Laboratorio #2.B

Luis Esteban Chacón Chaves – C22039

1. Números, strings , booleanos, símbolos, objetos, arreglos, variables nulas y variables indefinidas
2. Indicando el nombre del objeto, los nombres de sus atributos y los valores de estos atributos

   Ejemplo:

```
let book = {
  title: "Efectos Personales",
  author: "Faustino Desinach",
  numberOfPages: 362
};
```

3. Se refiere a los lugares desde los que se puede acceder una variable
4. Que *undefined* es asignada a variables que no han sido definidas (pero sí declaradas) y *null* se puede asignar a una variable intencionalmente para indicar que no tiene un valor
5. Es el *Document Object Model.* Es la manera que tiene Javascript de interactuar con el contenido de un archivo HTML. Consiste en una estructura tipo árbol de nodos. Cada nodo representa una parte de la estructura HTML del archivo
6. **getElementById:**
   - Qué hace: retorna un elemento del DOM con el id especificado
   - Retorna: el primer elemento con el id especificado. Si no encuentra ninguno, retorna null
   - Para qué sirve: acceder a un elemento con un id específico

   getElementsByClassName:
   - Qué hace: retorna todos los elementos de una clase especificada
   - Retorna: una colección de todos los nodos con la clase especificada

- Para qué sirve: para coleccionar todos los elementos de una misma clase

getElementsByTagName:

- Qué hace: retornar los elementos de cierto tipo de tag
- Retorna: una colección de nodos con el tipo de tag especificado
- Para qué srive: para coleccionar todos los elementos de un tipo de tag

querySelector:

- Qué hace: retornar un elemento seleccionado mediante un selector de CSS
- Retorna: retorna el primer elemento que coincida con el selector de CSS o nulos si no encuentra ninguno
- Para qué sirve: seleccionar elementos utilizando la misma sintaxis que CSS

querySelectorAll:

- Es lo mismo que querySelector pero devuelve una colección de los elementos que coinciden con el selector de CSS.

Ejemplo:

```
var div = document.getElementById("myDiv");

var paragraphs = document.getElementsByClassName("paragraph");

var firstParagraph = document.querySelector(".paragraph");

var allParagraphs = document.querySelectorAll(".paragraph");
```

7. Por medio del método document.createElement(), después se puede insertar en el DOM con appendChild(), insertBefore() o append()

Ejemplo:

```
var newParagraph = document.createElement("p");
newParagraph.textContent = "Hello World";

var content = document.querySelector("div .content");
content.appendChild(newParagraph);
```

8. Es una forma de hacer referencia al objeto al que pertenece la función o el código que se está ejecutando.
9. Es un objeto que representa el estado de una operación asincrónica. Puede ser *pending, fullfiled o rejected*

   Ejemplo:

```
let promise = new Promise((resolve, reject) => {
  let operation = true;

  if(operation) {
      resolve("Task completed");
  } else {
      reject("An error ocured");
  }
});
```

10. Es una función que se utiliza para hacer solicitudes HTTP asincrónicas. Devuelve una *promise* que se resuelve con la respuesta de la solicitud.

    Ejemplo:

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

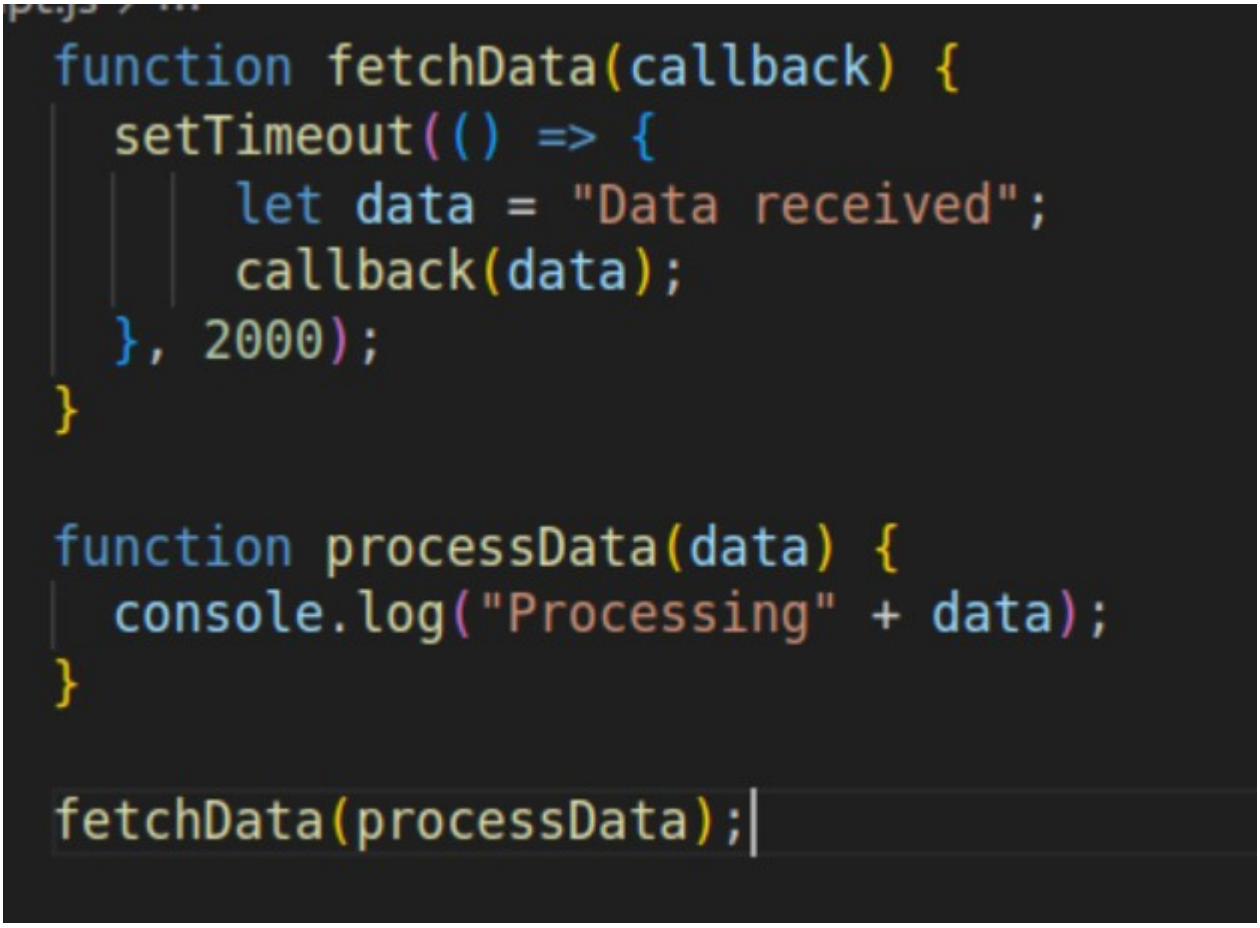
11. Son una forma de manejar operaciones asincrónicas de manera sencilla. Se usan junto a *promises* para escribir código asincrónico de forma más parecida a código sincrónico. *Async* se usa para declarar una función que contiene código asincrónico. *Await* se usa adentro de una función declarada con *async* para esperar a que se resuelva o rechace una *promise.*

    Ejemplo:

```
async function getData() {
  let response = await fetch('https://api.example.com/data');
  let data = await response.json();
  console.log(data);
}
```

12. Es una función que se pasa como argumento a otra función y se ejecuta después de que la función haya completado su tarea

    Ejemplo:



```
function fetchData(callback) {
  setTimeout(() => {
      let data = "Data received";
      callback(data);
  }, 2000);
}

function processData(data) {
  console.log("Processing" + data);
}

fetchData(processData);
```

13. Una función que tiene acceso a su propio scope, el de la función que la rodea y al scope global. Esto incluso después de que la función externa haya terminado su ejecución
14. Usando document.cookie y definiendo parámetros como nombre, valor, y opcionalmente fecha de expiración, dominio y la ruta, entre otros
15. Que *var* tiene *scope* de función o global, y se puede redefinir, además es *hoisted a undefined* antes de ejecución. *Let* tiene *scope* y no permite redefiniciones. *Const* también tiene alcance de bloque y no permite reasignación, pero si es un objeto o arreglo se pueden cambiar sus propiedades.