionar”, que al ser presionado especifique la información del estudiante en una salida animada.

Notas actividad 4
Existen APIs de todo tipo, incluso podemos construir aplicaciones frontend que emplean sólo datos de APIs. La actividad propone que en la carga inicial del sitio se solicite la información a Harry Potter API, y en consecuencia se genere el HTML dinámico.

CODERHOUSE

Actividad 5
Buscar estudiantes

Añadir una interfaz que permita introducir una palabra, con la intención de filtrar el array de estudiantes obtenido desde la API en la actividad 4, usando la cadena ingresada.
Luego, actualizar la interfaz, pero sólo con la información de los estudiantes cuyo valor en la propiedad coincida con el valor ingresado en la caja de búsqueda.

Notas actividad 5
Tener en cuenta que mientras los datos se soliciten al API, los mismos no están cargados en la aplicación. Esto implica que no se pueda realizar un búsqueda, hasta que el servidor responda la petición correctamente.

CODERHOUSE

Resoluciones

Solución propuesta Actividad 1

```
//FUNCION CALLBACK PARA EL GET DE PUBLICACIONES
function callbackGetPublicaciones(respuesta,estado){
    //NOS PREGUNTAMOS SI EL ESTADO RESPONDIDO POR EL SERVIDOR ES SUCCESS.
    if(estado === "success"){
        //SI ES SUCCESS ENTONCES EN LA RESPUESTA TENEMOS EL JSON DE PUBLICACION.
        for (const publicacion of respuesta) {
            //AGREGAMOS UNA ESTRUCTURA HTML POR PUBLICACION EN LA RESPUESTA.
            $("#domGenerado").append(`<div>
                                        <h2>${publicacion.title}</h2>
                                        <p>${publicacion.body}</p>
                                    </div></hr>`);
        }
    }
}
//ESTA URL ES PARA OBTENER TODAS LAS PUBLICACIONES
const URLGETPUBLICACIONES = "https://jsonplaceholder.typicode.com/posts";
//AGREGAMOS CONTENEDOR A GENERAR EL CONTENIDO
$('body').append("<div id='domGenerado'></div>");
//LLAMADA ASINCRONA GET CON EL METODO .get DE JQUERY
$.get(URLGETPUBLICACIONES,callbackGetPublicaciones);
```

Solución propuesta Actividad 2

```
function selectLista(array, id){
    let innerSelect = '';
    array.forEach(provincia => innerSelect += `<option value="${provincia.id}">${provincia.nombre}</option>`)
    return `<select id="${id}">${innerSelect}</select>`;
}
const APIPROVINCIAS = "https://apis.datos.gob.ar/georef/api/provincias";
const provincias = [{id:-1, nombre: "SELECCIONAR PROVINCIA"}];
$(document).ready(function () {
    //LLAMAR AL API DE PROVINCIAS
    $.get(APIPROVINCIAS, function(datos,estado){
        if(estado === "success"){
            provincias.push(...datos.provincias);
            //AGREGAMOS SELECT,ESCUCHAMOS EL EVENTO CHANGE Y FILTRAMOS LA SALIDA
            $('body').prepend(`<div>PROVINCIAS: ${selectLista(provincias,'provinciasSelect')}</div>`);
            $('body').prepend("<h4 id='salidaProvincias'></h4>");
            $('#provinciasSelect').change(function (e) {
                const seleccionado = provincias.find(obj => obj.id == e.target.value);
                $('#salidaProvincias').html(`SELECCIONADA ${seleccionado.nombre.toUpperCase()}`);
            });
        }
    });
});
```

Solución propuesta Actividad 3

```
let municipios = [];
$('body').append(`<div id="municipios"></div><hr>);
function cargarMunicios(provincia){
    let apiMunicipios = `https://apis.datos.gob.ar/georef/api/municipios?provincia=${provincia.id}&campos=id,nombre&max=100`;
    //SE REALIZA LA LLAMADA GET Y SE CARGAR EL SELECT
    $.get(apiMunicipios, function(datos,estado){
        if(estado === "success"){
            municipios= datos.municipios;
            $('#municipios').html(`<hr>PROVINCIAS: ${selectLista(municipios,'municipioSelect')}`);
            $('#municipioSelect').change(function (e) {
                const seleccionado = municipios.find(obj => obj.id == e.target.value);
                $('#salidaProvincias').html(`PROVINCIA ${provincia.nombre.toUpperCase()}
                                            MUNICIPIO${seleccionado.nombre.toUpperCase()}`);

            });
        }
    });
}
```

CODERHOUSE

Solución propuesta Actividad 4

```
const estudiantes   = [];
$(document).ready(function () {
    $.get("http://hp-api.herokuapp.com/api/characters/students", (respuesta, estado ) => {
        if(estado === "success"){
            estudiantes.push(...respuesta);
            $('body').append("<div id='contenidoGenerado'></div>");
            $('body').prepend("<div id='notificacion'></div><hr>");
            for (const estudiante of estudiantes) {
                let idGenerado = estudiante.name.replace(" ","-");
                $("#contenidoGenerado").append(`<div height="300" style="float:left">
                                                <img src=${estudiante.image}" width="200" height="250"  style="margin:1">
                                                <h3>${estudiante.name}</h3>
                                                <button id=${idGenerado} class="btnEstudiante">SELECCIONAR</button>
                                            </div>`);
            }
            $(".btnEstudiante").click(function (e) {
                $('#notificacion').html(`SELECCIONADO ${e.target.id.replace("-"," ").toUpperCase()}`);
            });

        }
    });
});
```

CODERHOUSE

Solución propuesta Actividad 5

```
$('body').prepend("<div>BUSCAR: <input id='busqueda' type='text'></div><hr>  ");
function generarEstudiantes (lista){
    $("#contenidoGenerado ").empty();
    for (const estudiante of lista) {
        let idGenerado  = estudiante.name.replace(" ", "-");
        $("#contenidoGenerado ").append(`<div height="300" style="float:left">
                                        <img src="  ${estudiante.image}" width="200" height="250"  style="margin:1">
                                        <h3>  ${estudiante.name}</h3>
                                        <button id=  ${idGenerado} class="btnEstudiante">SELECCIONAR</button>
                                        </div>`  );
    }
    $(".btnEstudiante").click(function  (e) {
        $('#notificacion').html(`SELECCIONADO  ${e.target.id.replace("-", " ").toUpperCase()}`);
    });
}
$('#busqueda').keyup(function  (e) {
    if(estudiantes.length  > 0){
        let value  = e.target.value;
        let filtrados  = estudiantes.filter(estudiante =>
                                        estudiante.name.includes(value)  ||
                                        estudiante.gender.includes(value)  ||
                                        estudiante.house.includes(value));
        if(filtrados.length  > 0){
            generarEstudiantes (filtrados);
        }else{
            generarEstudiantes (estudiantes);
        }
    }
});
```

CODERHOUSE

JAVASCRIPT

15

Actividades

Actividad 1
App email spa - rutas

Declarar un array de al menos tres (3) rutas, el cual usaremos para construir un simulador de tipo email. La app tendrá una arquitectura single-page application, y su funcionalidades mínimas deberán ser las siguientes:

- ✓ “Enviar”: una interfaz para enviar mails.
- ✓ “Enviados”: una interfaz para visualizar los enviados.
- ✓ “Papelera”: una interfaz donde ver los mails eliminados.

Notas actividad 1

Recordemos que una aplicación SPA presenta la característica de brindar un conjunto significante de funcionalidades, sin refrescar la página. Determinamos rutas para definir qué interfaz renderizar en la única página de la aplicación.

CODERHOUSE

Actividad 2
App email spa – enviar

Renderizar una interfaz que permita realizar la simulación del envío de un correo electrónico a cierta dirección. Con la intención de manejar un respuesta del servidor, se propone realizar una petición POST con AJAX a la API JSONPlaceholder (jsonplaceholder.typicode.com/posts).
Una vez confirmada la petición, instanciar un objeto Email, y asociarlo a un array de emails, efectuando una salida de confirmación de envío al usuario.

Notas actividad 2
La simulación del envío del mail, usando un POST con AJAX, nos permite codificar un comportamiento real de comunicación frontend-backend. Técnicamente, el email no se envía correctamente hasta que el backend confirme el envío.

CODERHOUSE

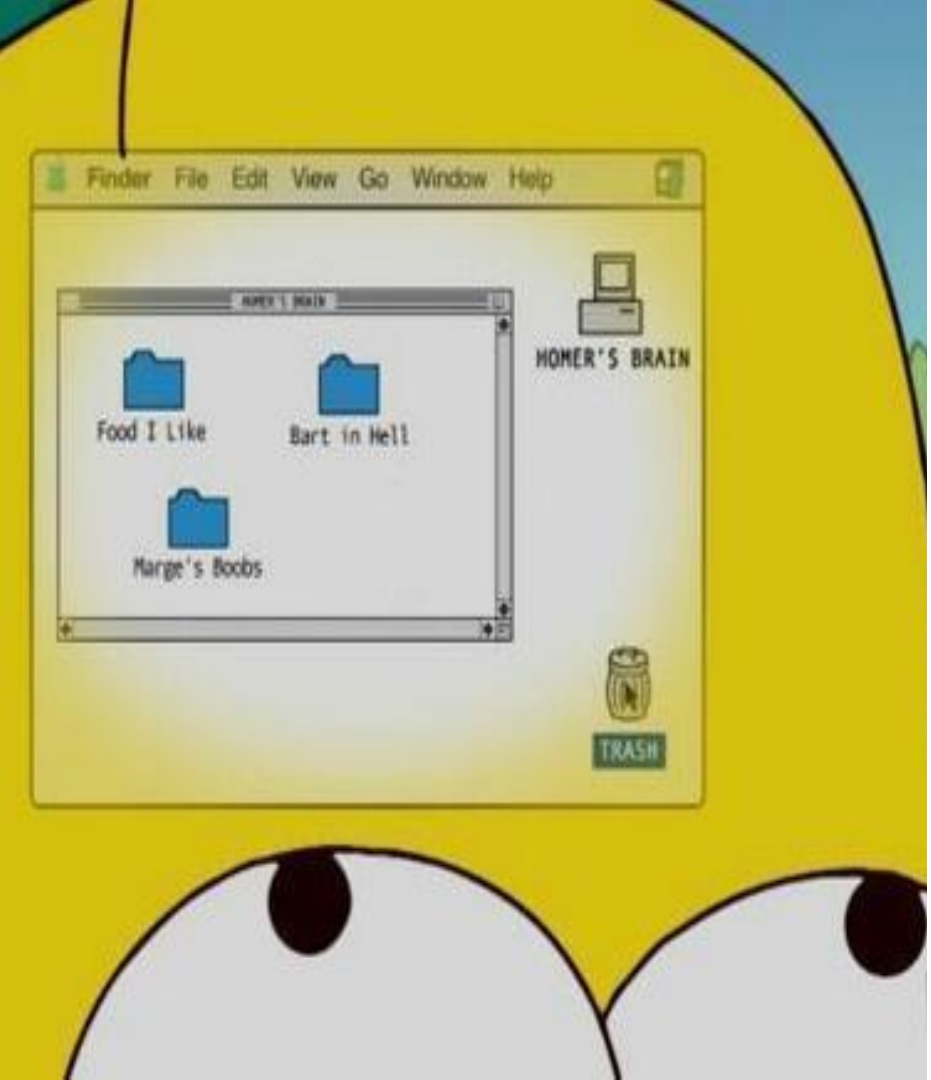
Actividad 3
App email spa - enviados

Renderizar una interfaz que permita visualizar los emails enviados de la aplicación. Ya que el backend al que realizamos los envíos de emails (JSONPlaceholder) no almacena nuestros correos, el array de “emails” generado en la actividad 2 representa la colección de enviados. Si queremos que la información persista al cierre de la aplicación, podemos almacenar el array en localstorage.

Notas actividad 3

Todo aplicación web cuenta con un backend, en el cual se almacena la información de la aplicación. Los emails enviados, dado que estamos simulando un comportamiento de envio, no se almacenan en el servidor con el POST realizado, por ende, esa información sólo se administra de forma local.

CODERHOUSE

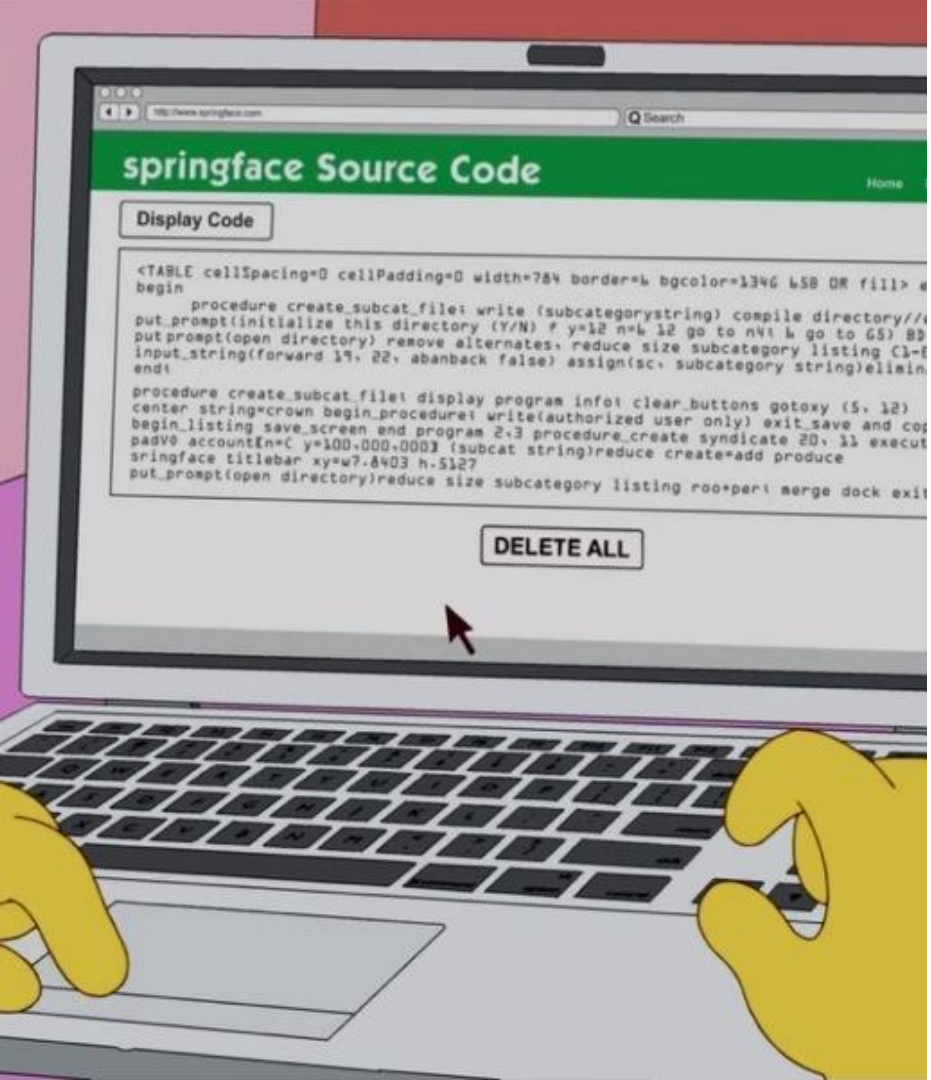


Actividad 4
App email spa – papelera

En la interfaz de la actividad 3, agregar un botòn por cada email, el cual al presionarlo elimine la instancia del array de enviados, asociando el objeto a una colección “papelera”. Luego, renderizar una interfaz que permita visualizar los emails en papelera de la aplicación.
También podemos manejar una respuesta del servidor, realizando una petición DELETE con AJAX a la API de JSONPlaceholder (jsonplaceholder.typicode.com/posts/1), por cada email enviado a la papelera.

Notas actividad 4
Con el método $.ajax() de jQuery, podemos simular la petición DELETE. Generalmente, el pedido de eliminación al backend se presupone uno a uno, por lo cual hay que enviar el ID del elemento a eliminar en la ruta.

CODERHOUSE



Actividad 5
App email spa – router

Programar el router SPA, que permitirá al usuario utilizar una interfaz en función del la ruta de acceso. El comportamiento del router de esta aplicación se resume a:

- ✓ Renderizar interfaz “enviar”, cuando el usuario solicite la ruta asociada a “enviar”.
- ✓ Renderizar interfaz “enviados”, cuando el usuario solicite la ruta asociada a “enviados”.
- ✓ Renderizar interfaz “`papelera”, cuando el usuario solicite la ruta asociada a “papelera”.

Notas actividad 5

El usuario puede solicitar una u otra ruta, haciendo clic en distintas etiquetas enlaces de la aplicación. Recordemos que la estrategia que mostramos en clase es emplear el hash(#), y detectar el evento hashchange para determinar la solicitud de cambio de ruta.

CODERHOUSE

Resoluciones

Solución propuesta Actividad 1

```
const routes = [
    { path: '/enviar'       ,  action: 'enviar' },
    { path: '/enviados', action: 'listar' },
    { path: '/papelera', action: 'eliminados' },
];
```

Solución propuesta Actividad 2

```
const emails = [];
class Email{
    constructor(emisor,receptor, mensaje){
        this.id       = emails.length;
        this.emisor   = emisor;
        this.receptor = receptor;
        this.mensaje  = mensaje;
    }
}
function enviarHTML(){
    return `<form id='enviarEmail'>
                <input type="text">
                <input type="text">
                <textarea></textarea>
                <input type="submit">
            </form>`
}
```

Solución propuesta Actividad 2

```
function enviarEvents(){
    $("#enviarEmail").submit(function (e) {
        e.preventDefault();
        const inputs = e.target.children;
        $.post( "https://jsonplaceholder.typicode.com/posts",
                {emisor: inputs[0].value ,receptor:inputs[1].value, detalle:inputs[2].value},
                (data,status)=>{
                    if(status === "success"){
                        let mail  = new Email(data.emisor, data.receptor, data.detalle)
                        emails.push(mail);
                        console.log(emails);
                        //$('body').prepend(`<h4> EMAIL ENVIADO ${mail.id} A
${mail.receptor}</h4>`);
                        enviadosRender('body');
                    }
                });
    });
}
function enviarRender(selector){
    $(selector).html(enviarHTML());
    enviarEvents();
}
enviarRender('body');
```

CODERHOUSE

Solución propuesta Actividad 3

```
function enviadoHTML(mail){
    return `<div>
                <h4>DE ${mail.emisor.toUpperCase()}</h4>
                <h4> A ${mail.receptor.toUpperCase()}</h4>
                <p>MENSAJE: ${mail.mensaje.toUpperCase()}</p>
                <button id=' ${mail.id}' class='btnDelete'>ELIMINAR</button>
            </div>`
}

function enviadosRender(selector){
      for (const mail of emails) {
        $(selector).append(enviadoHTML(mail));
      }
}
```

CODERHOUSE

Solución propuesta Actividad 4

```
const papelera = [];
function enviadosEvents(){
    console.log($(".btnDelete"));
    $(".btnDelete").click(function (e) {
        console.log(e.target.id);
        const mail = emails.find(email => email.id == e.target.id);
        $.ajax({
            url: `https://jsonplaceholder.typicode.com/posts/${mail.id}`,
            type: 'DELETE',
            success: function(data) {
                const idMail  = emails.indexOf(mail);
                //USAMOS SPLICE PARA ELIMINAR UN OBJETO POR POSICIÓN
                emails.splice(idMail, 1);
                papelera.push(mail);
            }
        });
    });
}
```

```
$('body').prepend(` <nav>
                        <a href="#/enviar"          class="button">ENVIAR</a>
                        <a href="#/enviados" class="button">ENVIADOS</a>
                        <a href="#/papelera" class="button">PAPELERA</a>
                    </nav>`);
$('body').append(`<div  id="app" style="height: 1000px;"></div>`);
//FUNCIONES PARA OBTENER Y GENERAR LA RUTA
const parseLocation = () => location.hash.slice(1).toLowerCase() || '/';
const findActionByPath = (path, routes) => routes.find(r => r.path == path || undefined);
//FUNCION ROUTER
const router = () => {
    //OBTENER RUTA ACTUAL
    const path    = parseLocation();
    const { action = 'error' } = findActionByPath(path, routes) || {};
    // LLAMAMOS AL MÈTODO CORRESPONDIENTE PARA LA ACCIÓN ENCONTRADA
    switch (action) {
      case 'enviar':
        enviarRender('#app');
        break;
      case 'listar':
        enviadosRender('#app');
        break;
      case 'eliminados':
        eliminadosRender('#app');
        break;
      default:
        $('#app').html("ERROR 404 - PAGINA NO ENCONTRADA")
        break;
    }
};
//CADA VEZ QUE SE DETECTA LA CARGA DE LA VENTANA SE LLAMA A LA FUNCION ROUTER
$(window).on('load', function () {
    router();
});
//CADA VEZ QUE SE DETECTA UN CAMBIO EN EL HASH  SE LLAMA A LA FUNCION ROUTER
$(window).on('hashchange', function () {
    router();
});
```

Solución propuesta Actividad 5

CODERHOUSE