EXERCISES QUE TRABAJAREMOS EN LA CUE

- EXERCISE 1: CONOCIENDO LAS SENTENCIAS CICLICAS DE CONTROL DE FLUJO
- EXERCISE 2: CONTROLANDO EXCEPCIONES CON TRY CATCH
- EXERCISE 3: CONOCIENDO LOS ARREGLOS DE JAVASCRIPT
- EXERCISE 4: CREANDO NUESTRO PRIMER OBJETO

EXERCISE 1: CONOCIENDO LAS SENTENCIAS CICLICAS DE CONTROL DE FLUJO

Existen distintas sentencias cíclicas de control de flujo en **JavaScript**. Estas nos permiten repetir una serie de instrucciones una y otra vez hasta que se cumpla cierta condición.

Comenzaremos hablando de la sentencia **for.**

FOR

For es una sentencia cíclica que crea un bucle que consiste en tres expresiones opcionales, encerradas en paréntesis y separadas por punto y coma.

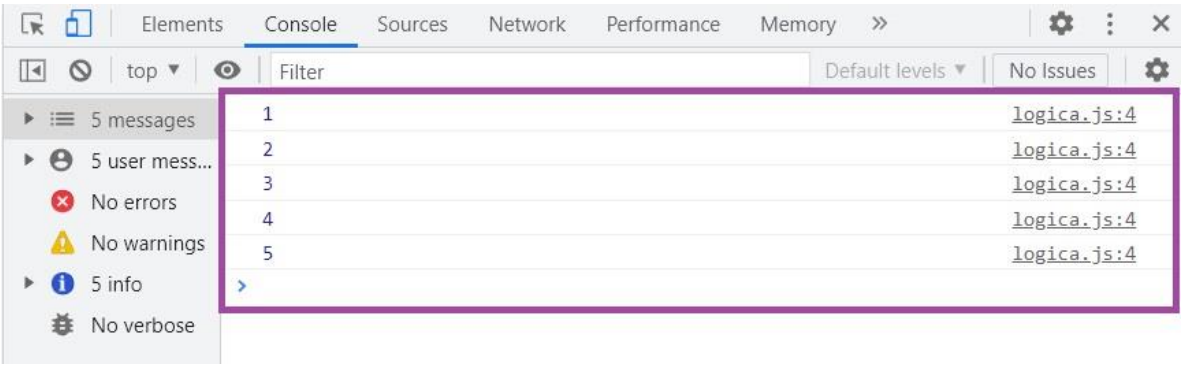
Cuenta con un valor de inicio, un valor final y una instrucción de aumento o decremento.

```
for (valor de inicio; valor de salida; incremento){
    instrucciones que se repetirán
}
```

Para comenzar a trabajar, debemos tener un archivo **HTML** y un archivo **JS** enlazado a él. Vamos a solicitar un número y la instrucción se repetirá la misma cantidad de veces que el número indicado.

```
1 var numero = parseInt(prompt("Ingrese un número"));
2
3 for(var inicio = 1; inicio <= numero; inicio ++){
4     console.log(inicio);
5 }
```

Si observamos nuestro código, indicamos una variable de nombre “inicio” cuyo valor es 1. Luego, como valor de salida, indicamos que el ciclo se repetirá siempre y cuando la variable “inicio” sea igual o menor al número ingresado por el usuario. Finalmente, indicamos que la variable “inicio” se incrementará de uno en uno, es decir, si el usuario ingresa 5, comenzará en 1 y terminará teniendo el valor 5.



Podemos observar que la línea impresa en las 5 ocasiones es siempre la línea 4 del archivo **lógica.js**.

Si deseamos leer más en detalle sobre el ciclo for podemos hacerlo en **mozilla.org**.

WHILE

While es una sentencia cíclica de control de flujo o bucle que se ejecuta mientras cierta condición se evalúe como verdadera. La condición siempre se evaluará antes de que se ejecute la sentencia por lo que, si se evalúa como falsa la primera vez, no se ejecutará el bloque de instrucciones dentro de **while** ninguna vez.

La sintaxis es la siguiente:

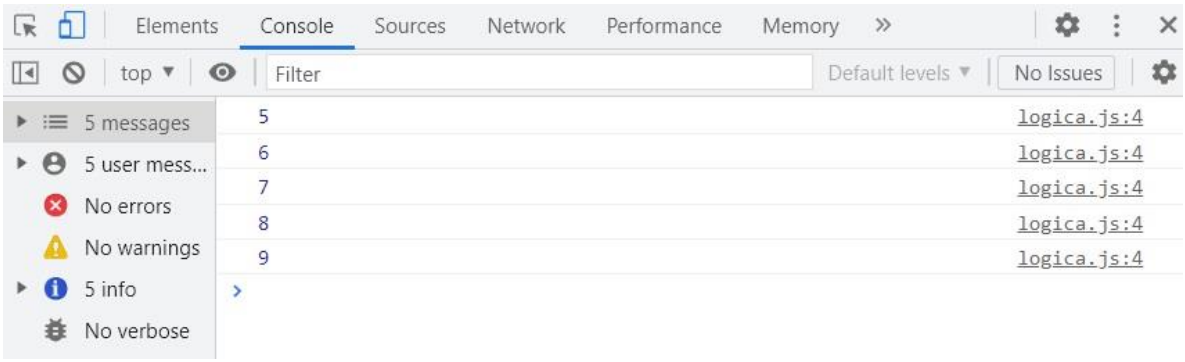
```
while (condición){
    instrucciones que se repetirán
}
```

Vamos a eliminar nuestra instrucción **for** y escribiremos la palabra reservada **while**. En el paréntesis escribiremos la condición, para este caso, evaluaremos que un número sea menor a 10. Se imprimirá el número mientras sea menor a 10.

Debemos tener la precaución de indicar un incremento, puesto que, si el número ingresado no se incrementa, entraremos en un bucle infinito.

```
var numero = parseInt(prompt("Ingrese un número"));

while(numero < 10){
    console.log(numero);
    numero = numero +1;
}
```

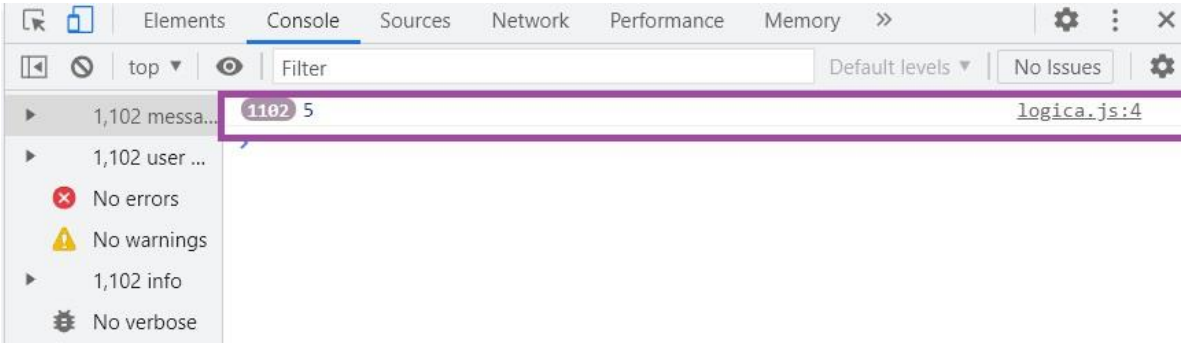
De esta forma, ingresando 5 obtenemos como resultado:



Es decir, se ingresó el número 5, evalúo el número y era menor a 10 así que se imprimió y se elevó en uno su valor. Volvió a evaluar, era menor, imprimió el valor, que era 6 y lo elevó en uno. Esto se repite hasta que la condición pasa a ser *false*, es decir, cuando deja de ser menor a 10.

Si ingresamos 10 u 11, simplemente no ingresará el flujo al ciclo **while**.

Si ingresamos un número menor a 10, pero olvidamos colocar la instrucción **numero = numero + 1**; entraremos en un ciclo infinito que no terminará jamás.



Podemos observar que ha ejecutado la misma instrucción 1102 veces.

Para obtener más información, podemos leer la documentación de while en **mozilla.org**.

DO WHILE

Do while es una sentencia cíclica conocida como “hacer mientras”. Crea un bucle que ejecuta una sentencia hasta que la condición evaluada se evalúe como falsa.

A diferencia de **while**, este ciclo primero ejecuta la instrucción y después evalúa la condición, por lo que incluso si la primera vez la condición no se cumplía, se va a ejecutar siempre al menos una vez.

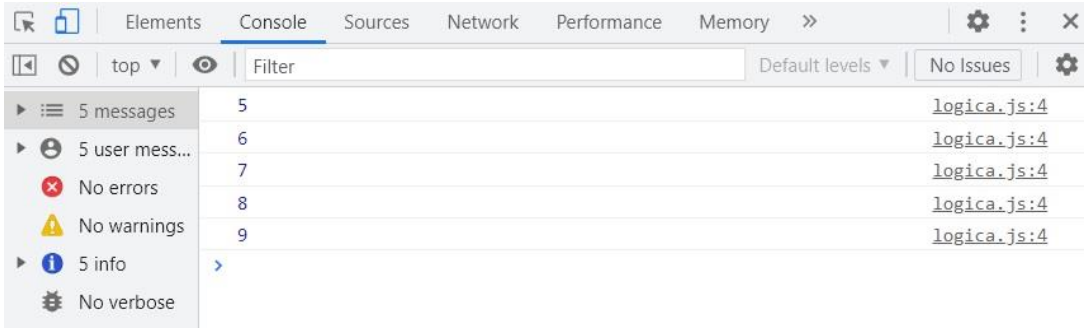
La sintaxis es la siguiente:

```
do {
    instrucciones que se repetirán
} while (condición);
```

Vamos a realizar el mismo ejemplo utilizando **do while**.

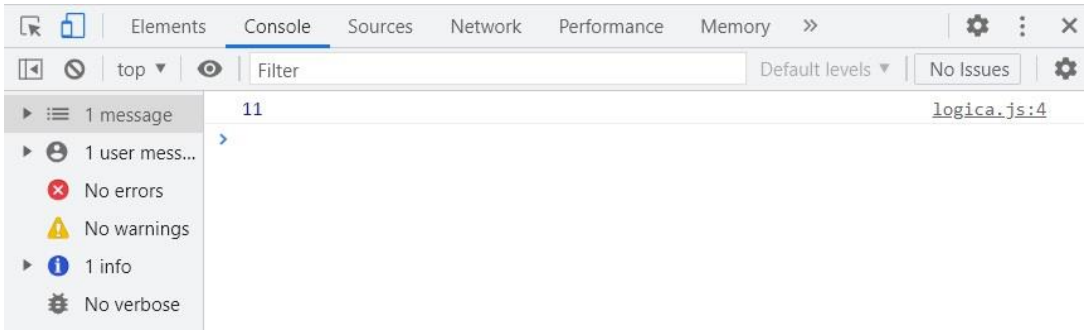
```
var numero = parseInt(prompt("Ingrese un número"));

do{
    console.log(numero);
    numero = numero +1;
}while (numero < 10);
```

Obteniendo como resultado:



Si no colocamos el incremento, entraremos en un ciclo infinito también.

Ingresemos 11 como número para ver que ocurre:



Podemos ver que ingreso de igual forma al ciclo, imprimió el número y posteriormente evaluó la condición. Al no cumplirse dejo de ejecutar el ciclo.

Para profundizar sobre el ciclo **do while** podemos leer la documentación de do while en **mozilla.org**.

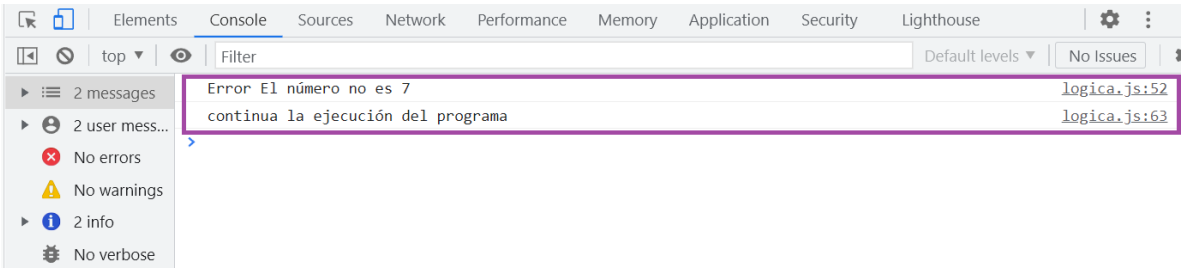
EXERCISE 2: CONTROLANDO EXCEPCIONES CON TRY CATCH

Try Catch es un bloque de instrucciones que declara una instrucción para cuando todo sale bien, es decir, para intentar que el código se ejecute correctamente (**try**) y un bloque de instrucciones para ejecutarse cuando ocurra una excepción (**catch**).

Vamos a declarar una variable llamada “número” con el valor 10 y vamos a generar una excepción si el número es distinto a 7.

```
var numero = 10;
try {
    if(numero != 7) throw new Error("El número no es 7");
} catch (error) {
    console.log(error.name, error.message)
}
console.log("continua la ejecución del programa")
```

Dándonos como resultado que, cada vez que el número ingresado sea distinto a 7, se captura la excepción y el programa continúa sin problema:



Si el número cumple con lo solicitado, la ejecución del programa continua sin ingresar al **try catch**.

> Vale destacar que la expresión **throw** se utiliza para lanzar una excepción. Esta expresión especifica el valor que se lanzará.

Podemos leer y profundizar más sobre **try catch** en developer.mozilla.org

EXERCISE 3: CONOCIENDO LOS ARREGLOS DE JAVASCRIPT

JavaScript es un lenguaje de programación (al igual que otros lenguajes) que cuenta con una amplia gama de objetos predefinidos que podemos utilizar día a día en nuestro desarrollo de software, por ejemplo, **Array** o Math.

Justo ahora nos centraremos en los **Arrays** o Arreglos.

QUE ES UN ARRAY

Array es un objeto global de **JavaScript** que es usado en la construcción de arreglos, que son objetos de tipo lista de alto nivel.

Comenzaremos creando un arreglo, para eso, definimos el nombre y, entre corchetes, agregamos los datos de la lista.

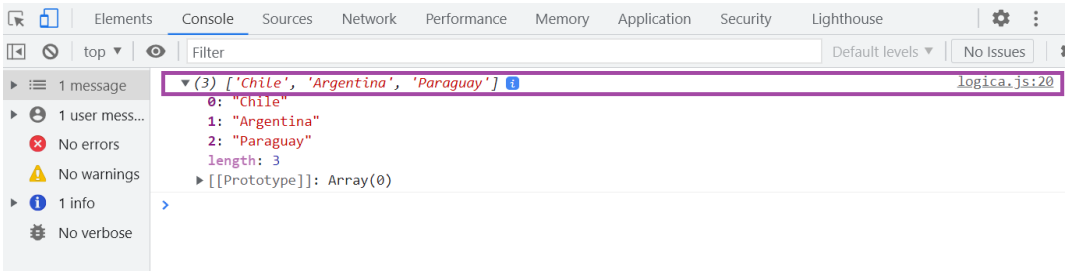
```
1 var paises =["Chile", "Argentina", "Paraguay"];
```

Para este ejemplo, el arreglo países cuenta con tres datos de tipo **String** dentro de él.

Para imprimir el arreglo completo, simplemente lo enviaremos a la consola.

```
1 console.log(paises);
```

Obteniendo como resultado:



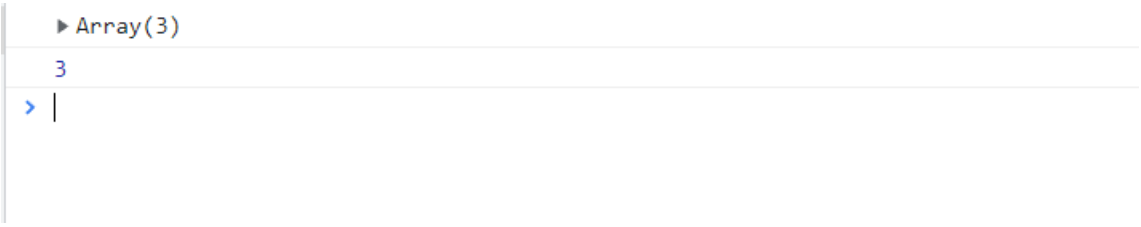
La primera posición de un arreglo siempre será el índice cero. Comenzará en cero y terminará en 2 cuando tenemos un arreglo de 3 espacios.

CONSEGUIR EL LARGO DE UN ARREGLO

Para obtener el largo de un arreglo usaremos la propiedad **length**.

```
1 console.log(paises.length);
```

Esto nos da como resultado:



Nos indica que el arreglo tiene un largo de 3 (recordando que sus índices son cero, uno y dos).

ACCEDER A UN ELEMENTO DEL ARREGLO POR SU INDICE

Para poder acceder a un valor especifico dentro del arreglo, podemos hacerlo mediante su índice, por ejemplo, para mostrar Argentina:

```
1 console.log(paises[1]);
```

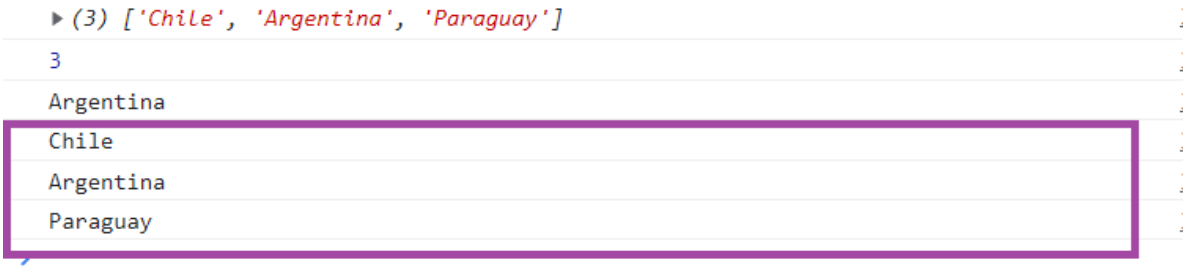
Escribimos el nombre del arreglo y, entre llaves, el número del índice al cual queremos acceder:

RECORRER UN ARREGLO UTILIZANDO FOR

Para recorrer un arreglo elemento por elemento y poder acceder a cada dato como un elemento separado utilizaremos un ciclo iterativo.

```
1 for(let i = 0; i < paises.length; i ++){
2     console.log(paises[i]);
3 }
```

Debemos comenzar siempre por el índice cero para poder ingresar a cada uno de los datos:

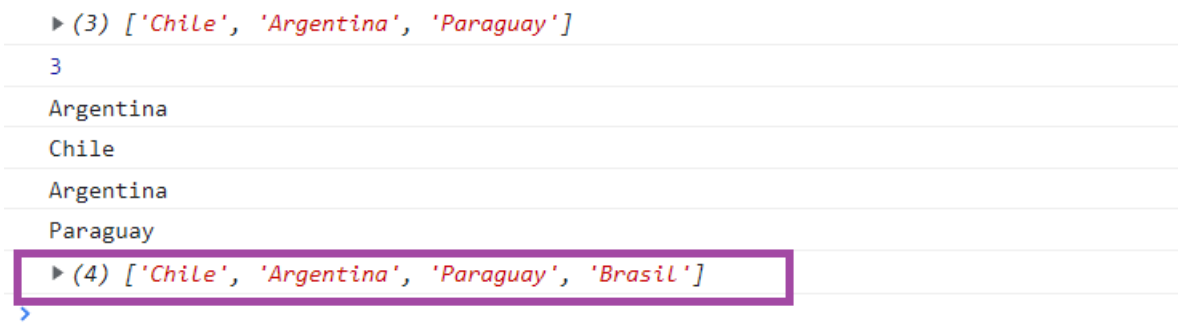


AÑADIR UN ELEMENTO AL FINAL DE UN ARREGLO

Podemos incorporar un nuevo elemento al final del arreglo utilizando la función **push()**.

```
1 paises.push("Brasil");
2 console.log(paises);
```

Al imprimir nuevamente nuestro arreglo obtenemos como resultado:



AÑADIR UN ELEMENTO AL PRINCIPIO DEL ARREGLO

Utilizando el método **unshift()** el elemento añadido ocupa el índice cero, desplazando al resto un lugar:

```
1 paises.unshift("Uruguay");
2 console.log(paises);
```

Obtenemos como resultado:

```
▶ (3) ['Chile', 'Argentina', 'Paraguay']
3
Argentina
Chile
Argentina
Paraguay
▶ (4) ['Chile', 'Argentina', 'Paraguay', 'Brasil']
▶ (5) ['Uruguay', 'Chile', 'Argentina', 'Paraguay', 'Brasil']
```

ELIMINAR EL ULTIMO ELEMENTO DE UN ARREGLO

Para eso usaremos la función **pop()**:

```
1 paises.pop();
2 console.log(paises);
```

Obteniendo como resultado:

```
▶ (3) ['Chile', 'Argentina', 'Paraguay']
3
Argentina
Chile
Argentina
Paraguay
▶ (4) ['Chile', 'Argentina', 'Paraguay', 'Brasil']
▶ (5) ['Uruguay', 'Chile', 'Argentina', 'Paraguay', 'Brasil']
▶ (4) ['Uruguay', 'Chile', 'Argentina', 'Paraguay']
```

ELIMINAR EL PRIMER ELEMENTO DE UN ARREGLO

Por el contrario, si deseamos eliminar el primer elemento de un arreglo podemos hacerlo usando la función **shift()**.

```
paises.shift();
console.log(paises);
```

Obteniendo como resultado:

```
▶ (3) ['Chile', 'Argentina', 'Paraguay']
3
Argentina
Chile
Argentina
Paraguay
▶ (4) ['Chile', 'Argentina', 'Paraguay', 'Brasil']
▶ (5) ['Uruguay', 'Chile', 'Argentina', 'Paraguay', 'Brasil']
▶ (4) ['Uruguay', 'Chile', 'Argentina', 'Paraguay']
▶ (3) ['Chile', 'Argentina', 'Paraguay']
```

Ahora que conocemos los **Arrays** o arreglos de **JavaScript** podemos comenzar a almacenar nuestros datos y objetos en arreglos y manipularlos a través de bucles para, por ejemplo, mostrarlos en pantalla.

Para obtener información más detallada sobre Array podemos ingresar a la documentación.

EXERCISE 4: CREANDO NUESTRO PRIMER OBJETO

JavaScript es un lenguaje multiparadigma, es decir, cumple con distintas maneras o estilos de programar software, pero nos enfocaremos en la **Programación Orientada a Objetos** (*Object Oriented Programming*) abreviado como **POO**.

CREANDO NUESTRO PRIMER OBJETO

En nuestro archivo **JS** vamos a crear nuestro primer objeto, el cual será un auto que contendrá atributos como color, número de puertas y marcas. Además, tendrá la función de acelerar y frenar.

Para crear el objeto comenzamos declarando una variable usando **var** y colocamos el nombre del objeto el cual será "auto".

Continuamos indicando que será un objeto escribiendo la instrucción **new Object()**.

```
1 var auto = new Object();
```

Luego, incluiremos cada uno de los atributos de nuestro auto (color, número de puertas y marcas) describiendo el objeto y su atributo más el dato.

```
1 auto.color = "rojo";
2 auto.numeroPuertas = 4;
3 auto.marca = "Samsung";
```

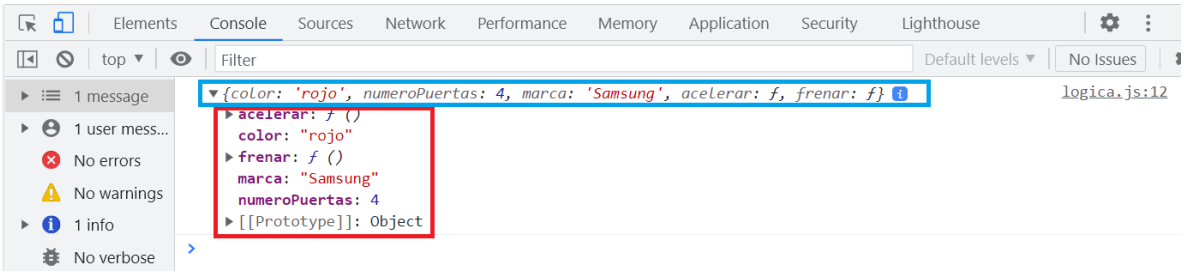
Finalmente, agregaremos las funciones **acelerar()** y **frenar()**. Ambos métodos solo van a mostrar un mensaje en la consola.

```
auto.acelerar = function () {
    console.log("El auto aceleró");
}
auto.frenar = function () {
    console.log("El auto fren{o");
}
```

Ahora hemos creado nuestro objeto “auto”, pero, es necesario ver como podemos ver los datos de estos objetos.

Primero, imprimiremos el objeto completo:

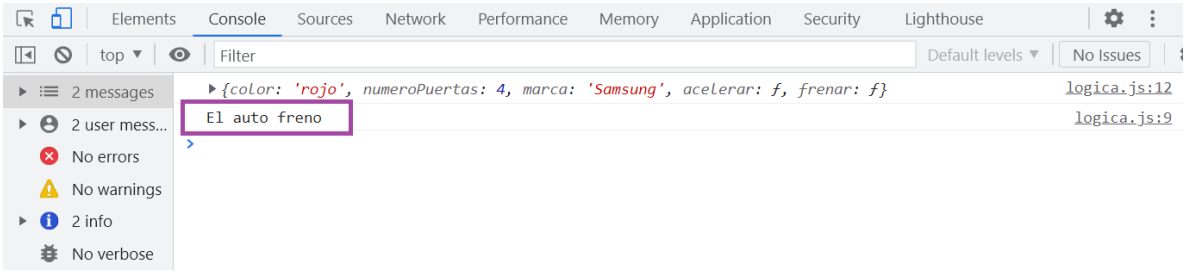
```
console.log(auto);
```



Podemos llamar la función del objeto:

```
auto.frenar();
```

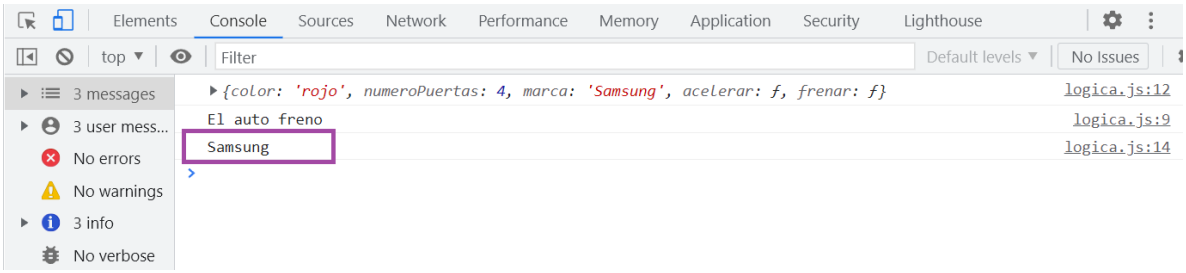
Obteniendo como resultado:



Finalmente, veamos el valor de uno los atributos específicamente:

```
1 console.log(auto.marca);
```

Obtenemos como resultado:



Con esto hemos creado nuestro primer objeto, pero, veamos otra forma de definir este mismo objeto.

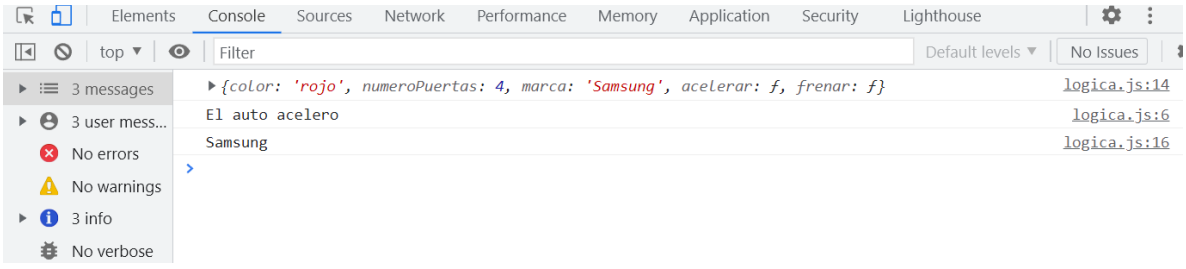
Para eso, vamos a modificar nuestro objeto. Crearemos una lista delimitada por comas de pares de propiedades y valores asociados y todo, encerrado por llaves.

```
var auto = {
    color: 'rojo',
    numeroPuertas: 4,
    marca: 'Samsung',
    acelerar: function () {
        console.log("El auto acelero");
    },
    frenar: function () {
        console.log("El auto freno");
    },
};
```

Para imprimir los mismos valores, usamos las mismas sentencias.

```
console.log(auto);
auto.acelerar();
console.log(auto.marca);
```

Y obtenemos como resultado:



FUNCIÓN CONSTRUCTORA

Podemos definir un tipo de objeto creando una función para el objeto que especifique su nombre, propiedades y métodos.

Continuemos con el ejemplo del auto. El tipo de objeto será **Auto** y mantendrá las propiedades que escribimos en el ejemplo anterior. Para definir la función constructora la sintaxis será la siguiente:

```
function Auto(color, numeroPuertas, marca) {
    this.color = color;
    this.numeroPuertas = numeroPuertas;
    this.marca = marca;
}
```

This se utiliza para asignar valores a las propiedades del objeto en función de los valores pasados a la propia función.

Para crear objetos de tipo **Auto** usaremos la indicación **new**, de la siguiente manera:

```
var miAuto1 = new Auto("Rojo", 4, "Nissan");
var miAuto2 = new Auto("Negro", 2, "Suzuki");
```

De esta forma hemos creado distintos objetos a partir de un mismo “molde”.

OBJETOS COMO PROPIEDADES DE OBJETOS

Los objetos pueden tener dentro de sus propiedades o atributos, un objeto completo, es decir, no solo puede almacenar un **String** o un numérico si no, un valor que representa todo un objeto con sus propios atributos.

Veámoslo en el siguiente código:

```
function Auto(color, numeroPuertas, marca, conductor) {
    this.color = color;
    this.numeroPuertas = numeroPuertas;
    this.marca = marca;
    this.conductor = conductor;
}
function Conductor(nombre, tipoLicencia, edad) {
    this.nombre = nombre;
    this.tipoLicencia = tipoLicencia;
    this.edad = edad;}
```

Hemos incorporado a **Auto** el atributo conductor y hemos creado una función constructora de **Conductor** con sus atributos propios.

Ahora creamos un objeto **Conductor**.

```
var conductor1 = new Conductor("Luis Ochoa", "B", 28);
```

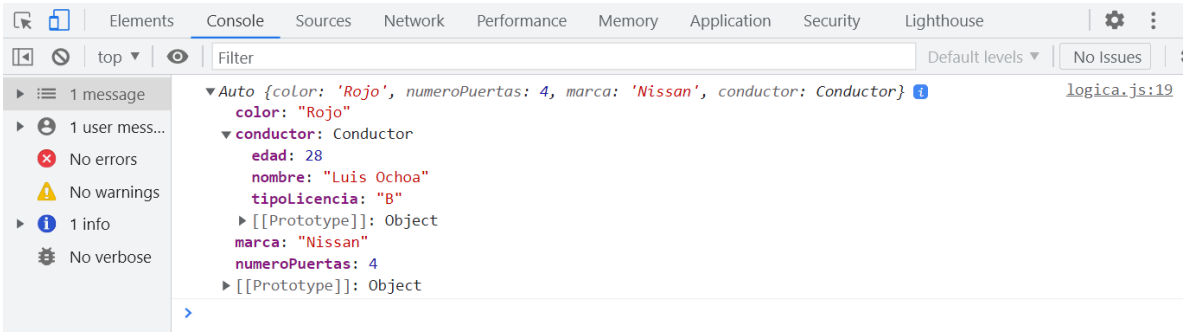
Y finalmente usamos este objeto “conductor1” como valor del atributo conductor de los objetos **Auto** creados.

```
var miAuto1 = new Auto("Rojo", 4, "Nissan", conductor1);
var miAuto2 = new Auto("Negro", 2, "Suzuki", conductor1);
```

Ahora, para ver cómo se almacena este objeto dentro del otro, imprimamos en la consola el objeto “miAuto1”.

```
console.log(miAuto1);
```

Y obtenemos:



De esta forma hemos conocido los objetos en **JavaScript**, pero el mundo de los objetos es mucho más amplio y complejo. Podemos revisar más información en la documentación, en el apartado de Conceptos básicos de los objetos JavaScript, en el apartado Trabajando con objetos y en JavaScript orientado a objetos para principiantes.