# 8. JavaScript DOM manipulation

## 8.1 Relationship between JavaScript and HTML: why do we need JS?

- We covered the relationship between HTML, CSS and JavaScript when we discussed the class on Introduction to Programming Basics. We said HTML provides the basic structure of websites, which is enhanced and modified by other technologies like CSS and JavaScript. CSS is used to control presentation, formatting, and layout. JavaScript is a logic-based programming language that can be used to modify website content and make it behave in different ways in response to a user's actions.
- In short, the relationship between HTML, CSS and JavaScript is explained below:
    - HTML defines the structure of a web page
    - CSS provides the style/look of the HTML page
    - JavaScript adds interactivity to the HTML page

## 8.2 How do we add JS into our HTML?

- We have also covered the different ways we can include JavaScript into our HTML under 1.7 section of phase 2. However, it is important to revise this section before diving into DOM.
- Just like HTML and CSS, JavaScript is also a text file. HTML file uses. There are different ways of including JavaScript in our HTML.
- **Including JavaScript code inline**:
    - Insert your JavaScript code directly inside the HTML tag using the special tag attributes such as onclick, onmouseover, onkeypress, onload, etc. Example:

```
<body>
<button onclick="alert('Hello World!')">Click
Me</button>
</body>
```

    - **Note**: You should avoid placing large amount of JavaScript code inline as it clutters up your HTML with JavaScript and makes your JavaScript code difficult to maintain

- **Embedding the JavaScript Code in your HTML**:
  - Embed the JavaScript code directly within your web pages by placing it between the <script> and </script> tags.
  - The <script> tag indicates the browser that the contained statements are to be interpreted as executable script and not HTML. Here's an example:

```
<body>
<script>
function someFunction() {
alert ('Test');
}
someFunction();
</script>
</body>
```

- **Including an external JavaScript file in HTML (Recommended):**
  - Use the script tag with the attribute src. The <script> tag is an HTML tag that is specifically used to include/inject JavaScript to an HTML page. You've already used the src attribute when using images. The value for the src attribute should be the path to your JavaScript file. Include the script tag in between the <head> tags in your HTML document. Example:

```
<script src="script.js"></script>
```

  - Once you include your external JavaScript file into your HTML, you can use the "console.log()" method to display what you want on the console. Below is an example of code you will write if you want the code to log “Hello World” on the console: console.log("Hello World")
  - **Recommended place to add the <script> tag**:
    - **Adding <script> tag at the bottom**: If we include our <script> tag within our html <head> tag, the HTML parser on our browser loads and executes the script as soon as it encounters it. In modern websites, scripts are often “heavier” than HTML, so, their download size is larger, and processing time is also longer. It is recommended to add our <script> tag at the bottom

in our <body> tag because we want the HTML and CSS to finish loading before our JavaScript. Putting the <script> tag at the bottom allows the browser to see the HTML elements above the script, and it doesn't block the HTML page content from showing. For example, if our JavaScript is having issues or we have a slow internet connection, the browser will still load the rest of the page and doesn't "hang".

- **"defer" attribute:** If we include our <script> tag within our html <head> tag, we have to add the "defer" attribute to tell the HTML parser to load the script after it finishes loading all of our HTML elements. The defer attribute tells the browser not to wait for the script. Instead, the browser will continue to process the HTML and build the page. Note that, "defer" is an attribute that is added on HTML 4 just for this purpose. Example:

```
<head>
<script src="script.js" defer>
</script>
</head>
```

- **"async" attribute**: With async attribute, the JavaScript file gets downloaded asynchronously. Meaning, after the script file is downloaded, the browser will pause the HTML parser, execute the script and then resume parsing. So, we can say with async, the script file is executed as soon as it is downloaded. With defer, the JavaScript file gets downloaded asynchronously but it is executed only after the document parsing is completed.
- **Note**: Both the defer and async attributes don't have any effect on inline scripts.

### 8.3 Understanding DOM: how do HTML and JS work together?

- HTML is a markup language. JavaScript is a programing language that usually deals with objects. How do they work together then?
- From what we have learned so far, we can manipulate data types like array using JavaScript. We have also seen how we can access and manipulate JavaScript objects.
    - **Example of manipulating array**:

```
let someArray = [1, 6, 8];
someArray[1] = 44; // reassigning the 2nd
element of the array
console.log(someArray); // prints [1, 44, 8]
```

- **Example of manipulating object**:

```
let somePerson = {
    name: "Abebe",
    education: {
    school: "Evangadi",
    field: "Full stack",
    grade: "3.9"
    }
    }
    somePerson['education']['grade'] = "4.0"; //
reassigning the grade value
    console.log(somePerson); // prints object
with updated grade of 4
```

- **What if we have a way to convert the HTML elements into a structured object like the "somePerson" and try to manipulate the elements?**
  - If we can convert the HTML elements into an object form, we can apply the JavaScript object manipulation techniques on them, right?
  - To do this, DOM is the answer for that
- **What is the DOM Object?**
  - **DOM (Document Object Model) explained in plain English**:
    - Assume you have your TV on, but you want to change the show that's being streamed, and you also want to increase the TV's volume. Changing the show and increasing the TV's volume needs a way by which you interact with your television, a remote. The remote is basically the bridge that allows you to interact with your television. You make the TV interact with you via the remote. You make HTML interact with you via JavaScript

by using DOM. Just like your TV can't do much without the remote, JavaScript doesn't do much (other than doing some calculations or work with basic strings) to make HTML interactive without using DOM to select and update the HTML elements.

- We know that JavaScript needs to access the HTML document and it also needs to know when the user is interacting with the HTML. It is through DOM structure that JavaScript can access and update the HTML document whenever a user interacts with the HTML.
- For example, if you want a button on your website to change its color to green when a user clicks on this button, you need to be able to select this specific button and write a logic that changes the color of the selected element into green using JavaScript. DOM organizes our HTML document in an object form so that JavaScript can manipulate (access and modify/update the any HTML element). Therefore, DOM is the link between an HTML web page and JavaScript.

- **DOM explained in terms of programming:**
  - The DOM is a structure/standard/syntax that allow JavaScript to access, modify, and update the structure of an HTML page.
  - The DOM is a logical tree-like model/representation of our HTML. DOM represents the HTML page using a series of objects. The main object is the document object, which in turn houses other objects which also house their own objects, and so on. Click on the below link to look at the diagram explaining DOM:

    https://content.codecademy.com/courses/dom/dom_revision_1.svg

- **Understanding the DOM**
  - **Why do we need to understand DOM?** As a Frontend developer, your job will be to select and update the DOM elements when user interacts with a website. Selecting and updating HTML elements using JavaScript is possible if one has a good understanding of what DOM is. This is where you apply your basic programing skills
    - **Selecting DOM elements**: It simply means locating the element you want to work with

        - **Updating DOM elements**: It means interacting with the element, the text of the element, the attributes of the element or with its child elements

- **Relationship between the browser and DOM: How does a website work?**
    - **Webpage**:
        - A webpage is a properly tagged HTML text document. The main purpose of a web page is to display the text information that is included in the document in a well-organized and visually meaningful way to the user.
        - As a web page is simply a text document, we would need help to create some useful visual representation of the text. A software that helps us create a useful visual representation is called a Browser. This means a web page without the help of the browser is as helpless as a text document.
    - **The browser/web browser**:
        - In common usage, a web browser is usually shortened to "browser."
        - Web browser is a software program that loads files from a remote server (or perhaps a local disk) and displays them to a user. The main role of a web browser is to create a visual representation of the HTML document based on the tags used on the text document.
        - The process of creating the visual representation of the HTML file is called rendering. For the browser to be able to render your HTML file, it needs to get the HTML tagged text first. When the HTML file is saved locally, all the browser needs is a way to locate this resource file containing the HTML. The way we achieve this is by giving the URL (Uniform Resource Locator) to the browser. Since we are on a local computer this URL is just the path to the file. Path is structured by the Operating System on the computer.
        - Within the browser software, there is a piece of software that locates the URL of the resource file containing the HTML and figures out what to display to you based on the HTML files it receives. This is called the browser engine.
        - When you write some HTML, CSS, and JS, and attempt to open the HTML file in your browser, the browser reads the HTML from your hard disk. To read your HTML file, the browser transforms your HTML into object via DOM.

- Common web browsers include Google Chrome, Firefox, Apple Safari, Mozilla, and Microsoft Edge.

- **How does a browser render/build an HTML web page?**
  - When a web page is loaded, the browser first reads the HTML text, then browser creates nodes from the HTML document and it will create a tree-like structure of these node objects, the DOM tree. A DOM tree starts from the topmost element which is the <html> tag and branches out to the HTML elements nested under the <html> tag.
    - Whenever the browser encounters a script tag, the DOM construction is paused. The entire DOM construction process is halted until the script finishes executing. This is because JavaScript can alter both the DOM tree and the CSS tree. That is why the location of your script tag in your HTML file matters. If you add the "async" or "differ" keyword to the script tag, the DOM construction will not be halted.
    - **Note**: We discussed previously that browser understands only HTML and CSS. For the browser to understand the code on our JavaScript file, every browser has a JavaScript engine that takes our JavaScript code and converts it into something that the browser can understand
  - After constructing the DOM tree, the browser reads CSS from all the sources (external, embedded, inline, user-agent, etc.) and construct a CSS object model which is a tree like structure just like DOM. Each node in this tree contains CSS style information that will be applied to DOM elements that it targets (specified by the selector)
  - After constructing these two trees, the browser then combines these trees together. The combined tree-like structure is called Render Tree. Once the Render-Tree is constructed, then the browser starts the printing individual elements on the screen. Note, if the browser constructed the HTML and CSS, there is no page to be printed on the screen unless Render-Tree isn't constructed.
- **Here is a diagram showing the interactions of HTML, CSS and JavaScript files with the browser**

- https://www.oreilly.com/api/v2/epubs/9781449355517/files/httpatomoreillycomsourceoreillyimages1416220.png.jpg

- **Note**:
  - DOM is not part of HTML
  - DOM is not part of JavaScript. However, JavaScript interacts with the created DOM object
  - DOM is a language-agnostic structure. Meaning, DOM can interact with various languages other than JavaScript

## 8.4 The DOM tree

- **The DOM tree**: When the browser reads HTML code and encounters an HTML element like <html>, <body>, , or encounters HTML attribute like "src" and "ref" or texts in HTML, it converts each one of them into objects. The objects the browser creates from our HTML document are called a node. After the browser has created node objects from the HTML document, it will treat the node objects as a tree-like hierarchy.
  - **DOM node**: In DOM terminology, all elements of an HTML document are nodes, including the built-in DOM elements such as document document.body, HTML tags such as <p> (called element node), a text in HTML tags (text node) and an attribute (attribute node) of an HTML tag. DOM nodes are any one of the above in an object form in the DOM hierarchy. Thus, node objects have methods in them that we can use to access and alter them.
  - **Dom element**: DOM element on the other side is one type of a DOM node representing only HTML elements. Meaning, an element node is a node that's written using a tag in the HTML document. <html>, <head>, <title>, <body>, <h2>, <p> are all elements because they are represented by tags. <!DOCTYPE >, the comment and the text nodes are not DOM elements
- In simple terms, we can say DOM tree is constructed from nodes. There are 12 node types available to build the DOM tree. However, we will focus below on the five most used node types present in almost all HTML documents.
- **The common node types under the DOM tree**: Please note than under the DOM tree, all the below nodes are represented as objects.

  **1. The "document node"**: Document node is the root of DOM tree. It represents the entire HTML document that is loaded in the browser window. It is the entry point to

the document. The document node has multiple methods that we can access all nodes in the DOM tree. Also, the top-level nodes in an HTML document (<html>, <head>, <body>) can be directly accessed using properties of document node as follows:

- document.documentElement to access the <html> tag
- document.head to access the <head> tag
- document.body to access the <body> tag

**2: The "element node"**: Each HTML tag becomes element node under the DOM tree. Examples of element node objects include:

- <html>, <head>, <body>, <h1>, , 

**3: The "attribute node"**: Each attribute of an HTML element becomes element node under the DOM tree. Once again, attribute nodes are also objects. Means they have their own method that we can use to read or change the attribute values. Examples of attribute node objects: <html lang="en-US">, <a href="">, <img src="" width="" alt="">,  , etc.

**4. The "text node"**: Text nodes represents the textual content in an element or an attribute. Examples of text node objects include:

-  Some text 

**5. The "comment node"**: HTML comments (<!-- comment -->) are represented by comment nodes in a DOM tree. That means, comment node can also be accessed using JavaScript. You may be asking yourself "why is a comment added to the DOM, if it doesn't affect the visual representation of an HTML?". However, there is a rule under DOM stating that anything in the HTML must be represented in the DOM tree. Meaning, everything in HTML, even comments, become a part of the DOM.

- Refer to the following links/diagrams to see how the DOM tree represents the HTML elements, attributes and texts as objects.
  - https://content.codecademy.com/courses/dom/dom_revision_1.svg
  - https://miro.medium.com/max/1040/1*YSA8lCfCVPn3d6GWAVokrA.png
  - https://upload.wikimedia.org/wikipedia/commons/thumb/5/5a/DOM-model.svg/1200px-DOM-model.svg.png

## 8.5 DOM manipulation: introduction

- **DOM manipulation with JavaScript**:
  - **What do we mean by manipulating?**
    - Earlier we said that DOM represents the HTML page using a series of objects, the document object being the main object. The document object in turn houses other objects which also house their own objects, and so on.
    - So, when we say, “DOM manipulation”, it simply means changing the DOM (the HTML document). This in turn means, changing the HTML elements, that DOM converted into objects, in response to a user’s action
  - **Manipulating DOM involves two steps:**
    - 1: Finding/selecting the element we want to work with
    - 2: Altering the text or attributes of that element
  - **The “document object” makes manipulation of the HTML document possible**: The "document" object in JavaScript is the door to the DOM structure. This is the name of the JavaScript object that contains all the methods and properties to help us access and manipulate the DOM elements. The "document object” allows us to access the root node of the DOM tree. You can access the other child nodes of the DOM as properties of the “document object”.
    - Example, to access the body node: console.log(document.body);

## 8.6 DOM manipulation: selecting elements (part 1)

- JavaScript is most used language to modify HTML elements or their content and apply effects like show/hide, animations, etc. But before we perform any action on an element, we need to select the target HTML element.
- **What does selecting DOM elements mean?** In terms of DOM manipulation, the term “selecting” an element(s) means finding an element that we plan to manipulate or apply a change on.
- **Selecting in JavaScript can also be divided into three types**

- **1. Select an individual element**:
    - Example: Finding one specific  element
- **2. Select multiple elements**: Example:
    - Finding all  elements
- **3. Traversing between multiple elements**:
    - Example: Finding a <p> element within a particular 

## 8.7 DOM manipulation: selecting an individual element (part 2)

- **Selecting an individual element**: There are three common methods provided by the document object that we can use to select one specific element. Let's look at them below.
    - **1. getElementById() method**: The getElementById method accepts CSS' id selector as its argument and returns an element whose id matches the passed string/id. Since the ids of elements are always unique to that element, this is a faster way to select an element. If the id is not found, then this method returns null.
    - **2. querySelector() method**: The querySelector method takes CSS selectors as its argument and returns the first element that matches the passed selector. Meaning, the method can take ids, classes and tag names as its argument.
        - document.querySelector("#hi"); // selects the element with "hi" id name
        - document.querySelector(".bye"); // selects all elements with "bye" class name, but returns first element in the document with class name
        - document.querySelector("div"); // selects all elements with "div" tag, but returns first div
        - document.querySelector("div span.hello"); // selects all "span" elements within a "div" tag but returns the first span element with class name "hello", inside a div
    - **3. Traverse the DOM**: Traversing is the act of selecting an element from another/neighboring element. This is discussed in detail under section 8.9 of this class.

## 8.8 Selecting multiple elements (HTML collection vs NodeList )

- **Selecting multiple elements**: There are three common ways of selecting multiple elements with just one query. Please note that, you can pass in any valid CSS selector for these methods, including, comma-separated CSS selectors for targeting multiple different selectors.

- **querySelectorAll() method**: The querySelectorAll() method returns all elements in the document that matches a specified CSS selector(s) as NodeList.
    - **Syntax**: document.querySelectorAll(CSS selectors)

```
<ul id="listOfFruits">
 <li class="red">Apple</li>
            <li class="yellow">Mango</li>
            <li class="yellow">Peach</li>
    <li class="red">Rasberries</li>
 </u>
```

```
Ex: document.querySelectorAll("li")) //returns
a nodeList of all <li>
```

- **Selecting one element from a nodeList**
    - **Selecting using the item() method** : See example code above

```
var el = document.querySelectorAll(".yellow");
el.item(0);// selects the first li with yellow class
el.item(1);// selects the 2nd li with yellow class
```

- **Selecting using array syntax**: See example code above

```
var el =
document.getElementsByClassName("red");
console.log(el[1]); //selects the 2nd li with
red class
```

- **getElementsByTagName() method**: The method takes a tag name and returns a collection of all elements in the document with the specified tag name, as an HTMLCollection object.

    **Syntax**:

```
document.getElementsByTagName(tagname)
getElementsByTagName("li"); //returns
HTMLCollection with all <li>
```

- **Selecting one element from an HTMLCollection**: See example code above

```
var el = document.getElementsByTagName("li");
console.log(el.item(0));// returns the 1st li
element
console.log(el[1]); // returns the 2nd li
element
```

- **getElementsByClassName()**: Just like getElementsByTagName()**,** this method also returns a live HTMLCollection representing an array-like object of all elements with the specified class name.

```
Syntax:
document.getElementsByClassName(classname)
getElementsByClassName(“red”)// returns HTML
collection with all elements having the “red” class
```

- **Selecting one element from an HTMLCollection**: See example code above

```
var el = document.getElementsByClassName("red");
console.log(el[1]); //selects the 2nd li element
with red class
console.log(el.item(0));// selects the 1st li
element with red class
```

- **HTMLCollection vs NodeList**
  - One thing to understand is that most JavaScript selector methods return multiple elements with a single query. Because of this, by default many selectors will return array-like lists that store the matching elements.
    - **Note**: When we use the methods that return multiple elements by default, even if there is only one matching element in the document, a list will still be returned.
    - **Note 2**: If you want to do something with an element returned in a list, you must specifically identify it from the list first in order to properly target it. There are several ways you can do this.
  - **NodeList**: To refresh your memories, everything in the DOM is a node (contains element nodes, text nodes, atrribute nodes, etc.) NodeList returns a static NodeList. Meaning, even if the DOM changes after you use the method, the NodeList does not change and does not reflect the current UI.
    - querySelectorAll() method returns a static NodeList.
  - **HTMLCollection:** Element/HTML node in DOM contains only elements. To refresh your memory, an element node in DOM is one specific type of node, called element node (it can be <li>, , <body>, etc.) HTMLCollection returns a live HTMLCollection where, each nodes can be accessed by index numbers 0 being the first index. Meaning, if the DOM changes after you use the method, the HTMLCollection is updated to reflect the current UI.
    - getElementsByClassName() and getElementsByTagName() methods return a live HTMLCollection.
  - **NodeList(static collection) vs HTMLCollection (live collection) in example**: We said HTMLCollection returns a live HTMLCollection. Meaning, if the DOM changes after you use the method, the HTMLCollection is updated to reflect the change. We also said NodeList returns a static NodeList. Meaning, even if the DOM changes after you use the method, the NodeList does not reflect the current change.
    - **Assume this is your html**

```
<div id="my-div">
<p class="firstPar">I am the first paragraph</p>
<p class="secondPar">I am the second paragraph</p>
    </div>
```

- **Assume you have your script below**: If you add a third paragraph into the above HTML document dynamically (using JavaScript) and check the length of the list
    - Using the querySelectorAll() method returns a Nodelist and the length of the collection containing paragraphs will not update from 2 to 3 even if a new paragraph is added dynamically. This is because querySelectorAll() returns a static/non-live collection of items
    - Using the getElementsByTagName() method returns an HTMLCollection and the length of the collection containing the paragraphs will update from 2 to 3 after a new paragraph is added dynamically. This is because methods like getElementsByTagName() and getElementsByClassName() return a live collection of items
- **Look at the script below to explain static and live collections**

```
//  NodeList (non-live/static collection)
const staticList = document.querySelectorAll("p");
console.log(staticList.length); // prints: 2
 //  HTMLCollection (live collection)
const liveList = document.getElementsByTagName("p");
console.log(liveList.length); // prints: 2
// add a third paragraph using JS
const thirdPar = document.createElement("p");
thirdPar.className = "thirdPar";
 document.getElementById("my-div").appendChild(thirdPar);
console.log(staticList.length); // prints: 2
 console.log(liveList.length); // prints: 3
```

- **Note**: If you have a situation where the DOM is going to be changing and you specifically want your collection of elements to reflect the current UI, a method that returns a live list is a better option.

### 8.9 Selecting elements (traversing between multiple elements)

- **Traversing between multiple elements**: Traversing is the act of selecting an element from another/neighboring element.
- **Traversing directions**: we can traverse in three directions:

```
<div class="hello" >
<ul id="fruitId" class="hello">
      <li id="one" class="red">Apple</li>
      <li id="two" class="yellow">Mango</li>
       <li id="three" class="red">Peach</li>
</ul>
</div>
```

  - **Traversing downwards**:
    - **firstElementChild()**: The firstElementChild property returns the first child element of the specified element.

```
console.log(document.getElementById("listOfFruits"
).firstEleme  ntChild);// prints the first li
element under the ul
```

    - **lastElementChild()**: The lastElementChild property returns the last child element of the specified element.

```
console.log(document.getElementById("fruitId").las
tElementChi   ld);// prints the last li element
under the ul
```

  - **Traversing upwards**:
    - **parentElement()**: parentELement is a property that lets you select the parent element. parentElement is great for selecting one level upwards

```
console.log(document.getElementById("one").parentE
lement);//    prints the <ul> which is the parent
to all the <li>
```

- **closest()**: To find an element that can be multiple levels above the current element, you use the closest method. closest lets you select the closest ancestor element that matches a selector.

```
console.log(document.getElementById("one").closest
(". hello")); //prints the closest parent or the
<ul>.
```

  - **Note:** The <li>s selected here has the  and <ul> as its parents, but closest allows us choose the closest parent

- **Traversing sideways**:
  - **previousElementSibling**: The previousElementSibling property returns the previous element of the specified element, in the same tree level. Look at the example above:

```
console.log(document.getElementById("three").previ
ousElement Sibling)//prints the 2nd li
```

  - **nextElementSibling**: The nextElementSibling property returns the element immediately following the specified element, in the same tree level. Look at the example above:

```
console.log(document.getElementById("two").nextEle
mentSibli ng); // prints the 3rd li
```

- **Question to ask**: Why do we need to learn to traverse the DOM? When can we easily use document.querySelector() ? Let's look at this example:

```
<div class="ulDiv" >
 <ul class="greetings" id="greetingsID" >
```

```
<li class="bye">Bye 1 here</li>
<li class="bye">Bye 2 here</li>
</ul>
</div>
 <div class="noULDiv">
<div class="hello">Bye 5 here</div>
</div>
```

- **Based on the above example code**:
  - console.log(document.querySelector("div")),
    - What querySelector() method does above is basically full search on the document to find all elements with "div" tag and print only the first div (the div with "ulDiv " class name).
  - console.log(document.getElementById("greetingsID").parentElement)
    - Here, the getElementById method goes straight to the element with "greetingsID" id and the parentElement method will find and the parent of that element parent

### 8.10 Altering values (working with HTML content)

- At the beginning of this class, we said that there are two steps involved in DOM manipulation: selecting elements and altering the element, its text content or its attributes. We have spent a lot of time on selecting elements. It is now time to discuss on altering
- **Altering HTML content**: The HTML DOM allows JavaScript to change the content of HTML elements. Meaning, we can add, update or remove the HTML tag, the text content in an HTML element, or the entire block of HTML code from your page. These are the most common methods that allow us add, update or remove contents in an HTML document.
- **Let's use the below HTML document as an example to explain altering**:

```
<div id="bigId">
<div>Hello</div>
<ul id="listOfFruits">
<li id="one" class="red">Apple</li>
```

```
<li id="two" class="yellow">Mango</li>
</ul>
</div>
```

- **createElement() method**: The createElement() method creates an Element Node with the specified element name.
    - **Syntax**: document.createElement(nodename)
    - **Note 1**: After the element is created, use the element.appendChild() or element.insertBefore() method to insert it to the HTML document.
    - **Note 2**: HTML elements often contains text. To insert a text into the element you created, use the innerText or innerHTML properties.
    - **Let's create a new <li> and insert it as the last child of our <ul>**

```
var liElem = document.createElement("li");
var parent =
document.getElementById("listOfFruits");
parent.appendChild(liElem);
console.log(parent); // shows that a new <li> is
added as the last child of the parent node (the <ul>)
```

- **appendChild() method**: The appendChild() method appends a set of node objects as the last child of a node. See the above example.
    - **Syntax**: node.appendChild(node)
- **prepend () method:** The method inserts a set of node objects before the first child of the Element. See the above example.
    - **Syntax**: node. prepend(node)
    - **Let's insert an <li> as the first child of our <ul>**

```
var liElem2 = document.createElement("li");
var parent = document.getElementById("listOfFruits");
parent.prepend(liElem2);
console.log(parent);// shows that a new <li> is added
as the first child of the parent node (the <ul>)
```

- **innerHTML() method**: The innerHTML property changes or prints the HTML content (inner HTML) of an element. This method returns HTML, as its name indicates.
    - **Syntax**: HTMLElementObject.innerHTML
    - **Let's change the inner HTML of our  with " bigId " id**

```
var content = document.getElementById(" bigId ");
console.log(content.innerHTML); // prints all the
inner HTML    under   the div with " bigId " id
content.innerHTML = "<li>Kiwi</li>"; // changing
inner HTML of div (with " bigId " id) to <li>Kiwi</li>
console.log(content.innerHTML);// now prints
<li>Kiwi</li>
```

- **textContent() method**: The textContent property changes or returns the text content of the specified node, and all its descendants. TextContent() returns all text contained by an element and all its children.
    - **Syntax**: node.textContent
    - **Let's change the text content of our  that has "bigId" as its id**

```
var content = document.getElementById("bigId");
console.log(content.textContent);// prints
“hello”, “apple”, and “mango”
content. textContent = "New text here";
console.log(content.textContent);// prints "New
text here"
```

- **removeChild() method**: The removeChild() method removes a specified child node of the specified element. Note: Use the appendChild() or insertBefore() methods if you want to insert the removed node back into the same document.
    - **Syntax**; removeChild(child);
    - **Let's remove the <li> child from our <ul>**

```
var ULcontainer =
document.getElementById("listOfFruits");
var unwantedLi = document.getElementById("two");
console.log(ULcontainer); // prints <ul> with all
its <li> children
ULcontainer.removeChild(unwantedLi); // prints
<ul>, but the <li> with "two" id is removed
```

### 8.11 Altering values (working with HTML attribute)

- **Working with attributes**: This is when you want to add, change or remove the attribute value of an element. Let's use the example code below to explain how we can alter attributes

```
<form id=" formID " >
<label >First Name</label>
 <input type="text" name="firstName">
 <label for="">Last Name</label>
<input type="text" name="lastName" >
</form>
```

  - **className() method**: The className property sets or returns the class name of an element (the value of an element's class attribute). **Note**: To apply multiple classes, separate them with spaces, like "test demo"
    - **Syntax**: HTMLElementObject.className
    - **Let's add 2 class names for <form> and print the added class names**

```
var myForm = document.getElementById("formID");
myForm.className = "formClass1, formClass2"; //
"formClass1" and "formClass2" class names added
console.log(myForm.className);// prints
formClass1, formClass2
```

```
console.log(myForm.classList); // Also prints a list containing the new class names "formClass1" and "formClass2"
```

- **ClassList() method**: The classList property returns the class name(s) of an element. This property can be used to add, remove and toggle CSS classes on an element.
    - **Syntax**: element.classList
    - **Let's add a class name for our last name <input> and print the added class name**

```
var classForLastName=document.getElementsByName("lastName")[0];
classForLastName.classList.add("myLastName");// adds class name
console.log(classForLastName.classList);// prints a DOMToken list containing the "myLastName" we just added above
```

- **Id property**: The id property sets or returns the id of an element (the value of an element's id attribute).
    - **Syntax**: HTMLElementObject.id
    - **Let's print the id for our <form> element**

```
var myForm = document.getElementById("formID");
console.log(myForm.id); // prints the id name "formID"
```

- **hasAttribute() method**: The hasAttribute() method returns a Boolean value (true or false) indicating whether the specified element has the specified attribute or not.
    - **Syntax**: element.hasAttribute(attributename)
    - **Let's check if our <form> element has "name" and "id" attributes**

```
var myForm = document.getElementById("formID");
console.log(myForm.hasAttribute("name")); // prints false because the <form> element doesn't have a name attribute
```

```
console.log(myForm.hasAttribute("id")); //prints
true because <form> element has id attribute.
```

- **getAttribute() method**: The getAttribute() method returns the value of the attribute of an element.
    - **Syntax**: element.getAttribute(attributename)
    - **Let's print the value of the "name" attribute for our first name <input>**

```
var firstInputVal =
document.getElementsByTagName("input")[0];
console.log(firstInputVal.getAttribute("name"));
// returns "firstName" which is the value of the
"name" attribute for first name <input>
```

- **setAttribute() method**: The setAttribute() method adds the specified attribute to an element and gives it the specified value. If the specified attribute already exists, only the value is set/changed. Otherwise, a new attribute is added with the specified name and value.
    - **Syntax**: element.setAttribute(attributename, attributevalue)
        - Attributename parameter specifies the name of the attribute whose value is to be set. Even if you put upper case for the name, the attribute name is automatically converted to all lower-case when setAttribute() is called
        - attributevalue  parameter is the value to assign to the attribute
    - **Let's add a "value" attribute to our first name input. The value will be a placehoder for our First Name input.**

```
var firstInputVal =
document.getElementsByTagName("input")[0];
firstInputVal.setAttribute("value ", "first name
");// placeholder input value of "first name"
added to our <input>
```

- **removeAttribute() method**: The removeAttribute() method removes the specified attribute from an element.
    - **Syntax**: element.removeAttribute(attributename)
    - **Let's remove the "type" attribute form the first name <input>**

```
var firstInputVal =
document.getElementsByTagName("input")[0];
firstInputVal.removeAttribute("type");// removes
the "type" attribute
console.log(firstInputVal.getAttribute("type"));
// returns null because we just removed the
"type" attribute
```

### 8.12 Altering values (working with inline styling)

- **Changing CSS Values with the DOM "style" property**
    - We use the style property to manipulate the inline style of the HTML elements using JavaScript
    - The style property is used to get as well as add the inline style of an element using different CSS properties. We can set almost all the styles for the elements like colors, fonts, text alignments, margin, borders, background images, sizes, and more.
    - The style property is extremely helpful to dynamically change the visual representation of an HTML element using JavaScript. **Note**: the JavaScript syntax for setting CSS properties is slightly different than CSS (Ex: backgroundColor instead of background-color). Visit the below link to see the syntax to add CSS using JavaScript:
        - http://www.sitestepper.be/en/css-properties-to-javascript-properties-reference-list.htm
    - **Syntax**:
        - **To return style properties**: = element.style.property
        - **To set/add style properties**: = element.style.property = value

- **How to access the style property**: we can use the getElementById() method

- **Let's use the following example code to add or return CSS properties using style property**

```
<h1 id="title">Fruits</h1>
 <div class="containerDiv" id="divID">
<ul id="listOfFruits" class="Kelele">
<li id="one" class="red">Apple</li>
<li id="two" class="yellow">Mango</li>
</ul>
</div>
```

- **Style color property**: The color property sets or returns the color of the text.
  - **Let's add "red" color to the text of the first <li>**

```
var firstLi = document.getElementById("one");
firstLi.style.color = "red"; // changes text
color of <li> to red
console.log(firstLi.style.color);// prints "red"
```

- **Style font property**: The font property sets or returns up to six separate font properties, in a shorthand form. With this property, we can set/return the following font properties in the order they are written; font-style, font-variant, font-weight, font-size, line-height and font-family.
  - **Let's add font (font-weight, font-size and font-family) to our 2nd <li> and print the font values we just added**

```
var secondLi = document.getElementById("two");
secondLi.style.font = "bold 30px arial"; //
changes text font of 2nd <li>
console.log(secondLi.style.font); // prints
"bold 30px arial"
```

- **Style background color property**: The backgroundColor property sets or returns the background color of an element.
    - **Let's change the background color of 2nd <li> into yellow and print the changed color**

```
var secondLi = document.getElementById("one");
secondLi.style.backgroundColor = "yellow"; //
changes the background color of 2nd <li>to yellow
console.log(secondLi.style.backgroundColor); //
prints "yellow"
```

- **Style display property**: The display property sets or returns the element's display type. Elements are mostly block or inline. You can also hide the element using display:none
    - **Let's change the display property of our <li> into none and print the changed display**

```
var h1Element = document.getElementById("title");
h1Element.style.display = "none"; // hides our <h1>
console.log(h1Element.style.display); // prints
"none" because we just hid our <h1> above
```

- **Style border property**: The border property sets or returns up to three separate border properties, in a shorthand form. With this property, you can set/return one or more of the following (in any order): border-width, border-style and border-color.
    - **Let's change the border property of our  into thick, solid, and red and print the properties**

```
var divElement = document.getElementById("divID");
divElement.style.border = "thick solid red";
console.log(divElement.style.border);// returns
“thick solid red”
```