Laboratorio 2B

Dannyel Obaldía Cordero

Escuela de Ciencias de la Computación e Informática, Universidad de Costa Rica

CI-0126 Ingeniería de Software

Rebeca Obando Vásquez

26 de marzo de 2025

1. **¿Cuáles son los data types que soporta javascript ?**

String, Number, Bigint, Boolean, Undefined, Null, Symbol, Object.

2. **¿Cómo se puede crear un objeto en javascript? De un ejemplo**

Javascript tiene objetos default y el usuario puede definir objetos.
Los default son objetos, arrays, dates, maps, sets, intarrays, floatarrays, promises, entre otros. Estos son declarativos: let var = 16 crea un objeto numérico, también pueden ser strings o booleanos. Pueden ser objetos como const person = {name:"Dannyel", lastName:"Obaldia"} o arreglos como const array = ["Hola", "yo", "soy", "nadie"].
Un objeto también puede ser complejo, con diferentes propiedades y funciones, por ejemplo:

```
const house = {
        number : 345,
        city: "San Pedro",
        owner: "Dannyel",
        wholeDirection: function() {
                return this.city + " " + this.owner;
        }
};
```

3. **¿Cuáles son los alcances (scope) de las variables en javascript?**

El alcance de las variables se refiere en dónde es utilizable o el área en la que es referenciable.
Los alcances son global, local y de bloque; global es alcanzable desde cualquier parte del programa, esto incluye diferentes archivos. El alcance local es dentro de una función, para aquellas declaradas con let, var o const. En alcance de bloque hay variables declaradas dentro de bloques de ciclo como for, if o while, sea con const o let, pero no var.

4. **¿Cuál es la diferencia entre undefined y null?**

Undefined significa que hay una variable declarada, pero no tiene un valor asignado, mientras que null es un valor explícito. Es decir, null es un valor deliberado para una variable, que significa que no tiene ningún contenido, y undefined simplemente es que no tiene un valor asignado explícitamente.

5. **¿Qué es el DOM?**

A la hora de cargar una página web, se crea un "modelo de objeto de documento" (DOM, en inglés), que es como un árbol de objetos para representar el programa. Esto aplica para los elementos html, para que Javascript pueda modificar y dinamizar el contenido de html y los estilos de CSS.

6. **Usando Javascript se puede acceder a diferentes elementos del DOM. ¿Qué hacen, que retorna y para qué funcionan las funciones getElement y querySelector? Cree un ejemplo**

Los elementos de DOM accesibles desde Javascript devuelven miembros del DOM, lo que vendrían siendo estructuras de html. Element se refiere a un nodo del DOM, document devuelve un objeto document, y attribute devuelve una referencia a un objeto de interfaz particular.

QuerySelector() es una función que devuelve el primer elemento que coincida con un selector CSS, o un grupo de estos. Estos selectores son los grupos a los que modifica CSS en un documento .css.

getElement() por sí sola no es una función para acceder al DOM, si no que tiene diferentes versiones para buscar dentro del árbol de acuerdo con diferentes identificadores. Estos son getElementById(“”), getElementByTagName() y getElementByClassName() principalmente. Estos identificar y devuelven el objeto de DOM que primero coincida con la ID, el tag o la clase seleccionada.

```
<html>

<head>

        <title>Hola Mundo</title>

    </head>

</html>
```

En Javascript, podría acceder al elememento del DOM que representa al tag *head* con

```
const element = document.getElementsByTagName(“head”);
```

7. **Investigue cómo se pueden crear nuevos elementos en el DOM usando Javascript. De un ejemplo**

Desde Javascript, uno puede crear un nuevo elemento en el DOM con la función createElement. Luego de crearlo se debe insertar en el documento para que se vea reflejado.

Un ejemplo es el siguiente código:

```
Function addElement() {
const newSpan = document.createElement(“span”);
newSpan.textContent = “Hola hola hola hola”
    const div = document.getElementeByTag(“div”);
```

```
        div.appendChild(newSpan);
}
```

8. **¿Cuál es el propósito del operador this?**

This es una keyword multifuncional que se refiere a un contexto donde un código debería de correr. Se utiliza en métodos de un objeto para referenciarse a sí mismo y de esa forma poder reutilizar un mismo método con objetos o instancias diferentes. Cuando es fuera de un objeto, se puede referir al objeto global, o mejor dicho al contexto en el que sucede. En general, su propósito es poder referirse dentro de un contexto a sí mismo, de forma que no se debe declarar otro objeto por aparte para referirse a él.

9. **¿Qué es un promise en Javascript?De un ejemplo**

Promise es un placeholder o un valor que no es conocido a la hora de ser creado, es decir, es un valor pendiente. Se asocia a acciones asincrónicas que dependen del resultado de una función, y la promesa tiene valores de pending, fulfilled y rejected, que dan paso o no a diferentes acciones de acuerdo con lo declarado.

En código se ve como lo siguiente:

```
function successCallback(result) {
console.log('La creación de la solución fue exitosa y se encuentra en: ${result}');
}
funtion failureCallback(error) {
console.error('Error al crear la página: ${error}');
}

createPage(parameters).then(successCallback, failureCallback);
```

En este caso se comprobará de forma asíncrona si la operación de crear la página fue exitosa o no y eventualmente se actuará de forma acorde. El objeto de promise es devuelto por la función asíncrona, createPage, y se le pasan los callbacks como parámetros para que actúe acorde.

10. **¿Qué es Fetch en Javascript? De un ejemplo**

Fetch es una API que sirve para hacer solicitudes HTTP, esperar y comprobar su respuesta. Esta devuelve un response que es un promise, que luego puede ser tratado. Se puede utilizar de varias formas, con parámetros para indicar cómo tratar el proceso y qué parámetros enviar. Por ejemplo:

```
async function getData() {
const url = https://this.com/this;
try {
```

```
const response = await fetch(url);
if (!response.ok) {
                    throw new Error(‘Not ok');
                }

                const data = await response.json();
                console.log(data);
                } catch (error) {
                    console.error('There was an error:', error);
                }
        }
```

11. ¿Qué es Asynch/Await en Javascript ? De un ejemplo

Async y await son dos comandos que identifican promesas, y hacen más fácil el código para tratar con ellas.
Async se usa como un indicador frente a la declaración de una función, y lo que indica es que es una función asincrónica y que por lo tanto devuelve una promesa.
Await solo se puede utilizar dentro de funciones con el identificador Async, y lo que hace es esperar a la resolución de un promise de otra función para proceder con una ejecución.

```
async function fetchInfo() {
const response = await fetch (‘https://this.that/andthat’);
const data = await responde.json();
return data;
}
```

12. ¿Qué es un Callback? De un ejemplo

Callback son funciones que son pasadas como argumentos, utilizadas comúnmente en operaciones asincrónicas.

```
function successCallback(result) {
console.log(‘La creación de la solución fue exitosa y se encuentra en: ${result}’);
}
funtion failureCallback(error) {
console.error(‘Error al crear la página: ${error}’);
}

createPage(parameters).then(successCallback, failureCallback);
```

En este ejemplo que es el mismo que el de un punto anterior, las funciones de callback como respuestas de la promise reciben de parámetro a la promise y la procesan, y son funciones que se pasan como parámetros.

13. ¿Qué es Clousure?

Closure es un aspecto de JS que no es declarado, pero forma parte de este. Básicamente se refiere a cuando se accede a una variable en una función, pero fuera de su contexto. Esto significa que se puede utilizar el valor de una variable declarada fuera de una función, dentro de esta misma. Por ejemplo:

```
function raiseTwo() {
let mult= 4;
function multiply() {
mult=mult*mult;
console.log(mult);
        }
}
```

14. ¿Cómo se puede crear un cookie usando Javascript?

Las cookies son información pequeña almacenada, que JS puede manejar con la propiedad document.cookie.
Una cookie puede crearse como: document.cookie = "username=Dannyel";
Todos los valores guardados en una cookie se recuperar como: let x = document.cookie;
Y a las cookies se les puede agregar una fecha de expiración, entonces para borrarlas basta con ponerles una fecha de expiración anterior a la actual: document.cookie = "username=; expires=Thu, 01 Jan 2020 00:00:00 UTC; path=/;";
Entonces, para crear una cookie solo hay que asignar un valor a document.cookie.

15. ¿Cuál es la diferencia entre var, let y const?

Var es la forma de declarar variables más antigua de JS, y son variables que pueden ser locales o globales y pueden volver a ser modificadas después de haber sido declaradas. En JS estar variables también se declaran con undefined en el código antes de que se les asigne un valor, o sea, existen desde antes que se declaren en el código, a la hora de ejecutar. Además, las var pueden declararse varias veces, lo que hace que se puedan modificar indeseadamente si se usa dos veces una variable con el mismo nombre que en un principio no era su intención primera.
Let soluciona los errores de var permitiendo una única declaración y manteniendo el contexto de su utilización dentro del bloque (entre {}) que se declare. Esto permite que una variable del mismo nombre sea diferente si se declaran en ámbitos diferentes. Let se declara al principio del código igual que var pero no se inicializa con undefined.
Const por último se refiere a variables que mantienen valores constantes. Declarativamente son parecidas a let, con la característica de no poder ser modificables.

Referencias

- https://www.w3schools.com/html/html_forms.asp
- https://getbootstrap.com/docs/4.0/components/navs/
- https://www.w3schools.com/bootstrap/bootstrap_get_started.asp
- https://www.geeksforgeeks.org/how-many-ways-to-use-less/

- https://www.geeksforgeeks.org/css-preprocessor-less/