Universidad de Costa Rica

CI-0126 Ingeniería de Software

Grupo 3

Laboratorio:
#2B

Docente:
Msc. Rebeca Obando Vásquez

Asistente:
Esteban Iglesias

Alumno:

Jesús Mena Amador

Carnet:
B54291

Enlace al repositorio de github

Fecha de Entrega: 26 de Marzo 2025

Contents

1 ¿Qué es JavaScript?

JavaScript es un lenguaje de programación que los desarrolladores utilizan para hacer páginas web interactivas. Desde actualizar fuentes de redes sociales a mostrar animaciones y mapas interactivos, las funciones de JavaScript pueden mejorar la experiencia del usuario de un sitio web. Como lenguaje de scripting del lado del servidor, se trata de una de las principales tecnologías de la World Wide Web. Por ejemplo, al navegar por Internet, en cualquier momento en el que vea un carrusel de imágenes, un menú desplegable "click-to-show" (clic para mostrar), o cambien de manera dinámica los elementos de color en una página web, estará viendo los efectos de JavaScript.

1.1 ¿Cuáles son los data types que soporta javascript?

JavaScript admite los siguientes tipos de datos primitivos:

- **Boolean**
- **String**
- **Null:** Representa la ausencia intencional de cualquier valor de objeto
- **Undefined:** Indica que una variable ha sido declarada pero aún no se le ha asignado un valor
- **Number:** Números enteros o de punto flotante.
- **BigInt:** Números enteros de precisión arbitraria.
- **Symboll:** Representa un identificador único.

Además, JavaScript tiene el tipo de dato **Object**, que es una colección de propiedades. (JavaScript data types and data structures)

1.2 ¿Cómo se puede crear un objeto en javascript?

Para crear un objeto en JavaScript existen diversas formas:

1. **Usando un objeto literal:**

```
const persona = {
  nombre: "Juan",
  edad: 30
};
```

2. **Usando el constructor `Object`:**

```
const persona = new Object();
persona.nombre = "Juan";
persona.edad = 30;
```

3. **Usando `Object.create` para crear un nuevo objeto con un prototipo específico:**

```
const prototipo = {
  saludar() {
    console.log("Hola");
  }
};

const persona = Object.create(prototipo);
persona.nombre = "Juan";
```

(Working with objects)

1.3 ¿Cuáles son los alcances (scope) de las variables en javascript?

En JavaScript, el alcance de una variable determina dónde se puede acceder a ella,:

- **Alcance global:** Una variable declarada fuera de cualquier función o bloque tiene un alcance global y es accesible desde cualquier parte del código.
- **Alcance de función:** Una variable declarada dentro de una función utilizando `var` es accesible solo dentro de esa función.
- **Alcance de bloque:** Variables declaradas con `let` o `const` dentro de un bloque tienen un alcance limitado a ese bloque.

(Scope)

1.4 ¿Cuál es la diferencia entre undefined y null?

La diferencia es que aunque ambos son valores primitivos `undefined` indica que una variable ha sido declarada pero no se le ha asignado ningún valor, mientras que `null` representa la ausencia intencional de cualquier valor de objeto.
(undefined) (null)

1.5 ¿Qué es el DOM?

El Document Object Model (DOM) es una interfaz de programación que representa la estructura de un documento HTML o XML como un árbol de nodos. Permite a los programas y scripts acceder y manipular dinámicamente el contenido, la estructura y el estilo de los documentos. (Document Object Model (DOM))

1.6 ¿Qué hacen, que retorna y para qué funcionan las funciones getElement y querySelector?

1. La función `getElementBy*` devuelve una referencia al elemento que tiene el atributo * que coincide con el valor especificado. Si no se encuentra ningún elemento, devuelve null, en el lugar del asterisco pueden ir:
   - **Id** (getElementsByID())
   - **ClassName** (getElementsByClassName())
   - **Name** (getElementsByName())
   - **TagName** (getElementsByTagName())
   - **TagNameNS** (getElementsByTagNameNS())
2. La función `querySelector` devuelve el primer elemento dentro del documento que coincide con el selector CSS especificado. Si no se encuentra ningún elemento, devuelve null. (querySelector()

Un ejemplo de uso de ambas funciones, para `getElementBy*` utilizaremos `getElementByID`:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Ejemplo getElement y querySelector</title>
</head>
<body>
  <div id="miId">Elemento con ID</div>
  <div class="miClase">Primer elemento con clase</div>
  <div class="miClase">Segundo elemento con clase</div>

  <script>
    const porId = document.getElementById("miId");
    console.log(porId.textContent); // "Elemento con ID"

    const porClase = document.querySelector(".miClase");
    console.log(porClase.textContent); // "Primer elemento con clase"
  </script>
</body>
</html>
```

1.7 Investigue cómo se pueden crear nuevos elementos en el DOM usando Javascript.

Para crear nuevos elementos en el DOM, se utiliza el método `document.createElement()`, que crea un nuevo nodo de elemento del tipo especificado. (createElement()

Un ejemplo de uso es el siguiente:

```
// Crear un nuevo párrafo
const nuevoParrafo = document.createElement("p");
nuevoParrafo.textContent = "Este es un nuevo párrafo.";

// Agregar el nuevo párrafo al cuerpo del documento
document.body.appendChild(nuevoParrafo);
```

1.8 ¿Cuál es el propósito del operador this?

El operador `this` en JavaScript se refiere al objeto que está ejecutando la función actual. Su valor depende de cómo se invoque la función.

Casos principales de uso de this:

- En el contexto global (modo no estricto)
- Dentro de un objeto (Métodos de objetos)
- Dentro de una función (modo estricto)
- Dentro de una función en un objeto (pero con una función anidada)
- Con `call()`, `apply()` y `bind()`
- En una función flecha (`=>`): Las funciones flecha no tienen su propio this, sino que heredan el this del contexto donde fueron definidas

(this)

1.9 ¿Qué es un promise en Javascript?

Una promesa en JavaScript es un objeto que representa la eventual finalización (o falla) de una operación asincrónica y su valor resultante. Proporciona una forma más limpia de manejar operaciones asincrónicas en comparación con las devoluciones de llamada (callbacks). (Promise)

Un ejemplo con la función `obtenerDatos` que devuelve una promesa que se resuelve después de 2 segundos si la operación es exitosa y se rechaza si hay un error:

```
function obtenerDatos() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      const exito = true;
      if (exito) {
        resolve("Datos obtenidos exitosamente");
      } else {
        reject("Error al obtener datos");
      }
    }, 2000);
  });
}

obtenerDatos()
  .then((mensaje) => console.log(mensaje))
  .catch((error) => console.error(error));
```

1.10 ¿Qué es Fetch en Javascript?

La API Fetch proporciona una interfaz para realizar solicitudes HTTP de forma asincrónica. Es una alternativa moderna a `XMLHttpRequest` y devuelve promesas, facilitando el manejo de respuestas y errores. (Uso de Fetch)

En el siguiente ejemplo, se realiza una solicitud GET a una API y se maneja la respuesta o cualquier error que ocurra:

```
fetch('https://api.ejemplo.com/datos')
  .then(response => {
    if (!response.ok) {
      throw new Error('Error en la solicitud');
    }
    return response.json();
  })
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));
```

1.11 ¿Qué es Async/Await en Javascript?

`async` y `await` son palabras clave que permiten escribir código asincrónico de manera más síncrona y legible. Una función declarada con `async` devuelve una promesa, y `await` se utiliza dentro de funciones `async` para esperar la resolución de una promesa. (Función async)

En el siguiente ejemplo, `obtenerDatos` es una función asincrónica que utiliza `await` para esperar la respuesta de fetch y luego procesa los datos:

```
async function obtenerDatos() {
  try {
    const response = await fetch('https://api.ejemplo.com/datos');
    if (!response.ok) {
      throw new Error('Error en la solicitud');
    }
    const data = await response.json();
    console.log(data);
  } catch (error) {
    console.error('Error:', error);
  }
}

obtenerDatos();
```

1.12 ¿Qué es un Callback?

Una función de devolución de llamada (callback) es una función que se pasa como argumento a otra función y se invoca dentro de esa función para completar una acción o rutina. (Función Callback)

En el siguiente ejemplo, `mostrarNombre` se pasa como callback a `procesarUsuario` y se invoca con el objeto `usuario`:

```
function procesarUsuario(id, callback) {
  const usuario = { id: id, nombre: 'Juan' };
  callback(usuario);
}

function mostrarNombre(usuario) {
  console.log('Nombre del usuario:', usuario.nombre);
}

procesarUsuario(1, mostrarNombre);
```

1.13 ¿Qué es Clousure?

Un closure es la combinación de una función y el ámbito léxico en el que se definió. Permite que una función acceda a variables de su ámbito exterior incluso después de que ese ámbito haya finalizado. (Closures)

1.14 ¿Cómo se puede crear un cookie usando Javascript?

En el navegador, puedes crear nuevas cookies mediante JavaScript utilizando la propiedad `document.cookie`. (document.cookie)

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

La diferencia radica encuando es mejor utilizar una u otra:

- `var`: Tiene ámbito de función y puede ser redeclarada en el mismo ámbito. Las variables declaradas con var se elevan (hoisting) al inicio de su ámbito, hay que tratar de evitar utilizarla debido a su ámbito amplio y su comportamiento inesperado con hoisting.
- `let`: Tiene ámbito de bloque y no puede ser redeclarada en el mismo ámbito. Es preferible a var cuando se necesita reasignar el valor de la variable, es la recomendable en la mayoría de los casos, ya que evita problemas de ámbito.
- `const`: Tiene ámbito de bloque y no puede ser redeclarada ni reasignada. Se utiliza para declarar variables cuyo valor no cambiará, se utiliza cuando no se necesita reasignar valores (ej. configuraciones, constantes).

(var), (let) y (const)

2 Bibliografía

- msn web docs: JavaScript data types and data structures
- msn web docs: Working with objects
- msn web docs: Scope
- msn web docs: undefined
- msn web docs: null
- msn web docs: Document Object Model (DOM)
- msn web docs: getElementById() method
- msn web docs: getElementsByClassName() method
- msn web docs: getElementsByName() method
- msn web docs: getElementsByTagName() method
- msn web docs: getElementsByTagNameNS() method
- msn web docs: querySelector() method
- msn web docs: createElement() method
- msn web docs: this
- msn web docs: Promise
- msn web docs: Uso de Fetch
- msn web docs: Función async
- msn web docs: Función Callback
- msn web docs: Closures
- msn web docs: document.cookie
- msn web docs: var
- msn web docs: let
- msn web docs: const