**1. [querySelector()](https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector" \t "_blank)**

The querySelector() method **returns the first element that matches one or more CSS selectors**. If no match is found, it returns null.

Before querySelector() was introduced, developers widely used the [getElementById()](https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementById" \t "_blank) method which **fetches an element with a specified id value**.

Although getElementById() is still a useful method, but with the newer querySelector() and querySelectorAll() methods we are free to **target elements based on any**[**CSS selector**](https://www.w3.org/TR/css3-selectors/#selectors), thus we have more flexibility.

**Syntax**

|  |  |
| --- | --- |
|  | var ele = document.querySelector(selector); |

* ele – First matching element or null (if no element matches the selectors)
* selector – one or more CSS selectors, such as "#fooId", ".fooClassName", ".class1.class2", or ".class1, .class2"

**Code Example**

In this example, the first <div> gets selected with the querySelector() method and its color is changed to red.

|  |  |
| --- | --- |
| HTML | |
| 1  2  3  4  5 | <p>paragraph one</p>  <p>paragraph two</p>  <div>div one</div>  <p>paragraph three</p>  <div>div two</div> |
| JavaScript | |
| 1  2 | var firstDiv = document.querySelector('div');  firstDiv.style.color = 'red'; |

**2. [querySelectorAll()](https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelectorAll" \t "_blank)**

Unlike querySelector() that returns only the first instance of all matching elements, querySelectorAll() **returns all elements that match the specified CSS selector**.

The matching elements are returned as a [NodeList](https://developer.mozilla.org/en-US/docs/Web/API/NodeList" \t "_blank) object that will be an empty object if no matching elements are found.

**Syntax**

|  |  |
| --- | --- |
| 1 | var eles = document.querySelectorAll(selector); |

* eles – A NodeList object with all matching elements as property values. The object is empty if no matches are found.
* selector – one or more CSS selectors, such as "#fooId", ".fooClassName", ".class1.class2", or ".class1, .class2"

**Code Example**

The example below uses the same HTML as the previous one. However, in this example, all paragraphs are selected with querySelectorAll(), and are colored blue.

|  |  |
| --- | --- |
| HTML | |
| 1  2  3  4  5 | <p>paragraph one</p>  <p>paragraph two</p>  <div>div one</div>  <p>paragraph three</p>  <div>div two</div> |
| JavaScript | |
| 1  2  3 | var paragraphs = document.querySelectorAll('p');  for(var p of paragraphs)    p.style.color = 'blue'; |

**3. [addEventListener()](https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener" \t "_blank)**

**Events** refer to what happens to an HTML element, such as clicking, focusing, or loading, to which we can react with JavaScript. We can assign JS functions to *listen* for these events in elements and do something when the event had occured.

There are three ways you can **assign a function to a certain event.**

If foo() is a custom function, you can register it **as a click event listener** (call it when the button element is clicked) in three ways:

|  |  |
| --- | --- |
| HTML | |
| 1 | <button onclick=foo>Alert</button> |

|  |  |
| --- | --- |
| JavaScript | |
| 1  2 | var btn = document.querySelector('button');  btn.onclick=foo; |

|  |  |
| --- | --- |
| JavaScript | |
| 1  2 | var btn = document.querySelector('button');  btn.addEventListener('click', foo); |

The method addEventListener() (the third solution) has **some pros**; it is the latest standard – allowing the assignment of more than one function as event listeners to one event – and comes with a useful set of options.

**Syntax**

|  |  |
| --- | --- |
| 1 | ele.addEventListener(evt, listener, [options]); |

* ele – The HTML element the event listener will be listening for.
* evt – The targeted event.
* listener – Typically, a JavaScript function.
* options – (optional) An object with a set of boolean properties (listed below).

| **Options** | **What happens, when it is set to true?** |
| --- | --- |
| capture | Stops event bubbling, i.e. prevents calling of any event listeners for the same event type in the element’s ancestors.  To use this feature, you can use 2 syntaxes:   1. ele.addEventListener(evt, listener, true) 2. ele.addEventListener(evt, listener, {capture:true}); |
| once | Listener is called only the first time the event happens, then it is automatically detached from the event, and won’t be triggered by it anymore. |
| passive | The default action of the event cannot be stopped with the [preventDefault()](https://developer.mozilla.org/en/docs/Web/API/Event/preventDefault" \t "_blank) method. |

**Code Example**

In this example, we add a click event listener called foo, to the <button> HTML tag.

|  |  |
| --- | --- |
| HTML | |
| 1 | <button>Click Me</button> |
| JavaScript | |
| 1  2  3 | var btn = document.querySelector('button');  btn.addEventListener('click',foo);  function foo() { alert('hello'); } |

**Interactive Demo**

Assign the foo() custom function as an event listener to any of the three following events: input, click or mouseover & trigger the chosen event in the bottom input field by hovering, clicking or typing in it.

**4. [removeEventListener()](https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/removeEventListener" \t "_blank)**

The removeEventListener() method **detaches an event listener** previously added with the addEventListener() method **from the event it is listening for**.

**Syntax**

|  |  |
| --- | --- |
| 1 | ele.removeEventListener(evt, listener, [options]); |

Uses the same syntax as the aforementioned addEventListener() method (except for the once option that’s excluded). The options are used to be very specific about identifying the listener to be detached.

**Code Example**

Following the Code Example we used at addEventListener(), here we remove the click event listener called foo from the <button> element.

|  |  |
| --- | --- |
| JavaScript | |
| 1 | btn.removeEventListener('click',foo); |

**5. [createElement()](https://developer.mozilla.org/en-US/docs/Web/API/Document/createElement" \t "_blank)**

The createElement() method **creates a new HTML element** using the **name of the HTML tag** to be created, such as 'p' or 'div'.

You can later add this element to the web page by using different **methods for DOM insertion**, such as AppendChild() (see later in this post).

**Syntax**

|  |  |
| --- | --- |
| 1 | document.createElement(tagName); |

* tagName – The name of the HTML tag you want to create.

**Code Example**

With the following example, you can create a new paragraph element:

|  |  |
| --- | --- |
| 1 | var pEle = document.createElement('p') |

**6. [appendChild()](https://developer.mozilla.org/en-US/docs/Web/API/Node/appendChild" \t "_blank)**

The appendChild() method **adds an element as the last child** to the HTML element that invokes this method.

The child to be inserted can be either a **newly created element**, or an **already existing one**. In the latter case, it will be moved from its previous position to the position of the last child.

**Syntax**

|  |  |
| --- | --- |
| 1 | ele.appendChild(childEle) |

* ele – The HTML element to which childEle is added as its last child.
* childEle – The HTML element added as the last child of ele.

**Code Example**

In this example, we insert a <strong> element is as the child of a <div> element using the appendChild() and the aforementioned createElement() methods.

|  |  |
| --- | --- |
| HTML | |
| 1 | <div></div> |
| JavaScript | |
| 1  2  3  4 | var div = document.querySelector('div');  var strong = document.createElement('strong');  strong.textContent = 'Hello';  div.appendChild(strong); |

**Interactive Demo**

In the following interactive demo, letters from #a to #r are the child elements of the #parent-one, #parent-two, and #parent-three id selectors.

Check out how the appendChild() method works by **typing one parent and one child selector name into the input fields below.**You can choose children belonging to another parent as well.

**7. [removeChild()](https://developer.mozilla.org/en-US/docs/Web/API/Node/removeChild" \t "_blank)**

The removeChild() method **removes a specified child element** from the HTML element that calls this method.

**Syntax**

|  |  |
| --- | --- |
| 1 | ele.removeChild(childEle) |

* ele – The parent element of childEle.
* childEle – The child element of ele.

**Code Example**

In this example, we remove the <strong> element we added as a child to the <div> tag at the Code Example for the previous appendChild() method.

|  |  |
| --- | --- |
| 1 | div.removeChild(strong); |

**8. [replaceChild()](https://developer.mozilla.org/en-US/docs/Web/API/Node/replaceChild" \t "_blank)**

The replaceChild() method **replaces a child element with another one** belonging to the parent element that calls this method.

**Syntax**

|  |  |
| --- | --- |
| 1 | ele.replaceChild(newChildEle, oldChileEle) |

* ele – Parent element of which children are to be replaced.
* newChildEle – Child element of ele that will replace oldChildEle.
* oldChildEle – Child element of ele, that will be replaced by newChildEle.

**Code Example**

In this example the child element <strong> belonging to the <div> parent element is replaced with a newly created <em> tag.

|  |  |
| --- | --- |
| HTML | |
| 1  2  3 | <div>    <strong>hello</strong>  </div> |
| JavaScript | |
| 1  2  3  4  5 | var em = document.createElement('em');  var strong = document.querySelector('strong');  var div = document.querySelector('div');  em.textContent = 'hi';  div.replaceChild(em, strong); |

**9. [cloneNode()](https://developer.mozilla.org/en/docs/Web/API/Node/cloneNode" \t "_blank)**

When you have to create a new element that needs to be the **same as an already existing element** on the web page, you can simply **create a copy of the already existing element** using the cloneNode() method.

**Syntax**

var dupeEle = ele.cloneNode([deep])

* dupeEle – Copy of the ele element.
* ele – The HTML element to be copied.
* deep – (optional) A boolean value. If it’s set to true, dupeEle will have all the child elements ele has, if it’s set to false it will be cloned without its children.

**Code Example**

In this example, we create a copy for the <strong> element with cloneNode(), then we add it to the <div> tag as a child element with the aforementioned appendChild() method.

As a result, <div> will contain two <strong> elements, both with the hello string as content.

|  |  |
| --- | --- |
| HTML | |
| 1  2  3 | <div>    <strong>hello</strong>  </div> |
| JavaScript | |
| 1  2  3  4 | var strong = document.querySelector('strong');  var copy = strong.cloneNode(true);  var div = document.querySelector('div');  div.appendChild(copy); |

**10. [insertBefore()](https://developer.mozilla.org/en-US/docs/Web/API/Node/insertBefore" \t "_blank)**

The insertBefore() method **adds a specified child element before another child element**. The method is called by the parent element.

If the referenced child element does not exist or you pass explicitly null in its place, the child element to be inserted is added **as the last child of the parent** (similar to appendChild()).

**Syntax**

|  |  |
| --- | --- |
| 1 | ele.insertBefore(newEle, refEle); |

* ele – Parent element.
* newEle – New HTML element to be inserted.
* refEle – A child element of ele **before which** newEle will be inserted.

**Code Example**

In this example, we create a new <em> element with some text inside, and add it **before** the <strong> element inside the <div> parent element.

|  |  |
| --- | --- |
| HTML | |
| 1  2  3 | <div>    <strong>hello</strong>  </div> |
| JavaScript | |
| 1  2  3  4  5 | var em = document.createElement('em');  var strong = document.querySelector('strong');  var div = document.querySelector('div');  em.textContent = 'hi';  div.insertBefore(em, strong); |

**Interactive Demo**

This interactive demo works similarly to our previous demo belonging to the appendChild() method. Here you only need to type the id selectors of two child elements (from #a to #r) into the input boxes, and you can see how the insertBefore() method moves the first specified child **before** the second.

**11. [createDocumentFragment()](https://developer.mozilla.org/en-US/docs/Web/API/Document/createDocumentFragment" \t "_blank)**

The createDocumentFragment() method may not be as well known as the others in this list, nevertheless is an important one, especially if you want to **insert multiple elements in bulk**, such as adding multiple rows to a table.

It **creates a [DocumentFragment](https://developer.mozilla.org/en/docs/Web/API/DocumentFragment" \t "_blank) object**, which essentially is **a DOM node that is not part of the DOM tree**. It’s like a buffer where we can add and store other elements (e.g. multiple rows) first, before adding them to the desired node in the DOM tree (e.g. to a table).

Why don’t we just add elements directly to the DOM tree? Because DOM insertion **causes layout changes**, and it’s an **expensive browser process** since multiple consequent element insertions will cause more layout changes.

On the other hand, adding elements to a DocumentFragment object **doesn’t cause any layout changes**, so you can add as many elements to it as you want before passing them to the DOM tree, causing a layout change only at this point.

**Syntax**

|  |  |
| --- | --- |
| 1 | document.createDocumentFragment() |

**Code Example**

In this example, we create multiple table rows and cells with the createElement() method, then add them to a DocumentFragment object, finally add that document fragment to a HTML <table> using the appendChild() method.

As a result, five rows – each of them containing one cell with a number from 1 to 5 as content – will be inserted inside the table.

|  |  |
| --- | --- |
| HTML | |
| 1 | <table></table> |
| JavaScript | |
| 1  2  3  4  5  6  7  8  9  10  11  12 | var table = document.querySelector("table");  var df = document.createDocumentFragment();    for(var i=0; i<5; i++) {    var td = document.createElement("td");    var tr = document.createElement("tr");    td.textContent = i;    tr.appendChild(td)    df.appendChild(tr);  }    table.appendChild(df); |

**12. [getComputedStyle()](https://developer.mozilla.org/en-US/docs/Web/API/Window/getComputedStyle" \t "_blank)**

Sometimes you want to **check the CSS property values** of an element before making any changes. You can use the [ele.style](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/style" \t "_blank) property to do the same, however the getComputedStyle() method has been made just for this purpose, it **returns the read-only computed values** of all the CSS properties of a specified HTML element.

**Syntax**

|  |  |
| --- | --- |
| 1 | var style = getComputedStyle(ele, [pseudoEle]) |

* style – A [CSSStyleDeclaration](https://developer.mozilla.org/en-US/docs/Web/API/CSSStyleDeclaration" \t "_blank) object returned by the method. It holds all CSS properties and their values of the ele element.
* ele – The HTML element of which CSS property values are fetched.
* pseudoEle – (optional) A string that represents a pseudo-element (for example, ':after'). If this is mentioned, then the CSS properties of the specified pseudo-element associated with ele will be returned.

**Code Example**

In this example, we get and alert the computed width value of a <div> element by using the getComputedStyle() method.

|  |  |
| --- | --- |
| HTML | |
| 1 | <div></div> |
| JavaScript | |
| 1  2 | var style = getComputedStyle(document.querySelector('div'));  alert(style.width); |

**13. [setAttribute()](https://developer.mozilla.org/en/docs/Web/API/Element/setAttribute" \t "_blank)**

The setAttribute() method either **adds a new attribute** to an HTML element, or **updates the value** of an attribute that already exists.

**Syntax**

|  |  |
| --- | --- |
| 1 | ele.setAttribute(name, value); |

* ele – The HTML element to which an attribute is added, or of which attribute is updated.
* name – The name of the attribute.
* value – The value of the attribute.

**Code Example**

In this example, we add the [contenteditable](https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/contenteditable" \t "_blank) attribute to a <div> by making use of the setAttribute() method, which will turn its content editable.

|  |  |
| --- | --- |
| HTML | |
| 1 | <div>hello</div> |
| JavaScript | |
| 1  2 | var div = document.querySelector('div');  div.setAttribute('contenteditable', '') |

**14. [getAttribute()](https://developer.mozilla.org/en/docs/Web/API/Element/getAttribute" \t "_blank)**

The getAttribute() method **returns the value of a specified attribute** belonging to a certain HTML element.

**Syntax**

|  |  |
| --- | --- |
| 1 | ele.getAttribute(name); |

* ele – The HTML element of which attribute is requested.
* name – The name of the attribute.

**Code Example**

In this example, we alert the value of the contenteditable attribute belonging to the <div> element with the help of the getAttribute() method.

|  |  |
| --- | --- |
| HTML | |
| 1 | <div contenteditable=true>hello</div> |
| JavaScript | |
| 1  2 | var div = document.querySelector('div');  alert(div.getAttribute('contenteditable')) |

**15. [removeAttribute()](https://developer.mozilla.org/en-US/docs/Web/API/Element/removeAttribute" \t "_blank)**

The removeAttribute() method **removes a given attribute** of a specific HTML element.

**Syntax**

ele.removeAttribute(name);

* ele – The HTML element of which attribute is to be removed.
* name – The name of the attribute.

**Code Example**

In this example, we remove the contenteditable attribute from the <div> element. As a result, the <div> won’t be editable any more.

|  |  |
| --- | --- |
| HTML | |
| 1 | <div contenteditable=true>hello</div> |
| JavaScript | |
| 1  2 | var div = document.querySelector('div');  div.removeAttribute('contenteditable'); |