Frontend Development II

Document Object Model (DOM) Part B

The Document Object Model (DOM) specifies how browsers should create a model of an HTML page and how JavaScript can access and update the contents of a web page while it is in the browser window.

The DOM is neither part of HTML, nor part of JavaScript; it is a separate set of rules. It is implemented by all major browser makers, and covers two primary areas:

MAKING A MODEL OF THE HTML PAGE

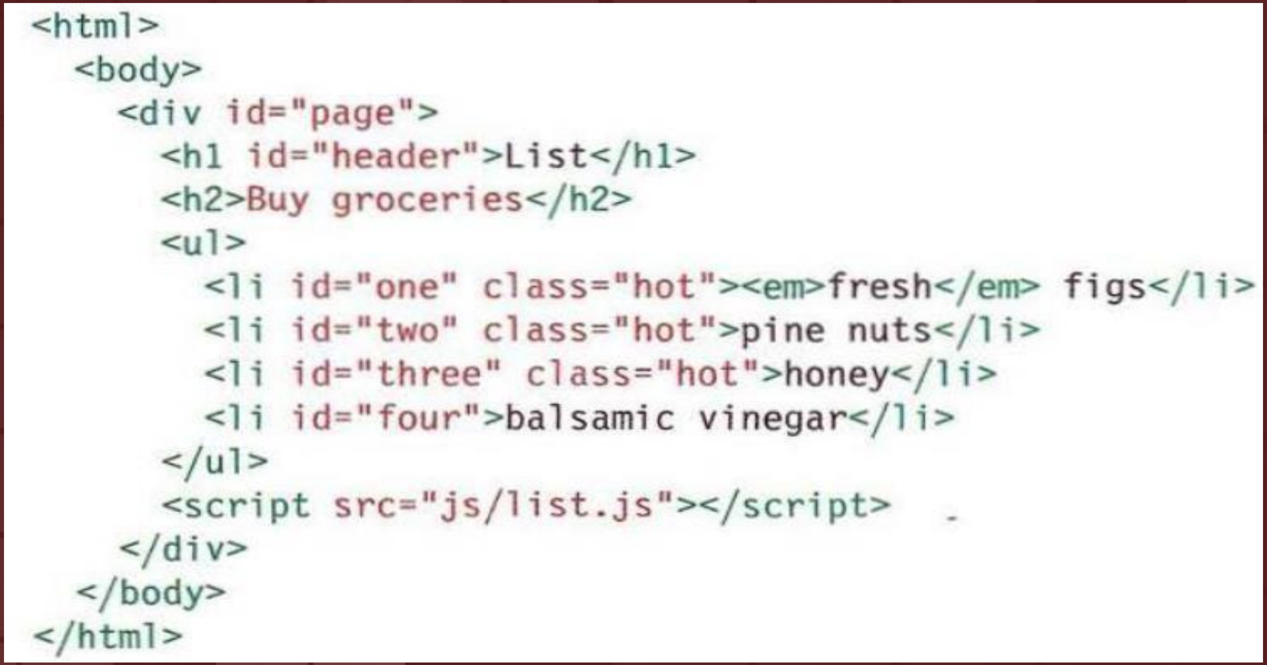
When the browser loads a web page, it creates a model of the page in memory. The DOM specifies the way in which the browser should structure this model using a DOM tree .

The DOM is called an object model because the model (t he DOM tree) is made of objects. Each object represents a different part of the page loaded in the browser window.

ACCESSING AND CHANGING THE HTML PAGE

The DOM also define s methods and properties to access and up date each object in this model, which in turn updates what the user sees in the browser.

As a browser loads a web page, it creates a model of that page. The model is called a DOM tree, and it is stored in the browsers' memory. It consists of four main types of nodes.



```
<html>
  <body>
    <div id="page">
      <h1 id="header">List</h1>
      <h2>Buy groceries</h2>
      <ul>
        <li id="one" class="hot"><em>fresh</em> figs</li>
        <li id="two" class="hot">pine nuts</li>
        <li id="three" class="hot">honey</li>
        <li id="four">balsamic vinegar</li>
      </ul>
      <script src="js/list.js"></script>
    </div>
  </body>
</html>
```

THE DOCUMENT NODE

Above, you can see the HTM L code for a shopping list, and on the right h an d page is its DOM tree. Every element, attribute, and piece of text in the HTML is represented by its own DOM node.

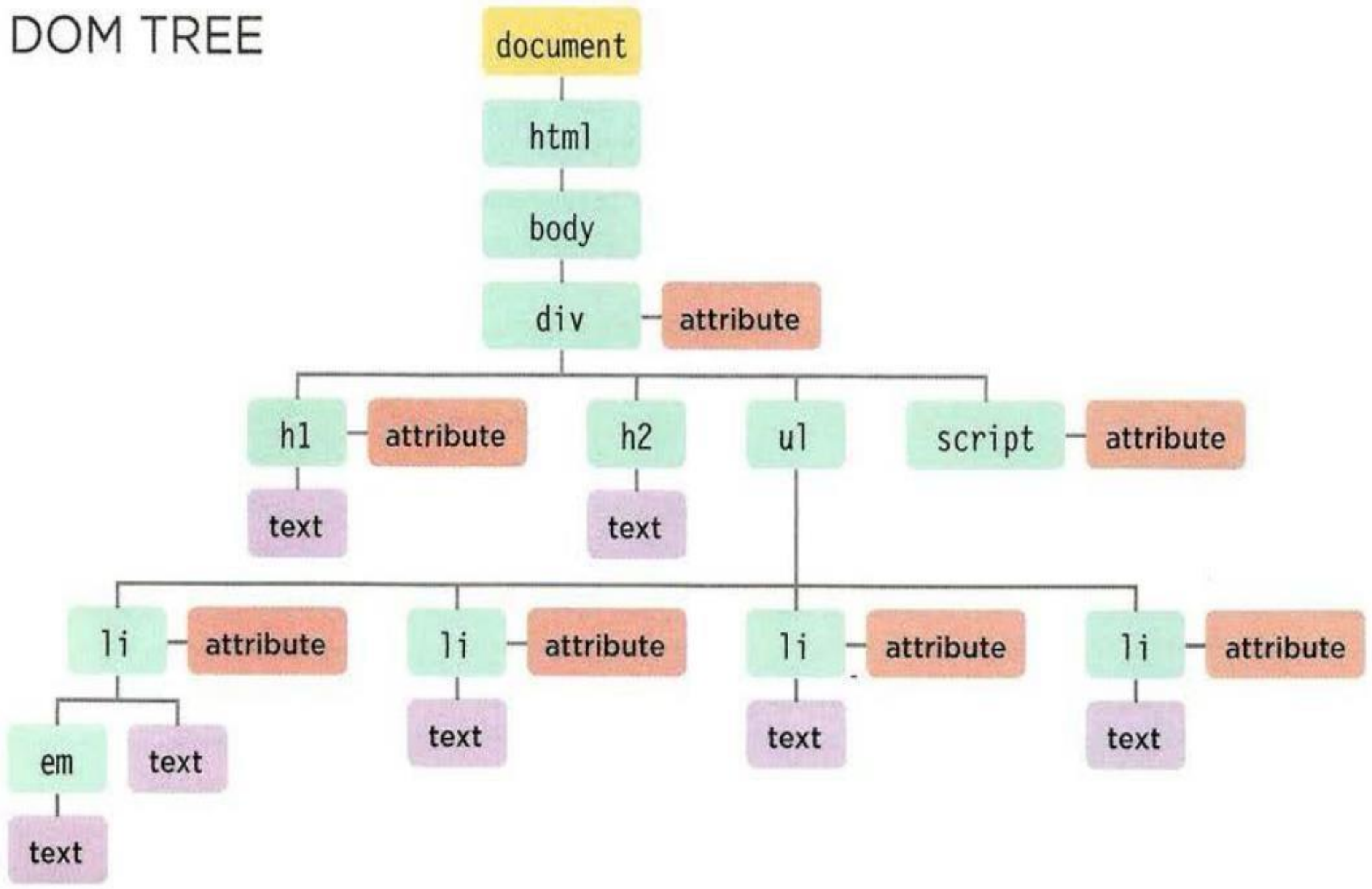
At the top of the tree a document nod e is added; it represents the entire page

When you access any element, attribute , or text node, you navigate to it via the document node. It is the starting point for all visits to the DOM tree.

ELEMENT NODES

HTML elements describe the structure of an HTML page. (The <h l > - <h6> elements describe what parts are headings; the <p> tags indicate where paragraphs of text start and finish; and so on.) To access the DOM tree, you start by looking for elements. Once you find the element you want , then you can access it s text and attribute nodes if you want to. This is why you start by learning methods that allow you to access element nodes, before learning to access and alter text or attributes.

DOM TREE


document
html
body
div
attribute
h1
attribute
h2
ul
script
attribute
text
text
li
attribute
li
attribute
li
attribute
li
attribute
em
text
text
text
text
text

WORKING WITH THE DOM TREE

Accessing and updating the DOM tree involves two steps:

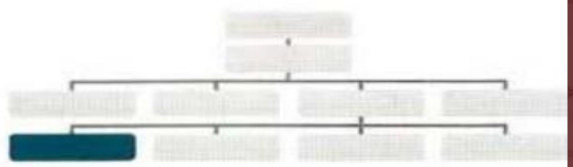
1: Locate the node that represents the element you want to work with.

2: Use its text content, child elements, and attributes.

STEP 1: ACCESS THE ELEMENTS

The first two columns are known as DOM queries. The last column is known as traversing the DOM

SELECT AN INDIVIDUAL ELEMENT NODE

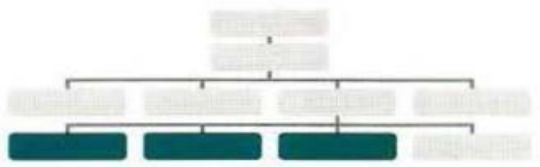


Here are three common ways to select an individual element:

`getElementById()`

`querySelector()`

SELECT MULTIPLE ELEMENTS (NODELISTS)



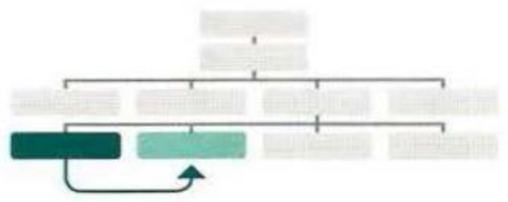
There are three common ways to select multiple elements.

`getElementsByClassName()`

`getElementsByTagName()`

`querySelectorAll()`

TRAVERSING BETWEEN ELEMENT NODES



You can move from one element node to a related element node.

`parentNode`

`previousSibling / nextSibling`

`firstChild / lastChild`

WORKING WITH THE DOM TREE

Accessing and updating the DOM tree involves two steps:

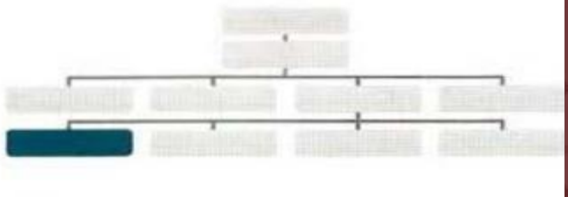
1: Locate the node that represents the element you want to work with.

2: Use its text content, child elements, and attributes.

STEP 1: ACCESS THE ELEMENTS

The first two columns are known as DOM queries. The last column is known as traversing the DOM

SELECT AN INDIVIDUAL ELEMENT NODE

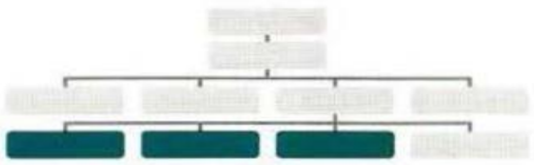


Here are three common ways to select an individual element:

`getElementById()`

`querySelector()`

SELECT MULTIPLE ELEMENTS (NODELISTS)



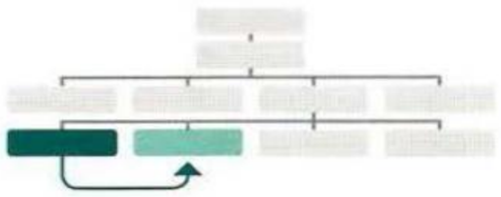
There are three common ways to select multiple elements.

`getElementsByClassName()`

`getElementsByTagName()`

`querySelectorAll()`

TRAVERSING BETWEEN ELEMENT NODES



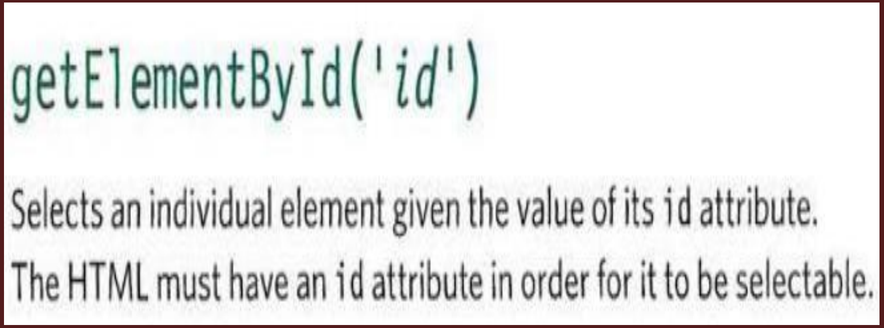
You can move from one element node to a related element node.

`parentNode`

`previousSibling / nextSibling`

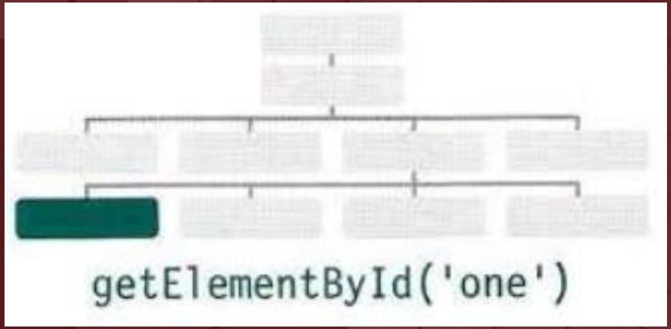
`firstChild / lastChild`

METHODS THAT RETURN A SINGLE ELEMENT NODE:

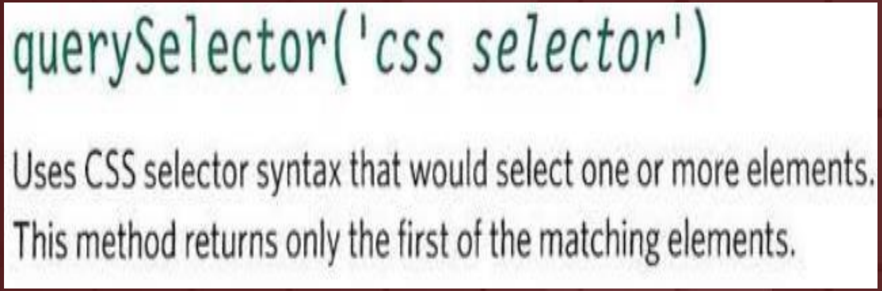


`getElementById('id')`

Selects an individual element given the value of its `id` attribute. The HTML must have an `id` attribute in order for it to be selectable.

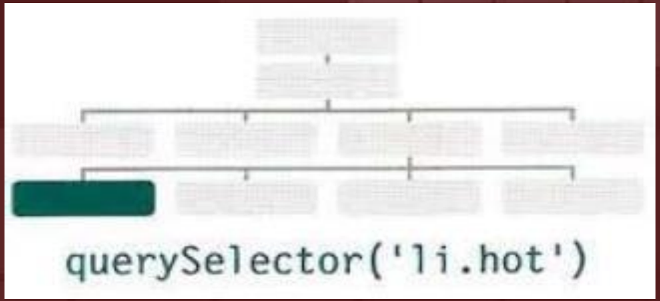


`getElementById('one')`



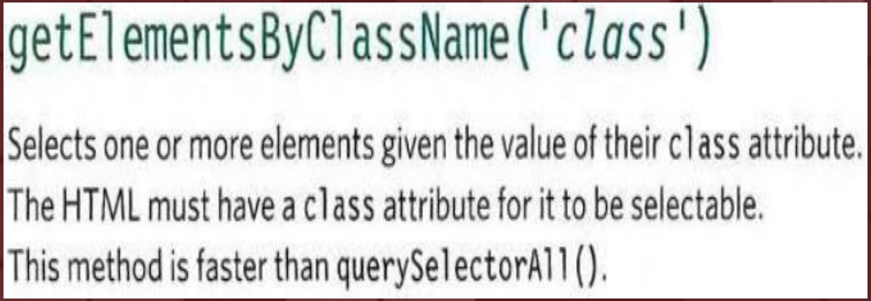
`querySelector('css selector')`

Uses CSS selector syntax that would select one or more elements. This method returns only the first of the matching elements.



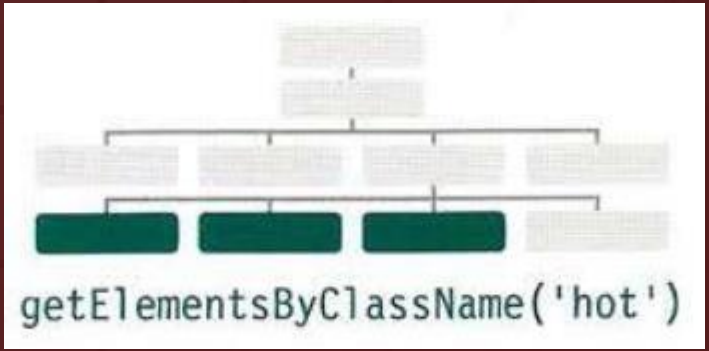
`querySelector('li.hot')`

ME THODS THAT RETURN ONE OR MORE ELEMENTS (AS A NODELIST)



`getElementsByClassName('class')`

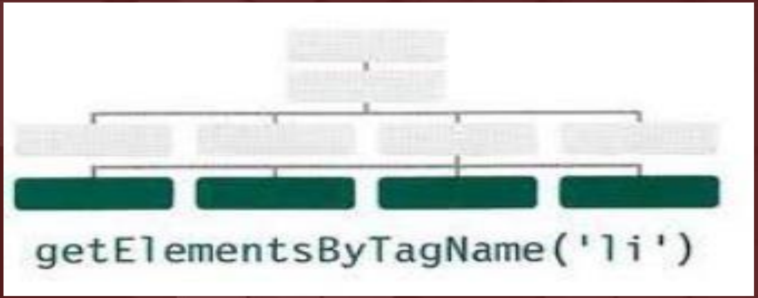
Selects one or more elements given the value of their `class` attribute. The HTML must have a `class` attribute for it to be selectable. This method is faster than `querySelectorAll()`.



`getElementsByClassName('hot')`

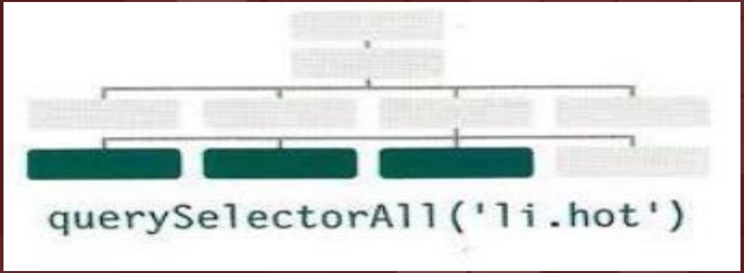
getElementsByTagName('*tagName*')

Selects all elements on the page with the specified tag name. This method is faster than querySelectorAll().



querySelectorAll('*css selector*')

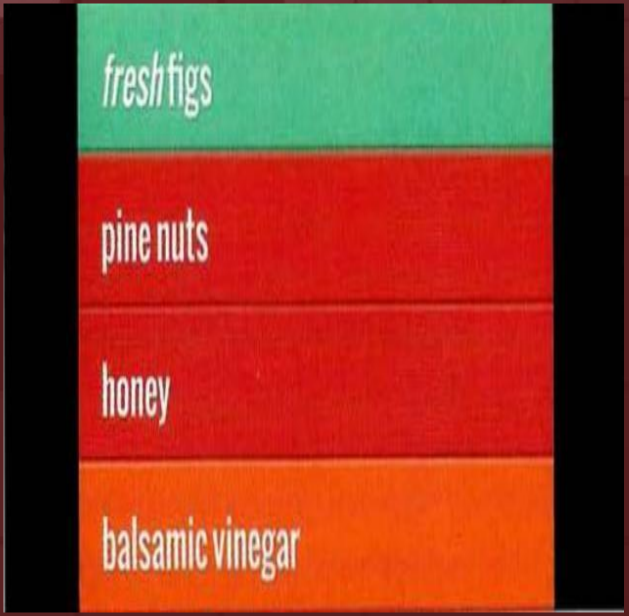
Uses CSS selector syntax to select one or more elements and returns all of those that match.



SELECTING ELEMENTS USING ID ATTRIBUTES

HTML:

```html
<h1 id="header">List King</h1>
<h2>Buy groceries</h2>
<ul>
  <li id="one" class="hot"><em>fresh</em>
    figs</li>
  <li id="two" class="hot">pine nuts</li>
  <li id="three" class="hot">honey</li>
  <li id="four">balsamic vinegar</li>
</ul>
```

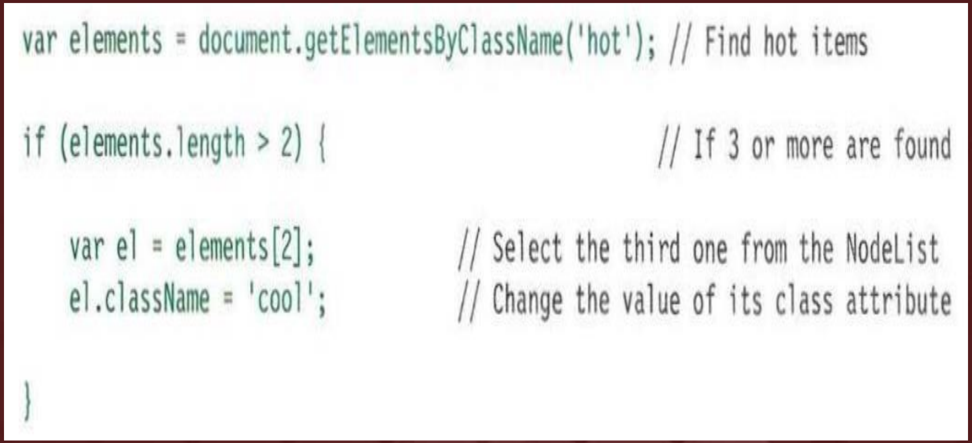


JavaScript

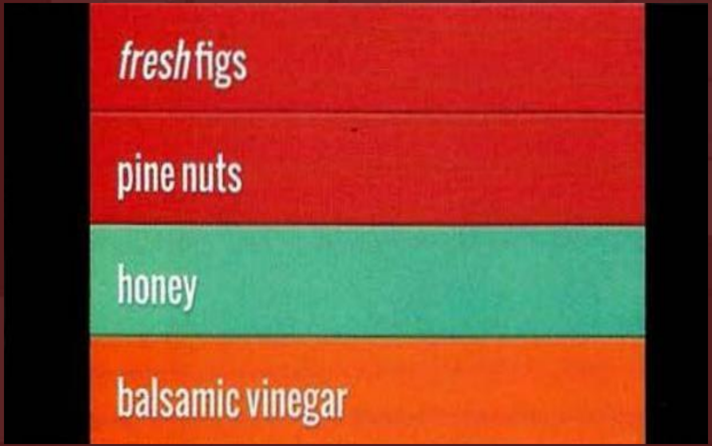
```javascript
// Select the element and store it in a variable.
var el = document.getElementById('one');

// Change the value of the class attribute.
el.className = 'cool';
```

NODELISTS: DOM QUERIES THAT RETURN MORE THAN ONE ELEMENT

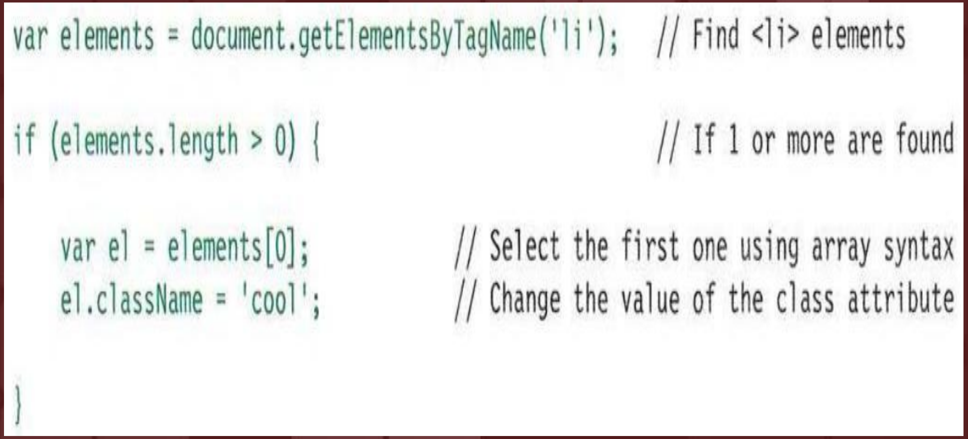
SELECTING ELEMENTS USING CLASS ATTRIBUTES



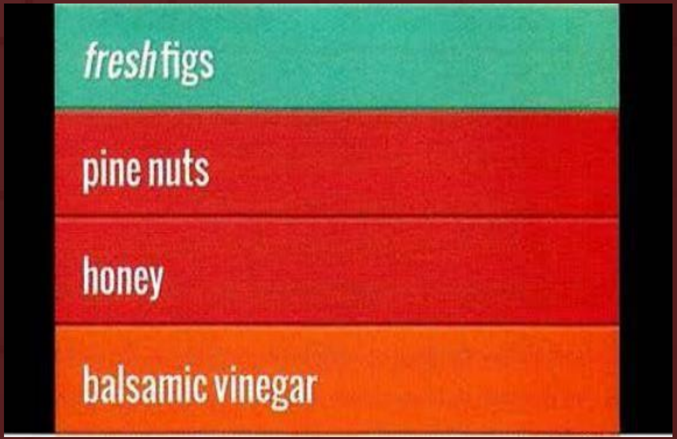
```
var elements = document.getElementsByClassName('hot'); // Find hot items

if (elements.length > 2) {                               // If 3 or more are found

    var el = elements[2];                  // Select the third one from the NodeList
    el.className = 'cool';                 // Change the value of its class attribute

}
```



SELECTING ELEMENTS BY TAG NAME

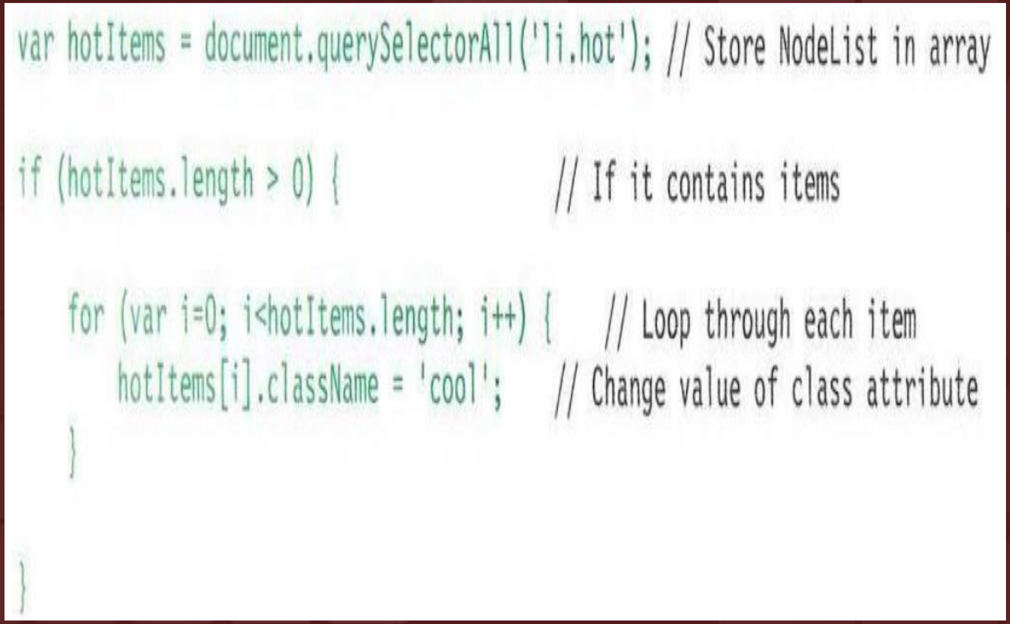


```
var elements = document.getElementsByTagName('li');   // Find <li> elements

if (elements.length > 0) {                               // If 1 or more are found

    var el = elements[0];                  // Select the first one using array syntax
    el.className = 'cool';                 // Change the value of the class attribute

}
```

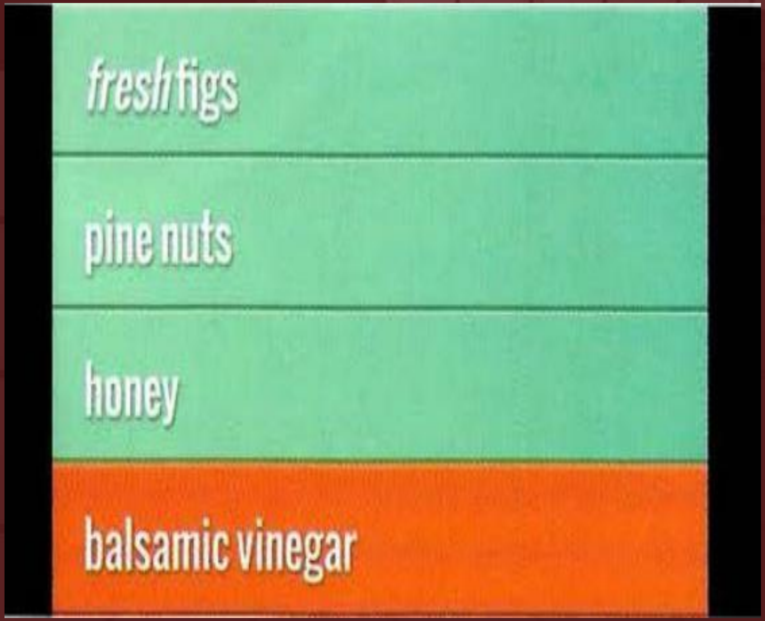


LOOPING THROUGH A NODELIST

If you want to apply the same code to numerous elements, looping through a Nodelist is a powerful technique. It involves finding out how many items are in the Nodelist, and then setting a counter to loop through them, one-by-one. Each time the loop runs, the script checks that the counter is less than the total number of items in the Nodelist.



```
var hotItems = document.querySelectorAll('li.hot'); // Store NodeList in array

if (hotItems.length > 0) {                        // If it contains items

  for (var i=0; i<hotItems.length; i++) {       // Loop through each item
    hotItems[i].className = 'cool';           // Change value of class attribute
  }

}
```



TRAVERSING THE DOM

When you have an element node, you can select another element in relation to it using these five properties. This is known as traversing t he DOM.

`parentNode`

This property finds the element node for the containing (or parent) element in the HTML.

(1) If you started with the first `<li>` element, then its *parent node* would be the one representing the `<ul>` element.

`previousSibling`
`nextSibling`

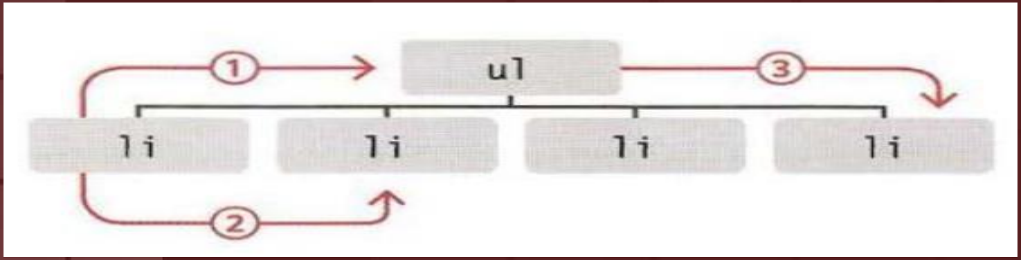
These properties find the previous or next sibling of a node if there are siblings.

If you started with the first `<li>` element, it would not have a *previous sibling*. However, its *next sibling* (2) would be the node representing the second `<li>`.

`firstChild`
`lastChild`

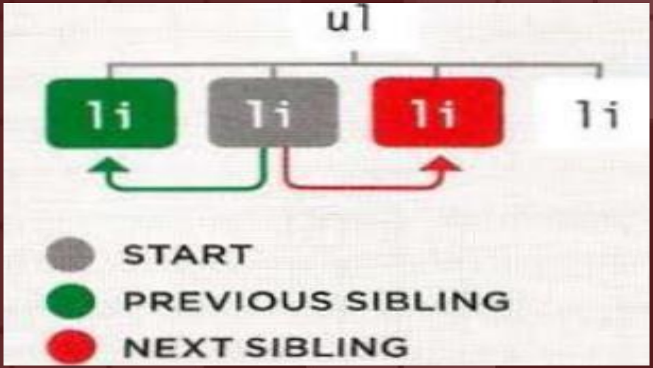
These properties find the first or last child of the current element.

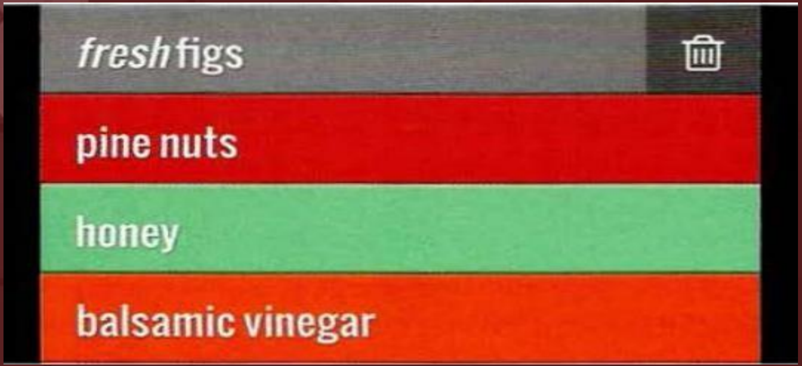
If you started with the `<ul>` element, the *first child* would be the node representing the first `<li>` element, and (3) the *last child* would be the last `<li>`.



```
<ul><li id="one" class="hot"><em>fresh</em> figs</li><li id="two"
class="hot">pine nuts</li><li id="three" class="hot">honey</li><li
id="four">balsamic vinegar</li></ul>
```

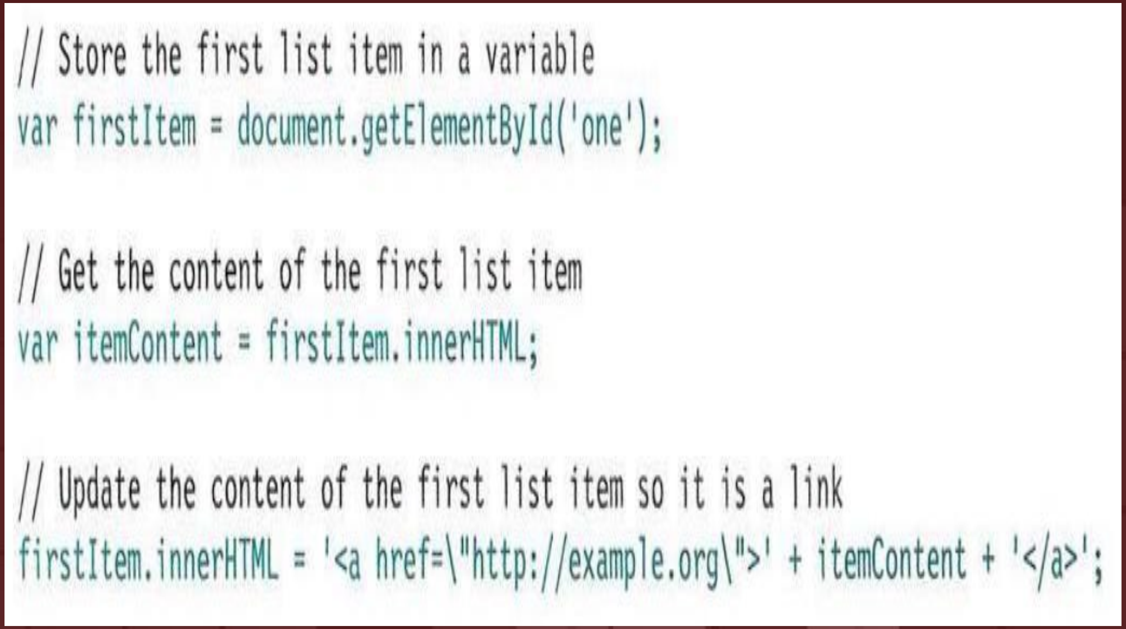
```
// Select the starting point and find its siblings
var startItem = document.getElementById('two');
var prevItem = startItem.previousSibling;
var nextItem = startItem.nextSibling;

// Change the values of the siblings' class attributes
prevItem.className = 'complete';
nextItem.className = 'cool';
```



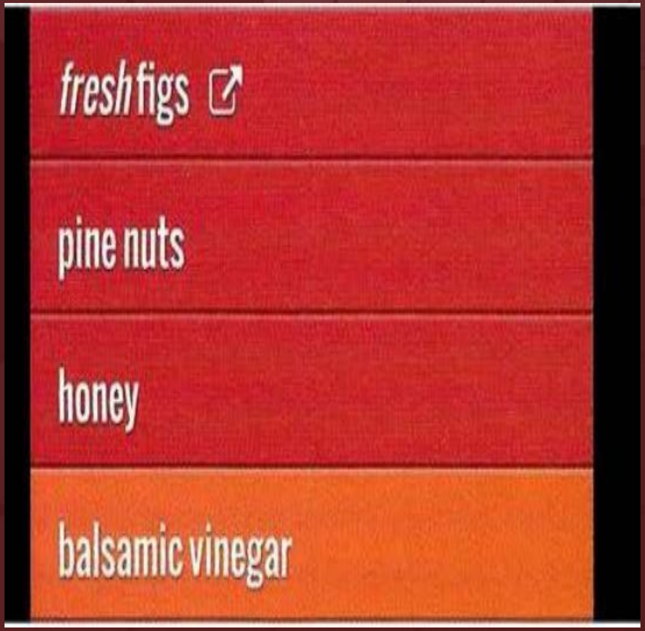


UPDATE TEXT & MARKUP

This example starts by storing the first list item in a variable called firstItem. It then retrieves the content of this list item and stores it in a variable called itemContent.



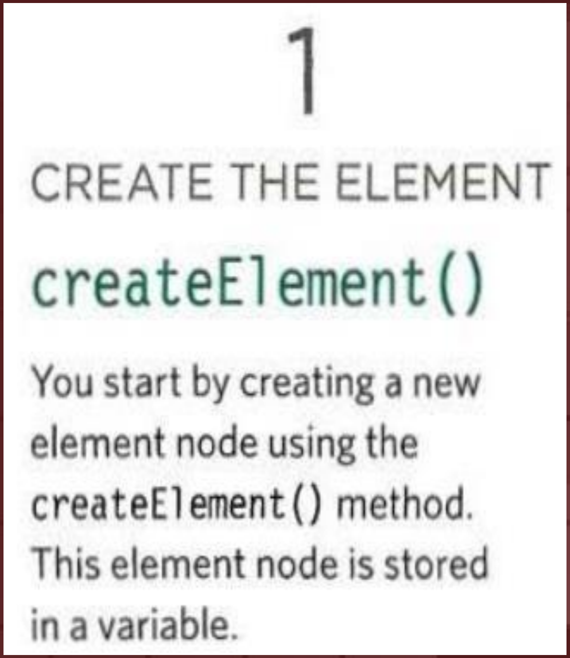
```
// Store the first list item in a variable
var firstItem = document.getElementById('one');

// Get the content of the first list item
var itemContent = firstItem.innerHTML;

// Update the content of the first list item so it is a link
firstItem.innerHTML = '<a href=\"http://example.org\">' + itemContent + '</a>';
```



ADDING ELEMENTS USING DOM MANIPULAT ION

DOM manipulation offers another technique to add new content to a page (rather than innerHTML).

It involves three steps:

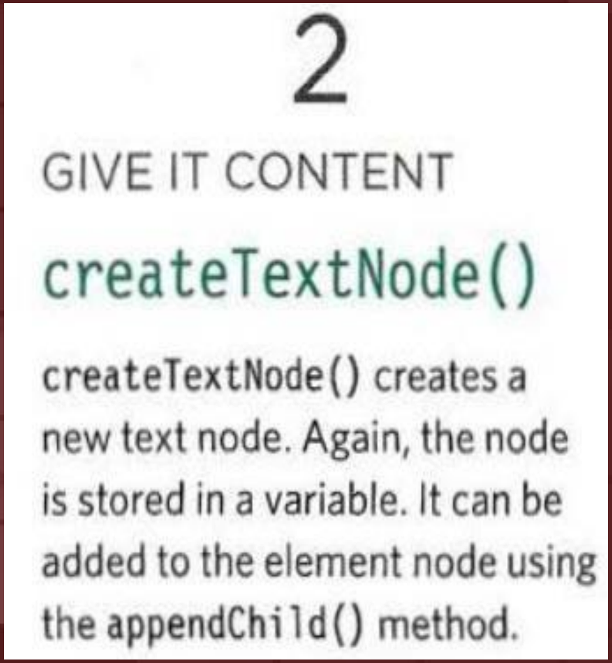


1

CREATE THE ELEMENT

`createElement()`

You start by creating a new element node using the `createElement()` method. This element node is stored in a variable.

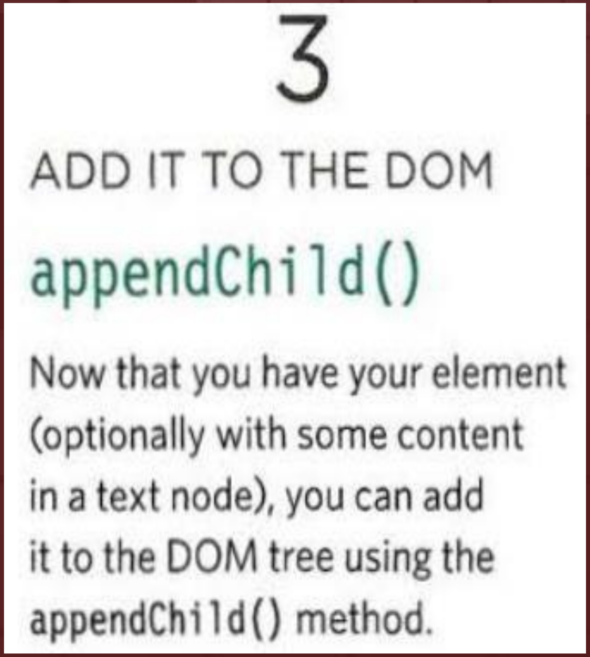


2

GIVE IT CONTENT

`createTextNode()`

`createTextNode()` creates a new text node. Again, the node is stored in a variable. It can be added to the element node using the `appendChild()` method.



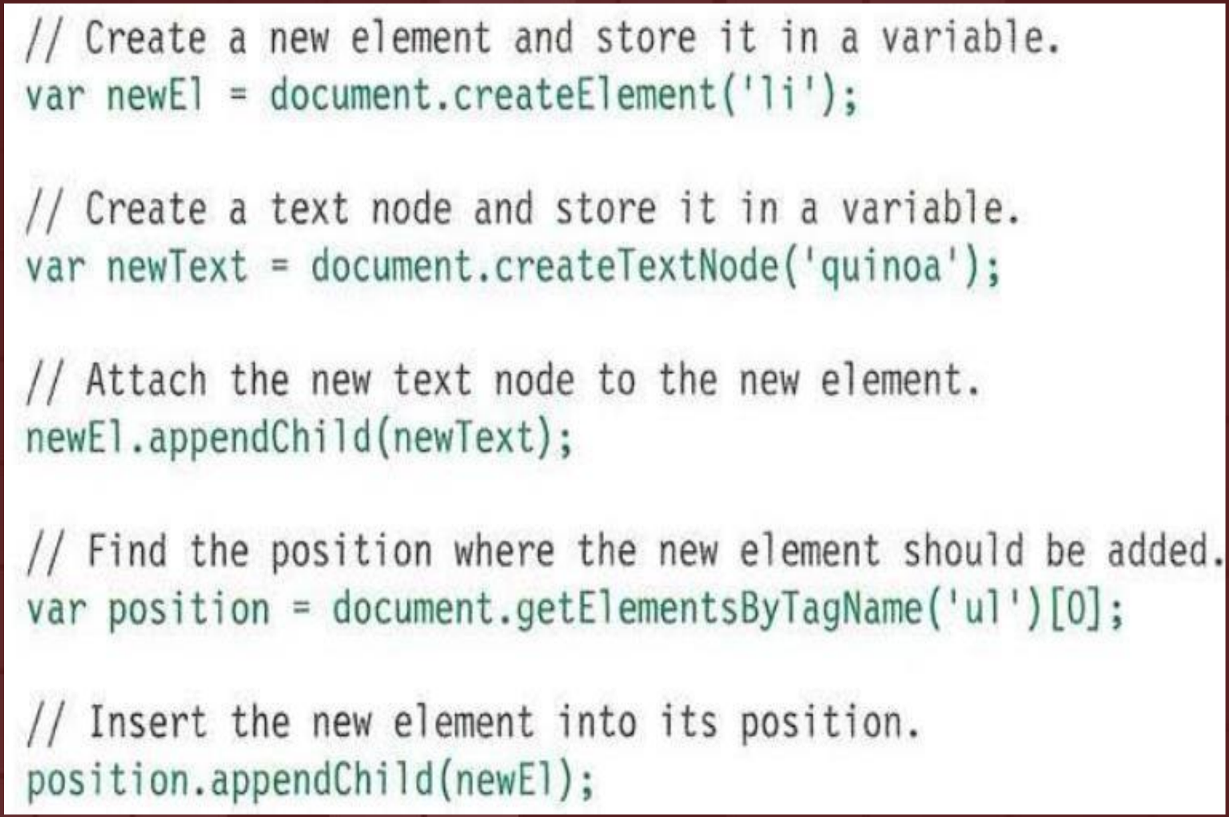
3

ADD IT TO THE DOM

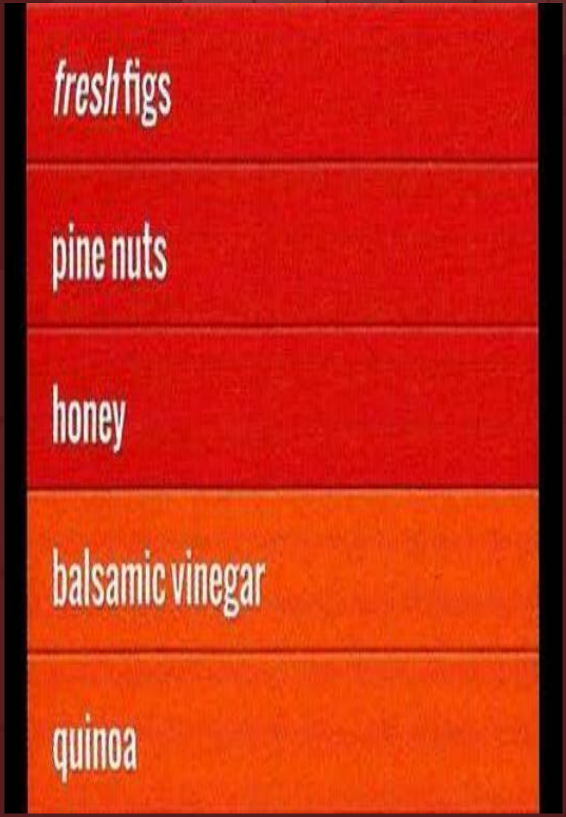
`appendChild()`

Now that you have your element (optionally with some content in a text node), you can add it to the DOM tree using the `appendChild()` method.

createElement () creates an element that can be added to the DOM tree, in this case an empty <li >element for the list. This new element is stored inside a variable called newel until it is attached to the DOM tree later on. createTextNode() allows you to create a new text node to attach to an element. It is stored in a variable called newText. The text node is added to the new element node using appendChild().

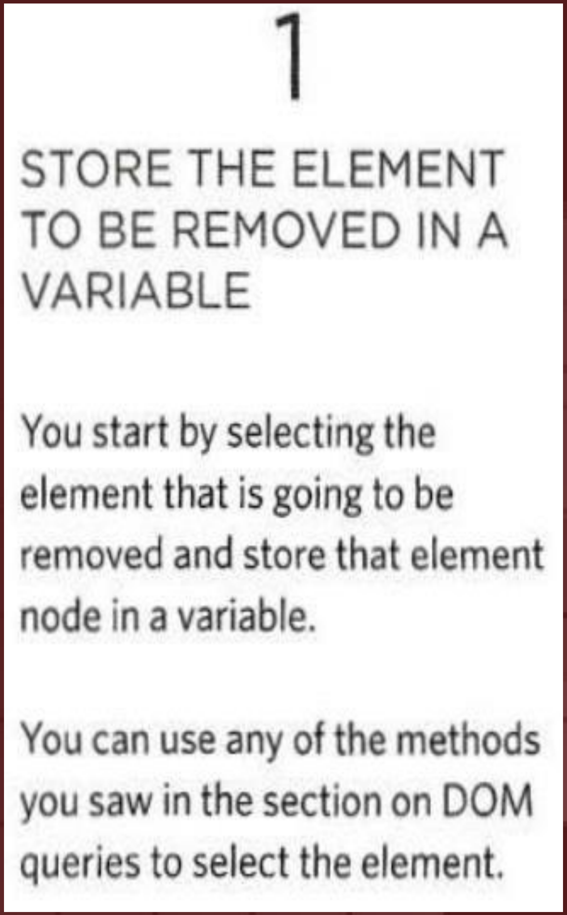


```
// Create a new element and store it in a variable.
var newEl = document.createElement('li');

// Create a text node and store it in a variable.
var newText = document.createTextNode('quinoa');

// Attach the new text node to the new element.
newEl.appendChild(newText);

// Find the position where the new element should be added.
var position = document.getElementsByTagName('ul')[0];

// Insert the new element into its position.
position.appendChild(newEl);
```



REMOVING ELEMENTS VIA DOM MANIPULATION

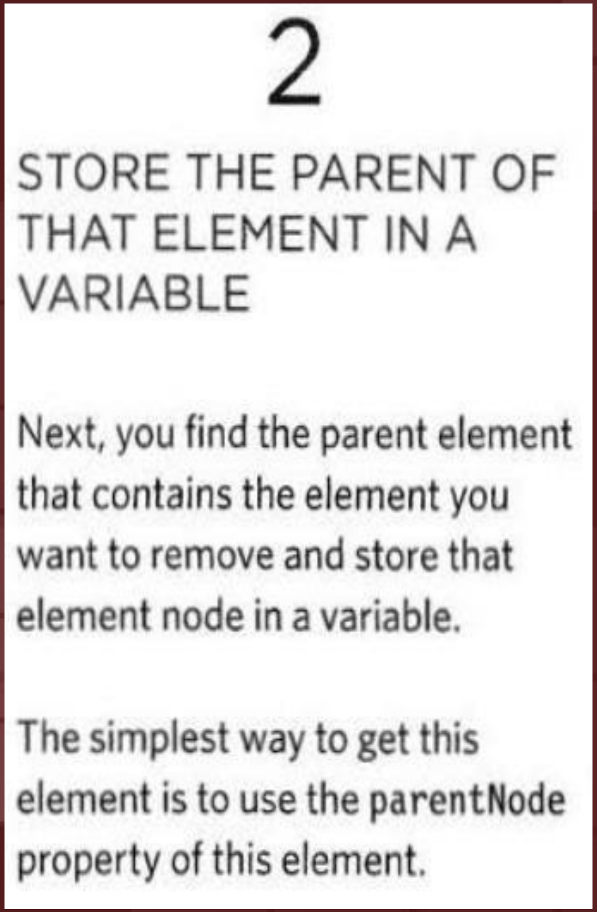
DOM manipulation can be used to remove elements from the DOM tree.


1

STORE THE ELEMENT TO BE REMOVED IN A VARIABLE

You start by selecting the element that is going to be removed and store that element node in a variable.

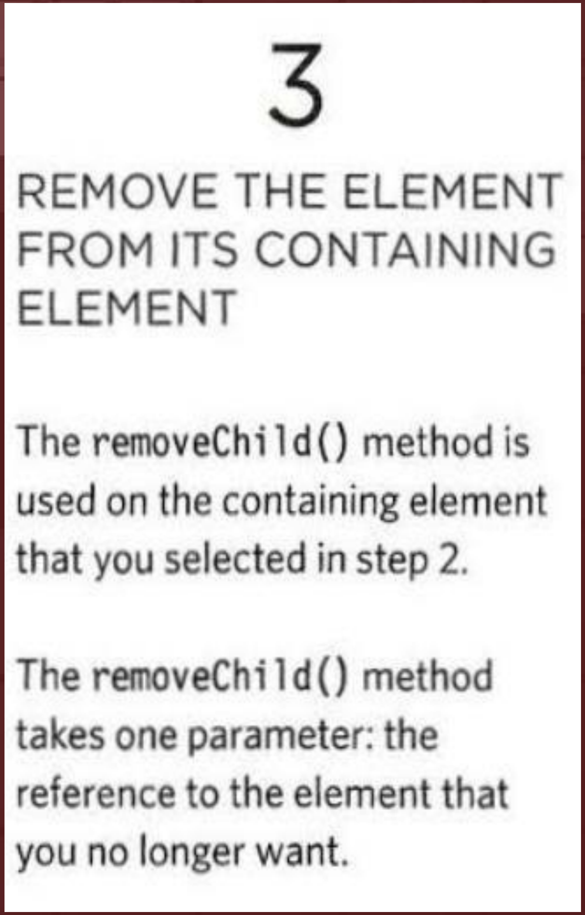
You can use any of the methods you saw in the section on DOM queries to select the element.


2

STORE THE PARENT OF THAT ELEMENT IN A VARIABLE

Next, you find the parent element that contains the element you want to remove and store that element node in a variable.

The simplest way to get this element is to use the `parentNode` property of this element.

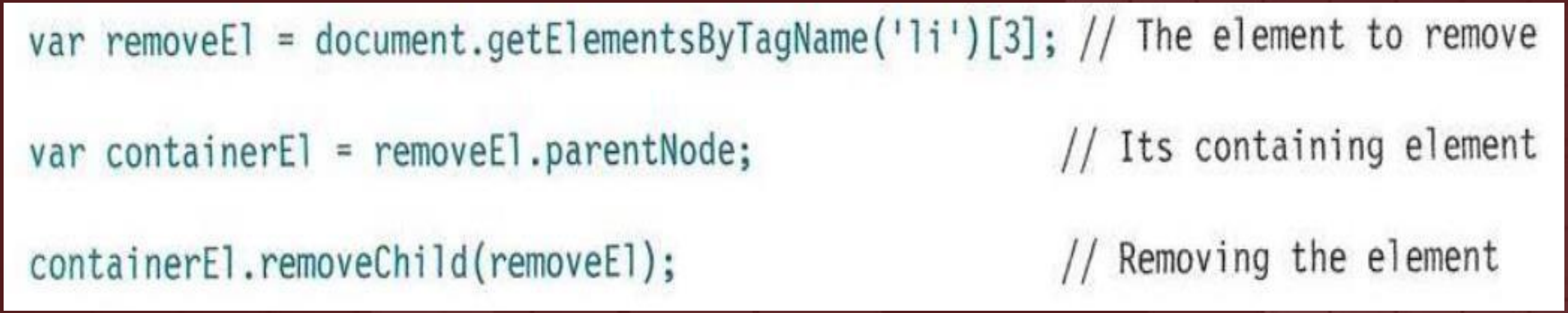

3

REMOVE THE ELEMENT FROM ITS CONTAINING ELEMENT

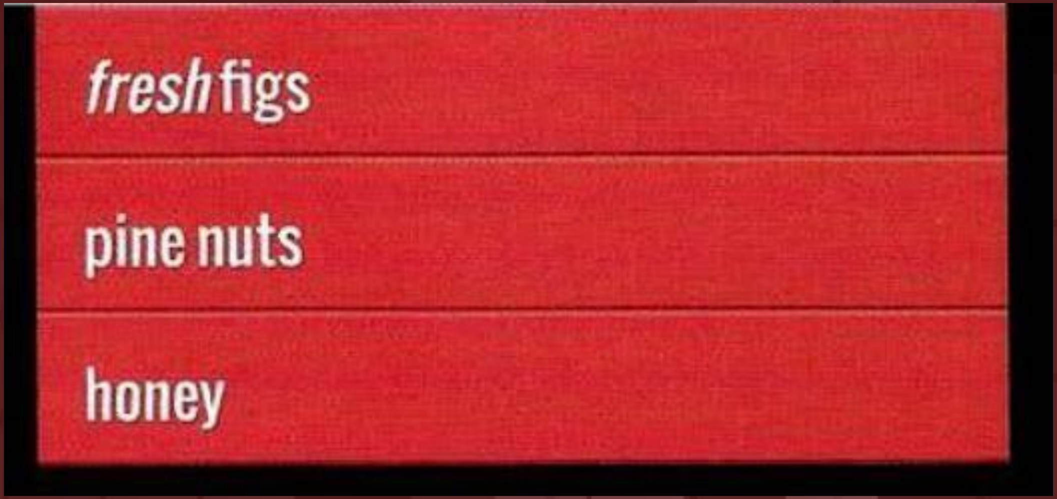
The `removeChild()` method is used on the containing element that you selected in step 2.

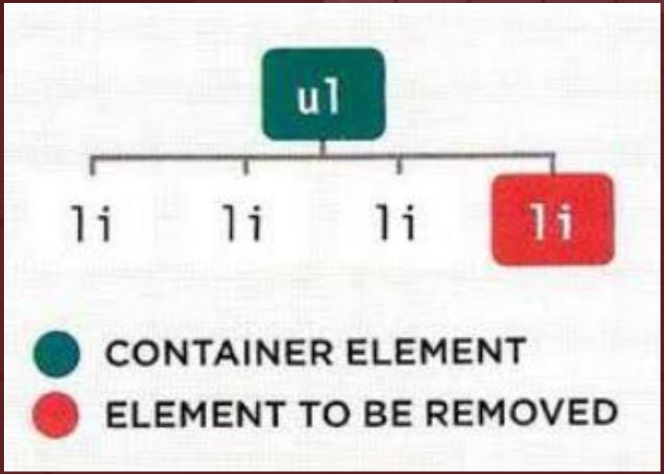
The `removeChild()` method takes one parameter: the reference to the element that you no longer want.

This example uses the removeChild() method to remove the fourth item from the list (along with its contents).



```
var removeEl = document.getElementsByTagName('li')[3]; // The element to remove
var containerEl = removeEl.parentNode;                 // Its containing element
containerEl.removeChild(removeEl);                     // Removing the element
```





The removeChild() method is used on the variable that holds the container node.

Next Week : jQuery

Questions ?