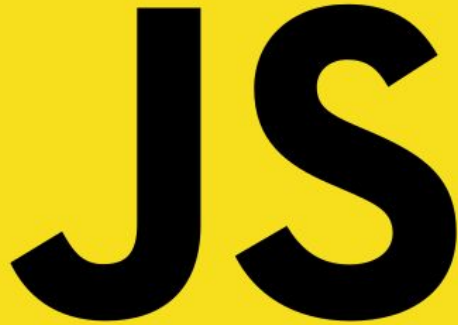


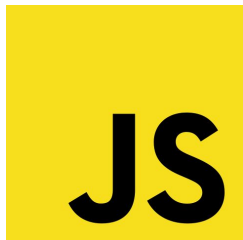
Arrays en Javascript



Un array es una colección de elementos.
En comparación con los objetos, los **arrays** solo tienen **valor**



```
1 // llave/valor
2 const product = {
3   name: 'T-shirt',
4   price: 50,
5   color: 'blue'
6 }
7
8 // Solamente tiene Valor
9 const myProduct = ['T-shirt', 50, 'blue'];
10 const Months = ['January', 'February', 'March'];
```

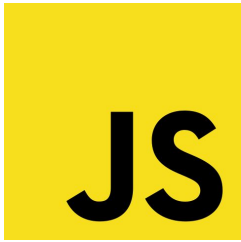


Arrays en Javascript



Los arrays también pueden guardar todo tipo de contenido:

Nótese que dentro de sus propiedades podemos guardar distintos tipos de datos, incluyendo arreglos u otros objetos.



Agregar y eliminar elementos de un objeto en Javascript



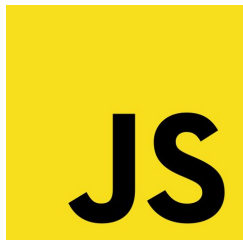
Puedo agregar elementos al objeto de manera manual, pero también probablemente desee agregarlos desde la ejecución. De igual modo tenemos un método para la eliminación de propiedades en el objeto.



```
1 // Eliminar uno de los datos del objeto
2 delete product.code;
```



```
1 // Agregar elementos al objeto
2 product.image = 'Imagen.jpg';
```



Destructuring de objetos en Javascript



Normalmente cuando deseo un elemento del objeto, sería de la siguiente forma:

objeto.propiedad;

Pero no siempre lo tendremos que hacer de esta manera ya que puedo de algún modo usar cada valor de manera independiente con **destructuring**

```
12 // Forma clásica
13 let response = `Su producto deseado es: ${product.detail} y cuesta ${product.price}`;
14 // Ayudándose de destructuring
15 let { detail, price } = product;
16 let response2 = `Su producto deseado es: ${detail} y cuesta ${price}`;
17
18 console.log(response);
19 console.log(response2);
20
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

bash + v [] [] ^ x

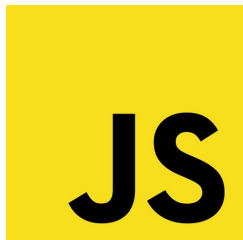
```
usuario@edwin-rozo:~/Documents/Rozo_Dev/Javascript/Basic/Javascript-Basic-2021/5.0
```

```
bjetos$ node Objetos.js
```

```
Su producto deseado es: T-shirt y cuesta 50USD
```

```
Su producto deseado es: T-shirt y cuesta 50USD
```

Edwin Rozo Gómez -
edwin.rozo1@misena.edu.co



Destructuring de objetos en Javascript



Normalmente cuando deseo un elemento del objeto, sería de la siguiente forma:

objeto.propiedad;

Pero no siempre lo tendremos que hacer de esta manera ya que puedo de algún modo usar cada valor de manera independiente con **destructuring**

```
12 // Forma clásica
13 let response = `Su producto deseado es: ${product.detail} y cuesta ${product.price}`;
14 // Ayudándose de destructuring
15 let { detail, price } = product;
16 let response2 = `Su producto deseado es: ${detail} y cuesta ${price}`;
17
18 console.log(response);
19 console.log(response2);
20
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

bash + - [] [] ^ x

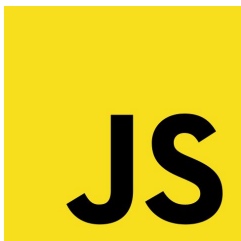
```
usuario@edwin-rozo:~/Documents/Rozo_Dev/Javascript/Basic/Javascript-Basic-2021/5.0
```

```
bjetos$ node Objetos.js
```

```
Su producto deseado es: T-shirt y cuesta 50USD
```

```
Su producto deseado es: T-shirt y cuesta 50USD
```

Edwin Rozo Gómez -
edwin.rozo1@misena.edu.co

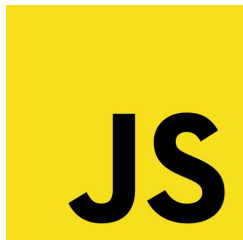


las propiedades de un objeto declarado const si se pueden modificar.



```
1  const user = {
2    name: 'Edwin',
3    age: 34,
4    available: true
5  }
6
7  console.log(user);
8
9  user.name = 'Gustavo';
10 //Detalles que no gustan mucho en js, es la posibilidad de cambiar de tipo de dato dentro de la misma variable.
11 user.age = 35;
12 user.available = false;
13
14 console.log(user);
15
```

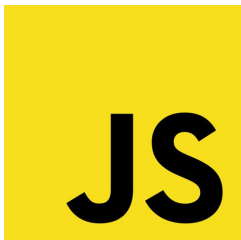
```
{ name: 'Edwin', age: 34, available: true }
{ name: 'Gustavo', age: 35, available: false }
```



Métodos para objetos



“use strict” -	Habilitar el modo estricto, me permite usar ciertos métodos para objetos.
Object.freeze(myObject)	Congelar el objeto para que no pueda ser modificado
Object.isFrozen(myObject)	Comprueba si el objeto está congelado o no.
Object.seal(myObject)	Modificar llaves del objeto, pero no eliminarlas
Object.isSealed(myObject)	comprobar si el objeto está sellado.



Unir 2 objetos



Algo que usaremos en algunos casos es unir dos objetos.

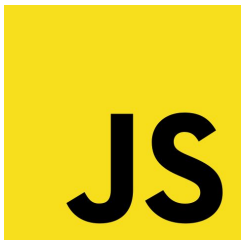
Ejemplo: objeto1 es **product** y objeto2 es **user**



```
1  const user = {  
2    name: 'Edwin',  
3    age: 34,  
4    available: true  
5  }
```



```
1  let product = {  
2    code: 001,  
3    detail: 'T-shirt',  
4    price: '50USD',  
5    availableColours: ['Blue', 'Black', 'White', 'Yellow', 'Red', 'Green'],  
6    stock: {  
7      adidas: 50,  
8      nike: 80  
9    }  
10 }
```

Unir 2 objetos

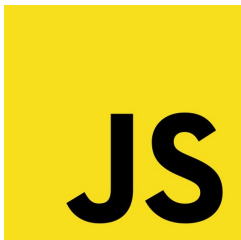


Algo que usaremos en algunos casos es unir dos objetos.

Ejemplo: objeto1 es **product** y objeto2 es **user**
con la sintaxis **Object.assign(product,user)** uniremos los dos objetos.



```
1 // Uniendo dos objetos
2 const newItem = Object.assign(product,user);
3 console.log(newItem);
```



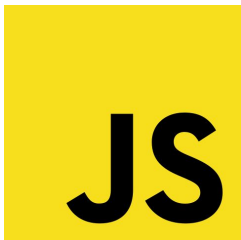
Unir 2 objetos



Como resultado:

```
1 // Uniendo dos objetos
2 const newItem = Object.assign(product,user);
3 console.log(newItem);
```

```
{
  code: 1,
  detail: 'T-shirt',
  price: '50USD',
  availableColours: [ 'Blue', 'Black', 'White', 'Yellow', 'Red', 'Green'
],
  stock: { adidas: 50, nike: 80 },
  name: 'Edwin',
  age: 34,
  available: true
}
```



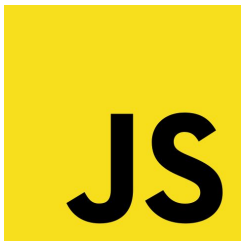
Unir 2 objetos



Podemos también añadir los dos objetos, utilizando spread operation(...)



```
1  let uniendoObjetos = {...product,...user};  
2  console.log(uniendoObjetos);
```



Usando this en los objetos



La propiedad **this** tiene como finalidad dar contexto a determinado valor dentro del objeto. Tal como se ve en el ejemplo:

```
1  const product = {  
2    nombre: 'camisa',  
3    code: '001',  
4    price: 10,  
5    resumen: function () {  
6      console.log(`El producto ${this.nombre} tiene un precio de ${this.price}`);  
7    }  
8  }
```