

EXERCISES QUE TRABAJAREMOS EN EL CUE

- EXERCISE 1: OBJETOS - ¿CÓMO SON? Y ¿CÓMO CREAMOS?
- EXERCISE 2: USO DE UN CONSTRUCTOR DE OBJETOS EN JAVASCRIPT.

EXERCISE 1: OBJETOS - ¿CÓMO SON? Y ¿CÓMO CREAMOS?

Hasta el momento, hemos aprendido a utilizar funciones, y como almacenar información en variables. Los objetos en JavaScript nos permiten agrupar estos dos conceptos en una unidad.

Recordemos que, según el paradigma de la programación orientada a objetos, la “unidad” u objeto debe modelar uno de la vida real, el cual puede tener características y habilidades propias que, al ser llevadas a código, se convierten en *propiedades* (variables) y *métodos* (funciones).

Considera el siguiente ejemplo que muestra cómo se crea un objeto.

El objeto lo almacenamos en una variable con el nombre que lo define, en este caso, empleamos nombres genéricos para identificar sus diferentes partes.

Entre llaves, encontraremos las distintas propiedades y métodos que definen a nuestro objeto. Dichas propiedades cumplen el objeto de almacenar información, y tal como una variable, pueden contener todos los tipos de datos que podemos usar en JS. Aparte de éstas, el objeto también puede contener métodos o funciones específicas. La sintaxis de las funciones dentro de un objeto, difiere de la habitual. Dentro de un objeto, las funciones se declaran partiendo con el nombre, en este caso, utilizamos uno genérico de “método”, seguido de dos puntos, y la palabra clave **function**. Después, vienen unos paréntesis, en donde podremos pasar valores por parámetro.

Es importante destacar que, para añadir propiedades o métodos, debemos colocar las comas correspondientes después de cada atributo.

```
1 // Objeto
2 var objeto = {
3     //Propiedades
4     propiedad1: "String",
5     propiedad2: 123,
6     propiedad3: false,
7
8     //Métodos
9     metodo: function () {
```

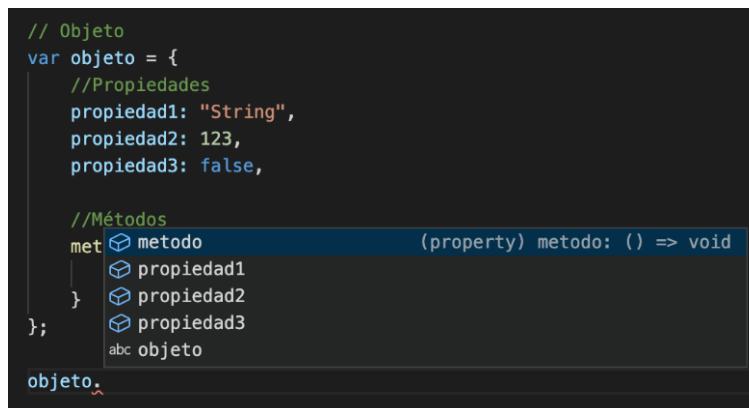
```
10         //cuerpo de funcion
11     }
12 };
```

Recalcando el punto, las propiedades son variables que pueden almacenar cualquier tipo, y los métodos simplemente son funciones dentro de un objeto. Esta forma de crear objetos en JS se llama: creación literal de objetos.

LLAMADA A OBJETOS O SUS ATRIBUTOS

Solo hemos revisado como declarar los objetos. Ahora, debemos llamarlos para poder usarlos. Para esto, basta con escribir su nombre. En este ejemplo, ese nombre es: "objeto".

Se puede acceder a las propiedades de un objeto con una simple notación de puntos. La siguiente imagen muestra que, al momento de escribir el nombre de nuestro objeto, seguido por un ".", el IDE, en este caso, empieza a mostrarnos todos los métodos y propiedades de dicho objeto a las cuales podemos acceder:



```
// Objeto
var objeto = {
    //Propiedades
    propiedad1: "String",
    propiedad2: 123,
    propiedad3: false,

    //Métodos
    metodo: () => void
};

objeto.
```

Incluso, podemos verificar que realmente se está accediendo a las propiedades, mediante un `console.log()`.

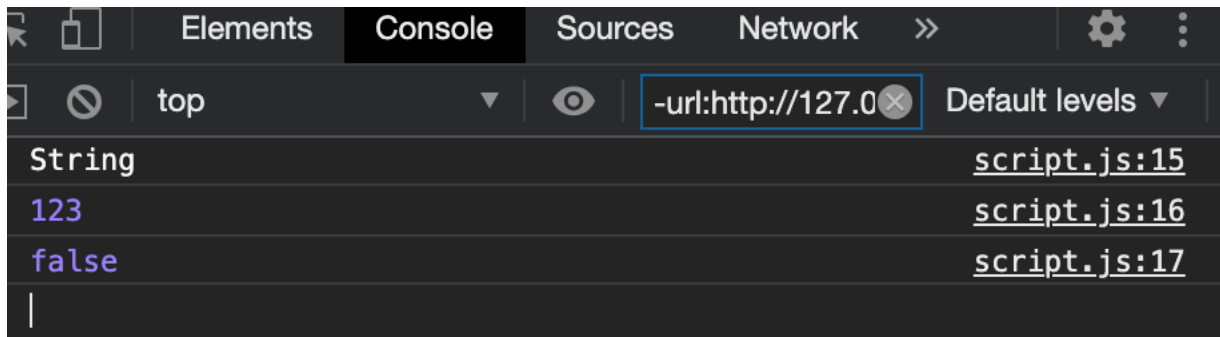
(En un punto aparte, cada vez que utilizamos un `console.log()`, estamos llamando al método `log()` del objeto `console`. ¡Sin haber notado que ya estábamos utilizando objetos en JavaScript!).

Mediante las siguientes líneas de código, verificaremos si realmente se puede acceder a los atributos de un objeto:

```
1 console.log(objeto.propiedad1);
```

```
2 console.log(objeto.propiedad2);  
3 console.log(objeto.propiedad3);
```

Por consola, el resultado es el siguiente:



Como se puede observar, se logra acceder a las tres propiedades de nuestro objeto "objeto". De momento, no accedemos al método pues lo hemos dejado vacío, pero al agregarle una funcionalidad, se puede acceder de la siguiente manera:

Funcionalidad nueva:

```
1 //Métodos  
2 metodo: function () { // Le agregamos funcionalidad al metodo.  
3     alert('Este mensaje proviene de un metodo de un objeto')  
4 }
```

Cómo acceder al método:

```
1 objeto.metodo();
```

El resultado en un navegador:

This page says

Este mensaje proviene de un metodo de un objeto

OK

CREACIÓN DE OBJETOS

Hemos visto cómo crear un objeto de manera literal, pero existe más de una forma hacerlo:

- Con la palabra clave `"new"`.
- Con la función `objeto.create()`.

A continuación, se demuestra cómo producir objetos mediante la palabra clave `"new"`. Su declaración consiste en declarar una variable con el nombre del objeto, y hacer la equivalencia a una nueva instancia del objeto `"Object()"`. Posteriormente, se pueden definir atributos y funciones usando la notación de `"."`:

```
1 var persona = new Object();  
2 persona.nombre = "Milton";  
3 persona.colorDeOjos = 'café';  
4 persona.edad = 5;
```

Si se requiere producir otro objeto que tenga las mismas características que éste, podemos emplear el método `assign()` de la clase `Object`. A continuación, demostraremos eso y, además, otorgaremos un atributo propio, solo al de la segunda instancia:

```
1 //tambien podemos introducir un objeto "esquema" como parametro de  
2 Objeto():  
3 var persona1 = Object.assign({}, persona);  
4 persona1.colorFavorito = 'verde';  
5  
6 console.log(persona)  
7 console.log(persona1)
```

Mediante la consola, obtenemos el siguiente resultado. En rojo aparece el objeto base, y en azul la nueva instancia que contiene los mismos atributos que dicho objeto base, pero con su propio atributo indicado con la flecha:

```
▼ {nombre: "Milton", colorDeOjos: "cafe", edad: 5} ⓘ
  colorDeOjos: "cafe"
  edad: 5
  nombre: "Milton"
  ▶ __proto__: Object
```

persona

```
▼ {nombre: "Milton", colorDeOjos: "cafe", edad: 5, colorFavorito: "verde"} ⓘ
  colorDeOjos: "cafe"
  colorFavorito: "verde"
  edad: 5
  nombre: "Milton"
  ▶ __proto__: Object
```

persona1



Otra manera de producir objetos en JavaScript, es mediante la función `Object.create()`. Según la documentación acerca de esta función otorgada por el mismo editor de texto, `create()` recibe por parámetro un objeto que puede ser usado como base para crear nuevas instancias, o con un objeto vacío que sirve para crear nuevos objetos.

A continuación, se observa la creación de un nuevo objeto, en donde el método `create()` recibe un objeto vacío, y posteriormente le agregamos un atributo `"nombre"`:

```
1 var obj = Object.create({});
2 obj.nombre = 'cualquier nombre';
```

Ahora, utilizaremos éste como base para crear un segundo objeto. Para hacerlo, debemos pasar el objeto como parámetro del método `create()`:

```
1 //obj2 tiene el mismo prototipo o características de obj
2 var obj2 = Object.create(obj);
```

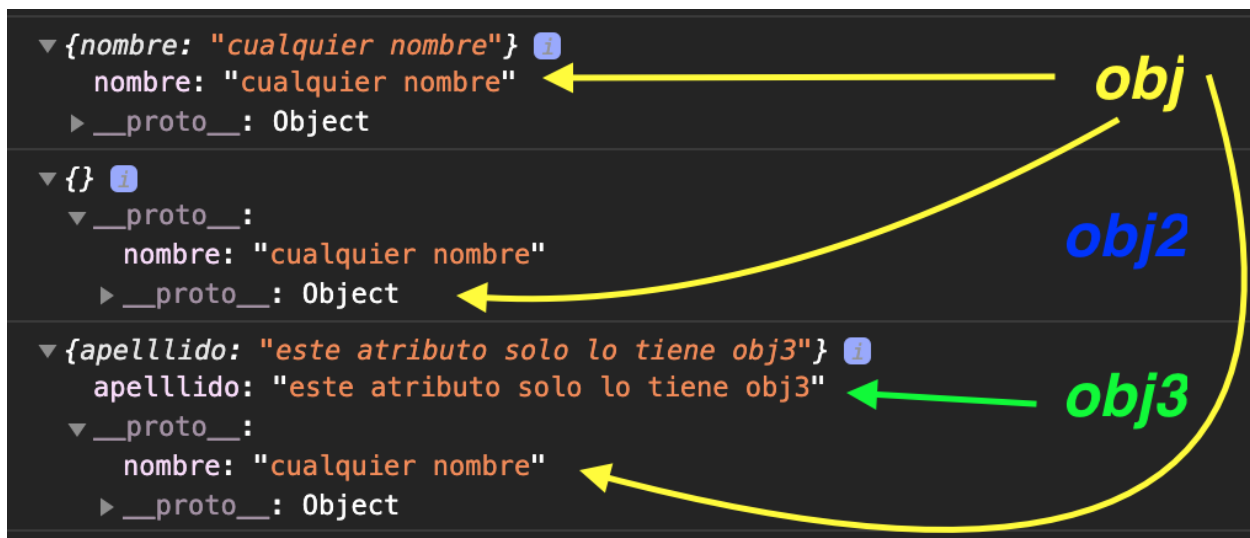
`obj2` ahora fue creado exactamente con las mismas características que `obj`, pero podemos agregarle sus propios atributos de la siguiente forma (note que estamos creando otro objeto, usando como base el primero, y a éste le agregamos el atributo apellido):

```
1 //obj3 tiene el mismo prototipo o características de obj pero con atributos
2 propios
3 var obj3 = Object.create(obj);
4 obj3.apelllido = 'este atributo solo lo tiene obj3';
```

Ahora, realizamos unos `console.log()` para ver el resultado.

```
1 console.log(obj)
2 console.log(obj2)
3 console.log(obj3)
```

La consola muestra tres objetos. El de color amarillo es el objeto base, y las flechas amarillas indican como los otros objetos obtuvieron la misma propiedad desde éste. Los objetos azul y verde, son objetos hijos que heredan las mismas propiedades, pero en cuanto al objeto verde, éste tiene su propio atributo indicado con una flecha del mismo color.



Hemos analizado tres formas en las que podemos crear un objeto. Ahora, debemos revisar otra bastante relevante, que nos facilitará la creación de múltiples objetos a la vez: el constructor de objetos, que veremos en el siguiente Exercise.

EXERCISE 2: USO DE UN CONSTRUCTOR DE OBJETOS EN JAVASCRIPT

Hasta el momento, hemos aprendido tres formas en las que podemos crear objetos en JavaScript: de manera literal, con la palabra clave `new`, y con la función `Object()`. Cada uno de estos enfoques logra el mismo objetivo, pero todos tienen el mismo problema.

Por ejemplo, en el caso de usar `Object()`, resulta que cada vez que generamos una nueva instancia de un objeto, tenemos que cambiar manualmente las propiedades al objeto creado, pues tiene los mismos valores que el base. ¿Y si tuviéramos que crear cientos de objetos?: bajo este contexto, se volvería muy difícil.

Aquí es donde el constructor de objetos nos facilita el trabajo. Para evitar manipular valores de propiedades repetidos, podemos utilizar los constructores, y crear funciones personalizadas que nos permitan generar múltiples instancias del mismo objeto. Primero, creamos una función constructora, y luego empleamos la palabra clave "nueva" para obtener esas instancias.

La sintaxis de los constructores empieza con la palabra clave `function`, seguida por el *tipo de objeto* que deseamos crear, y entre paréntesis estableciendo los parámetros que vamos a poblar dentro de éste. Dentro de las llaves, encontramos atributos que se van a *reutilizar* en este objeto; con esto nos referimos a que cuando originamos un nuevo objeto, pasando por parámetros, los atributos que deseamos poblar y las variables establecidas en el constructor, sirven como bosquejo para poblar dichas instancias, y la palabra clave `this` indica que esas propiedades serán propias del objeto.

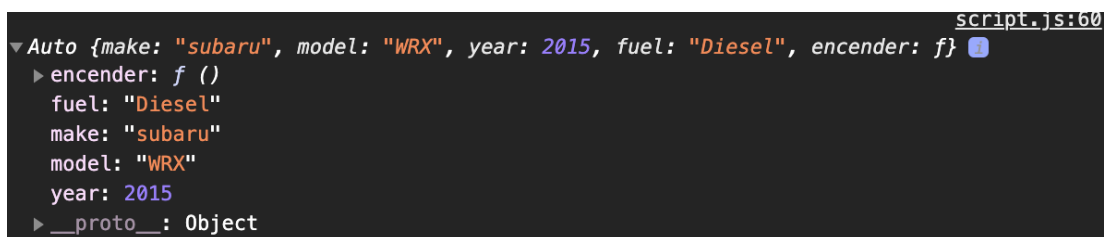
Se puede notar en el siguiente ejemplo, un constructor para un objeto de tipo auto, cuyos parámetros incluyen: la marca, el modelo, el año y el combustible. Dentro de las llaves utilizamos la palabra clave `this`, para indicar los valores: `"make"`, `"model"`, `"year"` & `"fuel"` (éstas no son claves, solo son las que corresponden a los parámetros, pero expresadas en inglés para establecer que no son los mismos), que servirán de "esquema" para crear nuevos objetos de tipo Auto.

```
1 function Auto(marca, modelo, anio, combustible) {  
2     this.make = marca;  
3     this.model = modelo;  
4     this.year = anio;  
5     this.fuel = combustible;  
6     this.encender = function () {  
7         console.log(`${marca} encendido.`)  
8     }  
9 }
```

Cuando creamos un nuevo objeto de tipo **auto**, debemos poblar sus atributos pasando los valores por parámetros. En cuanto a su función, viene por defecto, dado que son del mismo objeto. A continuación, vemos un ejemplo de este proceso, donde debemos poner la palabra clave **new** antes del constructor del tipo de objeto:

```
1 var subaru = new Auto('subaru', 'WRX', 2015, 'Diesel')
2
3 console.log(subaru);
```

El resultado en la consola muestra un objeto con todos los atributos y métodos que hemos definido en el constructor:



```
script.js:60
▼ Auto {make: "subaru", model: "WRX", year: 2015, fuel: "Diesel", encender: f()}
  ► encender: f ()
    fuel: "Diesel"
    make: "subaru"
    model: "WRX"
    year: 2015
  ► __proto__: Object
```

Se deben tener en cuenta los nombres de las propiedades del método que estaban en inglés, pues en la consola podemos ver que son las propias del objeto, y no simplemente unos marcadores, como lo fueron las propiedades que se definieron como parámetros del constructor. Si se requiere cambiar los valores de esas propiedades, debemos modificar el nombre de las que fueron encadenadas a la palabra clave **this**. Por ejemplo, si quisiéramos que todas las propiedades estuvieran en español, tendríamos que hacer la siguiente modificación al constructor:

```
1 function Auto(marca, modelo, anio, combustible) {
2   this.marca = marca;
3   this.modelo = modelo;
4   this.año = anio;
5   this.combustible = combustible;
6   this.encender = function () {
7     console.log(`${marca} encendido.`)
8   }
9 }
```

Ahora, vamos a crear un objeto utilizando el mismo constructor.


```
1 var toyota = new Auto(toyota, 'Supra', 2020, 'Diesel')
2
3 console.log(toyota);
```

Por consola vemos el siguiente resultado, el cual muestra ahora todas las propiedades en español:

```
▼ {marca: "toyota", modelo: "Supra", año: 2020, combustible: "Diesel", encender: f, ...} ⓘ
  año: 2020
  combustible: "Diesel"
  ▶ encender: f ()
    marca: "toyota"
    modelo: "Supra"
  ▶ venderAuto: f ()
  ▶ __proto__: Object
```

Hasta ahora hemos adquirido el conocimiento necesario para crear objetos en JavaScript, tomando nuestros primeros pasos en la programación orientada a objetos. Te recomendamos que intentes crear objetos, para familiarizarte con este nuevo paradigma.

De esta forma, ya hemos analizado exhaustivamente como declarar objetos, crear nuevas instancias, y primordialmente, hacer uso de los constructores de objetos para generar múltiples instancias de uno mismo.