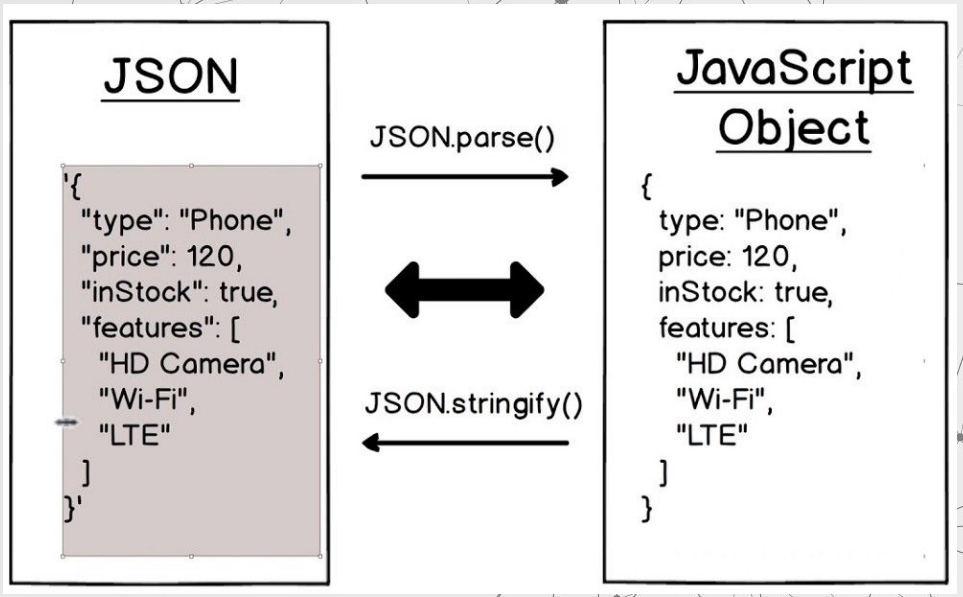
ALICIA VÁZQUEZ

JavaScript: Objetos genéricos

Alicia Vázquez
@aliciaFPInf

01

JSon vs Objetos



JSON vs XML

```
//XML en el fichero
<employees>
   <employee>
            <firstName>John</firstName>
<lastName>Doe</lastName>
   </employee>
   <employee>
        <firstName>Anna</firstName>
<lastName>Smith</lastName>
   </employee>
   <employee>
        <firstName>Peter</firstName>
<lastName>Jones</lastName>
   </employee>
</employees>
```

```
//JSON en el fichero
{[
  { "firstName":"John", "lastName":"Doe" },
  { "firstName":"Anna", "lastName":"Smith" },
  { "firstName":"Peter", "lastName":"Jones" }
]}
```

```
//JSON como texto.
let employees = `[
{ "firstName":"John", "lastName":"Doe" },
{ "firstName":"Anna", "lastName":"Smith" },
{ "firstName":"Peter", "lastName":"Jones" }
]`;
let object = JSON.parse(employees);
```

JSON vs Object

Ya hemos ido comentando que es un objeto y una clase.
En JS el objeto se define entre corchetes y separado por comas las propiedades que tenga junto con su valor. La sintaxis es muy parecida a la de JSon.

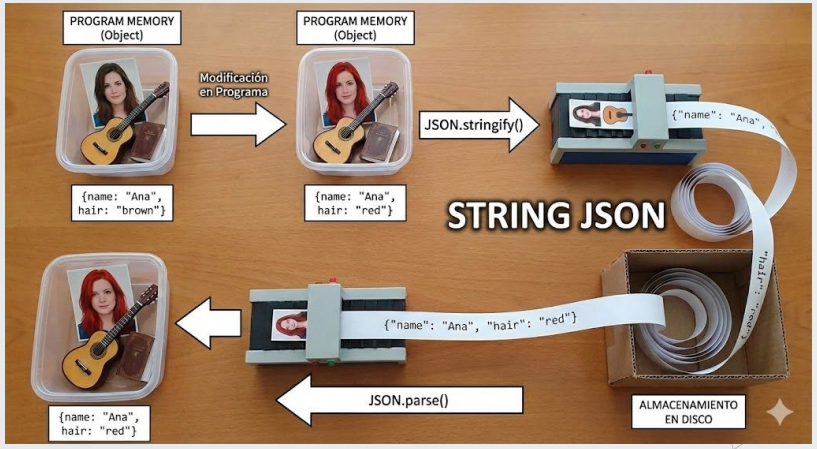
```
const employee = {
    firstName:"John",
    lastName:"Doe"
}
```

```
//JSON
{ "firstName":"John", "lastName":"Doe" }
```

Al ser la estructura de objetos de js tan parecida a la de json existen funciones que nos permiten pasar fácilmente de una a otra.

Método	Descripción
🔨 Parseo (De string a objeto)	
OBJECT `JSON.parse(str)`	Convierte el texto STRING `str` (si es un JSON válido) a un objeto y lo devuelve.
AB Convertir a texto (De objeto a string)	
STRING `JSON.stringify(obj)`	Convierte un objeto OBJECT `obj` a su representación JSON y la devuelve.
STRING `JSON.stringify(obj, props)`	Idem al anterior, pero filtra y mantiene solo las propiedades del ARRAY `props`.
STRING `JSON.stringify(obj, props, spaces)`	Idem al anterior, pero indenta el JSON a NUMBER `spaces` espacios.

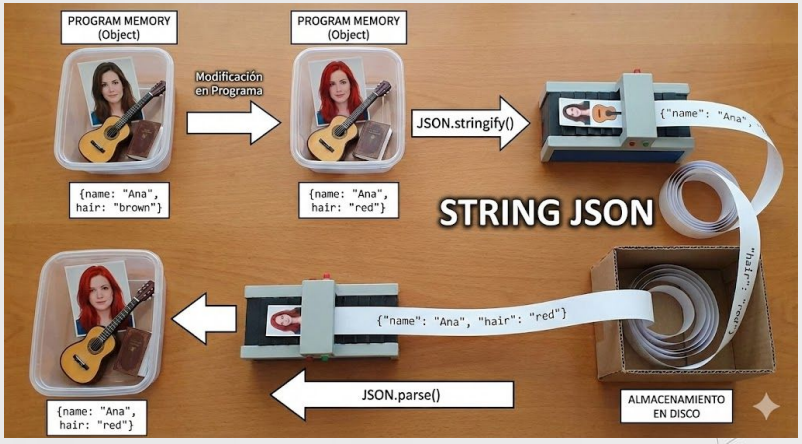
JSON.stringify()



JSON.stringify() nos permite pasar objetos a un JSON (un string), serializar los datos para poder ser enviados.
No serán pasados al string las propiedades de función (no está *getInfo*), ni las propiedades que su valor sea *undefined*

```
let jsonAlumno = JSON.stringify(alumno);
console.log(typeof(jsonAlumno) +" --> " +jsonAlumno); //string -->
{"nombre":"Aina","apellidos":"Garcia
López","edad":21,"ciclo":"DAMv1","telefono1":"666555777","telefono2":"666555777","tele
fono3":"666555777"}
```

JSON.parse()



JSON.parse() es la función inversa, transforma un Json (string) en un objeto con todas sus propiedades. Deserializamos los objetos que nos llegan desde el servidor

```
let a = JSON.parse(jsonAlumno);
console.log(typeof(a) +" --> " + a);  //object --> [object Object]

//Visualizamos toda la información que hay en el Objeto.
for (const key in a) {
      console.log(`${key} = ${a[key]}`);
}
```

Ejercicio alumnos.js

Partiendo del fichero *alumnos.js* donde tenemos una variable que contiene un JSON, recorrer el JSON y muestra por pantalla, en una tabla, tres de sus propiedades.

Juan	González Pérez	12345678A
María	López García	23456789B
Pedro	Martínez Ruiz	34567890C
Ana	Sánchez Jiménez	45678901D
David	Pérez Fernández	56789012E
Laura	González Martínez	12345678A
Carlos	Gómez López	67890123G
Elena	Fernández García	23456789B
Miguel	Martínez López	78901234H
Sara	Jiménez Sánchez	45678901D

02

Objetos genéricos

OBJETOS

En Javascript, existe un tipo de dato llamado **Object**. No es más que una variable especial que puede contener múltiples propiedades en su interior. De esta forma, tenemos la posibilidad de **organizar** múltiples variables de la misma temática dentro de un objeto. Las organizamos con las parejas **clave-valor**

En Javascript podemos definir cualquier variable como un objeto declarándola con **new** "construye el objeto" o declarando **literal object** (usando notación JSON). **Creamos una instancia de Object**.

```
//Construcción con new
const alumno = new Object();
//TAMBIEN const alumno = {}
alumno.nombre = "Lucas";
alumno.apellidos = "Fernandez Martinez";
alumno["edad"] = 20;
alumno.curso = "DAWe1";
```

```
//Declaración con literal object
const alumno2 = {
   nombre: "Aina",
   apellidos: "Garcia López",
   edad: 21,
   curso: "DAMv1"
};
```

Propiedades de un objeto

Podemos acceder a las propiedades de un objeto directamente con un punto (**.**) o bien usando los corchetes **[ ]**.

```
//ACCESO
console.log(alumno.nombre);
console.log(alumno.apellido);

console.log(alumno2["nombre"]);
console.log(alumno2["apellido"]);

//Añadimos más datos
alumno2.nota = 8;
console.log(alumno2); //Visualizamos todo el objeto.
```

Ejercicio 2: Objetos

1. Crea un objeto llamado **tvSamsung** con las propiedades:
   a. nombre (TV Samsung 42”)
   b. categoria (Televisores)
   c. unidades (4)
   d. precio (345.95)
   e. y con un método llamado **getImporte** que devuelve el valor total de las unidades (nº de unidades * precio)
2. Muestra los datos en pantalla (nombreAtributo: Valor)
3. Obtén el string que corresponde al JSON de **tvSamsung.**

Optional Chaining (?.)

Pero hemos de tener en cuenta que:

- Si intentamos acceder a propiedades que no existen no se produce un error, se devuelve *undefined*.
- Sin embargo se genera un error si intentamos acceder a propiedades de algo que no es un objeto.
    - Para evitar ese error hay que comprobar que existan las propiedades previas usando el operador **?**

```
console.log(alumno.ciclo)       // muestra undefined
console.log(alumno.ciclo.descrip)      // se genera un ERROR y deja de funcionar todo.
console.log(alumno.ciclo?.descrip)   // muestra undefined o el valor si existiera
```

Nullish Coalescing (??)

Este operador se usa para definir un valor por defecto. A diferencia del operador OR (||), el operador **??** es más preciso porque solo actúa si el valor es estrictamente null o undefined.

- Este operador se utiliza en caso de que el valor de la propiedad no sea ni null ni undefined, asignando un valor por defecto.
- Ejemplo: ***const stock = data.producto.cantidad ?? 0***; (Si la cantidad es 0, se mantiene el 0. Si es null, se asigna el valor por defecto 0).

```
// Combinación ganadora
const nombreUsuario = apiResponse.user?.profile?.name ?? "Invitado";
```

1. El `?.` asegura que si el objeto `user` o `profile` no existen, no se produzca un error catastrófico.
2. El `??` asegura que, si al final el nombre no se encuentra, el usuario vea "Invitado" en lugar de un feo `undefined`.

Recorrido FOR.. IN

Al ser un objeto un conjunto de propiedades podemos listarlas. En este caso el iterador no es un número, sino que corresponde al nombre de la **propiedad** del objeto.

```
function listaPropiedades(obj) {
    constresult = ``;
    for (var p in obj) {
            result += `${p} = ${obj[p]}\n`;
    }
    return result;
}
console.log(alumno2);
console.log(listaPropiedades(alumno2));
```

```
Printo el objeto directamente

▶ {nombre: "Aina", apellido: "Garcia", fechaNacimiento: "2/2
  curso: "DAMr1", …}
Printo el objeto usando la funcion

nombre = Aina
apellido = Garcia
fechaNacimiento = 2/2/2001
edad = 21
curso = DAMr1
nota = 8
```

Nota mental: Piensa que hasta ahora cuando recorremos un array, por ejemplo, el iterador era numérico y ya nos venía dado. Ahora es lo mismo, solo que le hemos dado un nombre al número.

Propiedades con funciones

Una propiedad de un objeto puede ser una función.

```
alumno.getInfo = function() {
    return 'El alumno ' + this.nombre + ' ' + this.apellidos + ' tiene ' + this.edad + ' años'
}
console.log(alumno.getInfo()); //El alumno Aina Garcia López tiene 21 años
```

This. hace referencia al objeto que está instanciado y por lo tanto al que ejecuta ese método.

Nota: En las funciones flecha no podemos usar **this.**

Ejercicio alumnos.js

Haz lo mismo, pero esta vez añade todas las propiedades que haya y añade la cabecera de la tabla.

nombre	apellidos	fecha_de_nacimiento	dni	telefono	mail	notam2	notam3	notam4	notam5	notam6	notam9
Juan	González Pérez	1999-05-15	12345678A	123456789	juan@example.com	7.5	8	6.5	9	7	8
María	López García	2000-08-25	23456789B	987654321	maria@example.com	8	7	8.5	9	6.5	7.5
Pedro	Martínez Ruiz	2001-03-10	34567890C	654321987	pedro@example.com	6	7	5.5	8	6	7
Ana	Sánchez Jiménez	1998-11-20	45678901D	123456789	ana@example.com	8	9	7	8.5	7.5	9
David	Pérez Fernández	2002-07-05	56789012E	987654321	david@example.com	7	8	6.5	8.5	7	8

Añade ahora una columna nueva para calcular la media. Piensa antes como hacerlo de la manera más óptima.

Propiedades calculadas

- Usando los **[ ]** podemos acceder a una propiedad a través de una variable. Esto nos permite ir creando propiedades del objeto dinámicamente. Además al usar los **[ ]** el nombre de las propiedades pueden tener el formato que queramos, espacio o ir creandolas iterativamente.

```
let propiedad = "nombre";
console.log(alumno[propiedad]); //Aina

propiedad = prompt("Qué propiedad quieres consultar", "apellidos");
console.log(alumno[propiedad]); //Garcia López

propiedad = prompt("Añade nueva propiedad", "curso");
let valor = prompt("Su valor", "1ero");
alumno[propiedad]=valor;
console.log(alumno); //{nombre: "Aina", apellidos: "Garcia López", edad: 21, ciclo: "DAMv1", curso: "1ero"}

for (let i =1; i<=3; i++){
   alumno["telefono"+i] = prompt("Telefono"+i,666555777);
}
console.log(alumno);
```

¿Existe la propiedad?

- Es posible querer comprobar si una propiedad existe o no dentro de un Objeto. Sabemos que si no existe directamente nos devuelve un *undefined*. Pero hay métodos más elegantes que nos devuelve un true/false si existe o no: **hasOwnProperty()** y **in**

```
console.log(alumno.notas); //undefined
console.log(alumno.notas?.m4); //undefined

console.log(alumno.hasOwnProperty("apellidos")); //true
console.log("edad" in alumno); //true
console.log("fecha nacimiento" in alumno); //false
let p = "telefono2";
console.log(p in alumno); //true
```

Más métodos (métodos estáticos)

- Por el mero hecho de ser un **Object**, tiene métodos, puesto que no deja de ser una instancia de la clase **Object**.

Método	Descripción
ARRAY `Object.keys(obj)` ES2015	Itera el `obj` y devuelve sus propiedades o keys.
ARRAY `Object.values(obj)` ES2015	Itera el `obj` y devuelve los valores de sus propiedades.
ARRAY `Object.entries(obj)` ES2015	Itera el `obj` y devuelve un ARRAY con los pares `[key, valor]`.
OBJECT `Object.fromEntries(array)` ES2015	Construye un objeto con un array de pares `[key, valor]`.

Más métodos (métodos estáticos)

```
const user = {
  name: "Manz",
  life: 99,
  power: 10,
  talk: function() {
    return "Hola!";
  }
};

Object.keys(user);// ["name", "life", "power", "talk"]
Object.values(user); // ["Manz", 99, 10, ƒ]
Object.entries(user);// [["name", "Manz"], ["life", 99],
["power", 10], ["talk", ƒ]]
```

```
// Obtener un objeto partiendo de dos arrays, uno de claves y otro
de valores.
const keys = ["name", "life", "power", "talk"];
const values = ["Manz", 99, 10, function() {
return "Hola" }];

const entries = [];
for (let i of Object.keys(keys)) {
  const key = keys[i];
  const value = values[i];
  entries.push([key, value]);
}


const user = Object.fromEntries(entries);
// {name: 'Manz', life: 99, power: 10, talk: ƒ}
```

Ejercicio 3: Objetos

1. Al objeto **tvSamsung** añadele iterativamente varias características y que sea al usuario quien indique el valor.
2. Crea una lista de propiedades que crees que debería tener un televisor y comprueba si **tvSamsung** las tiene o no. En caso de no tenerlas añadela e inicializarla a *"default value"*.
3. Muestra en un página HTML toda la información relativa a **tvSamsung.**

Ejercicio 4: Gestión de un coche

Crea un objeto llamado coche con las siguientes propiedades iniciales:

- marca: "Toyota"
- modelo: "Corolla"
- año: 2020
- encendido: false

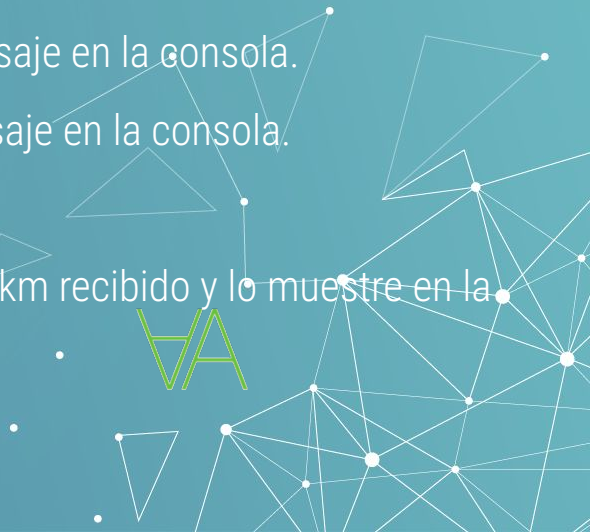
Añade un método arrancar() que cambie encendido a true y muestre un mensaje en la consola.

Añade un método apagar() que cambie encendido a false y muestre un mensaje en la consola.

Añade una nueva propiedad kilometraje con un valor inicial de 0.

Añade un método recorrer(km) que aumente el kilometraje según el valor de km recibido y lo muestre en la consola.

Prueba todas las funciones llamándolas con diferentes valores.



Ejercicio 5: Sistema de gestión de usuarios

Crea un array llamado usuarios que contenga al menos tres objetos representando personas. Cada objeto debe tener:

- nombre (string)
- edad (número)
- email (string)

Añade un nuevo usuario al array utilizando push().

Escribe una función mostrarUsuarios() que recorra el array e imprima cada usuario en formato: *"Nombre: Juan, Edad: 25, Email: juan@email.com"*.

Añade un método dentro de cada usuario llamado saludar(), que muestre un mensaje en la consola con su nombre.

Llama a saludar() en cada usuario para verificar su funcionamiento.

03

MAP y SET

SET

Es una colección de elementos que **no pueden estar repetidos**. La clase **SET** es muy parecida a la de **ARRAY**, pero tiene sus propias propiedades y métodos, es decir otro comportamiento.

```
//creación
let gadgets = new Set(['movil','tablet','portatil']);
console.log(gadgets);

//NO EXISTE!! console.log(gadgets.length);
console.log("Medida: "+gadgets.size);

//Añado elementos
gadgets.add("reloj");
gadgets.add("tv");
gadgets.add("tv"); //No lo agrega, ya existe.
gadgets.add("tv"); //No lo agrega, ya existe.

//recorrer y muestro
for(let item of gadgets){
    console.log("Item: "+item);
}
```

```
console.log("Elimino reloj ");
//eliminamos un elemento.
gadgets.delete("reloj");

console.log("Reloj?"+ gadgets.has("reloj"));

//Ordenado en el mismo
gadgets.forEach(item => console.log(item));

//Limpio todo el set, lo vacio.
console.log("VACIO SET");
gadgets.clear();
gadgets.forEach(item => console.log(item));
```

SET

Estas son algunas de las propiedades y métodos de la estructura de tipo **SET**

Propiedad o Método	Descripción
NUMBER .size	Propiedad que devuelve el número de elementos que tiene el conjunto.
SET .add(element)	Añade un elemento al conjunto (si no está repetido) y devuelve el set. Muta
BOOLEAN .has(element)	Comprueba si element ya existe en el conjunto. Devuelve si existe.
BOOLEAN .delete(element)	Elimina el element del conjunto. Devuelve si lo eliminó correctamente.
.clear()	Vacía el conjunto completo.

Es posible crear un Set a partir de un Array, pero debemos saber que si hay elementos repetidos en el array, en el set solo habrá uno.

```
let nombres = ['jose', 'maria', 'lola', 'jose', 'david', 'lola'];
let nombresUnicosSet = new Set(nombres)
console.log(nombresUnicosSet);// Expected output: Set(4) {'jose', 'maria', 'lola', 'david'}
//si lo queremos en un array:
let nombresUnicosArray = Array.from(nombresUnicosSet);
console.log(nombresUnicosArray);// Expected output:  (4) ['jose', 'maria', 'lola', 'david']
```

Ejercicio 1: Set

1. **Eliminar duplicados de un array utilizando un Set**
   Crea una función que reciba un array como entrada y devuelva un nuevo array con los elementos únicos del array original, es decir, eliminando los elementos duplicados.

```
const numeros = [1, 2, 3, 4, 4, 5, 6, 6, 7];
const numerosUnicos = eliminarDuplicados(numeros);
console.log(numerosUnicos); // Output: [1, 2, 3, 4, 5, 6, 7]
```

2. **Comprobar si dos arrays tienen elementos en común utilizando un Set**
   Crea una función que tome dos arrays como entrada y devuelva true si tienen al menos un elemento en común, y false en caso contrario.

```
const array1 = [1, 2, 3, 4, 5];
const array2 = [4, 5, 6, 7, 8];
console.log(tienenElementosEnComun(array1, array2)); // Output: true
```

MAP

Los Map en Javascript son estructuras de datos nativas que permiten implementar una estructura de tipo mapa, es decir, una estructuras donde tiene valores guardados a través de una **clave** para identificarlos. Comúnmente, esto se denomina pares **clave-valor**.

```
const map1 = new Map(); // Mapa Vacio
map1.set('a', 1);
map1.set('b', 2);
map1.set('c', 3);

console.log(map1.get('a')); // Expected output: 1

map1.set('a', 97); //Cómo ya existe se actualiza el valor.
console.log(map1.get('a')); // Expected output: 97

console.log(map1.size); // Expected output: 3
map1.delete('b');
console.log(map1.size); // Expected output: 2
console.log(map1.keys()); // Expected output: a,c
```

MAP

Estas son algunas de las propiedades y métodos de la estructura de tipo **MAP**

Propiedad o Método	Descripción
NUMBER .size	Propiedad que devuelve el número de elementos que tiene el mapa.
MAP .set(key, value)	Establece o modifica la clave key con el valor value. Muta
BOOLEAN .has(key)	Comprueba si key ya existe en el mapa y devuelve si existe o no.
OBJECT .get(key)	Obtiene el valor de la clave key del mapa.
BOOLEAN .delete(key)	Elimina el elemento con la clave key del mapa. Devuelve si lo eliminó correctamente.
.clear()	Vacía el mapa completamente.

Existen dos métodos muy interesantes: **`.keys()`** y **`.values()`**, ambos devuelven un objeto de tipo iterator, sin embargo usando la función **`Array.from()`**, podemos covertilo en arrays.

```
let array = Array.from(map1.keys());
console.log(array) // Expected output: [a,c]
array.push('z')
array.push('a')
console.log(array) //Expected output: ['a', 'c', 'z', 'a']
```

Ejercicio 2: Map

Partiendo de estos dos Arrays, crea una MAP que te permita guardar la relación de las notas con el módulo.

```
const modulos = ["m02", "m03", "m04", "m05", "m06", "m07", "m08", "FOL"]
const alumnoNotas = [[8, 9, 4], [9, 10], [6, 8, 10], [4], [8, 4, 7], [], [7, 5, 9, 10],[10]]
```

- Actualiza las notas de m07 a `[7.5, 6]`
- Obten un SET de las claves.
- Pasa este MAP a un objeto de Alumno.