6. JavaScript Browser BOM

JS Window, JS Screen, JS Location, JS History, JS Navigator, JS Popup Alert, JS Timing, JS Cookies

Browser object Model(BOM)

The **Browser Object Model** (BOM) is used to interact with the browser.

The default object of browser is window means you can call all the functions of window by specifying window or directly. For example:

window.alert("hello world");

is same as:

alert("hello world");

You can use a lot of properties defined underneath the window object like document, history, screen, navigator, location, innerHeight, and innerwidth.

JS Windows

The **JS window object** represents a window in the browser. An object of the window is created automatically by the browser.

Window is the object of browser, **it is not the object of javascript**. The javascript objects are string, array, date etc.

Methods of window object:

Examples of each methods:

1. **alert() :** It displays an alert dialog box. It has a message and ok button.

```
<script type="text/javascript">
function msg(){
alert("Hello Alert Box");
}
</script>
<input type="button" value="click" onclick="msg()"/>
```

Output:

2. **Confirm() :** It displays the confirm dialog box. It has a message with ok and cancel buttons.

```
<script type="text/javascript">
function msg(){
var v= confirm("Are u sure?");
if(v==true){
alert("ok");
}
else{
alert("cancel");
}
}
</script>
<input type="button" value="delete record" onclick="msg()"/>
```

Output:

3. **Prompt():**It displays a prompt dialog box for input. It has a message and textfield.

```
<script type="text/javascript">
```

```
function msg(){
var v= prompt("Who are you?");
alert("I am "+v);
}
</script>
 <input type="button" value="click" onclick="msg()"/>
```

Output:

4. **Open() :** It displays the content in a new window.

```
<script type="text/javascript">
function msg(){
open("https://www.uvpce.ac.in/");
}
</script>
<input type="button" value="UVPCE" onclick="msg()"/>
```

Output:

5. **setTimeout():**It performs its task after the given milliseconds.

```
<script type="text/javascript">
function msg(){
setTimeout(
function(){
```

```
alert("Welcome to UVPCE after 2 seconds")
},2000);}
</script>

<input type="button" value="click" onclick="msg()"/>
```

Output:

6. Calculating Width and Height of window

The window object provides the innerWidth and innerHeight property to find out the width and height of the browser window viewport (in pixels) including the horizontal and vertical scrollbar, if rendered.

Example:

```
<script>
function windowSize(){
        var w = window.innerWidth;
        var h = window.innerHeight;
        alert("Width: " + w + ", " + "Height: " + h);
        }
</script>
<button type="button" onclick="windowSize();">Get Window Size</button>
```

It displays the current size of the window on the click button .

Output:

JS Screen

The **JavaScript screen object** holds information of browser screen. It can be used to display screen width, height, colorDepth, pixelDepth etc.

The navigator object is the window property, so it can be accessed by:

window.screen or screen

Properties of JS screen:

Example:

```
<script>
document.writeln("<br/>screen.width: "+screen.width);
document.writeln("<br/>screen.height: "+screen.height);
document.writeln("<br/>screen.availWidth: "+screen.availWidth);
document.writeln("<br/>screen.availHeight: "+screen.availHeight);
document.writeln("<br/>screen.colorDepth: "+screen.colorDepth);
document.writeln("<br/>screen.pixelDepth: "+screen.pixelDepth);
</script>
```

Output:

JS Location

The location property of a window (i.e. window.location) is a reference to a Location object; it represents the current URL of the document being displayed in that window.

Since window object is at the top of the scope chain, so properties of the window.location object can be accessed without window. prefix, for example window.location.href can be written as location.href. The following section will show you how to get the URL of page as well as hostname, protocol, etc. using the location object property of the window object.

Examples:

1. Getting current page URL: Use **window.location.href** property to get the entire URL of the current page.

```
<script>

function getURL() {

alert("The URL of this page is: " + window.location.href);

}

</script><button type="button" onclick="getURL();">Get Page URL</button>
```

Output:

2. Getting different part of URL: Use other properties of the location object such as protocol, hostname, port, pathname, search, etc. to obtain different part of the URL.

```
// Prints complete URL

document.write(window.location.href);

// Prints protocol like http: or https:

document.write(window.location.protocol);

// Prints hostname with port like localhost or

localhost:3000

document.write(window.location.host);

// Prints hostname like localhost or www.example.com
document.write(window.location.hostname);

// Prints port number like 3000
```

```
document.write(window.location.port);

// Prints pathname like /products/search.php

document.write(window.location.pathname);

// Prints query string like ?q=ipad

document.write(window.location.search);

 // Prints fragment identifier like #featured

document.write(window.location.hash);
```

Output:

3. Loading new Documents: You can use the **assign()** method of the location object i.e. window.location.assign() to load another resource from a URL provided as parameter, for example:

```
<script>

function loadHomePage() {

                window.location.assign("https://www.ganpatuniversity.ac.in");}

</script>

<button type="button" onclick="loadHomePage();">Load Home Page</button>
```

You can also use the replace() method to load new document which is almost the same as assign(). The difference is that it doesn't create an entry in the browser's history, meaning the user won't be able to use the back button to navigate to it. Here's an example:

```
<script>

function loadHomePage()

{

window.location.replace("https://www.uvpce.ac.in");

}

</script>

<button type="button" onclick="loadHomePage();">Load Home Page</button>
```

Output:

4. Reloading the Page Dynamically: The **reload()** method can be used to reload the current page dynamically.

You can optionally specify a Boolean parameter true or false. If the parameter is true, the method will force the browser to reload the page from the server. If it is false or not specified, the browser may reload the page from its cache.

```
<script>

function forceReload() {

window.location.reload(true);

}

</script>

<button type="button" onclick="forceReload();">Reload Page</button>
```

JS History

The history property of the Window object refers to the History object. It contains the browser session history, a list of all the pages visited in the current frame or window.

Since Window is a global object and it is at the top of the scope chain, so properties of the Window object i.e. **window.history** can be accessed without window. prefix, for example **window.history.length** can be written as history.length.

The following section will show you how to get the information of user's browsing history.

Examples:

1. Getting the Number of pages visited: The **window.history.length** property can be used to get the number of pages in the session history of the browser for the current window. It also includes the currently loaded page.

```
<script>

function getViews() {

alert("You've accessed " + history.length + " web pages in this session.");

}

</script>

<button type="button" onclick="getViews();">Get Views Count</button>
```

Output:

2. Going back to the Previous Page: You can use the back() method of the History object i.e. **history.back()** to go back to the previous page in session history. It is the same as clicking the browser's back button.

```
<script>

function goBack() {

window.history.back();

}

</script>

<button type="button" onclick="goBack();">Go Back</button>
```

Output:

3. Going forward to the next page: You can use the forward() method of the History object i.e. **history.forward()** to go forward to the next page in session history. It is the same as clicking the browser's forward button.

```
<script>

function goForward() {

        window.history.forward();

}

</script>

<button type="button" onclick="goForward();">Go Forward</button>
```

Output:

4. Going to a Specific page: You can also load specific page from the session history using the go() method of the History object i.e. **history.go()**. This method takes an integer as a parameter. A negative integer moves backward in the history, and a positive integer moves forward in the history.

Example:

```
window.history.go(-2);  // Go back two pages

window.history.go(-1); // Go back one page

window.history.go(0);  // Reload the current page

window.history.go(1);  // Go forward one page

window.history.go(2);  // Go forward two pages
```

JS Navigator

The navigator property of a window (i.e. **window.navigator**) is a reference to a Navigator object; it is a read-only property which contains information about the user's browser.

Since Window is a global object and it is at the top of the scope chain, so properties of the Window object such as window.navigator can be accessed without window. prefix, for example **window.navigator.language** can be written as **navigator.language**.

Example:

1. Detect Whether the Browser is Online or Offline: You can use the **navigator.onLine** property to detect whether the browser (or, application) is online or offline. This property returns a Boolean value true meaning online, or false meaning offline.

```
<script>

function checkConnectionStatus() {

        if(navigator.onLine) {

                alert("Application is online.");

        } else { alert("Application is offline.");

         }

}

</script>

<button type="button" onclick="checkConnectionStatus();">Check Connection Status</button>
```

Output:

2. Check Whether Cookies are enabled or not: You can use the **navigator.cookieEnabled** to check whether cookies are enabled in the user's browser or not. This property returns a Boolean value true if cookies are enabled, or false if it isn't.

```
<script>

function checkCookieEnabled(){

            if(navigator.cookieEnabled) {

            alert("Cookies are enabled in your browser.");

           } else {

           alert("Cookies are disabled in your browser.");

           }

}

</script>

<button type="button" onclick="checkCookieEnabled();">Check If Cookies are Enabled</button>
```

Output:

3. Detecting the browser language: You can use the **navigator.language** property to detect the language of the browser UI. This property returns a string representing the language, e.g. "en", "en-US", etc.

```
<script>

function checkLanguage() {

        alert("Your browser's UI language is: " + navigator.language);

}

</script>

<button type="button" onclick="checkLanguage();">Check Language </button>
```

Output:

4. Getting Browser name and Version information: The Navigator object has five main properties that provide name and version information about the user's browser. The following list provides a brief overview of these properties:

- appName — Returns the name of the browser. It always returns "Netscape", in any browser.
- appVersion — Returns the version number and other information about the browser.
- appCodeName — Returns the code name of the browser. It returns "Mozilla", for all browser.
- userAgent — Returns the user agent string for the current browser. This property typically contains all the information in both appName and appVersion.
- platform — Returns the platform on which browser is running (e.g. "Win32", "WebTV OS", etc.)

```
<script>

function getBrowserInformation() { var info = "\n App Name: " + navigator.appName;

info += "\n App Version: " + navigator.appVersion;
```

```
info += "\n App Code Name: " + navigator.appCodeName;

 info += "\n User Agent: " + navigator.userAgent;

info += "\n Platform: " + navigator.platform;

alert("Here're the information related to your browser: " + info);

}

</script>

<button type="button" onclick="getBrowserInformation();">GetBrowser Information</button>
```

Output:

5. Check whether the browser is Java enabled or not: You can use the method javaEnabled() to check whether the current browser is Java-enabled or not. This method simply indicates whether the preference that controls Java is on or off, it does not reveal whether the browser offers Java support or Java is installed on the user's system or not.

```
<script>

function checkJavaEnabled() {

        if(navigator.javaEnabled()) {

        alert("Your browser is Java enabled.");
```

```
        } else {

        alert("Your browser is not Java enabled.");

        }

}

</script>

<button type="button" onclick="checkJavaEnabled();">Check If Java is Enabled</button>
```

Output:

JS Popup Alert

An alert dialog box is the most simple dialog box. It enables you to display a short message to the user. It also includes an OK button, and the user has to click this OK button to continue.

You can create alert dialog boxes with the alert() method.

Example:

```
var message = "Hi there! Click OK to continue.";

alert(message);

/* The following line won't execute until you dismiss previous alert */ alert("This is another alert box.");
```

Output:

JS Timing

A timer is a function that enables us to execute a function at a particular time.

Using timers you can delay the execution of code so that it does not get done at the exact moment an event is triggered or the page is loaded. For example, you can use timers to change the advertisement banners on your website at regular intervals, or display a real-time clock, etc. There are two timer functions in JavaScript: setTimeout() and setInterval().

Example:

1. Executing code after a Delay: The setTimeout() function is used to execute a function or specified piece of code just once after a certain period of time. Its basic syntax is setTimeout(function, milliseconds).

   This function accepts two parameters: a *function*, which is the function to execute, and an optional *delay* parameter, which is the number of milliseconds representing the amount of time to wait before executing the function (1 second = 1000 milliseconds). Let's see how it works:

```
<script>
function myFunction() {
    alert('Hello World!');
}
</script>
<button onclick="setTimeout(myFunction, 2000)">Click Me</button>
```

   The above example will display an alert message after 2 seconds on click of the button.

2. Executing code at Regular intervals: Similarly, you can use the setInterval() function to execute a function or specified piece of code repeatedly at fixed time intervals. Its basic syntax is setInterval(*function*, *milliseconds*).

This function also accepts two parameters: a *function*, which is the function to execute, and *interval*, which is the number of milliseconds representing the amount of time to wait before executing the function (1 second = 1000 milliseconds). Here's an example:

```
<script>
function showTime() {
    var d = new Date();
    document.getElementById("clock").innerHTML =
    d.toLocaleTimeString();
}
setInterval(showTime, 1000);
</script>
<p>The current time on your computer is: <span id="clock"></span></p>
```

Output:

The above example will execute the showTime() function repeatedly after 1 second. This function retrieves the current time on your computer and displays it in the browser.

3. Stopping Code Execution or Cancelling a Timer: Both setTimeout() and setInterval() method return an unique ID (a positive integer value, called timer identifier) which identifies the timer created by the these methods. This ID can be used to disable or clear the timer and stop the execution of code beforehand. Clearing a timer can be done using two functions: clearTimeout() and clearInterval().

The setTimeout() function takes a single parameter, an ID, and clear a setTimeout() timer associated with that ID, as demonstrated in the following example:

```
<script>
var timeoutID;
```

```
function delayedAlert() {
        timeoutID = setTimeout(showAlert, 2000);
       document.writeln(timeoutID);
}
function showAlert() {
alert('This is a JavaScript alert box.');
}
function clearAlert() {
clearTimeout(timeoutID);
}
</script>
<button onclick="delayedAlert();">Show Alert After Two Seconds</button>
<button onclick="clearAlert();">Cancel Alert Before It Display</button>
```

Output:

JS Cookie

What is Cookie?

A cookie is a small text file that lets you store a small amount of data (nearly 4KB) on the user's computer. They are typically used for keeping track of information such as user preferences that the site can retrieve to personalize the page when the user visits the website next time.

Cookies are an old client-side storage mechanism that was originally designed for use by server-side scripting languages such as PHP, ASP, etc. However, cookies can also be created, accessed, and modified directly using JavaScript, but the process is a little bit complicated and messy.

Creating a Cookie in JavaScript: To create or store a new cookie, assign a *name=value* string to this property, like this

```
document.cookie = "firstName=Christopher";
```

A cookie value cannot contain semicolons, commas, or spaces. For this reason, you will need to use JavaScript's built-in function encodeURIComponent() to encode the values containing these characters before storing it in the cookie. Likewise, you'll need to use the corresponding decodeURIComponent() function when you read the cookie value.

```
document.cookie = "name=" + encodeURIComponent("Christopher Columbus");
```

By default, the lifetime of a cookie is the current browser session, which means it is lost when the user exits the browser. For a cookie to persist beyond the current browser session, you will need to specify its lifetime (in seconds) with a max-age attribute. This attribute determines how long a cookie can remain on the user's system before it is deleted, e.g., the following cookie will live for 30 days.

```
document.cookie = "firstName=Christopher; max-age=" + 30*24*60*60;
```

You can also specify the lifetime of a cookie with the expires attribute. This attribute takes an exact date (in GMT/UTC format) when the cookie should expire, rather than an offset in seconds.

```
document.cookie = "firstName=Christopher; expires=Thu, 31 Dec 2099 23:59:59 GMT";
```

Here's a function that sets a cookie with an optional max-age attribute. You can also use the same function to delete a cookie by passing the value 0 for daysToLive parameter.

Exercise:

```
<script type="text/javascript">

function createCookie(cookieName,cookieValue,daysToExpire)
```

```
    {
            var date = new Date();
            date.setTime(date.getTime()+(daysToExpire*24*60*60*1000));
       document.cookie = cookieName + "=" + cookieValue + "; expires=" +
date.toGMTString();
    }
            function accessCookie(cookieName)
    {
            var name = cookieName + "=";
            var allCookieArray = document.cookie.split(';');
            for(var i=0; i<allCookieArray.length; i++)
            {
            var temp = allCookieArray[i].trim();
            if (temp.indexOf(name)==0)
            return temp.substring(name.length,temp.length);
              }
            return "";
    }
            function checkCookie()
    {
            var user = accessCookie("testCookie");
            if (user!="")
            alert("Welcome Back " + user + "!!!");
            else
            {
            user = prompt("Please enter your name");
            num = prompt("How many days you want to store your name on your computer?");
            if (user!="" && user!=null)
            {
         createCookie("testCookie", user, num);
            }
            }
    }

checkCookie();

    </script>
```

Output: