Referencias de objetos y copia

Fuente: https://es.javascript.info/

Una de las diferencias fundamentales entre objetos y primitivos es que los objetos son almacenados y copiados “por referencia”, en cambio los primitivos: strings, number, boolean, etc.; son asignados y copiados “como un valor completo”.

Esto es fácil de entender si miramos un poco “bajo cubierta” de lo que pasa cuando copiamos por valor.

Empecemos por un primitivo como string.

Aquí ponemos una copia de message en phrase:

```
let message = "Hello!";
let phrase = message;
```

Como resultado tenemos dos variables independientes, cada una almacenando la cadena "Hello!".

Bastante obvio, ¿verdad?

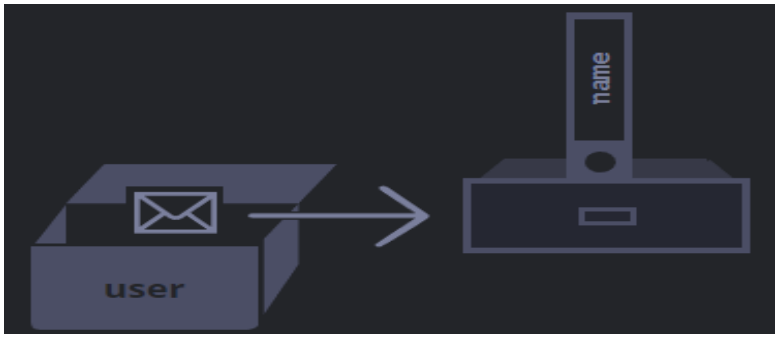
Los objetos no son así.

Una variable no almacena el objeto mismo sino su “dirección en memoria”, en otras palabras “una referencia” a él.

Veamos un ejemplo de tal variable:

```
let user = {
  name: "John"
};
```

Y así es como se almacena en la memoria:



El objeto es almacenado en algún lugar de la memoria (a la derecha de la imagen), mientras que la variable user (a la izquierda) tiene una “referencia” a él.

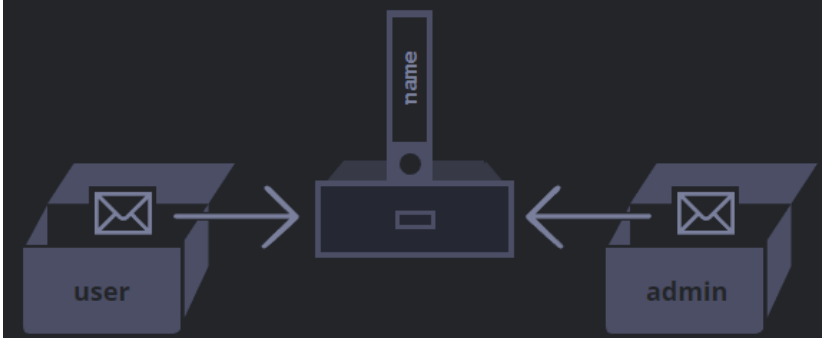
Podemos pensar de una variable objeto, como user, como una hoja de papel con la dirección del objeto escrita en ella.

Cuando ejecutamos acciones con el objeto, por ejemplo tomar una propiedad user.name, el motor JavaScript busca aquella dirección y ejecuta la operación en el objeto mismo.
Ahora, por qué esto es importante.
Cuando una variable de objeto es copiada, se copia solo la referencia. El objeto no es duplicado.
Por ejemplo:

```
let user = { name: "John" };

let admin = user; // copia la referencia
```

Ahora tenemos dos variables, cada una con una referencia al mismo objeto:



Como puedes ver, aún hay un objeto, ahora con dos variables haciendo referencia a él.

Podemos usar cualquiera de las variables para acceder al objeto y modificar su contenido:

```
let user = { name: 'John' };

let admin = user;

admin.name = 'Pete'; // cambiado por la referencia "admin"

alert(user.name); // 'Pete', los cambios se ven desde la referencia "user"
```

Es como si tuviéramos un gabinete con dos llaves y usáramos una de ellas (admin) para acceder a él y hacer cambios. Si más tarde usamos la llave (user), estaríamos abriendo el mismo gabinete y accediendo al contenido cambiado.

Comparación por referencia

Dos objetos son iguales solamente si ellos son el mismo objeto.

Por ejemplo, aquí a y b tienen referencias al mismo objeto, por lo tanto son iguales:

```
let a = {};
let b = a; // copia la referencia

alert( a == b ); // true, verdadero. Ambas variables hacen referencia al mismo objeto
alert( a === b ); // true
```

Y aquí dos objetos independientes no son iguales, aunque se vean iguales (ambos están vacíos):

```
let a = {};
let b = {}; // dos objetos independientes

alert( a == b ); // false
```

Para comparaciones como obj1 > obj2, o comparaciones contra un primitivo obj == 5, los objetos son convertidos a primitivos. Estudiaremos cómo funciona la conversión de objetos pronto, pero a decir verdad tales comparaciones ocurren raramente y suelen ser errores de código.

IMPORTANTE: Los objetos
Un efecto importante de almacenar objetos como referencias es que un objeto declarado como const puede ser modificado.
Por ejemplo:

```
const user = {
  name: "John"
};

user.name = "Pete"; // (*)

alert(user.name); // Pete
```

Puede parecer que la línea (*) causaría un error, pero no lo hace. El valor de user es constante, este valor debe siempre hacer referencia al mismo objeto, pero las propiedades de dicho objeto pueden cambiar.
En otras palabras: const user da un error solamente si tratamos de establecer user=... como un todo.
Dicho esto, si realmente necesitamos hacer constantes las propiedades del objeto, también es posible, pero usando métodos totalmente diferentes.