<codoacodo/>

Agencia de Aprendizaje a lo largo de la vida

FULL STACK FRONTEND
Clase 17

Javascript 5

BA

Arrays, Storage y JSON

Les damos la bienvenida

Vamos a comenzar a grabar la clase

<codoacodo/>

Clase 16

Objetos

- Objetos. ¿Qué son y cómo se usan?
- Propiedades y métodos.
- Función constructora.
- El objeto String y sus métodos.
- El objeto Math, sus propiedades y métodos.

Clase 17

Arrays, Storage y JSON

- Arrays.
- Funciones para operar arrays.
- Trabajar con array de objetos.
- Web Storage.
- JSON. Formato y ejemplos de uso.

Clase 18

JS6 - DOM y Eventos

- Manipulación del DOM.
- Definición, alcance y su importancia..
- Eventos en JS.
- Eventos. ¿Qué son, para qué sirven y cuáles son los más comunes?
- Escuchar un evento sobre el DOM.

Arrays

Los arrays son objetos similares a una lista cuyo prototipo proporciona métodos para efectuar operaciones de recorrido y de mutación.
Tanto la longitud como el tipo de los elementos de un array son variables.
Si hemos pensado a las variables como una “caja” en la que se almacena un dato, un array podría considerarse una colección de cajas, cada una de ellas con un dato en su interior.
Toda la colección comparte un nombre (el nombre del array) y cada caja puede referenciarse para poder acceder a su contenido.

Array | ¿Qué son?

Un **array**, también conocido como **arreglo** o **vector**, es una colección o agrupación de elementos en una misma variable.

Los elementos del array pueden ser datos de diferentes tipos. Sin embargo, algunos de los métodos que poseen sólo funcionarán correctamente en arrays que tengan todos sus elementos del mismo tipo.

Cada elemento dentro del array posee un **índice**, un valor que nos permite identificarlo. Pensábamos a las variables como una “caja”. De forma similar, podemos imaginar un array como los vagones de un tren, donde cada vagón posee un contenido y un orden. El índice es el orden y el contenido dentro del vagón es el dato.

Array | ¿Cómo se crean?

Se pueden definir de varias formas:

Constructor	Descripción
new Array(len)	Crea un array de **len** elementos usando un constructor.
new Array(e1, e2...)	Crea un array vacío o con elementos.
[e1, e2...]	Enumeración de los elementos dentro de corchetes(**[]**). Notación preferida.

```
// Definición mediante un constructor (forma tradicional)
const array = new Array("a", "b", "c")

// Mediante la enumeración de sus elementos (forma preferida)
const array = ["a", "b", "c"] // Array con 3 elementos
const empty = [] // Array vacío (0 elementos)
const mixto = ["a", 5, true] // Array mixto (string, number, boolean)
```

Array | Acceso a elementos

Las posiciones de un array se numeran a partir de 0 (cero). Cuando usamos array[0] estamos haciendo referencia a la posición 0 del array cuyo contenido, en este caso, es la letra “a”:

```
const array = ["a", "b", "c"]

console.log(array[0]) // 'a'
console.log(array[2]) // 'c'
console.log(array[5]) // undefined
```

```
a
c
undefined
```

array[5] hace referencia a una posición que **no existe**, dado que el array tiene solamente 3 posiciones, con índices respectivos iguales a 0, 1 y 2.

Array | .length

.length es un método que devuelve la cantidad de elementos que posee un array:

```
const array = ["a", "b", "c"]
console.log(array.length) // 3
```

```
3
```

Para acceder al último elemento del array utilizamos:

```
let ultimo = nombreArray[nombreArray.length - 1]
```

Debemos restar 1 al valor devuelto por **.length** por que los valores de los índices del arreglo comienzan en **cero**.

Ejemplos (crear, acceder y mostrar elementos)

```
const vector= [1,3,5,8] //0, 1, 2, 3: cantidad de elementos - 1
const vectorVacio= [] //Vector vacío
const vectorDos = new Array("a", "b", "c")
const vectorTres = new Array (1, 5, 10, 15, 20)
```

```
console.log(vector)
document.write(vector)
console.log("Vector vacio:", vectorVacio)
console.log(vectorDos)
console.log(vectorDos[1])
console.log(vectorTres[2])
console.log(vectorTres[6])
```

1,3,5,8

Array | Bucle For

Utilizando un **bucle for** y la propiedad **.length** recorremos un vector. **vectorDos.length** devuelve la longitud del vector, 3 . Usamos **<** (menor que) para recorrer las posiciones desde 0 a 2, sino la última regresa *undefined*.

```
const vectorDos = new Array("a", "b", "c")
console.log("Elementos del vector 2:")
for(i = 0; i < vectorDos.length; i++){
    console.log(vectorDos[i])
}
```

```
Elementos del vector 2:
a
b
c
```

En este caso mostramos en el body cada elemento del **vectorTres**, separados por una coma.

```
const vectorTres = new Array (1, 5, 10, 15, 20)
document.write("Elementos del vector 3: <br>")
for(i = 0; i < vectorTres.length; i++){
    document.write(vectorTres[i] + ", ")
}
```

Elementos del vector 3:
1, 5, 10, 15, 20,

Array | Métodos (funciones)

Algunos métodos y funciones que podemos utilizar con los array:

Método	Descripción
.push(obj1, obj2...)	Añade uno o varios elementos al final del array. Devuelve tamaño del array.
.pop()	Elimina y devuelve el último elemento del array.
.unshift(obj1, obj2...)	Añade uno o varios elementos al inicio del array. Devuelve tamaño del array.
.shift()	Elimina y devuelve el primer elemento del array.
.concat(obj1, obj2...)	Concatena los elementos (o elementos de los arrays) pasados por parámetro.
.indexOf(obj, from)	Devuelve la posición de la primera aparición de obj desde from.
.lastIndexOf(obj, from)	Devuelve la posición de la última aparición de obj desde from.

Array | Métodos | Push y Pop

El método **push** agrega elementos al final de una lista, y el método **pop** los elimina, también desde el final:

```
var frutas = ["Banana", "Naranja"]
console.log(frutas)
frutas.push("Kiwi", "Pera")
console.log(frutas)
```

```
▶ (2) ['Banana', 'Naranja']
▶ (4) ['Banana', 'Naranja', 'Kiwi', 'Pera']
```

```
var frutas = ["Banana", "Naranja"]
console.log(frutas)
console.log(frutas.pop())
console.log(frutas)
```

```
▶ (2) ['Banana', 'Naranja']
Naranja
▶ ['Banana']
```

Array | Métodos | Unshift y Shift

El método **unshift** agrega elementos al comienzo de un array y regresa la nueva longitud del mismo. Por su parte, **shift** elimina el primer elemento y devuelve su valor:

```
var colores = ["Rojo", "Celeste"]
console.log(colores)
colores.unshift("Azul", "Naranja")
console.log(colores)
```

```
▶ (2) ['Rojo', 'Celeste']
▶ (4) ['Azul', 'Naranja', 'Rojo', 'Celeste']
```

```
var colores = ["Rojo", "Celeste"]
console.log(colores)
console.log(colores.shift())
console.log(colores)
```

```
▶ (2) ['Rojo', 'Celeste']
Rojo
▶ ['Celeste']
```

Array | Métodos | Concat

El método **concat** se usa para unir dos o más arrays. Este método no cambia los arrays existentes, sino que devuelve un nuevo array:

```
var colores = ["Rojo", "Celeste"]
console.log(colores)
var masColores = ["Verde", "Negro"]
console.log(masColores)
//Los elementos de masColores se concatenan
//al final de los elementos de colores:
var todos = colores.concat(masColores)
console.log(todos)
```

```
▶ (2) ['Rojo', 'Celeste']
▶ (2) ['Verde', 'Negro']
▶ (4) ['Rojo', 'Celeste', 'Verde', 'Negro']
```

Observa que el orden en que se anexa un array al otro varía según apliquemos a uno u otro el método **.concat**

Array | Métodos | IndexOf y LastIndexOf

Estos métodos devuelven la posición (*índice*) en la que se encuentra el valor buscado, a partir de la posición dada. **IndexOf** lo hace contando desde el principio del arreglo, y **LastIndexOf** lo hace desde el final:

```
var letras = ["A", "B", "C", "D", "E", "B", "C"]
//Buscamos de izquierda a derecha
var posB1 = letras.indexOf("B")
console.log("La primera 'B' tiene indice",posB1)
var posB2 = letras.indexOf("B",2)
console.log("La segunda 'B' tiene indice",posB2)
//Buscamos de derecha a izquierda
var posA = letras.lastIndexOf("A")
console.log("La última 'A' tiene indice",posA)
var posB = letras.lastIndexOf("B")
console.log("La última 'B' tiene indice",posB)
```

```
La primera 'B' tiene indice 1
La segunda 'B' tiene indice 5
La última 'A' tiene indice 0
La última 'B' tiene indice 5
```

Array | Otros métodos

Otros métodos y funciones que podemos utilizar con los array:

Método	Descripción
.splice()	Agrega o elimina elementos a un array. Regresa los elementos eliminados.
.slice()	Devuelve los elementos seleccionados en un array como un array nuevo.
.reverse()	Invierte el orden de elementos del array.
.sort()	Ordena los elementos del array bajo un criterio de ordenación alfabética.
.sort(func)	Ordena los elementos del array bajo un criterio de ordenación func.

Array | Métodos | Splice y Slice

slice(inicio, final) retorna la copia de un arreglo desde el índice **inicio** hasta **final**, excluyendo el final. No modifica el arreglo original. +info

Splice realiza operaciones sobre el arreglo, modificándolo. Es muy versátil, y permite tanto quitar elementos como agregarlos. +info

```
const arreglo = ['a','b','c','d','e','f']
let trozo1 = arreglo.slice(1,3) // ['b','c']
let trozo2 = arreglo.slice(5)   // ['f']
```

```
const arreglo = ['a','b','c','d']
// Insertamos un elemento en la pos. 2:
arreglo.splice(2,0,'n')
console.log(arreglo) //['a','b','n','c','d']
// Reemplazamos un elemento en la pos. 1:
arreglo.splice(1,1,'t')
console.log(arreglo)//['a','t','n','c','d' ]
```

Array | Métodos | Sort y Reverse

Estos métodos ordenan e invierten el orden, respectivamente, de un arreglo. Para que funcione correctamente, debemos asegurarnos que todos los elementos del arreglo sean del mismo tipo.

```
// Arreglo de cadenas: Orden alfabético
const arreglo = ['c','d','a','b','e']
arreglo.sort()
console.log(arreglo) //['a','b','c','d','e']

const a = ['3','10','1','31','5']
a.sort()
console.log(a) //['1','10','3','31','5']
```

```
// Arreglo de números: Orden "alfabético"
const arreglo = [4,45,5,59,1,2]
arreglo.sort()
console.log(arreglo) //[1,2,4,45,5,59]

// Arreglos mixtos: cuidado
const cuidado = ['a',2,"b",1, true]
cuidado.sort()
console.log(cuidado) //[1,2,'a','b',true]
```

Array | Métodos con funciones

Podemos aplicar funciones a cada elemento del array:

Método	Descripción
.forEach(cb, arg)	Realiza la operación definida en cb por cada elemento del array.
.every(cb, arg)	Comprueba si todos los elementos del array cumplen la condición de cb.
.some(cb, arg)	Comprueba si al menos un elem. del array cumple la condición de cb.
.map(cb, arg)	Construye un array con lo que devuelve cb por cada elemento del array.
.filter(cb, arg)	Construye un array con los elementos que cumplen el filtro de cb.

Array | Métodos con funciones

Podemos aplicar funciones a cada elemento del array:

Método	Descripción
.findIndex(cb, arg)	Devuelve la posición del elemento que cumple la condición de cb.
.find(cb, arg)	Devuelve el elemento que cumple la condición de cb.
.reduce(cb, arg)	Ejecuta cb con cada elemento (de izq a der), acumulando el resultado.
.reduceRight(cb, arg)	Idem al anterior, pero en orden de derecha a izquierda.

For in

For in recorre las propiedades de un objeto, por ejemplo, un string o un array:

```
for (let variable in object) {
   //bloque de código a ser ejecutado
}
```

variable es variable que itera a través de las propiedades del objeto. Y **object** es el objeto sobre el que iteramos. Veamos un ejemplo iterando por un array:

```
let arreglo=["P","r","u","e","b","a"]
for (let letra in arreglo) {
  console.log(letra)
}
```

```
0
1
2
3
4
5
>
```

```
let arreglo=["P","r","u","e","b","a"]
for (let letra in arreglo) {
  console.log(arreglo[letra])
}
```

```
P
r
u
e
b
a
>
```

For in con objetos

For in también recorre las propiedades de un objeto, de principio a fin, sin necesidad de indicar "*desde dónde*" ni "*hasta donde*" ni "*el paso*" como con un **for** "normal".

```
let persona = {
   nombre: "Ana",
   apellido: "Paz",
   edad: 25
};
for (let x in persona) {
   console.log(x)
}
```

```
nombre
apellido
edad
>
```

Muestra los nombres de las propiedades.

```
let persona = {
   nombre: "Ana",
   apellido: "Paz",
   edad: 25
};
for (let x in persona) {
   console.log(x + ": " +
persona[x])
}
```

```
nombre: Ana
apellido: Paz
edad: 25
>
```

Muestra los nombres de las propiedades y el valor asociado a cada una.

For of

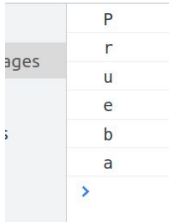
For of recorre una cadena (*string*) o arreglo (*array*), proporcionando acceso a cada uno de sus elementos. Su sintaxis es muy simple:

```
for (let variable of iterable) {
    //statement
}
```

En cada iteración el elemento (propiedad enumerable) correspondiente es asignado a variable.

Ejemplo: Definimos un arreglo, y lo recorremos guardando cada elemento en la variable **letra**, que mostramos por la consola:

```
let arreglo = ["P","r","u","e","b","a"]
for (let letra of arreglo){
     console.log(letra)
}
```



Web Storage

Javascript provee mecanismos para almacenar información en formato de texto en el dispositivo del usuario. La **API de almacenamiento web** proporciona los mecanismos mediante los cuales el navegador puede almacenar información de tipo clave/valor, de una forma mucho más intuitiva que utilizando cookies. Existen dos formas de hacerlo:

- **A nivel local (localStorage):** Al cerrar el navegador la información permanece en el dispositivo, y puede ser recuperada en una sesión posterior.
- **A nivel de sesión (sessionStorage):** Al finalizar la sesión la información almacenada se elimina.

Los objetos **localStorage** y **sessionStorage** permiten guardar pares clave / valor desde el navegador web.

LocalStorage

El objeto **localStorage** almacena datos sin fecha de vencimiento. Los datos no se eliminarán cuando se cierre el navegador y estarán disponibles en cualquier momento futuro.

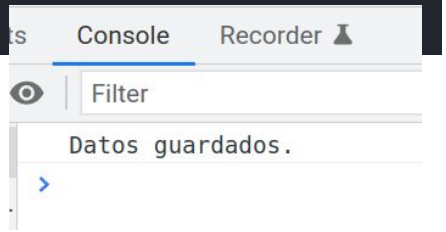
localStorage puede realizar esta tarea mediante los métodos **setItem** y **getItem**, que permiten guardar y recuperar información. Los datos se almacenan en formato de texto, como pares clave / valor.

No todos los navegadores soportan estas tecnologías. Si proporciona soporte, la condición `(typeof(Storage) !== "undefined")` es verdadera (true). Esto puede utilizarse para determinar si es posible grabar los datos o no.

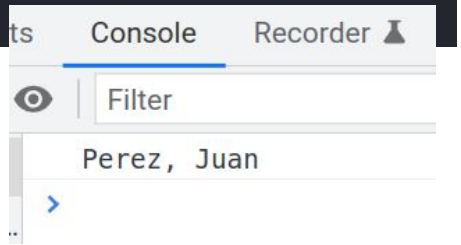
LocalStorage

El siguiente código almacena y recupera datos mediante **localStorage**:

```
// ¿El navegador soporta esta función?
if (typeof(Storage) !== "undefined") {
    // setItem guarda datos en el dispositivo
    localStorage.setItem("apellido", "Perez")
    localStorage.setItem("nombre", "Juan")
    console.log("Datos guardados.")
} else {
    console.log("Web Storage no soportado.")
}
```



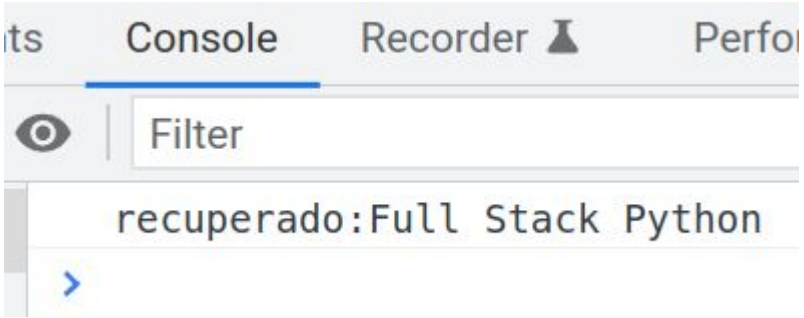

```
// ¿El navegador soporta esta función?
if (typeof(Storage) !== "undefined") {
    // getItem recupera datos del dispositivo
    ape = localStorage.getItem("apellido")
    nom = localStorage.getItem("nombre")
    console.log(ape + ", " + nom)
} else {
    console.log("Web Storage no soportado.")
}
```



SessionStorage

Los datos almacenados en **sessionStorage** son eliminados cuando finaliza la sesión de navegación, habitualmente al cerrar la pestaña en la que se muestra la página. Para ver el web storage en Chrome: F12/Application/Storage. +info

```
// ¿El navegador soporta esta función?
if (typeof(Storage) !== "undefined") {
    // setItem guarda datos en el dispositivo
    sessionStorage.setItem("curso", "Full
Stack Python")
    // getItem recupera datos del dispositivo
    curso = sessionStorage.getItem("curso")
    console.log("recuperado:" + curso)
} else {
    console.log("Web Storage no soportado.")
}
```

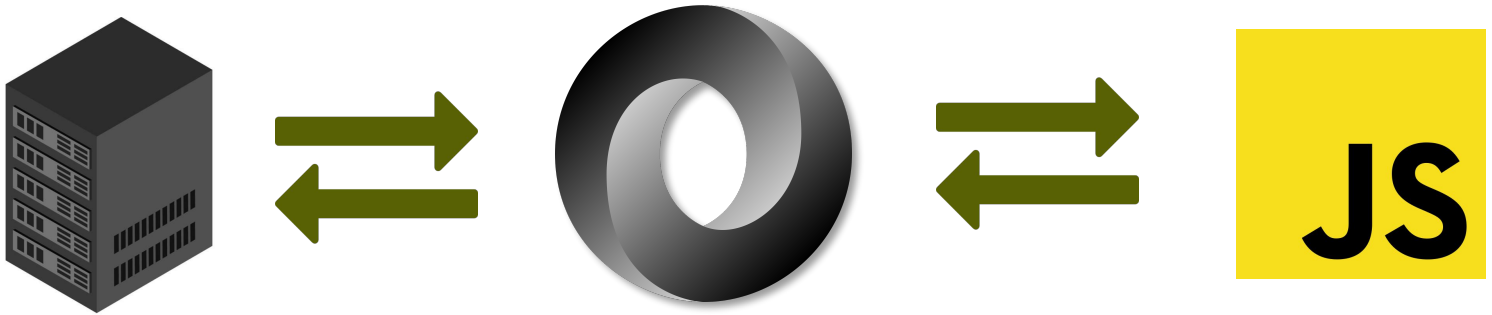


Se guarda un valor dentro de la clave "**curso**", y luego se recupera para mostrarlo en la consola. Esto solo tiene lugar si el navegador soporta **Storage**.

JSON

JSON: JavaScript Object Notation

JSON es una sintaxis propia de **objetos JS** utilizada para almacenar e intercambiar datos. Dado que JSON utiliza un formato de texto, es posible convertir cualquier objeto a JSON y enviarlo al servidor y viceversa. Esto permite procesar datos como objetos JS sin dificultades.



Servidor ***JSON*** ***Navegador***

JSON | Sintaxis

Reglas de sintaxis JSON:

La sintaxis JSON se deriva de la sintaxis de notación de objetos de JavaScript:

- Los datos se guardan en pares de nombre / valor
- Los datos están separados por comas
- Las {} contienen objetos
- Los corchetes se usan para indicar un array

En JSON , los valores deben ser uno de los siguientes tipos de datos:

- string
- number
- object (JSON object)
- array
- boolean
- null

La extensión por defecto para los archivos JSON es ".json"

JSON | Estructura de un archivo JSON

El JSON de la derecha posee una propiedad “empleados” compuesta por un arreglo de 3 elementos. Cada uno de ellos es un objeto con dos propiedades.

Este objeto JSON tiene varias propiedades con su valor. En el caso de la propiedad “hijos” el valor es un array.

```json
{
  "empleados": [
    { "nombre": "Juan", "apellido": "Pérez" },
    { "nombre": "Ana", "apellido": "López" },
    { "nombre": "Pedro", "apellido": "Uriarte" }
  ]
}
```

```json
{
  "nombre": "Luis",
  "apellido": "Fernández",
  "segundoNombre": null,
  "edad": 30,
  "hijos": ["Ana", "Luisa", "Marcelo"]
}
```

JSON | JSON.stringify( ) y JSON.parse( )

Podemos convertir datos almacenados en un objeto JavaScript al formato **JSON** usando **JSON.stringify( )**:

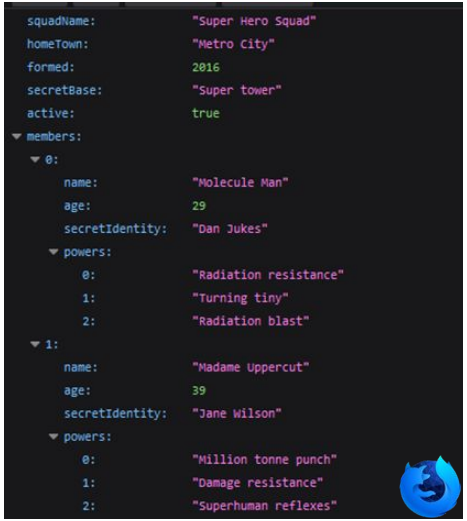
```
var myObj = { name: "John", age: 31, city: "New York" }
var myJSON = JSON.stringify(myObj)
// myJson= {"name":"John","age":31,"city":"New York"}
```

Si los datos están almacenados en JSON los podemos convertir a un objeto JS usando **JSON.parse( )**:

```
var myObj1=JSON.parse(myJSON)
//myObj1= { name: "John", age: 31, city: "New York" }
```

JSON | Otros ejemplos

Ejemplo superhéroes



Otro ejemplo: películas

JSON | API pública Randomuser

Esta API https://randomuser.me/api muestra datos de usuarios aleatorios, se utiliza para hacer pruebas. Es un string de JSON con un formato particular. Devuelve un usuario aleatorio, un array con un solo elemento.
Conviene leerlo desde **Firefox Developer Edition**, ya que la visualización es más simple.

Nosotros podremos ***consumir la API***, esto quiere decir leerla y traerla a nuestra aplicación.
Ingresando en https://randomuser.me/api/?results=5 podremos obtener 5 resultados, por ejemplo.

Material extra

Artículos de interés

Material de lectura:

- Fundamentos sobre arreglos
- Referencia sobre arreglos en W3Schools
- Métodos de los arreglos
- Superhéroes y películas en JSON

Videos:

- Arreglos y matrices en JavaScript
- Storage en JavaScript
- Curso de JSON (lista de reproducción . Ver videos 1 y 2)
- Página oficial de JSON
- Cargar archivo JSON en JavaScript

Material complementario

APIs gratuitas para programar proyectos. En el siguiente link podrán encontrar 1.500 APIs para utilizar en un desarrollo web. Tiempo, Películas, Libros, Eventos, Transporte, etc.

https://github.com/public-apis/public-apis

Las **APIs** son un protocolo de comunicación entre dos aplicaciones, lo que permite que un tercero pueda conectarse a un proveedor para consumir una serie de datos de manera sencilla. Simplifican procesos y habilitan el acceso a información o funcionalidades. Por lo tanto, los desarrolladores pueden crear nuevos servicios o mejorar los que ya tienen, sin tener que desarrollar todas las partes desde cero.

Actividades prácticas:

- Del archivo “**Actividad Práctica - JavaScript Unidad 2**” están en condiciones de hacer los ejercicios: 19 a 29.

No te olvides de dar el presente

Recordá:

- Revisar la Cartelera de Novedades.
- Hacer tus consultas en el Foro.
- Realizar los Ejercicios obligatorios.

Todo en el Aula Virtual.

Muchas gracias por tu atención.

Nos vemos pronto