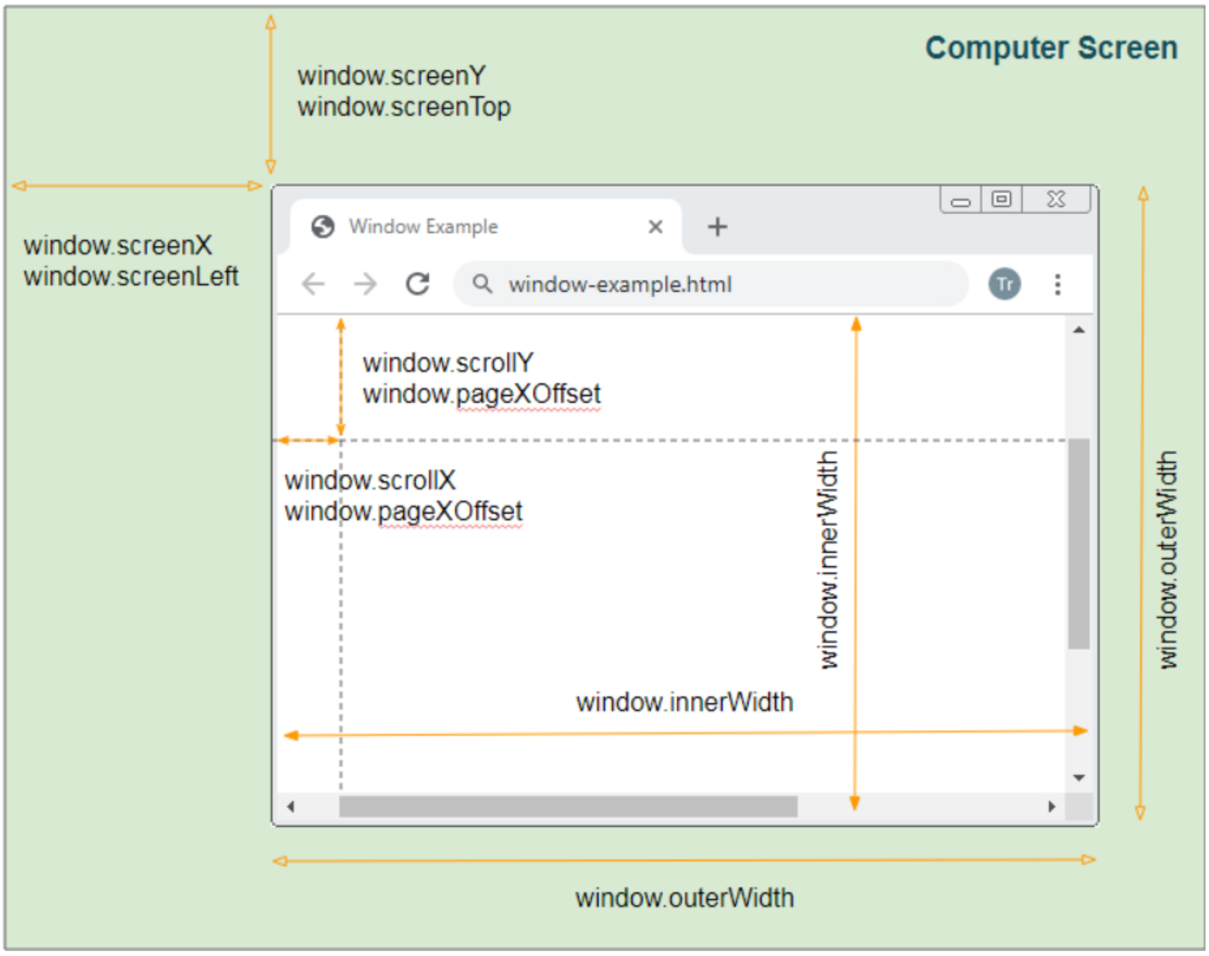
Window Object

The window object in JavaScript stands for the current web page that is being seen in a browser window. It gives access to the browser's methods and attributes as the global object in the browser environment. The window object is the global object in the web browser environment. It represents the current window or frames that the JavaScript code is running in.

The window object comes to the top position in the list of objects of the DOM(Document Object Model) hierarchy. The window object in JavaScript will display the contents of any webpage like all the HTML tags, data, links, images, etc.



A browser window is represented by the window object. The browser produces one window object for the HTML content and one extra window object for each frame when a document contains frames (iframe> elements).

Whenever we surf a browser and a webpage appears on the screen, it displays the contents of the document on the window screen, where the creation of the window object takes place.

Using the document.defaultView property, one may get a window for a certain document. The window object’s characteristics include global variables. The window object’s methods are known as global functions.

This window object is at top level of hierarchy in the The Browser Object Model (BOM)

The Browser Object Model (BOM) is a browser-specific convention referring to all the objects exposed by the web browser. Unlike the Document Object Model, there is no standard for implementation and no strict definition, so browser vendors are free to implement the BOM in any way they wish.

That which we see as a window displaying a document, the browser program sees as a hierarchical collection of objects. When the browser parses a document, it creates a collection of objects that define the document and detail how it should be displayed. The object the browser creates is known as the Document Object Model (DOM). It is part of a larger collection of objects that the browser makes use of. This collection of browser objects is collectively known as the Browser Object Model, or BOM.

Property Name	Description
window.document	The HTML document that is shown in the window is represented by the Document object, which is referred to by the document property.
window.console	The console gives the window’s Console Object back.
window.location	The location attribute makes reference to the Location object, which has data about the page’s current URL.
window.closed	If a window is closed, it returns a true boolean value.
window.frameElement	It gives the window’s current frame back.
window.frame	returns every window item currently active in the window.
window.history	Retrieves the window’s History object.
window.length	It gives the number of iframe> elements currently present in the window.
window.localStorage	provides the ability to store key/value pairs in a web browser. stores data without a time.

window.innerWidth and window.innerHeight	Without including the toolbars and scrollbars, these characteristics describe the width & height of the browser window.
window.opener	It returns a pointer to the window that created the window in the opener function.
window.outerHeight	You can use outerHeight to return the height of the browser window, including toolbars and scrollbars.
window.outerWidth	You can outerWidth to get the width of the browser window, including toolbars and scrollbars.
window.name	Returns or sets a window’s name.
window.parent	Brings up the current window’s parent window.
window.sessionStorage	Provides the ability to store key/value pairs in a web browser. Contains data for a single session.
window.self	It provides the window’s current state.
window.top	It provides the top-most browser window back.
window.screen	The screen attribute makes reference to the Screen object, which stands in for the screen that the browser is shown on.
window.history	The History object, which includes details about the current page’s browsing history, is the subject of the history property.
window.pageXOffset	The number of pixels that the current document has been scrolled horizontally.
window.pageYOffset	The number of pixels that the current document has been scrolled vertically.
window.screenLeft:	The x-coordinate of the current window relative to the screen.
window.screenTop	The y-coordinate of the current window relative to the screen.

window.screenX	The x-coordinate of the current window relative to the screen (deprecated).
window.screenY	The y-coordinate of the current window relative to the screen (deprecated).
window.navigator	An object representing the browser and its capabilities

In JavaScript, the window.document property refers to the current HTML document that is being displayed in a window or frame. It allows you to access and manipulate the content of the HTML document, including the element nodes, attributes, and text content.

Methods: A method in JavaScript is a function connected to an object. You may conduct operations or compute values by calling methods on objects.

Syntax:

window.MethodName()

NOTE: Depending on the attributes, a parameter could have any value, including a string, number, object, and more.

Here is a list of some of the methods of the window object:

Property Name	**Description**
window.open()	This method opens a new browser window or tab.
window.close()	This method closes the current window or tab.
window.alert()	This method displays an alert message to the user.
window.prompt()	This method displays a prompt message to the user and waits for their input.
window.confirm()	This method displays a confirm message to the user and waits for their response.window.focus: brings the current window or tab to the front.
window.blur()	Sends the current window or tab to the back.

window.postMessage()	Sends a message to the window or frame that is targeted by the specified WindowProxy object.
window.scrollTo()	Scrolls the window or frame to a specific position.
window.scrollBy()	Scrolls the window or frame by a specific number of pixels.
window.resizeTo()	Resizes the window to a specific size.
window.resizeBy()	Resizes the window by a specific number of pixels.
window.atob()	A base-64 encoded string is decoded via atob().
window.btoa()	Base-64 encoding is done with btoa().
window.clearInterval()	A timer set with setInterval() is reset.
window.clearTimeout()	The function clearTimeout() resets a timer specified with setTimeout().
window.focus()	It switches the focus to the active window.
window.getComputedStyle()	This function returns the element's current computed CSS styles.
window.getSelection()	It provides a Selection object corresponding to the user-selected text selection range.
window.matchMedia()	The provided CSS media query string is represented by a MediaQueryList object created by the matchMedia() function.

window.moveBy()	Relocates a window with respect to its present location.
window.moveTo()	Relocates a window to the given location.
window.print()	Displays what is currently displayed in the window.
window.requestAnimationFrame()	Before the subsequent repaint, the browser is asked to invoke a function to update an animation using the requestAnimationFrame() method.
window.setInterval()	At predetermined intervals, setInterval() calls a function or evaluates an expression (in milliseconds).
window.setTimeout()	When a certain amount of milliseconds have passed, the setTimeout() method calls a function or evaluates an expression.
window.stop()	It halts the loading of the window.

DOM Object

DOM stands for Document Object Model. It's the interface between JavaScript and the web browser.

With the help of the DOM, you can write JavaScript to create, modify, and delete HTML elements, set styles, classes and attributes, and listen and respond to events.

The DOM tree is generated from an HTML document, which you can then interact with. The DOM is a very complex API which has methods and properties to interact with the DOM tree.

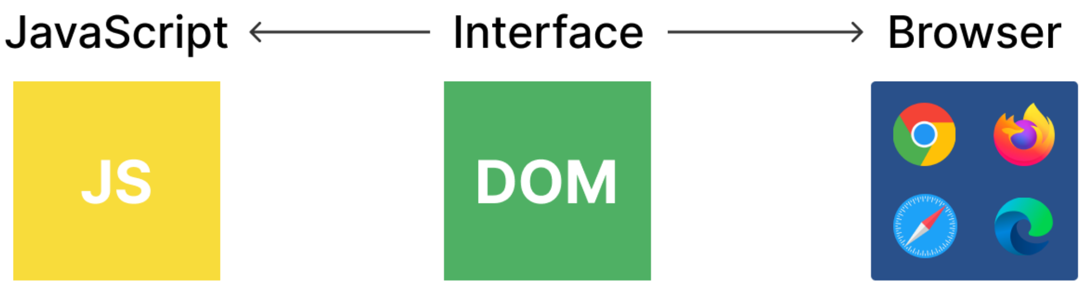
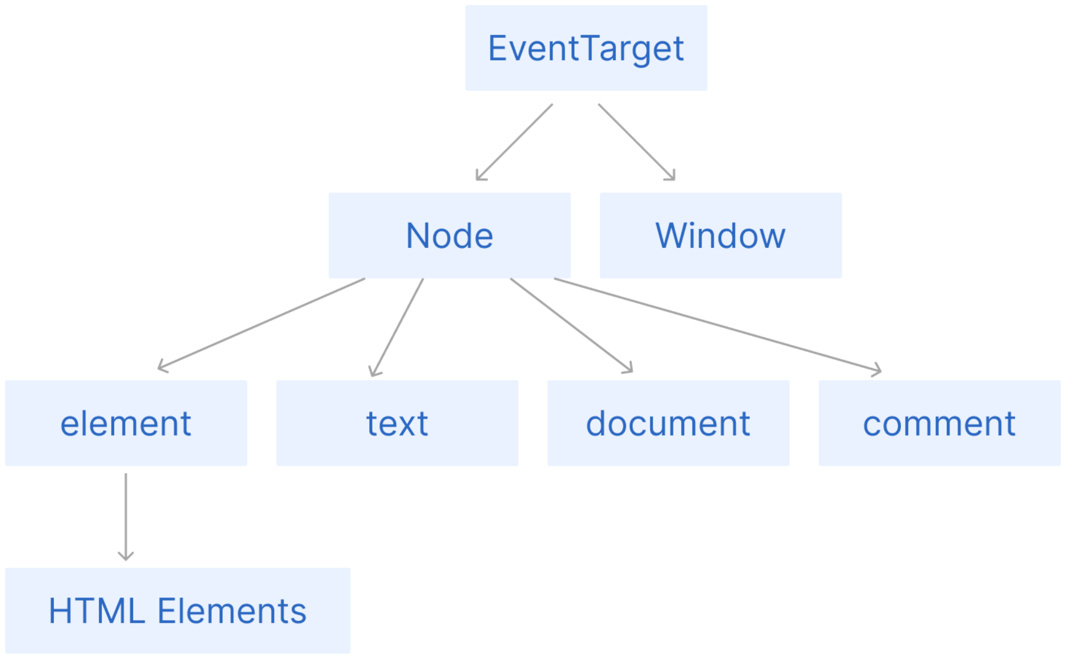


Illustration of the DOM

How the DOM Works – Behind the Scenes

The DOM is organized in a really clever manner. The parent element is called the EventTarget. You can understand better how it works with the help of the below diagram:



The EventTarget interface is implemented by objects which can receive events and may have listeners for them. In other words, any target of events implements the three methods associated with this interface. **Element**, and its children, as well as **Document** and **Window** are the most common event targets, but other objects can be event targets, too.
Window represents the browser's window. All global JavaScript objects, functions, and variables automatically become members of the window object. Global variables are properties of the window object. Global functions are methods of the window object. Even the document object (of the HTML DOM) is a property of the window object.

```
window.document.getElementById("header");

// Both are same

document.getElementById("header");
```

Nodes are in the DOM aka Document Object model. In the DOM, all parts of the document, such as elements, attributes, text, and so on are organized in a hierarchical tree-like structure that consists of parents and children. These individual parts of the document are known as nodes.
The Node in the above diagram is represented by a JavaScript object. We mostly work with the document which has most commonly used methods like document.queryselector(), document.getElementBy Id(), and so on.

How to Select, Create, and Delete Elements Using the DOM
With the help of the DOM, we can select, delete, and create element in JavaScript.

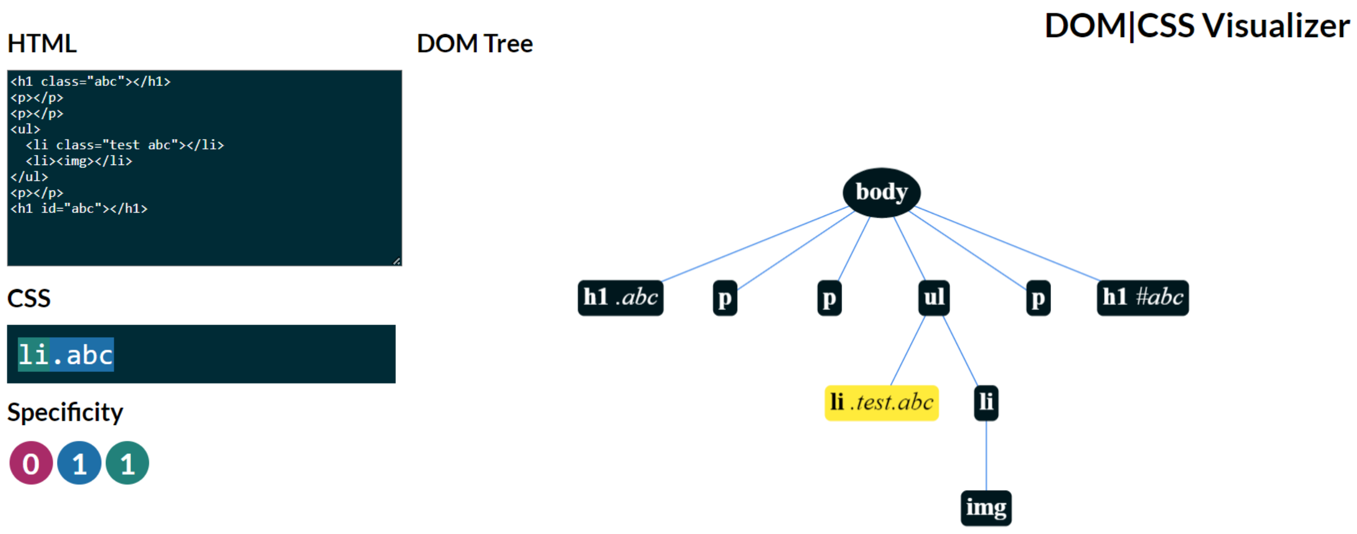
How to Select Elements
There are multiple ways we can select HTML elements in JavaScript. These are the methods we'll look at here:

- document.getElementById();
- document.getElementByClassName();
- document.getElementByTagName();
- document.querySelector();
- document.querySelectorAll();

How to use the `document.getElementById()` method
The getElementById() method returns an element whose id matches a passed string. Since the ids of HTML elements are supposed to be unique, this is a faster way to select an element with ids.

Example:

```
const ele = document.getElementById("IDName");
console.log(ele); // This will log the whole HTML element
```



```
<h1 class="abc"></h1>
<p></p>
<p></p>
<ul>
  <li class="test abc"></li>
  <li><img></li>
</ul>
<p></p>
<h1 id="abc"></h1>
```

```
li.abc
```

How to use the `document.getElementByClassName()` method

The document.getElementByClassName() method returns an HTMLCollection of elements that match the passed class's name. We can search for multiple class names by passing the class names separated by whitespaces. It will return a live HTMLCollection.
So what does it mean that the HTMLCollection is "live"? Well, it means that once we get the HTMLCollection for a class name, if we add an element with the same class name, then the HTMLCollection gets updated automatically.

Example:

```
const ele = document.getElementByClassName("ClassName");

console.log(ele); // Logs Live HTMLCollection
```

How to use the `document.getElementByTagName();` method

The document.getElementByTagName() method returns the HTMLCollection of elements that match the passed tag name. It can be called on any HTML element. It returns an HTMLCollection which is a live collection.

Example:

```
const paragraph = document.getElementByTagName("p");
const heading = document.getElementByTagName("h1");

console.log(paragraph); // p element HTMLCollection
console.log(heading);  // h1 element HTMLCollection
```

How to use the document.querySelector() method

The document.querySelector() method returns the first element that matches the passed selector. Now here, we can pass classname, id, and tagname. Take a look at the below example:

```
const id = document.querySelector("#idname");  // using id
const classname = document.querySelector(".classname");  // using class
const tag = document.querySelector("p");  // using tagname
```

Rules for selection:

- When selecting using class, use (.) at the start. For example (".classname")
- When selecting using id, use (#) at the start. For example ("#id")
- When selecting using a tag, simply select directly. For example ("p")

How to use the document.querySelectorAll() method

The document.querySelectorAll() method is an extension of the querySelector method. This method returns **all** the elements that match the passed selector. It returns the Nodelist collection which is not live.

const ele = document.querySelectorAll("p");

console.log(ele); // return nodelist collection of p tag

NOTE: HTMLCollection is a live collection, while the Nodelist collection is a static collection.

How to Create Elements

You can create HTML elements in JavaScript and add them to HTML dynamically. You can create any HTML element with document.createElement() by passing the tag name in parenthesis.
After you create the element, you can add the classname, attributes and textcontent to that element.

Here's an example:

```
const ele = document.createElement("a");
ele.innerText = "Click Me";
ele.classList.add("text-left");
ele.setAttribute("href", "www.google.com");

// update to existing element in HTML
document.querySelector(".links").prepend(ele);
document.querySelector(".links").append(ele);
document.querySelector(".links").befor(ele);
document.querySelector(".links").after(ele);

// Simalar to below anchor tag
// <a href="www.google.com">Click Me</a>
```

In the above example, we created an anchor tag in JavaScript and added attributes and a classname to that anchor tag. We have four methods in the above example to update the created element in the HTML.
prepend(): inserts the data at the top of its first child element.
append(): inserts the data or content inside an element at the last index.
before(): inserts the data before the selected element.
after(): puts the element after the specified element. Or you can say that it inserts data outside an element (making the content its sibling) in the set of matched elements.
How to Delete Elements
We know how to create elements in JavaScript and push them to the HTML. But what if we want to delete existing elements in the HTML? It's easy – we just need to use the remove() method on that element.
Here's an example:

```
const ele = document.querySelector("p");


// This will remove ele when clicked on
ele.addEventListner('click', function(){
        ele.remove();
})
```

How to Use Event Handlers

The change in the state of an object is known as an **Event.** The process of reacting to the events is called **Event Handling.**
Events happen when a user does something like click, hover over an element, press a key, and so on. So when an event happens and you want to do a certain thing or manipulate anything, you use event handlers to trigger that event.

We use event handlers to execute certain code when that particular event happens. There are multiple event handlers in JavaScript (here's a quick list of them), but you use the same process to add event handlers to any element.
Here's the syntax:

```
const ele = document.querySelector("a");


ele.addEventListner("event", function(){
        // callback function
});
```

Some events you can use:

- click
- mouseover
- mouseout
- keypress
- keydown

And here's an example of using the "click" event:

```
const ele = document.querySelector("a");

ele.addEventListner("click", function(){
        ele.style.backgroundColor = "pink";
});
```

Main Differences

DOM	Window/BoM
It has proper structure defined for use	BoM doesn't have any common structural, browsers, however, use inter-compatible structures
DOM is part of Window Object	Window object is constituted by DoM
There are no boundaries defined for DOM	Window has got internal and external boundaries
DOM has got elements that interact with Window	Manual changes in Window does not affect background code of DOM

References:

1) https://en.wikipedia.org/wiki/Browser_Object_Model
2) https://www.geeksforgeeks.org/window-object-in-javascript/
3) https://o7planning.org/12397/javascript-window
4) https://www.scaler.com/topics/window-object-in-javascript/
5) https://www.freecodecamp.org/news/what-is-dom-in-javascript/