Bom – Browser Object Model

The Browser Object Model (BOM) is a set of objects provided by web browsers to interact with the browser itself, beyond just manipulating the content of a web page. It provides JavaScript access to various components of the browser environment, such as the browser window, history, location, and more.

Window Object:

Get the width and height of the browser window

```
console.log(window.innerWidth);
console.log(window.innerHeight);
window.open("https://example.com","_blank",
"width=600,height=400");
```

Navigator objects

The Navigator object in JavaScript provides information about the browser's name, version, platform, and capabilities.

Properties

```
console.log("Browser Name:", navigator.appName);
```

Example output: "Browser Name: Netscape"

```
console.log("Browser Version:", navigator.appVersion);
```

Example output: "Browser Version: 5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.99 Safari/537.36"

```
console.log("Platform:", navigator.platform);
```

Example output: "Platform: Win32"

```
console.log("User Agent:", navigator.userAgent);
```

Example output: "User Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.99 Safari/537.36"

```
console.log("Cookies Enabled:", navigator.cookieEnabled);
Example output: "Cookies Enabled: true"
```

Methods

```
console.log("Java Enabled:", navigator.javaEnabled());
// Example output: "Java Enabled: false" (Depends on the browser and user settings)
```

Location objects

It provides properties that allow you to access and manipulate different parts of the URL

```
// Properties
console.log("Full URL:", location.href);
// Example output: "Full URL:
https://www.example.com/path/?query=string"

console.log("Hostname and Port:", location.host);
// Example output: "Hostname and Port: www.example.com"

console.log("Hostname:", location.hostname);
// Example output: "Hostname: www.example.com"

console.log("Protocol:", location.protocol);
// Example output: "Protocol: https:"

console.log("Pathname:", location.pathname);
// Example output: "Pathname: /path/"
```

Screen objects

Screen object in JavaScript provides information about the user's screen or display, such as its width, height, color depth, and pixel density.

```
console.log("Screen Width:", screen.width);
```

```
// Example output: "Screen Width: 1920" (Width of the screen in pixels)

console.log("Screen Height:", screen.height);
// Example output: "Screen Height: 1080" (Height of the screen in pixels)

console.log("Available Screen Width:", screen.availWidth);
// Example output: "Available Screen Width: 1920" (Available width of the screen in pixels)

console.log("Available Screen Height:", screen.availHeight);
// Example output: "Available Screen Height: 1040" (Available height of the screen in pixels)

console.log("Color Depth:", screen.colorDepth);
// Example output: "Color Depth: 24" (Color depth of the screen in bits per pixel)

console.log("Pixel Depth:", screen.pixelDepth);
// Example output: "Pixel Depth: 24" (Pixel depth of the screen in bits per pixel)
```

History objects

History object in JavaScript represents the user's navigation history for the current browser window. It allows you to navigate back and forward through the history stack

```
// Methods
history.back(); // Moves the browser back one page
history.forward(); // Moves the browser forward one page
history.go(-2); // Moves the browser back two pages
```

Cookie objects

Cookies are small pieces of data stored in the user's web browser. They are typically used by websites to remember users' preferences, authentication status, and other information related to their browsing session. Cookies can be set, retrieved, and deleted using JavaScript and are commonly used for tasks like personalization, tracking, and session management on the web.

Timing functions

Timing functions are crucial for managing when and how often certain blocks of code execute. There are several methods and concepts related to timing functions in JavaScript:

1. setTimeout()

The setTimeout() function is used to execute a specified block of code once after a specified time interval.

```
setTimeout(function, delay, param1, param2, ...)
```

- function: A function or code snippet to execute.
- delay: Time in milliseconds (1000 ms = 1 second) before executing the function.
- param1, param2, ...: Optional additional parameters to pass to the function.

```
setTimeout(function() {
        console.log("This message will appear after 2000 milliseconds.");
}, 2000);
```

2. setInterval()

The setInterval() function is used to repeatedly execute a specified block of code at a fixed interval.

```
setInterval(function, delay, param1, param2, ...)
```

- function: A function or code snippet to execute.
- delay: Time in milliseconds between each execution of the function.
- param1, param2, ...: Optional additional parameters to pass to the function.

```
var count = 0;
var intervalId = setInterval(function() {
    count++;
    console.log("Counter: " + count);
    if (count === 5) {
        clearInterval(intervalId); // Stop the interval after 5 executions
    }
}, 1000); // Execute every 1000 milliseconds (1 second)
```

clearTimeout() and clearInterval()

- clearTimeout(timeoutId): Clears the timeout specified by timeoutId, typically returned by setTimeout(), canceling the execution of the function.
- clearInterval(intervalId): Clears the interval specified by intervalId, typically returned by setInterval(), stopping further executions.

Session storage and local storage

Session storage is a part of the Web Storage API in web browsers that provides a way to store key-value pairs locally on the client-side.

sessionStorage maintains a separate storage area for each given origin that's available for the duration of the page session (as long as the browser is open, including page reloads and restores).
Data stored in **sessionStorage** is cleared when the page session ends.
Data is only accessible within the window/tab that set it.

```
// Storing data in sessionStorage
sessionStorage.setItem('username', 'John');
// Retrieving data from sessionStorage
let username = sessionStorage.getItem('username');
console.log(username); // Output: John
// Removing data from sessionStorage
sessionStorage.removeItem('username');
```

localStorage is a feature of web browsers that allows web applications to store key-value pairs locally on the client-side. It provides a persistent storage mechanism, meaning that the data stored in **localStorage** remains available even after the browser is closed and reopened, and across browser sessions.

localStorage does almost the same thing as **sessionStorage**, but it persists even when the browser is closed and reopened.
Data stored in **localStorage** has no expiration time.
Data is accessible across windows and tabs within the same origin.

```
// Storing data in localStorage
localStorage.setItem('email', 'example@example.com');
```

```
// Retrieving data from localStorage
let email = localStorage.getItem('email');
console.log(email); // Output: example@example.com
// Removing data from localStorage
localStorage.removeItem('email');
```

how to display some data from one page to another page using local storage

local storage limited to handle only string key/value pairs you can do like below using JSON.stringify and while getting value JSON.parse
var testObject ={name:"test", time:"Date 2017-02-03T08:38:04.449Z"};
Put the object into storage:

localStorage.setItem('testObject', JSON.stringify(testObject));

Retrieve the object from storage:

var retrievedObject = localStorage.getItem('testObject');

console.log('retrievedObject: ', JSON.parse(retrievedObject));