Session-06

Javascript in Browser

Thanos is on a mission to manipulate the website from browser using JavaScript pesto

Web Fundamentals

Session-06

JavaScript in Browser

Agenda : JavaScript in Browser

01 DOM Intro

02 DOM Manipulation

03 Window Object & APIs

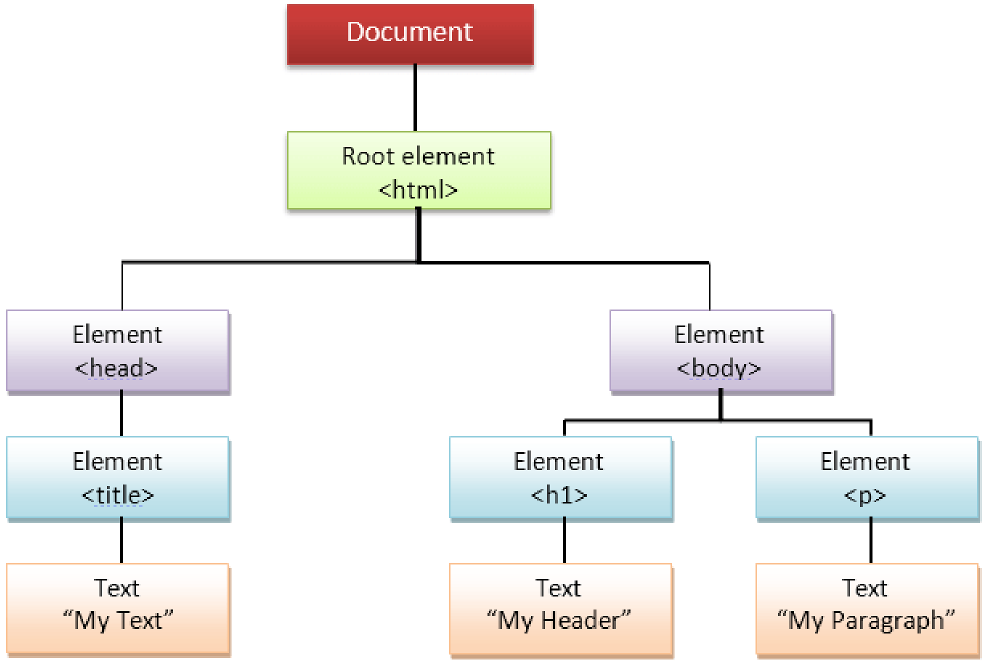
04 Local Storage, Session Storage, Cookies

05 PopUp, Location

06 Hands-On

DOM - Intro

- When a **web page is loaded, the browser creates** a Document Object Model of the page.
- DOM **is a tree of objects**
- It defines **properties, methods , events for all HTML elements**



JS DOM

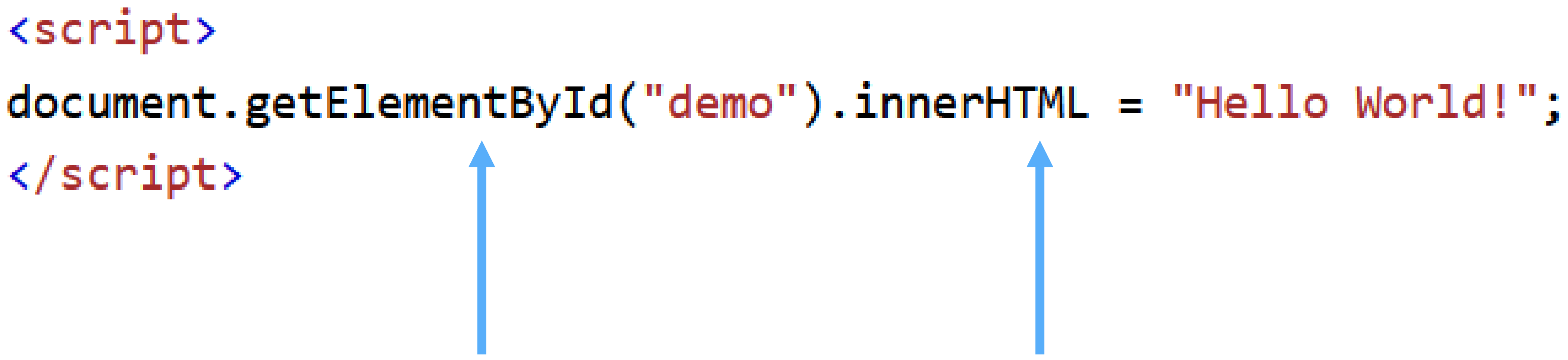
- JavaScript can **change all the HTML elements** in the page
- JavaScript can **change all the HTML attributes** in the page
- JavaScript can **change all the CSS styles** in the page
- JavaScript can **remove existing HTML elements** and attributes
- JavaScript can **add new HTML elements** and attributes
- JavaScript can **react to all existing HTML events** in the page
- JavaScript can **create new HTML events** in the page

Document Object

- The document object represents your web page.
- If you want to access any element in an HTML page, you always start with accessing the document object.

```
document.getElementById(id)
```

DOM Method & Property

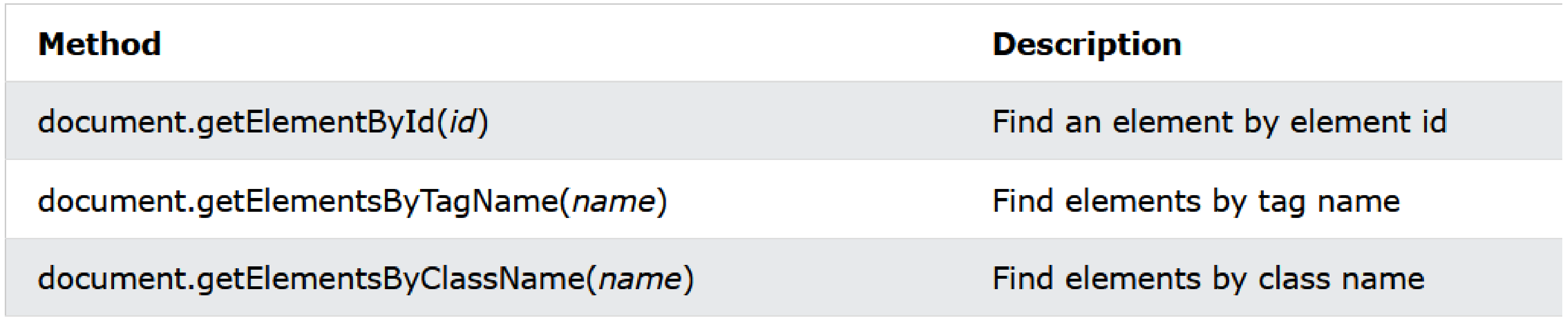


```
<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>
```

In the example above, **getElementById is a method,** while **innerHTML is a property.**

Finding Elements

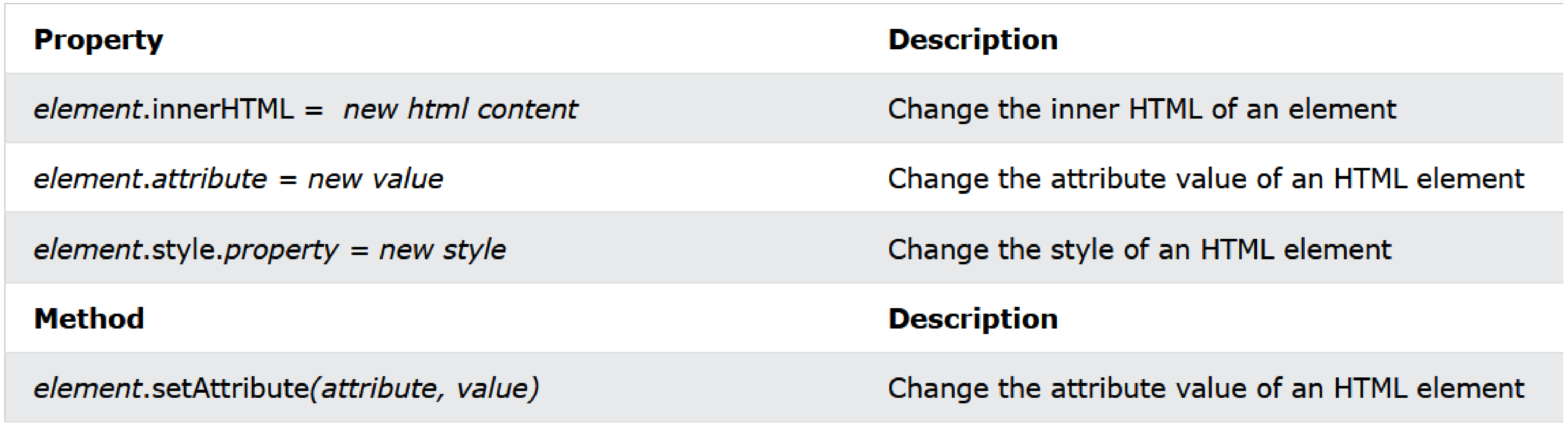
JavaScript can **Find any HTML elements** in the DOM by different attributes



Method	**Description**
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

Modify Elements

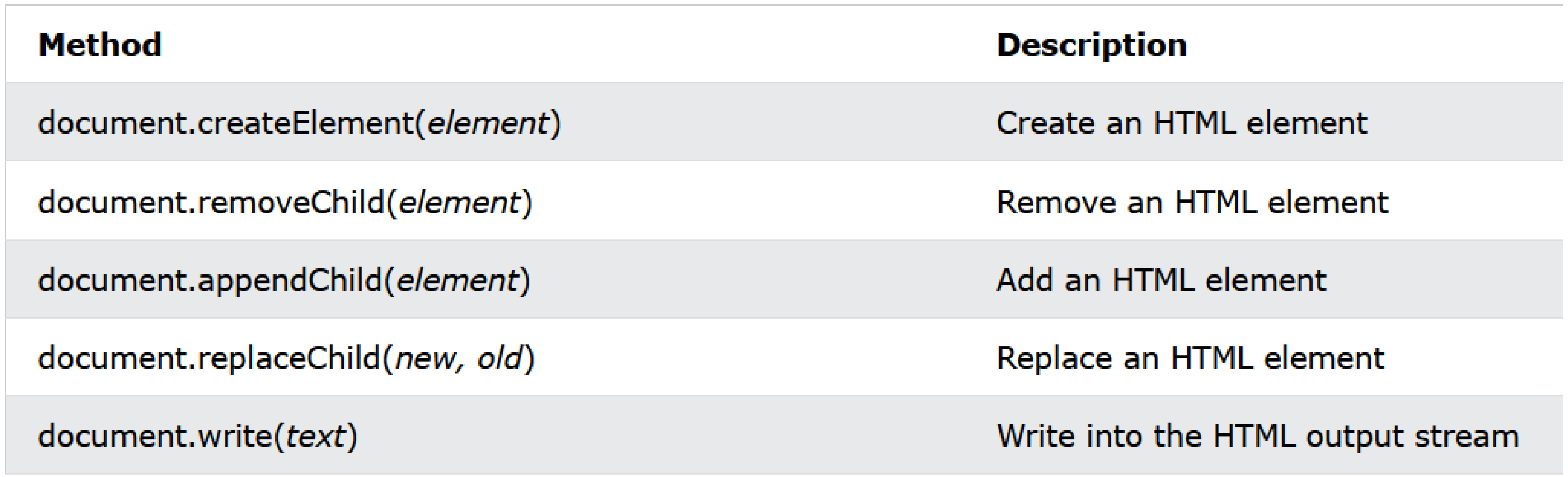
JavaScript can **Modify any HTML elements** in the DOM



Property	**Description**
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = *new value*	Change the attribute value of an HTML element
element.style.*property* = *new style*	Change the style of an HTML element
Method	**Description**
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element

Adding/Deleting Elements

JavaScript can **add or delete any HTML elements** in the DOM



Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*new, old*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

Changing HTML

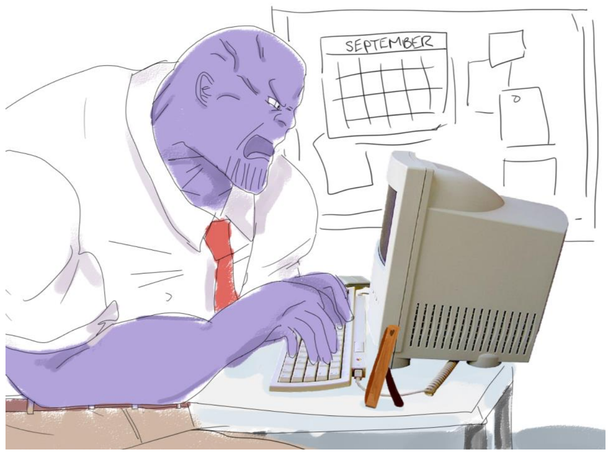
JS can modify **the content of an HTML element** is by using the ***innerHTML* property**.

```
<script>
document.getElementById("p1").innerHTML = "New text!";
</script>
```

Changing CSS

JS can modify **CSS Style of an HTML element** is by using the ***style* property**.

```
<script>
document.getElementById("p2").style.color = "blue";
</script>
```



This is good,How to handle click or any events?

DOM Events

JS can react to **any HTML events**

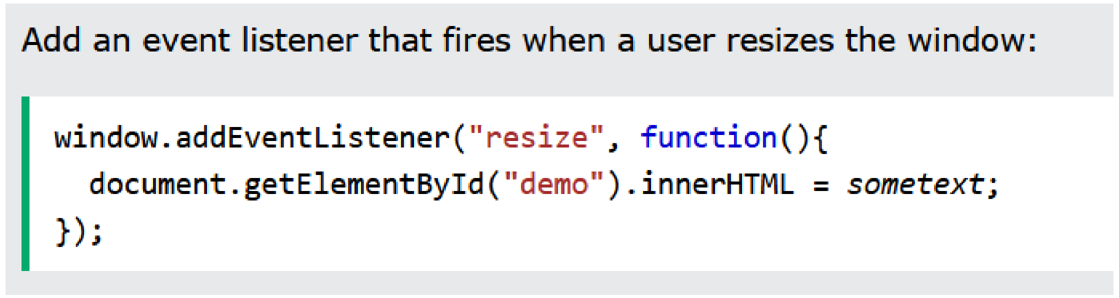
Examples of HTML events:

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

```
<h1 onclick="changeText(this)">Click on this text!</h1>

<script>
function changeText(id) {
  id.innerHTML = "Ooops!";
}
</script>
```

DOM Event Listener

- The **addEventListener()** method attaches an event handler to the specified element.
- The **addEventListener()** method attaches an event handler to an element without overwriting existing event handlers.
- You can **add many event handlers** to one element.
- You can **add many event handlers of the same type** to one element, i.e two "click" events.
- You can **add event listeners to any DOM object not only HTML elements**. i.e the window object.



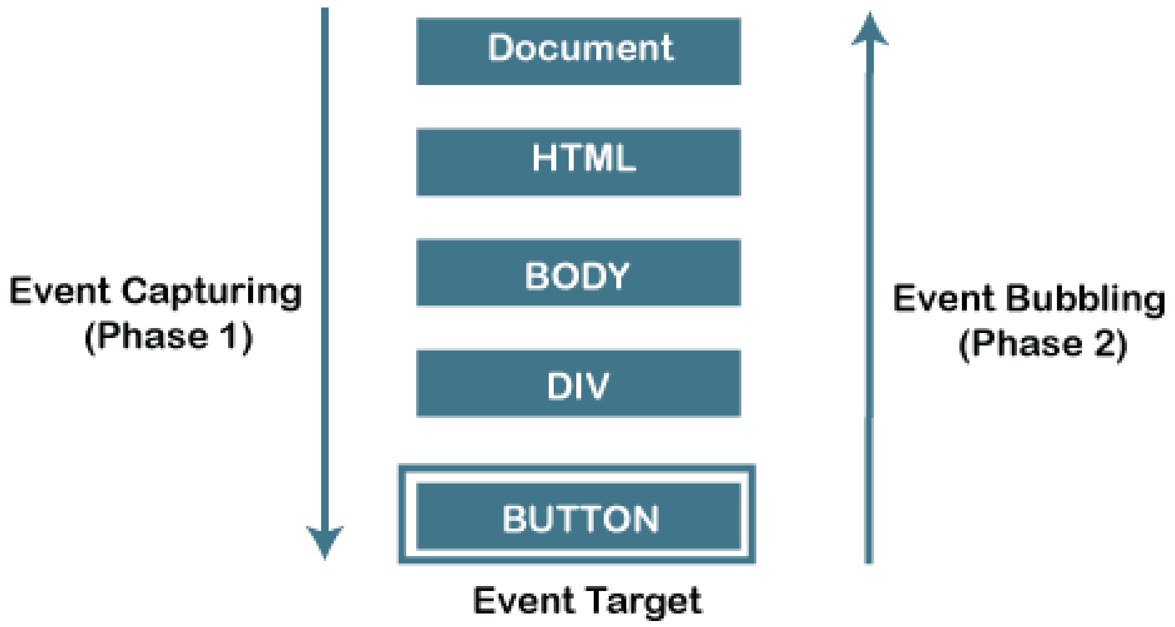
Add an event listener that fires when a user resizes the window:

```
window.addEventListener("resize", function(){
  document.getElementById("demo").innerHTML = sometext;
});
```

Event Capturing & Bubbling Phase

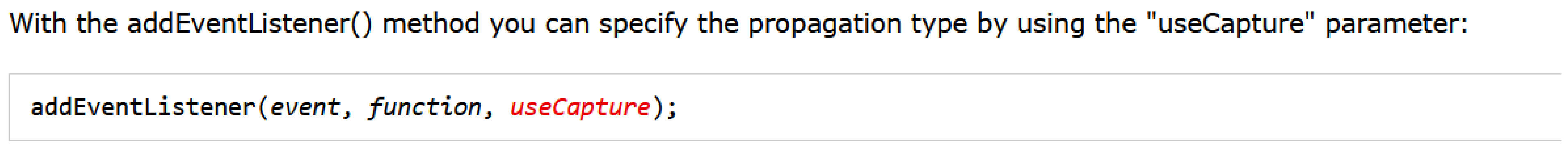
There are two ways of **event propagation in the HTML DOM, bubbling and capturing**.

- **Event Bubbling** is the process by which **an event travels from the deepest child element that triggered the event, up to the document root.**
- **Event capturing** is the process of capturing an event as it travels down the DOM tree, from the outermost element to the target element.



Event Capturing & Bubbling: Syntax

- **Event Bubbling** is the process by which **an event travels from the deepest child element that triggered the event, up to the document root.**
- **Event capturing** is the process of capturing an event as it travels down the DOM tree, from the outermost element to the target element.

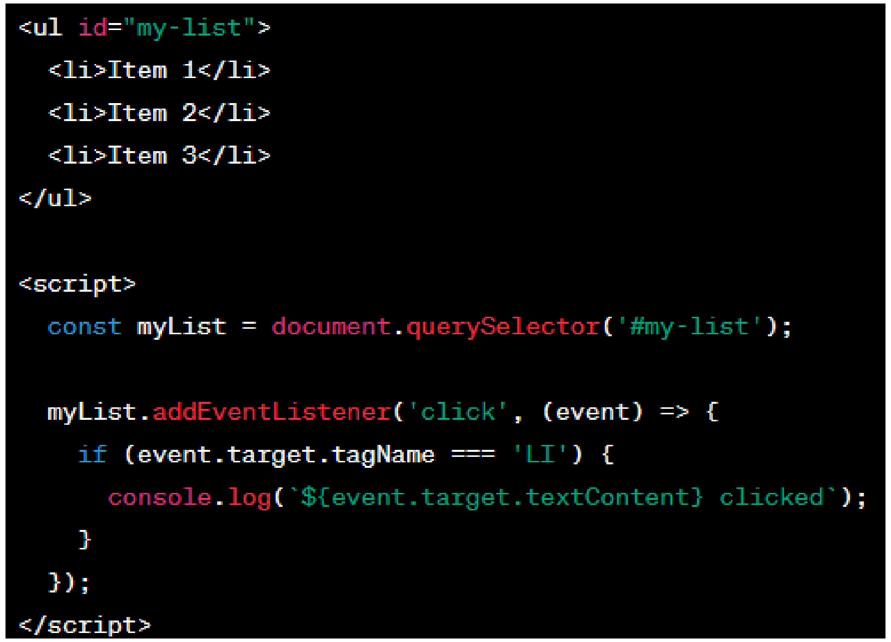

With the addEventListener() method you can specify the propagation type by using the "useCapture" parameter:

```
addEventListener(event, function, useCapture);
```

- The default value is false, which will use the bubbling propagation,
- when the value is set to true, the event uses the capturing propagation.

Event Delegation

- **Event delegation** is a technique that involves **attaching event listeners to a parent element** instead of individual child elements.
- When an event occurs on a child element**, the event bubbles up to the parent element where the event listener is triggered.**
- This technique **can improve performance and reduce memory consumption** when working with **a large number of child elements.**



```html
<ul id="my-list">
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>

<script>
  const myList = document.querySelector('#my-list');

  myList.addEventListener('click', (event) => {
    if (event.target.tagName === 'LI') {
      console.log(`${event.target.textContent} clicked`);
    }
  });
</script>
```

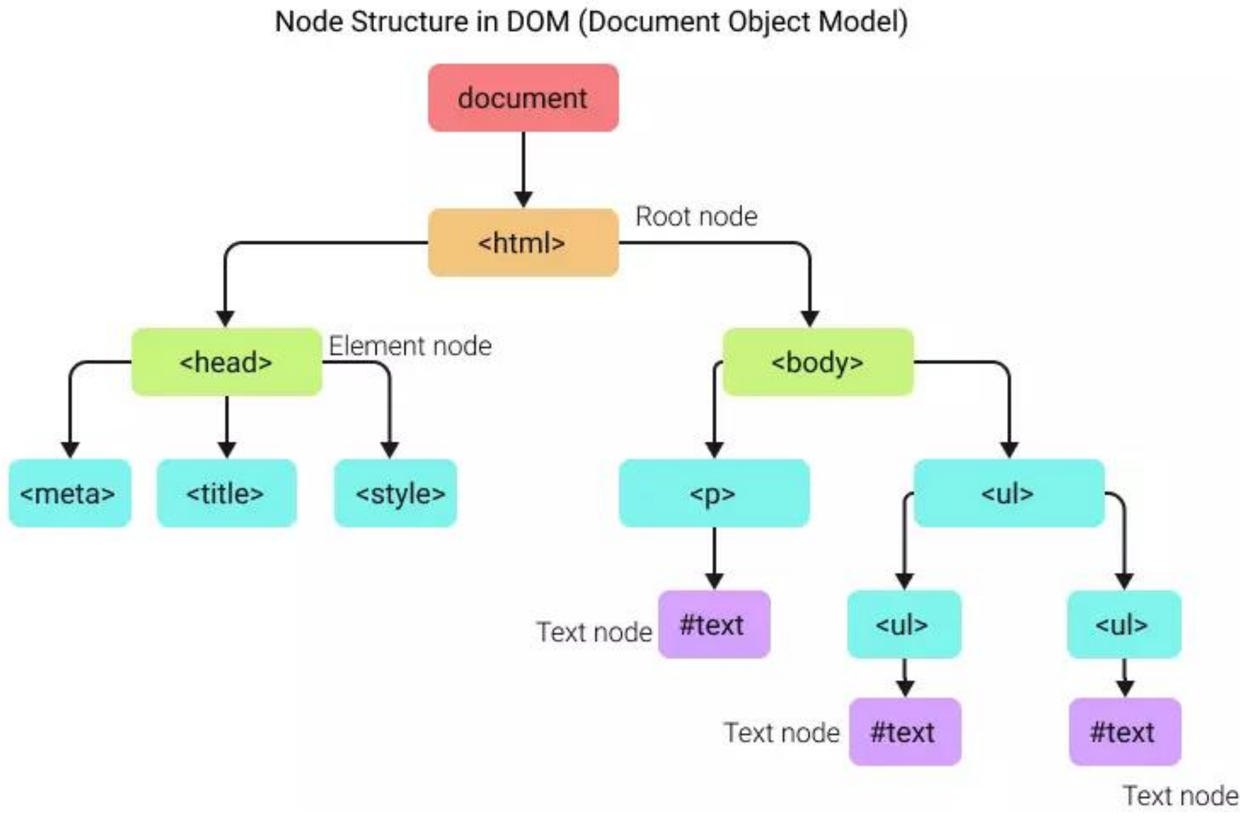
Yes, I use Event delegation, bubbling to handle performance in DOM

Knowledge check

Give real example use case of Event delegation

DOM Nodes

- In the (DOM), a web page is represented as a **tree structure of nodes.**
- Each node can contain other nodes, and **all the nodes are connected through parent-child relationships.**



DOM Nodes

- There are several types of nodes in the DOM, but the most commonly used ones are:
- **Element nodes:** represent HTML elements, such as div, p, img, etc. They can have attributes, child nodes, and text content.
- **Text nodes:** represent the textual content inside an element node. They don't have any child nodes, but they can have a parent node.
- **Comment nodes:** represent HTML comments in the source code. They don't have any child nodes, but they can have a parent node.

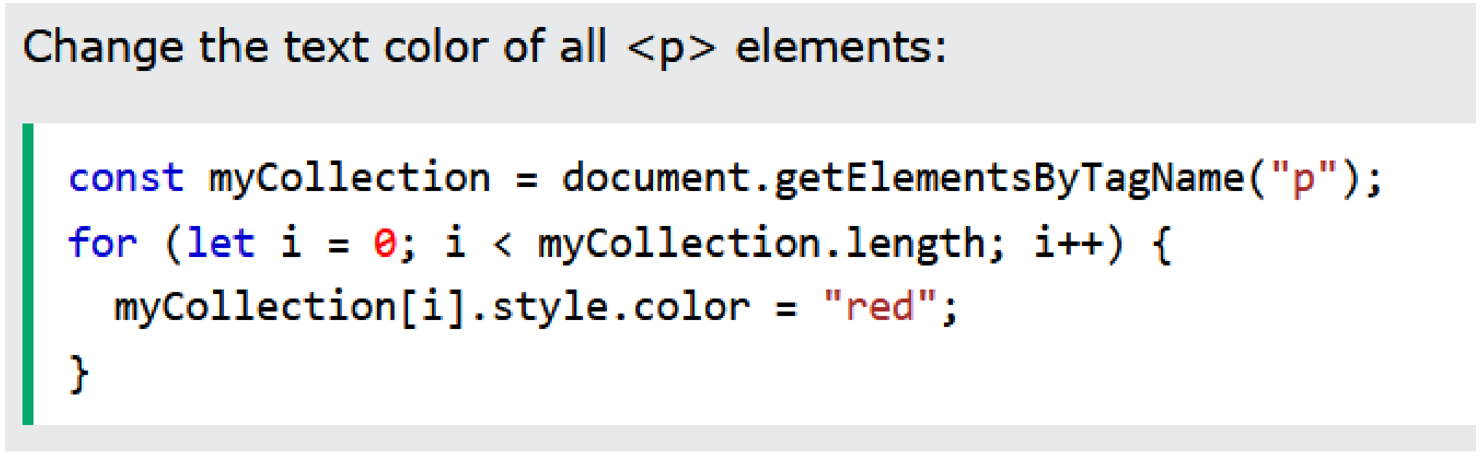
Below is snippet which has reference to DOM Node

```
const element = document.getElementById('my-element');
element.textContent = 'New text content';
```

DOM Collections

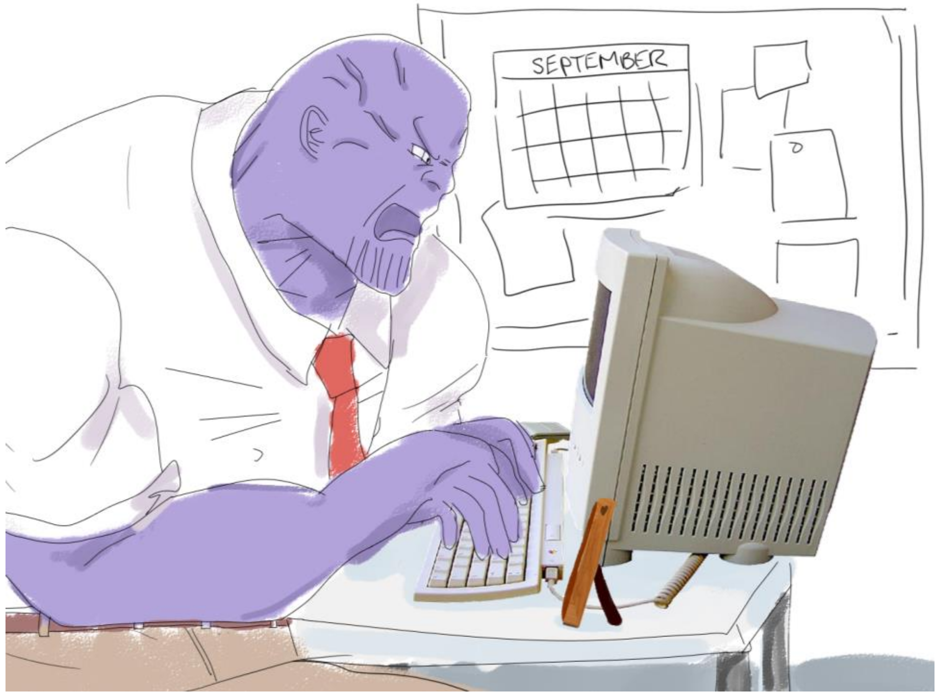
An HTMLCollection object is an array-like list (collection) of HTML elements.

The getElementsBy(XYZAttribute)method returns an HTMLCollection object.



Change the text color of all <p> elements:

```
const myCollection = document.getElementsByTagName("p");
for (let i = 0; i < myCollection.length; i++) {
  myCollection[i].style.color = "red";
}
```



This is good bro, but how browser communicates to web application?

What connects both?

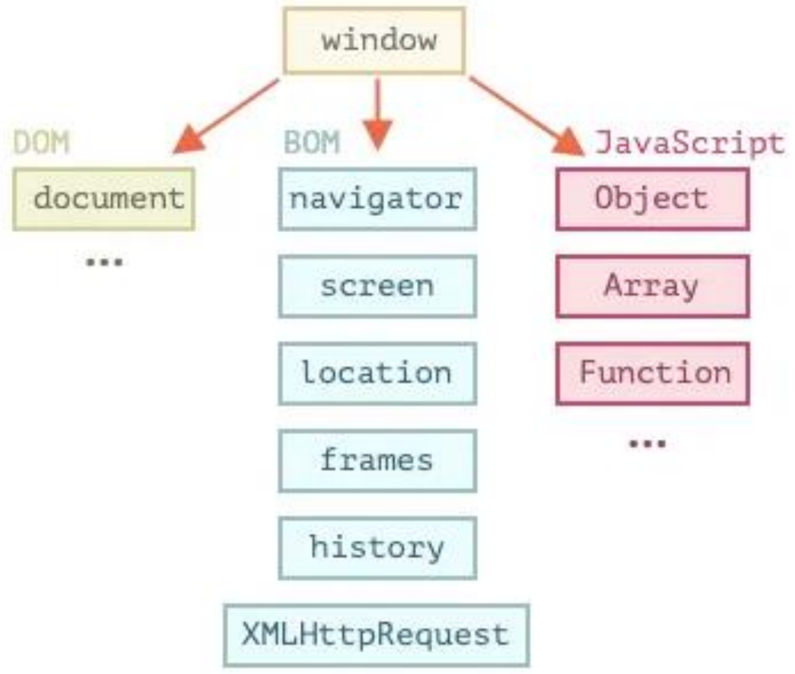
“Window” Object

'Window' Object or API

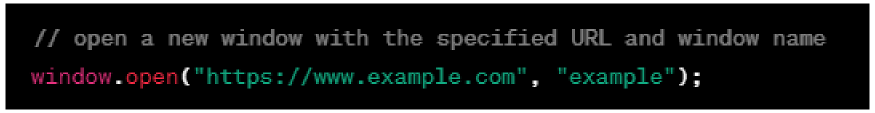
- The ***window*** object is supported by all browsers. It represents the browser's window.
- The **object at the "root" of the web browser** is window.
- All global JavaScript objects, functions, and variables automatically become members of the window object.
- Every **Browser Tab has it's own window** object
- **The Browser Object Model (BOM) allows JavaScript to "talk to" the browser.**

Now you are aware of what DOM is, **BOM(Browser)** is a similar tree-like structure which represents a collection of objects,

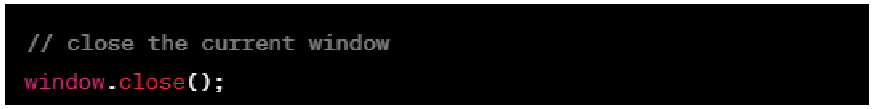
BOM enable us to interact with the user's browser



‘Window’ Object & Document

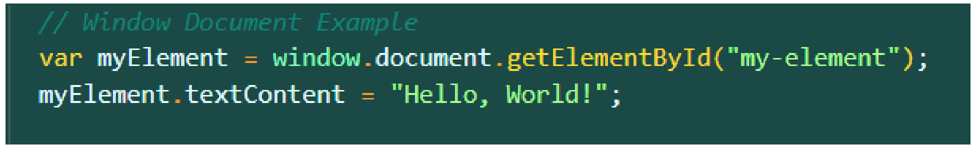


```
// open a new window with the specified URL and window name
window.open("https://www.example.com", "example");
```



```
// close the current window
window.close();
```

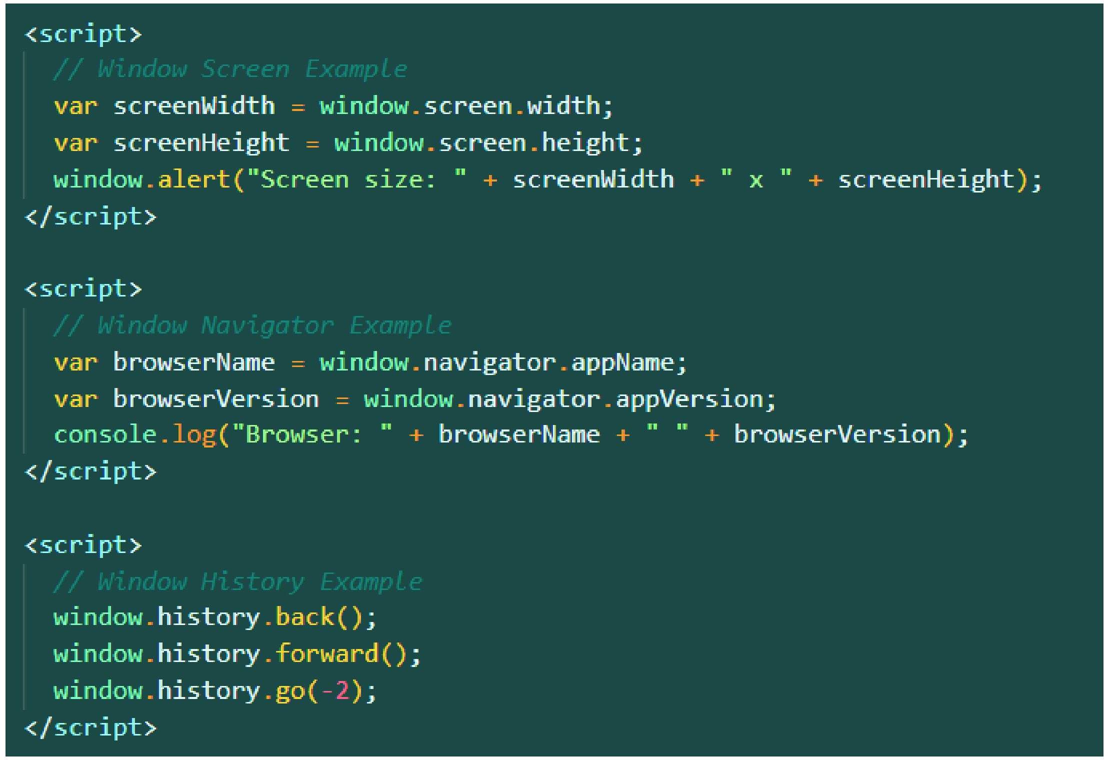
Example of using window document methods



```
// Window Document Example
var myElement = window.document.getElementById("my-element");
myElement.textContent = "Hello, World!";
```

‘Window’ -BOM

- ***The Browser Object Model (BOM) allows JavaScript to "talk to" the browser.***
- ***document***: Represents the web page loaded into the current browser
- ***screen***: information about the user's screen, such as its
- ***navigator***: information about the user's browser, including its name, version, and platform.
- ***location***: information about URL and provides methods for navigating to new URLs.
- ***history***: Provides access to the user's browser history



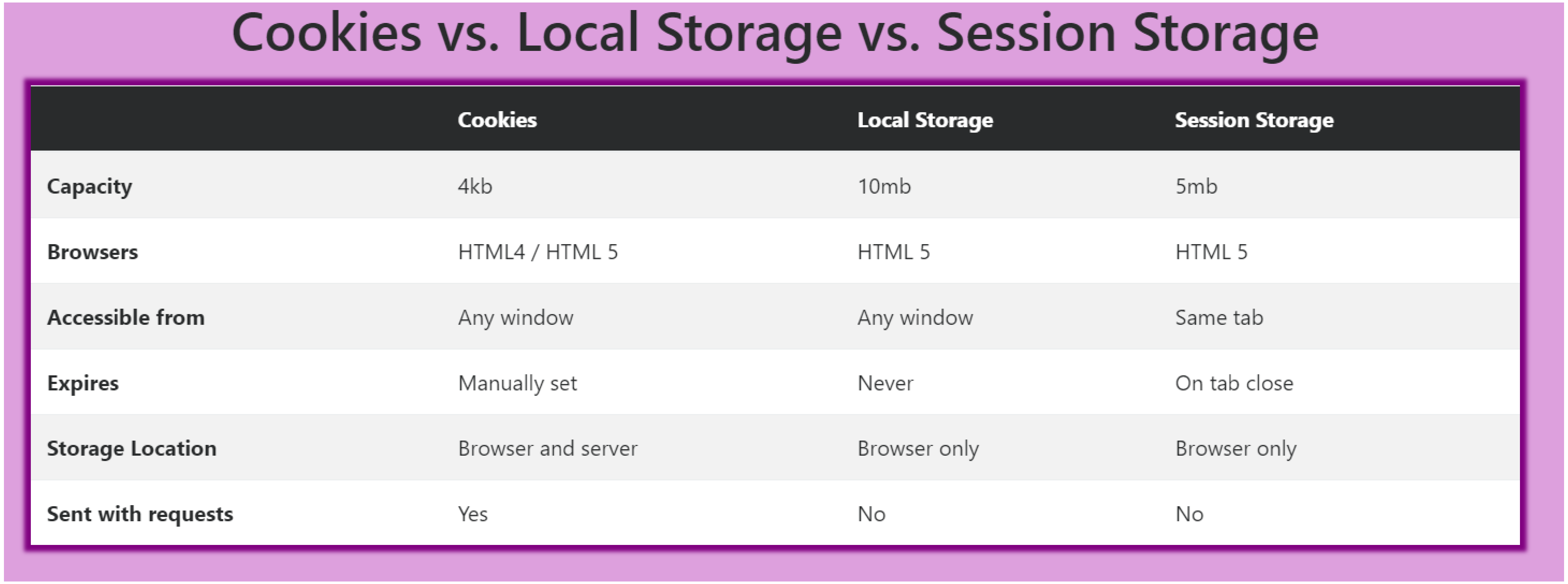
```
<script>
  // Window Screen Example
  var screenWidth = window.screen.width;
  var screenHeight = window.screen.height;
  window.alert("Screen size: " + screenWidth + " x " + screenHeight);
</script>

<script>
  // Window Navigator Example
  var browserName = window.navigator.appName;
  var browserVersion = window.navigator.appVersion;
  console.log("Browser: " + browserName + " " + browserVersion);
</script>

<script>
  // Window History Example
  window.history.back();
  window.history.forward();
  window.history.go(-2);
</script>
```

Some examples

<‖>esto

Browser- Data Storage

Let's see in browser

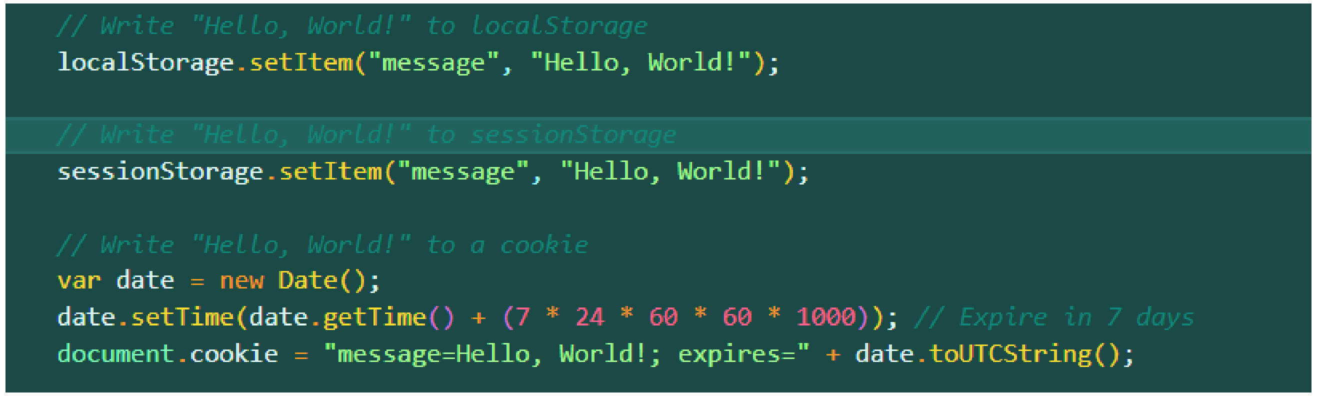


Cookies vs. Local Storage vs. Session Storage

	Cookies	Local Storage	Session Storage
Capacity	4kb	10mb	5mb
Browsers	HTML4 / HTML 5	HTML 5	HTML 5
Accessible from	Any window	Any window	Same tab
Expires	Manually set	Never	On tab close
Storage Location	Browser and server	Browser only	Browser only
Sent with requests	Yes	No	No

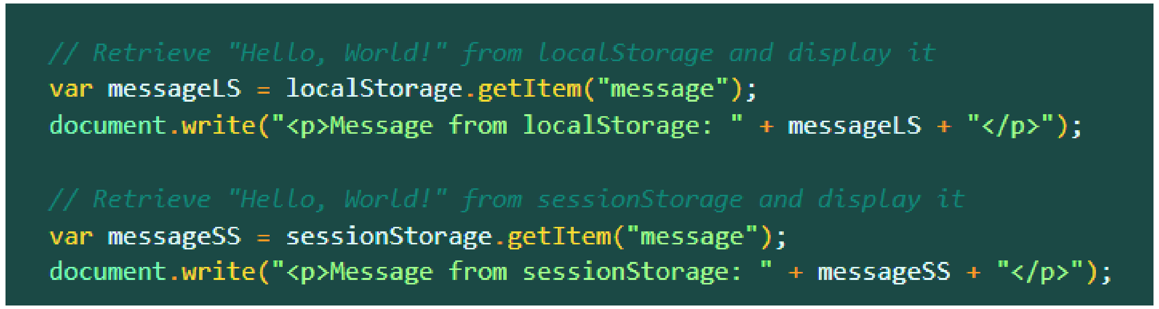
Web Storage

Set Storage & Cookies



```javascript
// Write "Hello, World!" to localStorage
localStorage.setItem("message", "Hello, World!");

// Write "Hello, World!" to sessionStorage
sessionStorage.setItem("message", "Hello, World!");

// Write "Hello, World!" to a cookie
var date = new Date();
date.setTime(date.getTime() + (7 * 24 * 60 * 60 * 1000)); // Expire in 7 days
document.cookie = "message=Hello, World!; expires=" + date.toUTCString();
```

Get Storage



```javascript
// Retrieve "Hello, World!" from localStorage and display it
var messageLS = localStorage.getItem("message");
document.write("<p>Message from localStorage: " + messageLS + "</p>");

// Retrieve "Hello, World!" from sessionStorage and display it
var messageSS = sessionStorage.getItem("message");
document.write("<p>Message from sessionStorage: " + messageSS + "</p>");
```

Knowledge check

When would you use session storage versus local storage?

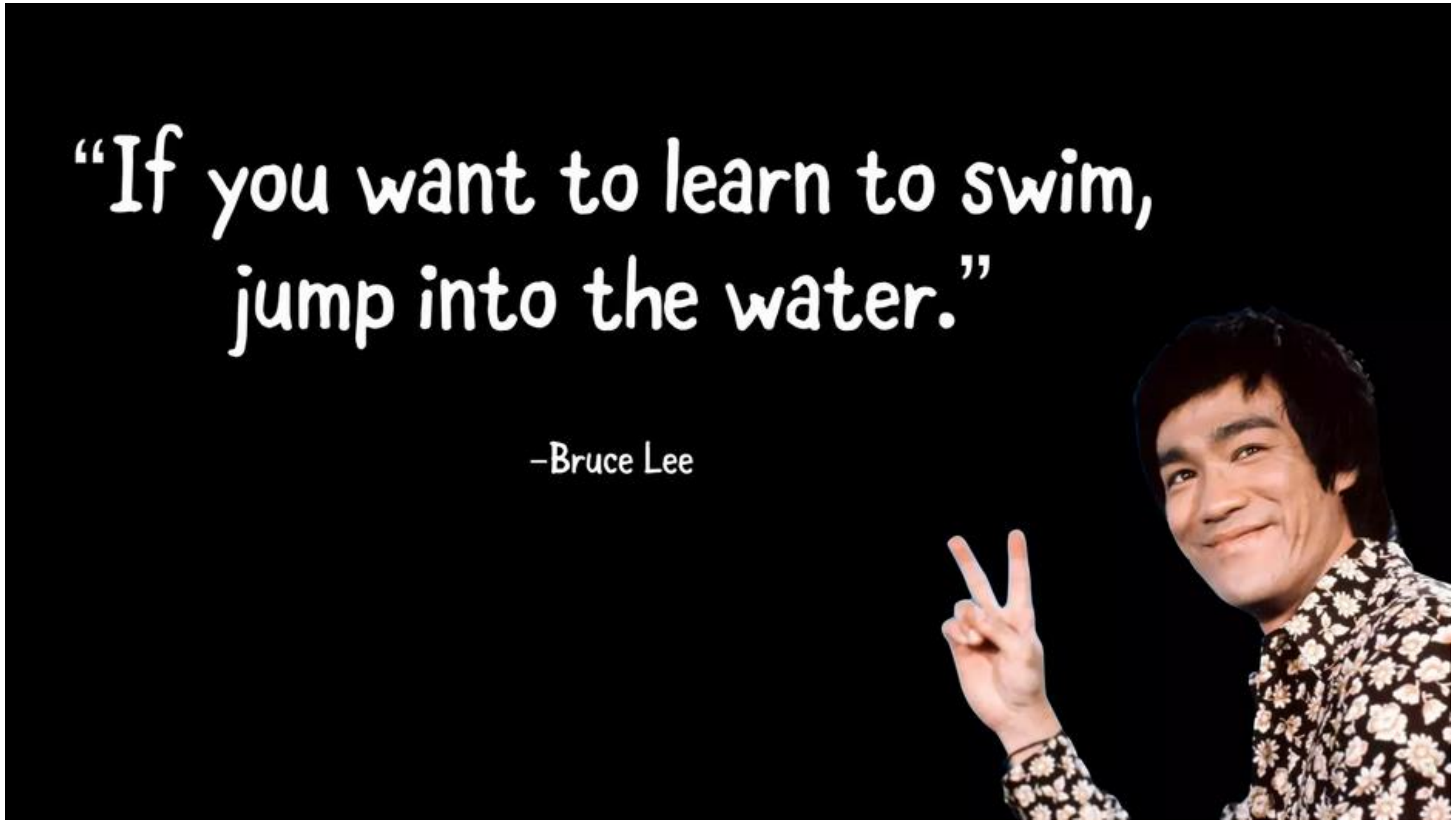
Situation 1: You are building a web application that requires users to log in to access certain features. You want to store the user's login credentials so that they don't have to log in again if they close the browser or refresh the page.

Situation 2: You are building a web application that allows users to save their preferences, such as font size, color theme, and language. You want to store these preferences so that they persist across browser sessions.

Answer:

For Situation 1, you would use session storage because it stores data for the current browsing session only. This means that the data is cleared when the user closes the browser or navigates away from the website. This is a good option for storing temporary data such as login credentials because it provides an additional layer of security by clearing sensitive data after the user is done using the website.

For Situation 2, you would use local storage because it stores data indefinitely until it is manually cleared by the user or the website. This means that the data will persist across browser sessions, allowing users to save their preferences and settings for future use. Local storage is a good option for storing non-sensitive data that the user might want to access repeatedly.


"If you want to learn to swim, jump into the water."
–Bruce Lee

Q & A