**How to Manipulate DOM Elements in JavaScript ?**

In [**web development**](https://www.geeksforgeeks.org/web-development/), one of the main features to enable interactivity is [**DOM Manipulation**](https://www.geeksforgeeks.org/dom-document-object-model/). [**DOM manipulation**](https://www.geeksforgeeks.org/dom-document-object-model/) allows developers to interact and modify the structure, style, and content of web pages dynamically.

The below list contains different methods to manipulate DOM.

**Table of Content**

* [DOM Element Selectors](https://www.geeksforgeeks.org/how-to-manipulate-dom-elements-in-javascript/#dom-element-selectors)
* [DOM content manipulators](https://www.geeksforgeeks.org/how-to-manipulate-dom-elements-in-javascript/#dom-content-manipulators)
* [DOM element creators](https://www.geeksforgeeks.org/how-to-manipulate-dom-elements-in-javascript/#dom-element-creators)
* [DOM CSS manipulators](https://www.geeksforgeeks.org/how-to-manipulate-dom-elements-in-javascript/#dom-css-manipulators)

**DOM Element Selectors**

There are multiple methods available in [**DOM**](https://www.geeksforgeeks.org/dom-document-object-model/) to select the [**HTML elements**](https://www.geeksforgeeks.org/html-elements/) and manipulate them as listed below:

* **querySelector() method:** The [**querySelector( ) method**](https://www.geeksforgeeks.org/html-dom-queryselector-method/)is used to select the first element that matches the specified selector provided as an argument.
* **querySelectorAll() method:**The [**querySelectorAll() method**](https://www.geeksforgeeks.org/html-dom-queryselectorall-method/) returns a NodeList which contains all elements that match the selector within the document.
* **getElementById( ) method:**The [**getElementId() method**](https://www.geeksforgeeks.org/html-dom-getelementbyid-method/) is used to select a single element using its Id.
* **getElementByClassName( ):**The [**getElementsByClassName() method**](https://www.geeksforgeeks.org/html-dom-getelementsbyclassname-method/) in [**JavaScript**](https://www.geeksforgeeks.org/javascript/) is used to select elements in the HTML document that have a specific class name.
* **getElementByTagName() method:**The [**getElementsByTagName()** **method**](https://www.geeksforgeeks.org/html-dom-getelementsbytagname-method/) is used to select elements using the specified tag name.
* **getElementByName() method:** The [**getElementsByName() method**](https://www.geeksforgeeks.org/html-dom-getelementsbyname-method/) is used to select elements that have a specific name attribute. This method returns a NodeList of elements with the specified name.

**Example:**The below code explain the use of the above listed **DOM selectors**.

HTML

<!DOCTYPE html>

<**html**>

<**head**>

<**title**>

GeeksForGeeks

</**title**>

</**head**>

<**body** style="text-align: center;">

<**div** id="target">

<**h1**>Hello Geeks</**h1**>

<**h2** class="example">

This is a sample sub heading

</**h2**>

<**p**>

This is sample paragraph1

</**p**>

<**p**>

This is sample paragraph2

</**p**>

<**input** type="text" name="username"

id="userName">

</**div**>

<**script**>

**const** queryS =

document.querySelector(".example");

**const** gebi =

document.getElementById("target");

**const** gebcn =

document.getElementsByClassName('example');

**const** gebtn =

document.getElementsByTagName('p');

**const** gebn =

document.getElementsByName('username');

**const** querySAll =

document.querySelectorAll("p");

console.log(queryS);

console.log(gebi);

console.log(gebcn);

console.log(gebn);

console.log(gebtn);

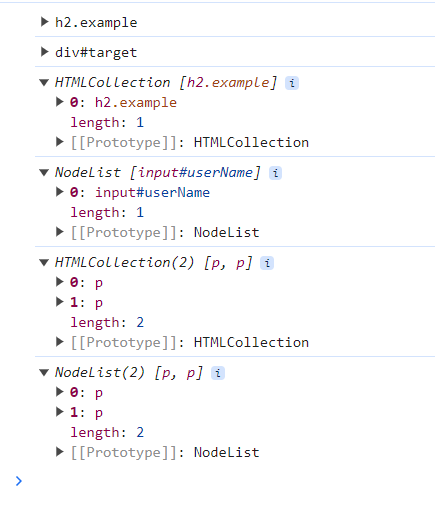
console.log(querySAll);

</**script**>

</**body**>

</**html**>

**Output:**



**DOM content manipulators**

The below [**DOM**](https://www.geeksforgeeks.org/dom-document-object-model/) properties can be used to change the [**HTML**](https://www.geeksforgeeks.org/html-tutorial/) and the text content of the [**HTML elements**](https://www.geeksforgeeks.org/html-elements/).

* **textContent:**The [textContent](https://www.geeksforgeeks.org/html-dom-textcontent-property/) property is used to set and get the text content and it’s descendants.
* **innerHTML:**The [innerHTML](https://www.geeksforgeeks.org/html-dom-innerhtml-property/) property is used to get and set the text and [**HTML**](https://www.geeksforgeeks.org/html-tutorial/) inside an [**HTML element**](https://www.geeksforgeeks.org/html-elements/).
* **innerText:** The [innerText](https://www.geeksforgeeks.org/html-dom-innertext-property/) property represents the rendered text content of a node and its descendants. It is aware of the elements which will be rendered and ignores the hidden elements.

**Example:**The below code will explain the use of the above listed DOM manipulators.

HTML

<!DOCTYPE html>

<**html**>

<**head**>

<**title**>

DOM Manipulation Techniques

</**title**>

</**head**>

<**body** style="text-align: center;">

<**h1**>Hello</**h1**>

<**h2**>Hello</**h2**>

<**h3**>Hello</**h3**>

<**script**>

**const** tcontentexample =

document.querySelector("h1");

**const** innerhtmlexample =

document.querySelector("h2");

**const** innertextexample =

document.querySelector("h3");

tcontentexample.textContent =

"Hello Geeks!- textContent";

innerhtmlexample.innerHTML =

"Hello Geeks!- innerHTML";

innertextexample.innerText =

"Hello Geeks! - innerText";

</**script**>

</**body**>

</**html**>

**Output:**



**DOM element creators**

You can also create dynamic [**HTML elements**](https://www.geeksforgeeks.org/html-elements/) using the below [**DOM**](https://www.geeksforgeeks.org/dom-document-object-model/) methods.

* **createElement( ) method:** The [createElement( )](https://www.geeksforgeeks.org/html-dom-createelement-method/) method is used to create an element in the HTML DOM document
* **appendChild( ) method:** The [appendChild( )](https://www.geeksforgeeks.org/html-dom-appendchild-method/) method is used to append a node to a list of child nodes of a specified parent node.
* **createTextNode( ) method:** The [createTextNode( )](https://www.geeksforgeeks.org/html-dom-createtextnode-method/) method is used to create a text node.

**Example:**The below code will explain the use of the above listed DOM manipulators.

HTML

<!DOCTYPE html>

<**html**>

<**head**>

<**title**>

DOM Manipulation Techniques

</**title**>

</**head**>

<**body** style="text-align: center;">

<**h1**>Hello 1</**h1**>

<**h2**>Hello 2</**h2**>

<**h3**>Hello 3</**h3**>

<**script**>

**const** para =

document.createElement("p");

para.textContent =

"This paragraph is created using createElement!";

**const** para1 =

document.createTextNode("p");

para1.textContent =

"This paragraph is created using createTextNode";

document.body.appendChild(para);

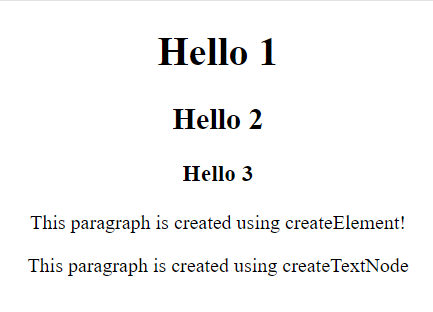
document.body.appendChild(para1);

</**script**>

</**body**>

</**html**>

**Output:**



**DOM CSS manipulators**

The below [**DOM**](https://www.geeksforgeeks.org/dom-document-object-model/) properties can be used to manipulate the [**CSS**](https://www.geeksforgeeks.org/css-tutorial/) of the [**HTML elements**](https://www.geeksforgeeks.org/html-elements/).

* **style: CSS** can be manipulated using (Syntax : **element.style.propertyName)**which helps us to dynamically manipulate styles and access the inline styles of an element.
* **cssText:** **CSS** can be directly manipulated using the [cssText](https://www.geeksforgeeks.org/cssstyledeclaration-csstext-property/) property of the style object to set multiple inline styles at a time. It is similar to how we add inline CSS to an element.
* **classList:** As Inline CSS is not recommended, mostly we use [classList](https://www.geeksforgeeks.org/html-dom-classlist-property/) property to dynamically add and remove CSS classes which are defined in external CSS.

**Example**: The below code explain the use of the above listed DOM Manipulation techniques.

HTML

<!DOCTYPE html>

<**html**>

<**head**>

<**title**>

DOM Manipulation Techniques

</**title**>

<**style**>

.**classList** {

**color**: **white**;

**background-color**: **black**;

**text-align**: **center**

}

</**style**>

</**head**>

<**body**>

<**h1**>Hello 1</**h1**>

<**h2**>Hello 2</**h2**>

<**h3**>Hello 3</**h3**>

<**script**>

**const** tcontentexample =

document.querySelector("h1");

**const** innerhtmlexample =

document.querySelector("h2");

**const** innertextexample =

document.querySelector("h3");

tcontentexample.style.

color = "white";

tcontentexample.style.

backgroundColor = "black";

tcontentexample.style.

textAlign = "center";

innerhtmlexample.style.cssText =

`color: white;

background-color: black;

text-align:center`;

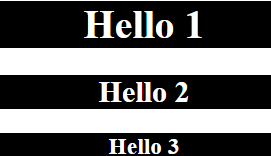
innertextexample.classList.add("classList");

</**script**>

</**body**>

</**html**>

**Output:**



Want to be a Software Developer or a Working Professional looking to enhance your **Software Development Skills**? Then, master the concepts of Full-Stack Development. Our [**Full Stack Development - React and Node.js Course**](https://gfgcdn.com/tu/Q3I/) will help you achieve this quickly. Learn everything from **Front-End to Back-End Development** with hands-on **Projects**and real-world examples. This course enables you to build scalable, efficient, dynamic web applications that stand out. Ready to become an expert in Full-Stack? Enroll Now and Start Creating the Future!

**Creating and removing HTML elements with JavaScript**

[[](https://medium.com/@tom.hendrych?source=post_page-----372bbd4cfdcc--------------------------------)](https://medium.com/@tom.hendrych?source=post_page-----372bbd4cfdcc--------------------------------)

[Tom Hendrych](https://medium.com/@tom.hendrych?source=post_page-----372bbd4cfdcc--------------------------------)

·

Follow

7 min read

·

Jan 26, 2024

2

In my second article in a series dedicated to **DOM API**, I will focus on **adding and removing HTML elements from the Document Object Model (DOM).** These actions are the bread and butter for creating modern highly interactive web applications.

The article is again divided into two main sections. The first section is dedicated to some basic theory, how to create and remove HTML elements and which methods are available to us. In the second part, we will together create a super simple application that will be able to create title and subtitle. Set their content and set class attributes on them.

**Theoretical part**

The process of displaying a newly created HTML element involves 3 steps:

1. Create element
2. Set Content / Attribute
3. Insert element to the DOM

**Create element**

To create HTML elements there is a method called document.createElement('tagName') . The tagName parameter can be any valid HTML element e.g. <h1> , <a> etc. The method returns a new HTML element.

const title = document.createElement("h1");

**Set Content / Attribute**

So, We have created an element, but what next? Our new element has no content. We can use one of the methods from [my previous article](https://medium.com/@tom.hendrych/a-guide-to-changing-html-content-with-javascript-0bf0495cf5b0). In this example, we will use textContent, but we could potentially use .innerHTML method if we wanted to use the more complex structure with nested elements.

title.textContent = 'Our new H1 element';  
title.innerHTML = "Our new <strong>H1 element</strong>"

The method .setAttribute(key, value) accepts two arguments. The first argument is key and it can be any valid attribute name. The second parameter as the name suggests set the value to our attribute. If the attribute already exists, the current value will be overridden by the new value.

const button = document.querySelector("button");  
button.setAttribute("class", "button");

**Insert element to the DOM**

Finally, we must insert our element with content to DOM in order to be visible to our users. There is a several methods that we can use.

We have four possible methods:

1. .appendChild()
2. .append()
3. .prepend()
4. .insertBefore()

The appendChild() method and append() are fairly similar. Both methods add our new element as a **the** **last child of theselected parent element**. The appendChild(element) accepts a single argument, that we want to insert, while append(element1, element2, ... elementN) accepts 1 to N elements. The append() method was introduced in ECMAScript 2015, so it has worse browser support than appendChild() method, but should be supported in all modern browsers. Here you can compare browser support in [this link](https://caniuse.com/?search=append).

// appendChild() example  
const parent = document.getElementById("parentElement");  
const child = document.createElement("div");  
parent.appendChild(child);  
  
// append() example  
const parent = document.getElementById("parentElement");  
const child1 = document.createElement("div");  
const child2 = document.createElement("span");  
parent.append(child1, child2, "Some text");

To insert a new element **at the start of the parent element** we got prepend() , this method inserts an element or string before the first child of the specified parent.

const parentElement = document.getElementById('parentElement');  
const newElement = document.createElement('p');  
newElement.textContent = 'Lorem ipsum';  
  
// Prepend the new element to the parent element  
parentElement.prepend(newElement);

We can also insert new element before another element. To do, we can use method .insertBefore(newElement, referenceElement) . The method accepts two required arguments. The first parameter is new element, which we want to insert. The second parameter is reference element before which we want to insert our new element.

const newElement = document.createElement('p');  
newElement.textContent = 'Inserted content';  
  
const referenceElement = document.getElementById('referenceElement');  
referenceElement.parentNode.insertBefore(newElement, referenceElement);

**Remove element**

The process of removing elements from DOM is quite straightforward. We can use one of two methods:

1. .remove()
2. .removeChild()

const elementToRemove = document.getElementById('exampleElement');  
elementToRemove.remove();

// .removeChild() example  
const elementToRemove = document.getElementById('exampleElement');  
const parentElement = elementToRemove.parentNode;  
parentElement.removeChild(elementToRemove);

With both methods we achieve the same result, but the element.remove() method is more modern and concise and requires slightly less code because we do not have to access parentNode to remove his child. On the other hand .removeChild() has better browser support, but all modern browsers should support .remove() .

**Practical part**

In our practical section, we will build a super simple application, which will be able to create a title and subtitle on button click action. Then, We will remove those elements.

This is our final app:

Let's start with the index.html file. Use the code below as a starting point for our app.

<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
 <link rel="stylesheet" href="styles.css" />  
 <script src="https://cdn.tailwindcss.com"></script>  
 <script src="index.js" defer></script>  
 <title>Changing page content</title>  
 </head>  
 <body>  
 <!-- our HTML will be here-->  
 </body>  
</html>

Then between body tag, we will add two buttons. One for creating and the second for removing elements. Do not forget to set id attribute accordingly, we will need them later in this tutorial. That should be all HTML for now.

<body>  
 <button id="createElement" type="button" class="button button--green">  
 Create element  
 </button>  
  
 <button id="deleteElement" type="button" class="button button--red">  
 Remove element  
 </button>  
</body>

Create styles.css file. All styles that we will need are included in the code below.

\* {  
 box-sizing: border-box;  
}  
  
body {  
 background: #111827;  
 font-family: 'Inter', sans-serif;  
 margin: 0 auto;  
 padding: 3rem 1.5rem;  
 max-width: 850px;  
}  
  
.title {  
 margin-bottom: 1rem;  
 font-size: 3.75rem;  
 line-height: 1;  
 font-weight: 800;  
 color: #ffffff;  
}  
  
.subtitle {  
 font-size: 1.5rem;  
 line-height: 2rem;  
 color: #ffffff;  
 margin-bottom: 12px;  
}  
  
.button {  
 padding-top: 0.625rem;  
 padding-bottom: 0.625rem;  
 padding-left: 1.25rem;  
 padding-right: 1.25rem;  
 border-radius: 0.5rem;  
 margin: 1rem 0;  
 margin-right: 1rem;  
 width: 100%;  
 font-size: 0.875rem;  
 line-height: 1.25rem;  
 font-weight: 500;  
 text-align: center;  
 color: #ffffff;  
  
 @media (min-width: 640px) {  
 width: auto;  
 }  
}  
  
.button--green {  
 background-color: #2ecc71 !important;  
}  
  
.button--green:hover {  
 background-color: #27ae60 !important;  
}  
  
.button--red {  
 background-color: #e74c3c !important;  
}  
  
.button--red:hover {  
 background-color: #c0392b !important;  
}

Finally, we can start write our Javascript code. So let's create index.js file. First of all, we need to get reference of our buttons. We can use the method getElementById .

const createBtnEl = document.getElementById('createElement');  
const deleteBtnEl = document.getElementById('deleteElement');

Now, we will create our new function createElementWithText which will help us write slightly less code because we want to reuse our code. Our factory function will accept 3 arguments: tagName , textContent , className . The tagName parameter will be used to create the HTML element we want, then we will use textContent parameter to set the content of our element and our last parameter will be used to set the CSS class attribute and help us with the styling of the new element. Then we will return our “prepared” element from the function.

const createBtnEl = document.getElementById('createElement');  
const deleteBtnEl = document.getElementById('deleteElement');  
  
// our new factory function  
function createElementWithText(tagName, textContent, className) {  
 const newElement = document.createElement(tagName);  
 newElement.textContent = textContent;  
 newElement.setAttribute('class', className);  
  
 return newElement;  
}

Now, we will add click event listeners to these buttons. So when the user clicks the respective button then the callback function will be called.

const createBtnEl = document.getElementById('createElement');  
const deleteBtnEl = document.getElementById('deleteElement');  
  
  
createBtnEl.addEventListener('click', () => {  
 // our code will be here  
});  
  
deleteBtnEl.addEventListener('click', () => {  
 // our code will be here  
});

In our createButtonEl callback function, we will create title and subTitle and assign them the element created in our function createElementWithText . So now we use append()method to insert the title and subtitle to the end of the body element.

createBtnEl.addEventListener('click', () => {  
   
 const newTitle = createElementWithText('h1', 'new Title', 'title');  
 const newSubtitle = createElementWithText(  
 'h2',  
 'This is a subtitle',  
 'subtitle'  
 );  
  
 document.body.append(newTitle, newSubtitle);  
});

If you click on the green button, the title and subtitle should appear on the page.

Let's move to our last missing piece of thepuzzle — removing elements. Create the new function removeElement(tagName) . Our function accepts only HTML element names. First of all, we need to find the element we want to remove. To do so, we will find the element with method querySelector() . Then we check if we found our desired element. If not, then we will print the message into the console and stop the execution of the rest of the code in our function.

If we find the element, then we will call the method remove() on itself.

function removeElement(tagName) {  
 const elementToRemove = document.querySelector(tagName);  
  
 if (!elementToRemove) {  
 console.warn(`No element ${tagName} found`);  
 return;  
 }  
  
 elementToRemove.remove(elementToRemove);  
}

Then will call our function removeElement with arguments h1 and h2 in order to remove title and subtitle .

I should note that finding elements by their tag name is not the best idea because the webpage can contain many instances of the same element and the method querySelector() finds and returns only first instance of the searched element. A better idea would be to give them some unique identifier e.g. id=”title” attribute. But for our simple app is it enough.

deleteBtnEl.addEventListener('click', () => {  
 removeElement('h1');  
 removeElement('h2');  
});

That is it. Try to click on the green button and then click on the red button. The title , and subtitle should appear on the green button click and disappear on the red button click.

Our final code: <https://jsfiddle.net/jimmzzz/qd51gw7L/76/>

And now we have completed our app 🎉  
Lets have a coffee :)

**Conclusion**

In this tutorial we have learned the following:

* How to create an HTML element with the method .createElement()
* How to insert a new element to DOM
* methods .append() and .appendChild() add an element as a **LAST** child of a respective parent element
* method .prepend() adds an element as a **FIRST** child of the respective parent
* .insertBefore() allows to insert element before another element
* To remove an element, we have two methods .remove() and .removeChild()