
ES6
ROCKS

DESTRUCTURING

DESTRUCTURING

Destructuring es una forma conveniente de extraer múltiples valores de objetos y/o arrays (pueden estar anidados).

```
// OBJECT DESTRUCTURING
const persona = { nombre: 'Ada', apellido: 'Lovelace' };
const { nombre: n, apellido: a } = persona; // n = 'Ada'; a = 'Lovelace'


  // {prop} es abreviado de {prop: prop}
const { nombre, apellido } = persona; // nombre = 'Ada'; apellido = 'Lovelace'


// ARRAY DESTRUCTURING
const unIterable = ['a', 'b'];
const [x, y] = unIterable; // x = 'a'; y = 'b'
```

DESTRUCTURING: CONSTRUCCIÓN DE DATOS VS. EXTRACCIÓN DE DATOS

Para terminar de entender destructuring, examinemos el contexto más amplio.
JS tiene operaciones para construir datos, una propiedad a la vez.

```
const persona = {};
persona.nombre = 'Ada';
persona.apellido = 'Lovelace';
```

La misma sintaxis puede utilizarse para extraer datos

```
const nombre = persona.nombre;
const apellido = persona.apellido;
```

DESTRUCTURING: CONSTRUCCIÓN DE DATOS VS. EXTRACCIÓN DE DATOS

Además, hay una sintaxis para construir múltiples propiedades al mismo tiempo, a través de un objeto literal.

```
const persona = { nombre: 'Ada', apellido: 'Lovelace' };
```

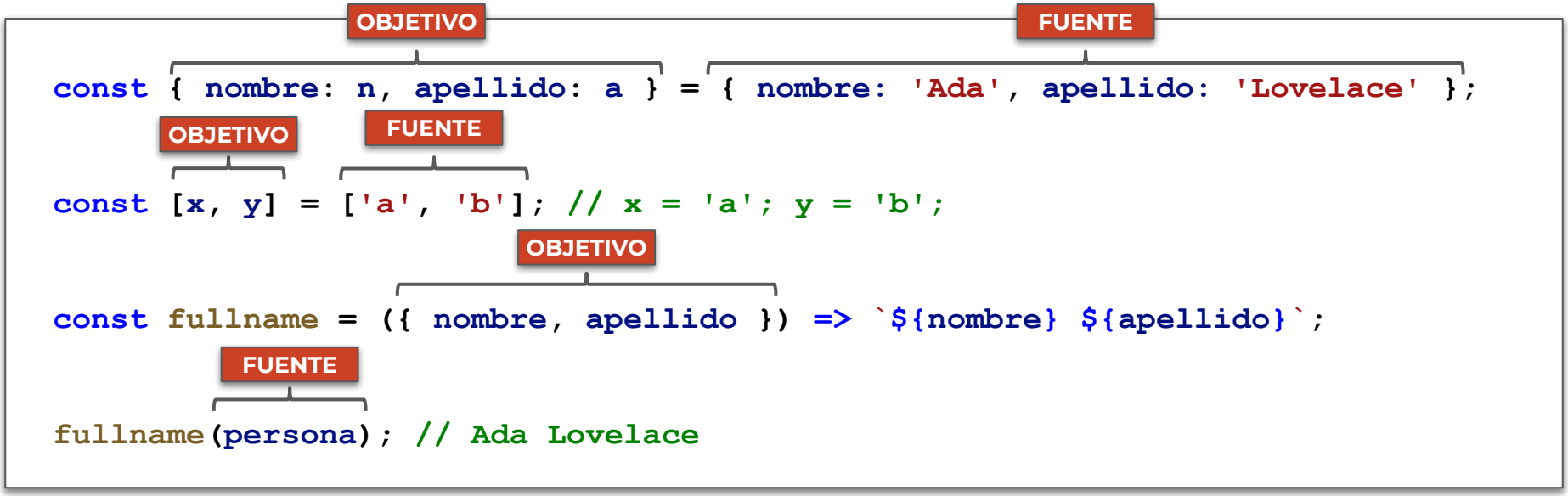
Pero antes de **ES6**, no había un mecanismo correspondiente para extraer datos. Eso es lo que hace **destructuring**. Permite extraer múltiples propiedades de un objeto:

```
const { nombre: n, apellido: a } = persona;

// con Arrays

const [x, y] = ['a', 'b']; // x = 'a'; y = 'b'
```

DESTRUCTURING: PARTES

En la asignación por destructuring tenemos dos partes involucradas:

- **Fuente del destructuring**: la información a desestructurar (la parte derecha de una asignación o los argumentos que pasamos a una función).
- **Objetivo del destructuring**: el patrón utilizado para desestructurar la fuente (la parte izquierda de una asignación o los parámetros de una función).



DESTRUCTURING: PARTES

El objetivo del destructuring puede ser alguno de los siguientes patrones:

- Una variable

```
persona
```

- Un patrón de objeto

```
{ nombre: <<patrón>>, apellido: <<patrón>> }
```

- Un patrón de array

```
[ <<patrón>>, <<patrón>> ]
```

DESTRUCTURING: EXTRAER SOLO LO NECESARIO

Si estamos desestructurando un objeto, podemos elegir sólo propiedades que nos interesan

```
const persona = {
 nombre: 'Ada',
 apellido: 'Lovelace'
};

const { nombre } = persona; // SIN cambio nombre de variable
const { nombre: nombrePersona } = persona; // Cambio nombre de variable

console.log( nombrePersona ); // Ada
console.log( nombre ); // Ada
```

DESTRUCTURING: EXTRAER SOLO LO NECESARIO

Si estamos desestructurando un array, podemos extraer solo un prefijo

```
const datos = ['Grace', 'Murray', 'Hopper'];


// solo tomo el primer y segundo elemento
const [ primerNombre, segundoNombre ] = datos;


console.log(primerNombre); // Grace
console.log(segundoNombre); // Murray
```

DESTRUCTURING: EXTRAER SOLO LO NECESARIO

Si estamos desestructurando un array, y necesitamos acceder solo a una posición específica, ignorando otras posiciones, podemos dejar el espacio en blanco. Por ejemplo, vamos a extraer sólo el 3er elemento del array.

```
const datos = ['Ada Lovelace', 'Jude Milhon', 'Hedy Lamarr', 'Grace Hopper',
'Evelyn Berezin'];


// solo tomo el tercer elemento
const [ , , tercerElemento ] = datos;


console.log(tercerElemento); // Hedy Lamarr
```

DESTRUCTURING: EJERCICIOS

1. Crear una función (*arrow function*) llamada **mostrarInfo** que un objeto y retorna la cadena “Tu nombre completo es ” concatenado con el valor de la propiedad **nombre**, un espacio y el valor de la propiedad **apellido**. Tenés que usar **destructuring** dentro de la función.

```
const persona = { nombre: 'Ada', apellido: 'Lovelace', edad: 35 };


const mostrarInfo = ; // ACA LA SOLUCION


console.log( mostrarInfo(persona) );
// Tu nombre completo es Ada Lovelace
```

DESTRUCTURING: EJERCICIOS

2. Utilizando **destructuring**, debemos extraer la propiedad ***nombre*** de cada objeto y crear un array en la variable ***nombres*** con los 3 nombres que obtuvimos de cada persona.

```
const persona1 = { nombre: 'Ada', apellido: 'Lovelace', edad: 35 };
const persona2 = { nombre: 'Grace', apellido: 'Hopper', edad: 25 };
const persona3 = { nombre: 'Hedy', apellido: 'Lamarr', edad: 45 };


const nombres = [];


console.log( nombres );
// [ 'Ada', 'Grace', 'Hedy' ]
```

DESTRUCTURING: EJERCICIOS

3. Tenemos un array **datosPersonales**, con el nombre, apellido, día nacimiento, mes nacimiento, año nacimiento, lugar nacimiento (respetando el orden) de una persona. Utilizando **destructuring**, obtener los datos necesarios para completar el siguiente código.

```
const datosPersonales = [ 'Ada', 'Lovelace', 10, 12, 1815, 'Londres' ];


// utilizando destructuring
// extraer las variables nombre, lugar, dia, mes, anio


console.log(`${nombre} nació en ${lugar}, el ${dia}/${mes}/${anio}.`);
// Ada nació en Londres, el 10/12/1815.
```

DESTRUCTURING: OBJETOS ANIDADOS

Los patrones a utilizar para hacer **destructuring** pueden ser más o menos complejos, según la estructura que tenga la fuente.

```
const persona = {
  nombre: 'Ada Lovelace',
  domicilio: {
    calle: 'Calle Falsa 123', localidad: 'CABA'
  }
};
const { nombre, domicilio: { localidad } } = persona;


console.log(`${nombre} vive en ${localidad}`); // Ada Lovelace vive en CABA
```

DESTRUCTURING: ARRAYS ANIDADOS

Otro ejemplo de patrones anidados, utilizando arrays.

```
const color = ['#FF00FF', [255, 0, 255], 'rgb(255, 0, 255)'];

// usamos destructuring anidado para asignar las variable R, G y B
const [hexa, [R, G, B]] = color;

console.log(hexa, R, G, B); // #FF00FF 255 0 255
```

DESTRUCTURING: ARRAY Y OBJETOS COMBINADOS

A veces trabajamos con estructuras más complejas, y (por ejemplo) tenemos un objeto con otro objeto y un array anidados. En estos casos podemos usar destructuring combinado.

```
const persona = {
 nombre: 'Ada Lovelace',
 lenguajes: ['Español', 'Inglés']
}


const { lenguajes } = persona;
const { lenguajes: [ lenguajePrincipal ] } = persona;


console.log(lenguajes); // [ 'Español', 'Inglés' ]
console.log(lenguajePrincipal); // Español
```

DESTRUCTURING: ARRAY Y OBJETOS COMBINADOS

En otro ejemplo, podemos tener un array con objetos y queremos obtener datos únicamente del primero.

```
const personas = [
 { nombre: 'Ada Lovelace', lenguajes: ['Español', 'Inglés'] },
 { nombre: 'Grace Hopper', lenguajes: ['Inglés'] }
]

const [ { nombre: nombrePrimeraPersona } ] = personas;

console.log(nombrePrimeraPersona); // Ada Lovelace
```

DESTRUCTURING: VALORES POR DEFECTO

Cuando estamos desestructurando y queremos acceder a una propiedad que no existe, obtenemos el siguiente resultado

```
const configuracion = {
  velocidad: 150
};
const { velocidad, ancho, alto: height } = configuracion;


console.log(velocidad, ancho, height); // 150 undefined undefined
```

DESTRUCTURING: VALORES POR DEFECTO

Con destructuring podemos configurar valores por defecto, por lo que si una propiedad no está en el objeto (o el array), se le asigna el valor que le indiquemos.

```
const configuracion = {
 velocidad: 150
}
const { velocidad = 750, ancho = 500, alto: height = 400 } = configuracion;


console.log(velocidad); // 150, toma el valor que está en el objeto
console.log(ancho); // 500, toma el valor que le indicamos como default
console.log(height); // 400, toma el valor que le indicamos como default
```

DESTRUCTURING: VALORES POR DEFECTO

Con array funciona de manera similar, según la posición del elemento que necesitemos ponerle un valor por defecto.

```
const rgb = [200];
// indicamos un valor por defecto de 255 a r y b
const [r = 255, g, b = 255] = rgb;


console.log(`R: ${r}, G: ${g}, B: ${b}`); // R: 200, G: undefined, B: 255
// R toma el valor que se corresponde con su posición
// G es undefined ya que no hay elemento en esa posición y no tiene valor
default
// B es 255 ya que no existe el elemento y toma su valor por defecto
```

DESTRUCTURING: EJERCICIOS

4. Tenemos un objeto **persona**, con sus datos personales. Utilizando **destructuring**, obtener los datos necesarios para completar el siguiente código.

```
const persona = {
 nombre: 'Ada', apellido: 'Lovelace',
 fechaNacimiento: [10, 12, 1815],
 lugarNacimiento: {
   pais: 'Inglaterra', ciudad: 'Londres'
 }
};


console.log(`${nombre} nació en ${lugar}, el ${dia}/${mes}/${anio}.`);
// Ada nació en Londres, el 10/12/1815.
```

DESTRUCTURING: EJERCICIOS

5. Utilizando **destructuring**, tenemos que extraer la información necesaria para completar el siguiente código.

```
const persona = ['Ada Lovelace', [ { edad: 30, idioma: 'inglés' } ]];


// usando destructuring, obtener las variables idioma y nombrePersona


console.log(`${nombrePersona} habla en ${idioma}.`);
// Ada Lovelace habla en inglés.
```

DESTRUCTURING: EJERCICIOS

6. Utilizando **destructuring**, tenemos que extraer la información necesaria para completar el siguiente código.

```
const persona1 = { nombre: 'Ada Lovelace', techs: ['JavaScript', 'C++'] };
const persona2 = { nombre: 'Grace Hopper', techs: [] };

console.log(`${nombrePersona1} programa en ${techPersona1}.`);
console.log(`${nombrePersona2} programa en ${techPersona2}.`);

// Ada Lovelace programa en JavaScript.
// Grace Hopper programa en JS.
```

DESTRUCTURING: REST OPERATOR

El operador **rest** permite guardar los elementos restantes (que no estamos extrayendo) en un array u objeto. Si se utiliza dentro de un array, tiene que ir último. El operador se escribe utilizando 3 puntos **...**

```
const [x, ...y] = ['a', 'b', 'c']; // x='a'; y=['b', 'c']


const datosPersona = {
 nombre: 'Ada Lovelace', edad: 35, idioma: 'inglés'
};
const { nombre, ...otrosDatos } = datosPersona;


console.log(nombre); // Ada Lovelace
console.log(otrosDatos); // { edad: 35, idioma: 'inglés' }
```

DESTRUCTURING: REST OPERATOR

Si el operador no puede encontrar más elementos, va a guardar un array/objeto vacío.

```
const [x, y, ...z] = ['a', 'b'];
console.log(x, y); // a b
console.log(z); // [ ]

const datosPersona = {
 nombre: 'Ada Lovelace', edad: 35
};
const { nombre, edad, ...otrosDatos } = datosPersona;
console.log(nombre, edad); // Ada Lovelace 35
console.log(otrosDatos); // { }
```

¡Gracias!