

M4C7 JavaScript Assignment

A.- Preguntas teóricas.

En este checkpoint vamos a seguir con la dinámica del checkpoint anterior. Necesito que realices una documentación sobre los conceptos que te proporcionaré, recuerda que debes ser lo más claro y comprensible posible, estás creando un material que lo visualizarán personas que recién se inician en el mundo del desarrollo, por lo tanto, debes también sustentarlo con ejemplos, porqué se utiliza, para qué se utiliza, sintaxis, etc.

Los conceptos son los siguientes.

1. ¿Qué diferencia a Javascript de cualquier otro lenguaje de programación?

JavaScript es un poderoso lenguaje de programación construido para el navegador Netscape en 1995.

Todos los navegadores modernos lo adoptaron desde entonces para añadir funciones a los sitios web y, más recientemente, a aplicaciones web.

JavaScript es un lenguaje de programación de alto nivel, interpretado y orientado a objetos, utilizado principalmente en el desarrollo web, **pensado para agregar potencial de interacción y dinamismo a las páginas web.**

Es un **lenguaje flexible y dinámico que permite a los desarrolladores crear experiencias interactivas y ricas en contenido.**

Es importante saber que JavaScript no está relacionado con Java, a pesar de tener un nombre similar. Son dos lenguajes de programación diferentes con propósitos y características distintas

Hay cuatro razones clave para aprender y utilizar JavaScript:

1. **Es el único lenguaje de programación que un navegador web puede entender de forma directa**, sin necesidad de herramientas intermedias.
2. Algunas de la aplicaciones más poderosas del mundo están integradas en Javascript. Según los datos recogidos en [W3techs](#), el 98,9 % de todos los sitios web funcionan con JavaScript actualmente.

3. Es un gran lenguaje para crear aplicaciones móviles. A lo largo de los años, los desarrolladores de JavaScript tomaron sus propias bibliotecas y crearon los marcos que permiten construir sus aplicaciones utilizando solamente Javascript, que se traduce directamente a la API de nuestro teléfono inteligente. Esto significa que podemos crear una aplicación que puede usar la cámara, que puede verificar la ubicación, que puede usar el acelerómetro, etc. Todas esas cosas, para las que antes había que aprender un lenguaje diferente, JavaScript lo hace posible ahora directamente.
4. JavaScript es uno de los recursos más poderosos que podamos tener para automatizar nuestro flujo de trabajo diario. Podemos crear un script en Javascript, de una forma simple, que ejecutará y automatizará tareas que hacemos de manera repetitiva, incluso evitando clicar manualmente.

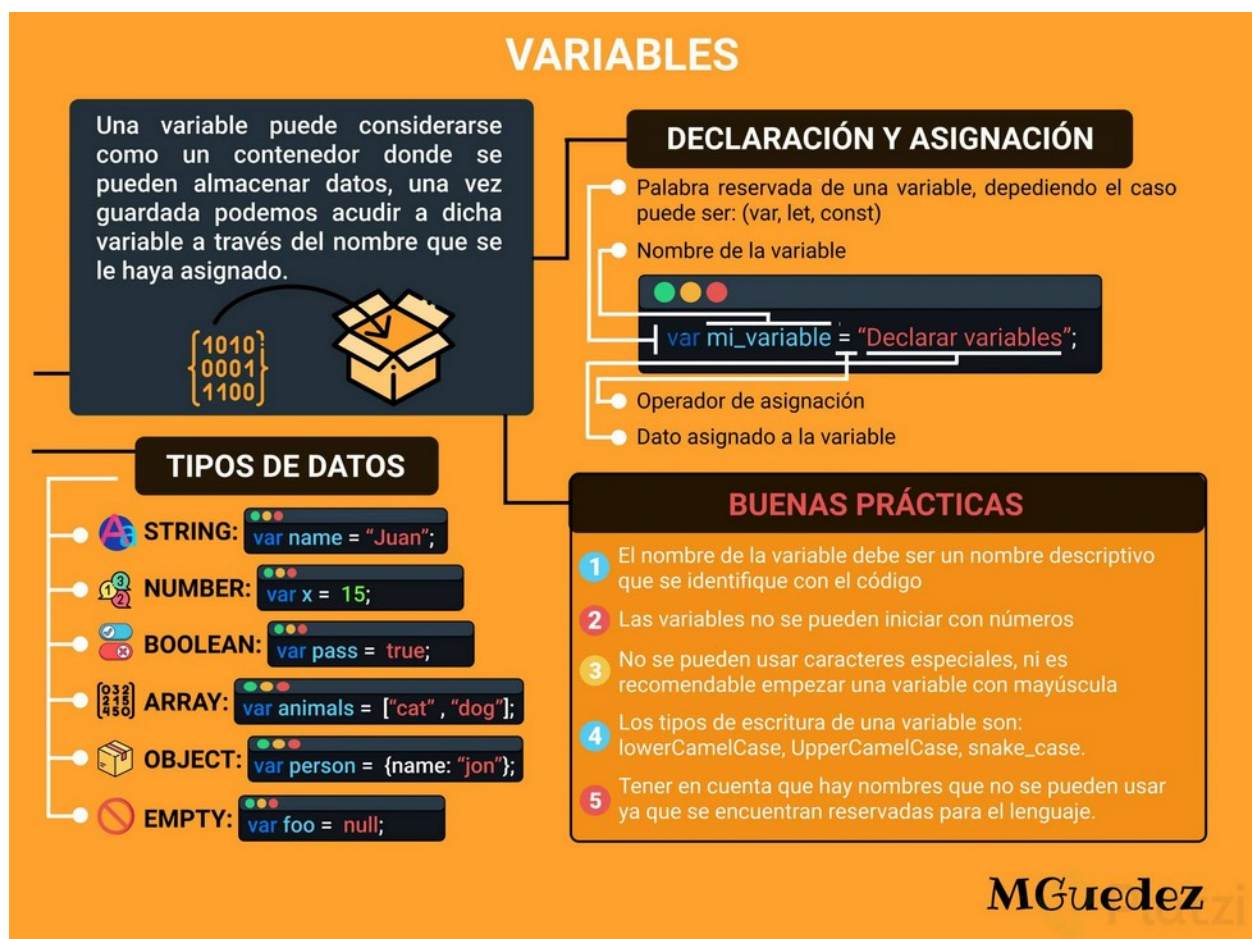
Podemos ver de manera esquemática los diferentes usos de JavaScript en el siguiente gráfico.



2. ¿Cuáles son algunos tipos de datos JS?

Tipos de datos es, esencialmente, cómo JavaScript categoriza todos nuestros puntos de datos. Estos son básicamente seis:

1. **Boolean**: true / false.
2. **Null**: null (vacío).
3. **Undefined**: undefined (no definido, valor por defecto aportado por JavaScript).
4. **Number**: valor numérico (sin necesidad de entrecomillado).
5. **String**: cadena de caracteres (entre comillados).
6. **Symbol** (objeto): similares a las cadenas excepto que tienen algunas reglas específicas: a) No pueden ser cambiados; b) Solo puede haber uno de ellos; c) Son lo más cercano que tiene JavaScript a lo que se conoce como "objeto de tipo inmutable". Está orientado a trabajar con objetos.



3. ¿Cuáles son las tres funciones de String en JS?

Una función en JavaScript es un bloque de código o un conjunto de instrucciones que realiza una tarea específica y que puede reutilizarse a voluntad. Por lo tanto, una función es uno de los building blocks fundamentales de JavaScript.

La informática (tal como la conocemos hasta ahora) está basada en el ya famoso concepto de «entrada-proceso-salida». Aunque las funciones de JavaScript no siempre tienen una entrada y una salida explícita, lo cierto es que no son la excepción a la regla y siguen este mismo sistema de procesamiento de información.

Existen tres formas de crear una función en JavaScript:

1. **Por declaración:** la más utilizada y la más fácil de recordar. Consiste en declarar la función con un nombre y sus parámetros de entrada entre paréntesis.

```
function add(a, b){ return a+b;};add(5,6); //11
```

2. **Por expresión:** consiste básicamente en guardar una función en una variable, para así ejecutar la variable como si fuera una función. Este nuevo recurso ha dado pie a las funciones anónimas.

```
const plus = function (a, b){ return a+b;};plus(5,6); //11
```

3. **Por constructor de objeto:** la menos recomendada y casi no se utiliza. Únicamente nos ayuda a recordar que las funciones también son tipos de objetos en JavaScript.

```
const greetings = new Function("return 'Hola';");greetings (); // 'Hola'
```

Estos tipos nos hablan de cómo definir nuestras propias funciones; sin embargo, los tipos de datos en los que se basa JavaScript cuentan con un conjunto de funciones disponibles para operar sobre ellos por defecto.

Algunas de las más utilizadas son las siguientes:

1. **String.split()**

Divide una cadena en un arreglo de subcadenas de la cadena original a partir de un carácter separador:

```
let cadena = "Hola,mundo,JavaScript";cadena.split(",");
```

2. String.substring()

Extrae caracteres desde un índice A hasta un índice B sin incluirlo:

```
let cadena = "Hola";cadena.substring(0,3);
```

3. String.trim()

Elimina espacios en blanco al inicio y al final de una cadena:

```
let cadena = " Hola ";cadena.trim();
```

4. Array.map()

Crea un arreglo nuevo con los resultados de la función enviada como parámetro:

```
let numbers = [2, 4, 8, 16];  
let halves= numbers.map(function(x) { return x / 2;});
```

5. Array.push()

Agrega elementos al final de un arreglo y regresa la longitud del arreglo con los nuevos elementos:

```
const things = ['dish', 'computer', 'dog'];const count = things.push('bed');
```

6. Array.pop()

Remueve el último elemento de un arreglo y lo devuelve:

```
const things = ['dish', 'computer', 'dog'];const last = things.pop();
```

7. String.slice()

Regresa una porción de una cadena y devuelve una cadena nueva:

```
let string1= "Bienvenido a JavaScript.";let slice = cadena1.slice(5, -1);
```

8. `Array.slice()`

Regresa una porción de un arreglo y devuelve un arreglo nuevo:

```
var names = ['Zita', 'Pepe', 'Mario', 'Cielo', 'Azul'];  
var slice = names.slice(1, 3);
```

9. `Object.toString()`

Todos los objetos en JavaScript tienen este método y se usa para representarlos en forma de cadena de texto:

```
let objeto = new Object();objeto.toString();
```

10. `Number.toFixed()`

Establece la cantidad de decimales que queremos definir en un número:

```
var num= 12345.6789;num.toFixed();// Returns '12346'  
num.toFixed(1);  
// Returns '12345.7'  
num.toFixed(6);// Returns '12345.678900'
```

11. `parseInt()`

Convierte una cadena a un entero de la base especificada:

```
parseInt("15", 10);parseInt("15.99", 10);
```

12. `Math.random()`

Devuelve un número flotante aleatorio entre 0 y 1 (sin incluir el 1):

```
const rnd = Math.random();
```

13. `console.log()`

Escribe un mensaje en la consola web o en el intérprete de JavaScript:

```
console.log("Este es un mensaje en la consola");
```

4. ¿Qué es un condicional?

Los condicionales son unos elementos básicos fundamentales de todo tipo de lenguaje de programación y, el motivo, es que permiten un comportamiento dinámico en la aplicación.

Los condicionales en JavaScript nos dan la habilidad de observar un par o múltiples valores, dependiendo de lo que necesitemos comprobar, y ver cómo se relacionan entre sí.

La condicional “if... else” se utiliza en el 95% de los casos cuando se programa.

La sintaxis básica de un condicional es como sigue:

```
if (condición) {  
    código a ejecutar si la condición es verdadera  
} else {  
    ejecuta este otro código si la condición es falsa  
}
```

1. La palabra clave **if** seguida de unos paréntesis.
2. Una **condición** a probar, puesta dentro de los paréntesis (típicamente “¿es este valor mayor que este otro valor?”, o “¿existe este valor?”). Esta **condición** usará los operadores de comparación que hemos hablado en el módulo anterior y retorna un valor true o false (verdadero o falso).
3. Un conjunto de llaves, en las cuales tenemos algún **código** – puede ser cualquier código que deseemos, código que se ejecutará solamente si la condición retorna true.
4. La palabra clave **else**.
5. Otro conjunto de llaves, dentro de las cuales tendremos **otro código** – puede ser cualquier código que deseemos, y sólo se ejecutará si la condición no es true.

Este código se leerá de la siguiente manera:

“si (if) la condición retorna verdadero (true), entonces ejecute el código siguiente, sino (else) ejecute este otro código”

Dentro de los condicionales tenemos los denominados operandos, que son los valores que están antes y después de los operadores de comparación o símbolos. Según los tipos de condicionales que queramos plantear, podemos distinguir una serie de operadores de comparación, o símbolos, como los que aparecen en el siguiente cuadro.

Tipos de condicionales

Símbolo	Explicación	Ejemplo
>	es mayor que	<code>if (1 > 0)</code>
<	es menor que	<code>if (1 < 0)</code>
&&	y	<code>if (1 > 0 && 67 > 0)</code>
	o	<code>if (1 > 10 67 > 0)</code>
==	es igual en valor	<code>if ("3" == 3)</code>
===	es igual en valor y tipo	<code>if ("3" === "3")</code>
!	no	<code>if (!(1 > 0))</code>
!=	no es igual	<code>if ("Doctor" != "Who")</code>
!==	no es igual en valor o tipo	<code>if ("Doctor" !== "Who")</code>
>=	es mayor o igual que	<code>if (10 >= 10)</code>
<=	es menor o igual que	<code>if (10 <= 20)</code>
true	Verdad	<code>if (true)</code>
false	Falso	<code>if (false)</code>

Fuente: programadorwebvalencia.com

5. ¿Qué es un operador ternario?

Un operador ternario es aquel que nos permite escribir un condicional entero en una sólo línea.

Un ejemplo:

```
function ageVerification(age) {  
  let answer = age > 25 ? "can" : "can't"; console.log(answer);  
}  
  
ageVerification(55)
```

Para comprender lo anterior vamos a comentar el mapping del código:

1. La primera parte del código es el condicional.
2. El signo de interrogación significa que partiremos del enunciado en positivo, como verdadero (true).
3. Los dos puntos equivalen a "else", que antecede al enunciado en negativo, en respuesta a que la condición no se cumpla.

Dentro de las recomendaciones de buenas prácticas en programación, se indica que el uso del ternario simple es bueno y nos permite reducir en tamaño el código pero se indica a modo de recomendación que hemos de evitar abusar de su utilización.

6. ¿Cuál es la diferencia entre una declaración de función y una expresión de función?

En la respuesta a la pregunta número 3 ya hemos hecho referencia a estas funciones. Señalemos tres de las diferencias importantes entre ellas.

1.- Las declaraciones de funciones pueden ser ejecutadas antes de su definición.

Gracias a que las declaraciones de funciones son ascendidas al momento de ejecución de nuestro programa, nuestra función se puede ejecutar incluso antes de su definición.

```
JS
1  nuevaFuncion()
2
3  function nuevaFuncion() {
4    console.log("Hola Mundo!")
5  }
6
7  // → Hola Mundo!
```

En cambio, esto no es posible con las expresiones de funciones, ya que no se sabe el valor que va a tener nuestra variable previamente declarada.

```
JS
1  nuevaFuncion()
2
3  var nuevaFuncion = function nuevaFuncion() {
4    console.log("Hola Mundo!")
5  }
6
7  // → TypeError: nuevaFuncion is not a function
```

2.- Teniendo en cuenta la sintaxis, las declaraciones de funciones están en el lado izquierdo, las expresiones de funciones en el lado derecho.

La palabra reservada function se puede usar tanto al lado izquierdo como al lado derecho del signo igual (=). Lo interesante de esto es que siguen diferentes reglas dependiendo de dónde la escribimos.

Cuando la usamos al lado izquierdo, estamos creando una declaración de función y es obligatorio asignar un nombre. En cambio, si la usamos a la derecha del símbolo igual, hablamos de una expresión de función.

```
JS
1 // declaración de función
2 function soyUnaDeclaracion() {
3   // ...
4 }
5
6 // expresión de función
7 var soyUnaExpresion = function() {
8   // ...
9 }
```

A modo de curiosidad, podemos observar que en las expresiones de funciones darle nombre a la función es opcional. En cambio para las declaraciones es obligatorio.

En este último caso, la función pasa a ser una función anónima, algo conveniente ya que nuestro programa sigue funcionando sin problemas, pero puede tener algunas implicaciones a la hora de inspeccionar la cola de ejecución del programa.

3.- Las expresiones de funciones son más difíciles de inspeccionar.

Poder identificar rápidamente los problemas y saber donde están para resolverlos es fundamental. Cuando le asignamos nombres a nuestras funciones, salen en la cola de ejecución y podemos seguir mejor los errores y ver de donde provienen.

The screenshot shows a code editor with the following code:

```
62 pintarEdad = function(numero) {
63   console.log(`la edad es de ${numero} años`)
64 }
65
66 // la famosa IIFE
67 (function vayaTiemposCuandoEraFamoso() {
68   console.log("7. IIFE ejecutado!");
69 })();
70
71 function pruebaDelStackTrace(cosas) {
72   cosas.map((palabra) => funcionInterna(palabra));
73   debugger;
74 }
75
76 function funcionInterna(entry) { entry = "foo"
77   debugger;
78   return entry;
79 }
80
81 pruebaDelStackTrace(["foo", "bar", "baz"]);
82
```

The call stack on the right shows the following frames:

- 1 evaluate index.js? [sm]:81
- 2 pruebaDelStackTrace index.js? [sm]:72
- 3 (anonymous) index.js? [sm]:72
- 4 funcionInterna index.js? [sm]:77

Arrows indicate the flow of execution: from the call to `pruebaDelStackTrace` (line 81) to the function `pruebaDelStackTrace` (line 71), then to the anonymous function (line 67), and finally to `funcionInterna` (line 76).

7. ¿Qué es la palabra clave "this" en JS?

En JavaScript, la palabra clave "this" siempre se refiere a un objeto. Lo que pasa es que el objeto al que se refiere variará dependiendo de cómo y dónde se llame "this".

"this" no es una variable, es una palabra clave, por lo que su valor no se puede cambiar ni reasignar.

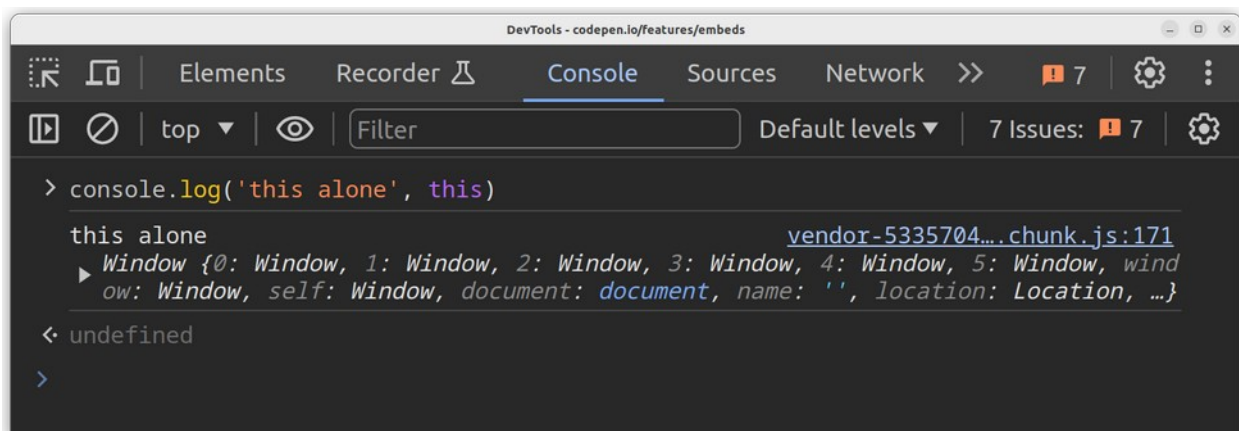
Hay algunas formas diferentes de usar la palabra clave "this" que nos ayudarán a comprender un poco mejor la utilidad del mismo.

1.- Si llamamos a "this" por sí mismo, es decir, no dentro de una función, objeto o lo que sea, se referirá al objeto de ventana global.

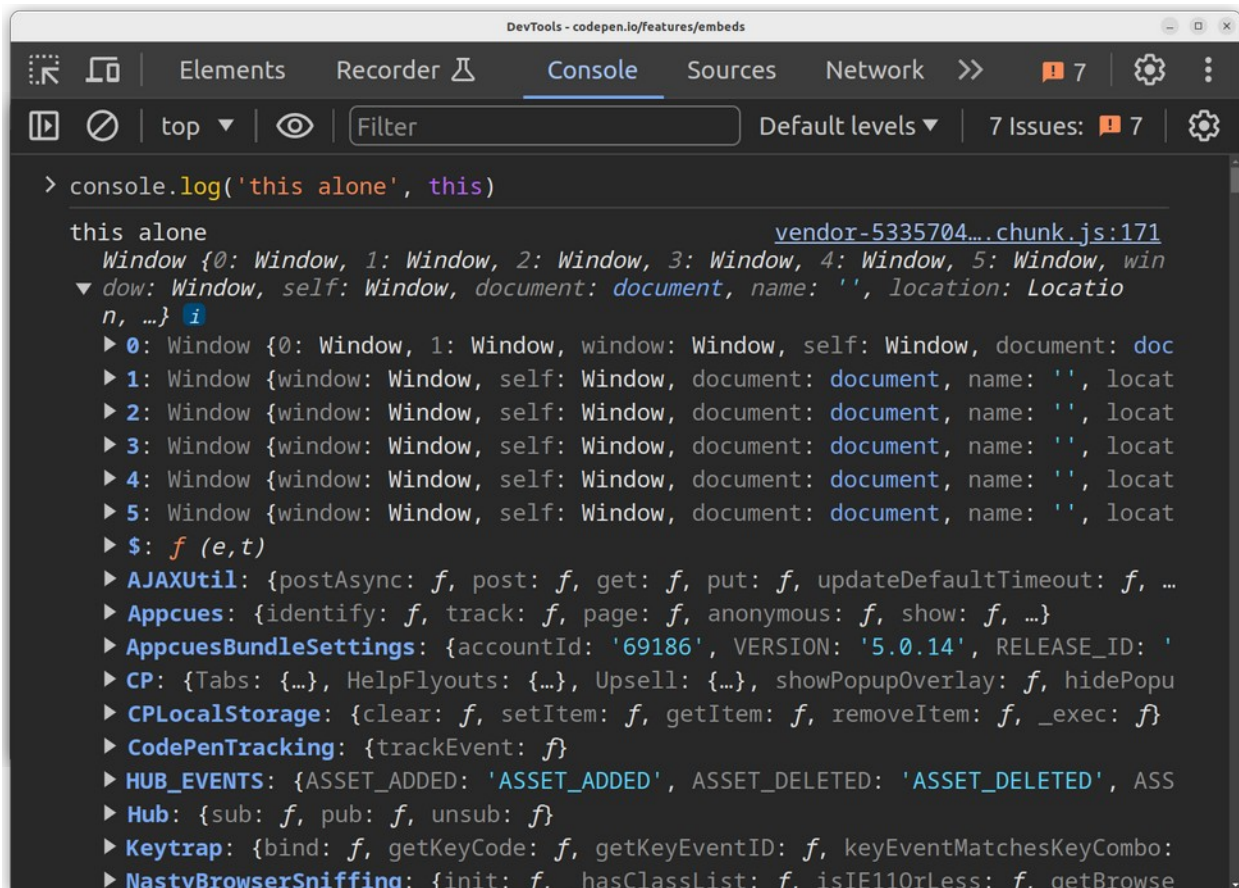
Si lo imprimimos como `console.log('this alone', this);`

obtendremos esto en nuestra consola:

[object Window]



Si lo expandimos:



```
DevTools - codepen.io/features/embeds

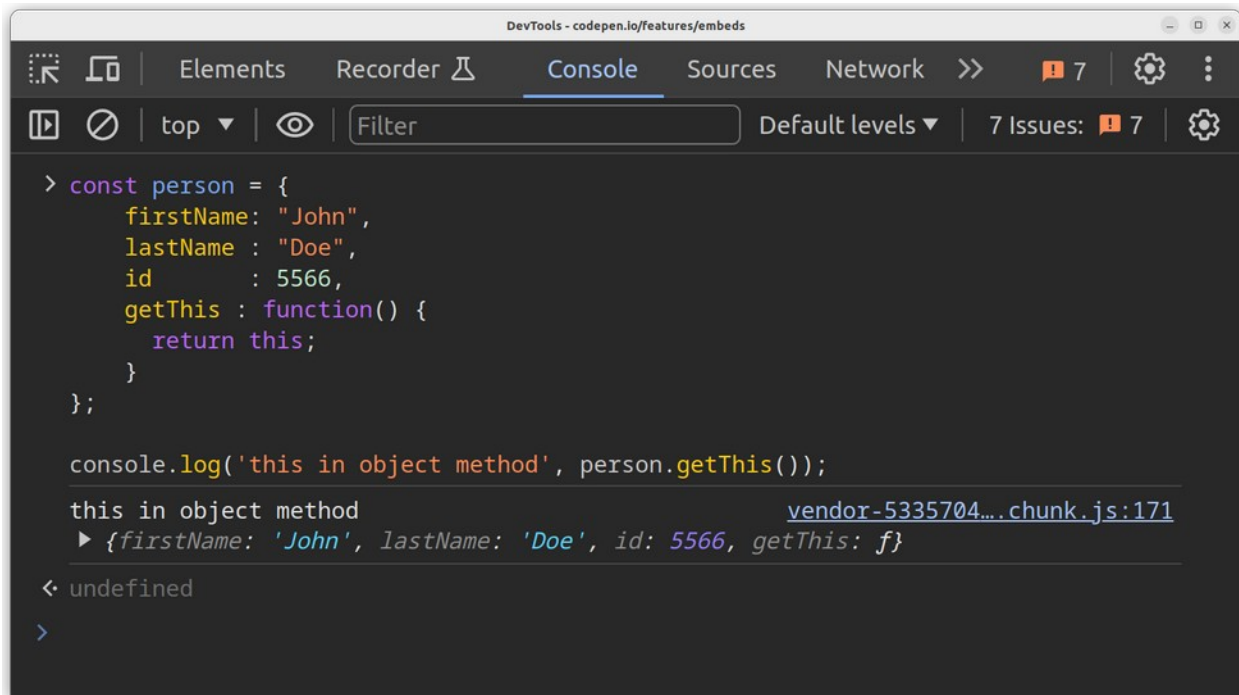
Elements Recorder Console Sources Network >> 7 Issues: 7

top Filter Default levels 7 Issues: 7

> console.log('this alone', this)

this alone vendor-5335704...chunk.js:171
Window {0: Window, 1: Window, 2: Window, 3: Window, 4: Window, 5: Window, win
▼ dow: Window, self: Window, document: document, name: '', location: Locatio
n, ...} ⓘ
  ▶ 0: Window {0: Window, 1: Window, window: Window, self: Window, document: doc
  ▶ 1: Window {window: Window, self: Window, document: document, name: '', locat
  ▶ 2: Window {window: Window, self: Window, document: document, name: '', locat
  ▶ 3: Window {window: Window, self: Window, document: document, name: '', locat
  ▶ 4: Window {window: Window, self: Window, document: document, name: '', locat
  ▶ 5: Window {window: Window, self: Window, document: document, name: '', locat
  ▶ $: f(e,t)
  ▶ AJAXUtil: {postAsync: f, post: f, get: f, put: f, updateDefaultTimeout: f, ...
  ▶ Appcues: {identify: f, track: f, page: f, anonymous: f, show: f, ...}
  ▶ AppcuesBundleSettings: {accountId: '69186', VERSION: '5.0.14', RELEASE_ID: '
  ▶ CP: {Tabs: {...}, HelpFlyouts: {...}, Upsell: {...}, showPopupOverlay: f, hidePopu
  ▶ CPLocalStorage: {clear: f, setItem: f, getItem: f, removeItem: f, _exec: f}
  ▶ CodePenTracking: {trackEvent: f}
  ▶ HUB_EVENTS: {ASSET_ADDED: 'ASSET_ADDED', ASSET_DELETED: 'ASSET_DELETED', ASS
  ▶ Hub: {sub: f, pub: f, unsub: f}
  ▶ Keytrap: {bind: f, getkeyCode: f, getKeyEventID: f, keyEventMatchesKeyCombo:
  ▶ NastyBrowserSniffing: {init: f, hasClassList: f, isIE11OrLess: f, getBrowse
```

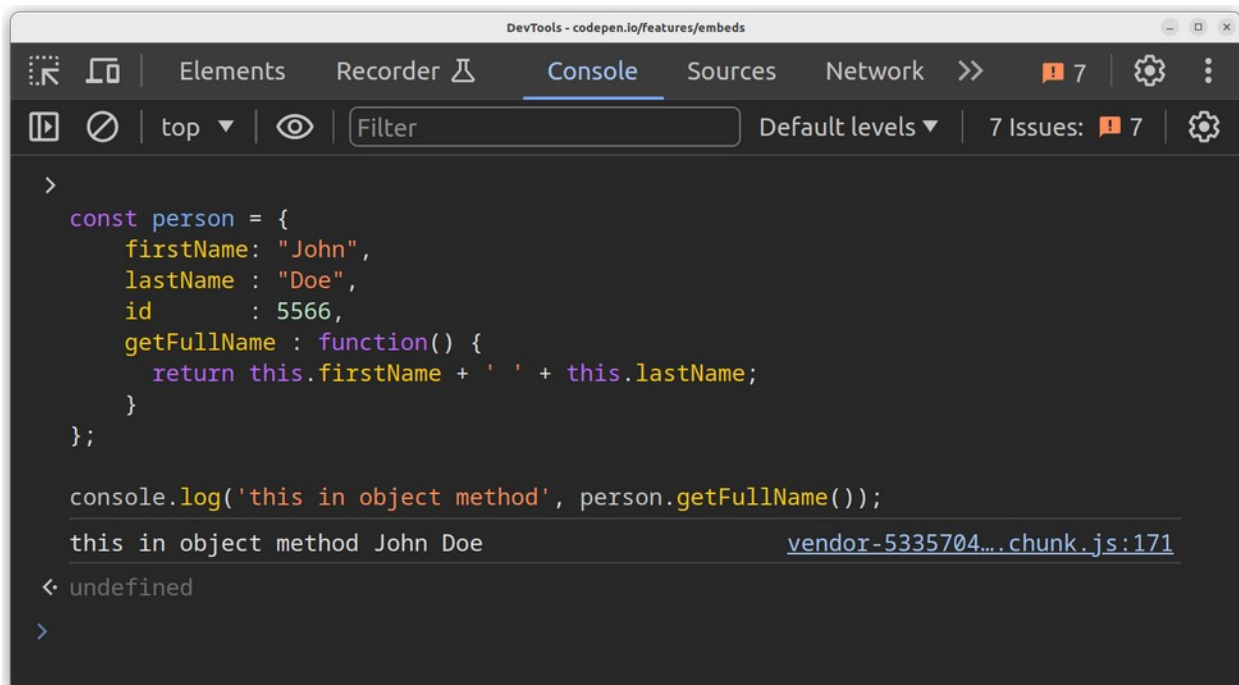
2.- Si llamamos a “this” dentro de un método de objeto, como en el siguiente ejemplo, veremos que “this” ya no se refiere al objeto en sí:



```
> const person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  getThis : function() {
    return this;
  }
};

console.log('this in object method', person.getThis());
this in object method vendor-5335704....chunk.js:171
▶ {firstName: 'John', lastName: 'Doe', id: 5566, getThis: f}
< undefined
>
```

Y con esto, podemos usar “this” para acceder a otras propiedades y métodos desde el mismo objeto:

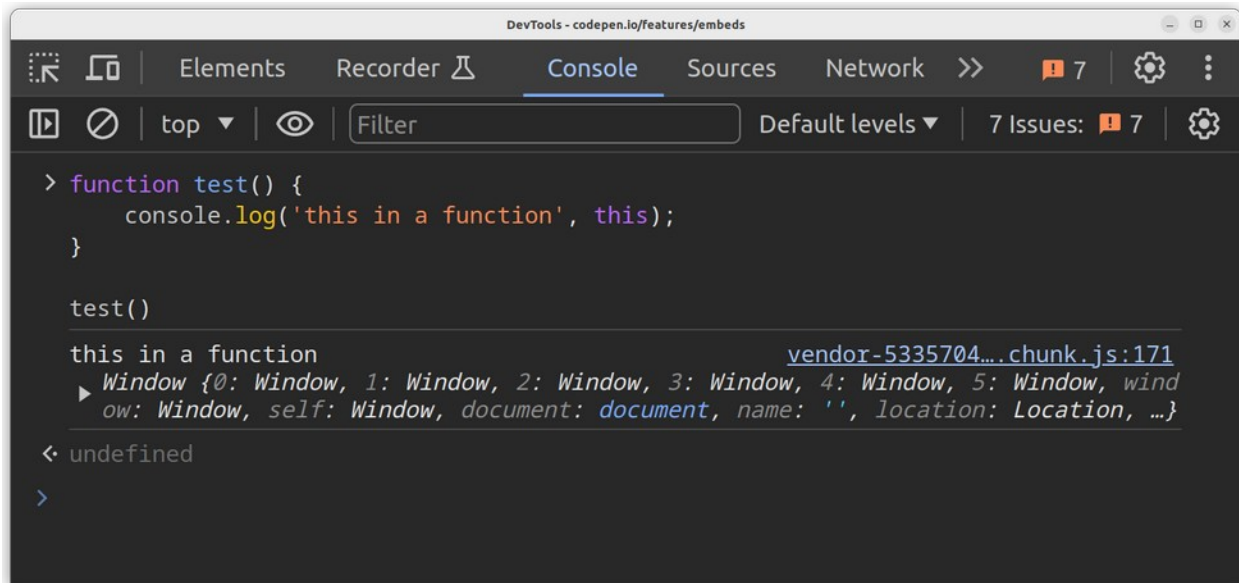


```
>
const person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  getFullName : function() {
    return this.firstName + ' ' + this.lastName;
  }
};

console.log('this in object method', person.getFullName());
this in object method John Doe vendor-5335704....chunk.js:171
< undefined
>
```

3.- Si llamamos a “this” dentro de una función como en el siguiente ejemplo, “this” ahora se referirá nuevamente al objeto de la ventana general, y obtendremos esto en nuestra consola:

[object Window].



The screenshot shows the Chrome DevTools Console with the 'Console' tab selected. The breadcrumb at the top reads 'DevTools - codepen.io/features/embeds'. The console toolbar shows 'top' as the current scope, a search filter, and '7 Issues'. The code being executed is a function named 'test()' that logs 'this in a function' with 'this' as the argument. The output shows the function call 'test()' followed by the log message 'this in a function' and a detailed object representation of the global 'Window' object. The object is expanded to show properties like '0: Window', '1: Window', '2: Window', '3: Window', '4: Window', '5: Window', 'window: Window', 'self: Window', 'document: document', 'name: ''', and 'location: Location, ...'. The prompt '>' is visible at the bottom of the console.

```
> function test() {  
    console.log('this in a function', this);  
}  
  
test()  
this in a function vendor-5335704...chunk.js:171  
▶ Window {0: Window, 1: Window, 2: Window, 3: Window, 4: Window, 5: Window, window: Window, self: Window, document: document, name: '', location: Location, ...}  
< undefined  
>
```

Fuente: freecodecamp.org

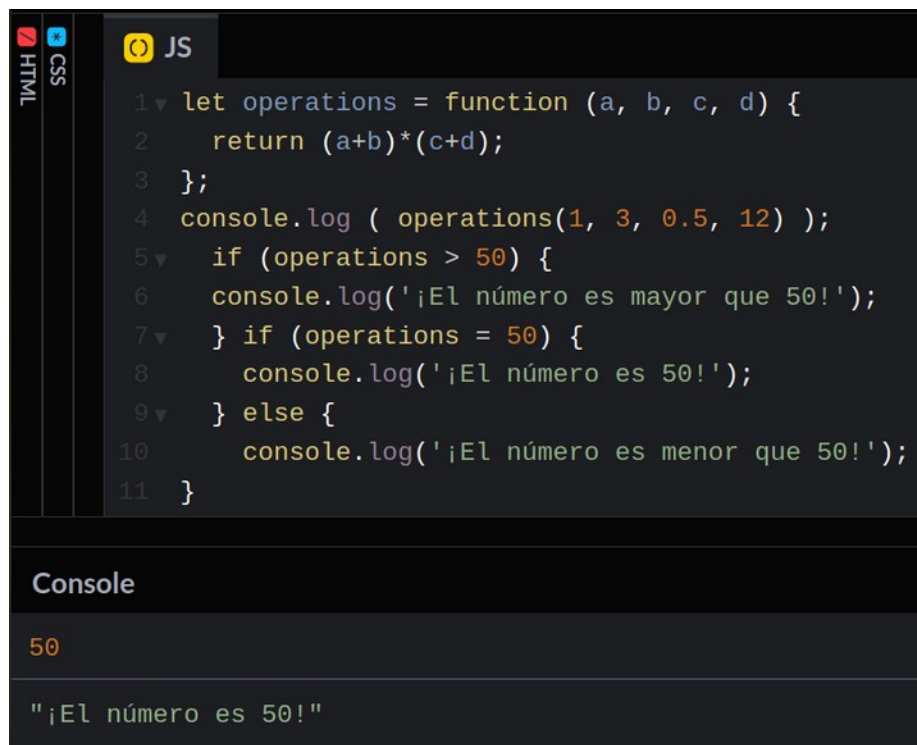
B.- Ejercicio práctico.

Y por último debes realizar el siguiente ejercicio práctico, y subirlo a tu repositorio en Git-Hub para revisarlo.

-Cree una función JS que acepte 4 argumentos. Suma los dos primeros argumentos, luego los dos segundos y multiplícalos. Si el número creado es mayor que 50, la consola registra "¡El número es mayor que 50!". Si es más pequeño, la consola registra "¡El número es menor que 50!"

Enlace: https://github.com/VicenteHeredia/Checkpoint_7.git

```
let operations = function (a, b, c, d) {  
  return (a+b)*(c+d);  
};  
console.log ( operations(1, 3, 0.5, 12) );  
if (operations > 50) {  
  console.log('¡El número es mayor que 50!');  
} if (operations = 50) {  
  console.log('¡El número es 50!');  
} else {  
  console.log('¡El número es menor que 50!');  
}
```



The screenshot shows a code editor with a dark theme. On the left, there are tabs for 'HTML', 'CSS', and 'JS'. The 'JS' tab is active. The code in the editor is the same as the one in the previous block. Below the code editor, there is a 'Console' panel. It shows the output of the code: the number '50' followed by the string '¡El número es 50!'.

```
1 let operations = function (a, b, c, d) {  
2   return (a+b)*(c+d);  
3 };  
4 console.log ( operations(1, 3, 0.5, 12) );  
5 if (operations > 50) {  
6   console.log('¡El número es mayor que 50!');  
7 } if (operations = 50) {  
8   console.log('¡El número es 50!');  
9 } else {  
10   console.log('¡El número es menor que 50!');  
11 }
```

Console

50

¡El número es 50!