# Objetos Literales

DEV.F:
DESARROLLAMOS(PERSONAS);

# OBJETOS LITERALES EN PROGRAMACIÓN

**Los objetos son una estructura de datos bastante usada en el lenguaje, de hecho, considero que es la más importante.**
**Un objeto en JavaScript es un conjunto agrupado entre llaves de claves y valores:**

```
NombreObjeto = {
    Valor
    Valor
    Valor
}
```

# ¿Qué es un literal?

**La definición de literal alude a algo textual, por ejemplo, si declaramos una variable de la siguiente manera:**

```
let colorDelSol = “AMARILLO”;
```

**Podemos decir la variable `colorDelSol` tiene asignada un string literal ya que se asigna el valor textualmente.**

**Exactamente lo mismo ocurre con los objetos literales, por ejemplo:**

```
let mascota = {
   nombre:"Scott",
   color:"Cafe",
   edad: 5,
   macho: true
};
```

**Donde:**

- **El nombre del objeto es mascota y sus claves/valores se describen en la siguiente tabla:**

| Clave | Valor | Tipo de dato |
|---|---|---|
| nombre | Scott | string |
| color | Cafe | string |
| edad | 5 | int |
| macho | true | bolean |

> *Los tipos de datos que puede almacenar un objeto pueden ser strings, enteros, boleanos, inclusive otros objetos.*

## Acceder a valores de un objeto

**Existen 2 maneras simples para poder acceder a los valores de un objeto:**

## Notación de punto

**Consiste en escribir el nombre del objeto seguido de un punto y el nombre de la propiedad a la cual se quiere acceder: objeto.clave o objeto.propiedad**

```
let mascota = {
nombre: "Scott",
tipo: "canino",
edad: 5,
macho: true
}
console.log(mascota.nombre);
console.log(mascota.edad);
```

```
Scott
```

```
5
```

## Notación de corchetes / llaves cuadradas o brackets

**Consiste en escribir el nombre del objeto anteponiendo entre corchetes la clave a la que se quiere acceder: objeto[clave] o objeto[propiedad]**

```
let mascota = {
nombre: "Scott",
tipo: "canino",
edad: 5,
macho: true
}
console.log(mascota['nombre']);
console.log(mascota['edad']);
```

```
Scott
5
```

## Conclusiones 📎

- Un objeto es la estructura de datos más usada en javascript, compuesta de pares ordenados y agrupados en claves y valores.
- Se denomina objeto literal al objeto cuyas propiedades están declaradas textualmente en el código.
- Los objetos pueden almacenar funciones(métodos) en su interior.
- Para acceder a las propiedades de un objeto dentro de un método es necesario usar la palabra reservada **this** por scope de las variables.
- A diferencia de otros lenguajes de programación se pueden añadir, actualizar, o eliminar propiedades de una manera muy poco usual pero bastante simple.

# ¿Qué es desestructuración?

La desestructuración es una característica bastante poderosa que nos permite separar **keys** o llaves de un objeto en variables independientes, ello para mejorar la legibilidad y escribir un código más compacto y simplificado. Dicha característica está presente desde la especificación ES6 del lenguaje.

## Sintaxis básica

```
const {key_1, key_2, ... key_n} = objeto;
```

**Imaginemos que contamos con un objeto literal y necesitamos imprimir su contenido, tendríamos que hacer algo como lo siguiente:**

```
const superheroe = {
  nombre: "Capitan América",
  edad: 30,
  peso: 100,
  empresa: "Marvel"
};
```

```
console.log(`${superheroe.nombre} tiene ${superheroe.edad} años, pesa ${superheroe.peso} kg y es de ${superheroe.empresa}`)
//salida: "Capitan América tiene 30 años, pesa 100 kg y es de Marvel"
```

**Como puedes apreciar, el ejemplo funciona bien, pero es poco mantenible y bastante redundante**

es acá donde la desestructuración de objetos puede ser implementada:

```
const superheroe = {
  nombre: "Capitan América",
  edad: 30,
  peso: 100,
  empresa: "Marvel"
};

const {nombre, edad, peso, empresa} = superheroe;
console.log(`${nombre} tiene ${edad} años, pesa ${peso} kg y es de ${empresa}`);
//salida: "Capitan América tiene 30 años, pesa 100 kg y es de Marvel"
```

Así de fácil podríamos reescribir el código usando desestructuración.

## Conclusiones 📎

- La desestructuración nos permite dividir las propiedades de un objeto en variables independientes para mejorar la legibilidad del código.
- Bastante útil en frameworks y librerías frontend como React.js.
- La desestructuración no se basa en el orden de las propiedades, sino el el nombre de las llaves del objeto en cuestión.
- Es posible escribir una desestructuración con las llaves desordenadas.