|  |
| --- |
| **BOM (Browser Object Model)** – expose objects like window, document, screen, history, location, navigator etc.  Window is the global object. |
| **DOM** - There are three spheres of concern that exist in the browser: **content, presentation, and behaviour** thus we have HTML, CSS, and JavaScript. The bridge between JavaScript programs and the content that is rendered in the browser is the **Document Object Model (DOM) API**. It is a **tree structure of nodes**. For most use cases, DOM manipulation will happen under document.body since it is the only part of the document that has rendered visual components. DOM defines HTML elements as objects, which have: **properties, methods, events**. The elements are nested in each other and create hierarchy. **DOM methods** are the actions we perform on HTML elements(add, delete, hide). **DOM Properties** are the values of the elements which we can set or change. Divs and paragraphs have .innerText property and the input fields have value property. They cannot hold these properties together at the same time. |
| **What is DOM API?** API is the acronym for **Application Programming Interface**, which is a software intermediary that **allows two applications to talk to each other**. Each time you use an app like Facebook, send an instant message, or check the weather on your phone, you’re using an API. The DOM API consists of the interface built to programmatically communicate with the DOM on the browser. The DOM is how the contents of a page are modelled in the browser, and the DOM API is how the browser provides programmatic access to this content.  => Imagine you’re sitting at a table in a restaurant with a menu of choices to order from. The kitchen is the part of the “system” that will prepare your order. What is missing is the critical link to communicate your order to the kitchen and deliver your food back to your table. That’s where the waiter or API comes in. The waiter is the messenger – or API – that takes your request or order and tells the kitchen – the system – what to do. Then the waiter delivers the response back to you. HTML elements are JS objects and can be handled as such:  1. Select an object to obtain its reference;  2. Modify its properties.  HTML attribute != DOM property but there is almost no difference between them since they have the same outcome.  **DOM API can:**   * change content and structure of HTML elements, * modify style and properties, * read user input and react to events – fired when there is some interaction with the user for example * create and remove elements |
| A **NodeList** is a collection of document nodes. A **NodeList** and an **HTML collection** is very much the same thing. Both an **HTMLCollection** object and a **NodeList** object is an array-like list (collection) of objects. Both have a length property defining the number of items in the list (collection).  Judge specifics oblige us to use **Array.from()** to first convert the collection into an array.  NodeList can contain any node type, including whitespaces and text. It has iterration methods. It is either live or static collection. Can be indexed. Can be converted to an array.  HTMLCollection contains only Element nodes. It has iterratin methods, but it also has **namedItem** method. It is a live collection. Can be indexed. Can be converted to an array. |
| **Changing HTML Content**  **- innerHTML – to access raw HTML. Changing innerHTML or textContent will remove all child nodes.**  **- textContent** - property of the Node interface represents the text content of the node and its descendants.  **Note:** textContent and [HTMLElement.innerText](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/innerText) are easily confused, but the two properties are [different in important ways](https://developer.mozilla.org/en-US/docs/Web/API/Node/textContent#differences_from_innertext). Element.innerHTML returns HTML, as its name indicates. Sometimes people use innerHTML to retrieve or write text inside an element, but textContent has better performance because its value is not parsed as HTML. Moreover, using textContent can prevent [XSS attacks](https://developer.mozilla.org/en-US/docs/Glossary/Cross-site_scripting).  **- value** – the values of input elements are string properties on them  **- style** – content can be hidden by manipulating the display -> **element.style.display = “none” / block, inline, or empty string ‘’’’;** |
| **Targeting HTML elements – these methods return a reference to the element which can be manipulated by JS**  By ID – **getElementById()** –ID attribute must be unique on the page  By class name – **getElementsByClassName()** – specifies the type of the element (div, p, a…).  Returns **live** HTMLCollection – changes status after we change something on the page!  By tag name – **getElementsByTagName()** – class names used for styling and easier selection.  Returns **live** HTMLCollection – changes status after we change something on the page!  By CSS selector:  **- querySelector()** – selects the first matching element  **- querySelectorAll()** – selects all matching elements and returns static NodeList (not live as HTMLCollection)  **- #id, .className, input[name=’’login”]** |
| **Parents & Children elements**  Every DOM elements has a parent which can be accessed by property .**parentElement** or .**parentNode**. Children also can be accessed by property **.children** |