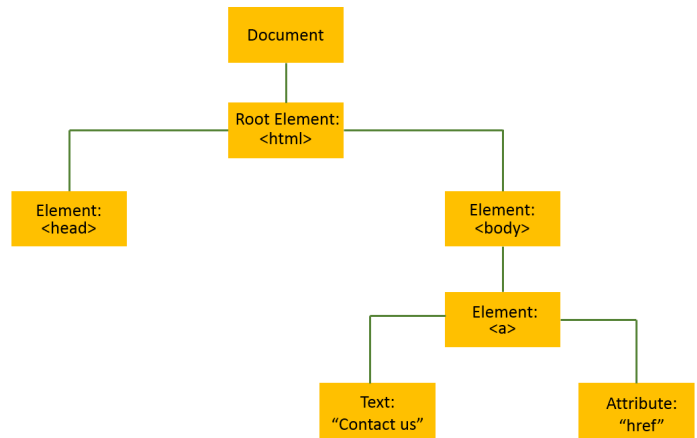


# JavaScript and the DOM

## Nodes in DOM tree

A *node* in the DOM tree is the intersection of two branches containing data. Nodes can represent HTML elements, text, attributes, etc. The *root node* is the top-most node of the tree. The illustration shows a representation of a DOM containing different types of node.



## HTML DOM

The DOM is an interface between scripting languages and a web page's structure. The browser creates a Document Object Model or DOM for each of the webpage it renders. The DOM allows scripting languages to access and modify a web page. With the help of DOM, JavaScript gets the ability to create dynamic HTML.

## Accessing HTML attributes in DOM

The DOM nodes of type Element allow access to the same attributes available to HTML elements. For instance, for the given HTML element, the `id` attribute will be accessible through the DOM.

```
<h1 id="heading">Welcome!</h1>
```

## The Document Object Model

The *Document Object Model*, or DOM is a representation of a document (like an HTML page) as a group of objects. While it is often used to represent HTML documents, and most web browsers use JavaScript interfaces to the DOM, it is language agnostic as a model.

The DOM is tree-like and heirarchical, meaning that there is a single top-level object, and other objects descend from it in a branching structure.

## DOM parent-child relationship

The parent-child relationship observed in the DOM is reflected in the HTML nesting syntax.

Elements that are nested inside the opening and closing tag of another element are the children of that element in the DOM.

In the code block, the two `<p>` tags are children of the `<body>`, and the `<body>` is the parent of both `<p>` tags.

## DOM Element Attributes

DOM Nodes of type Element provide access to attributes of those elements declared in HTML markup. For instance, if an HTML tag has attributes like `id` or `class` or `src`, those attributes can be accessed through the DOM.

```
<body>
  <p>first child</p>
  <p>second child</p>
</body>
```

```
<p id="intro">Welcome to my website</p>
<script>
  const paragraph =
document.querySelector('p');
  console.log(paragraph.id) // "intro"
</script>
```

## The element.parentNode Property

The `.parentNode` property of an element can be used to return a reference to its direct parent node. `.parentNode` can be used on any node.

```
<div id="parent">
  <p id="first-child">Some child text</p>
  <p id="second-child">Some more child
text</p>
</div>
<script>
  const firstChild =
document.getElementById('first-child');
  firstChild.parentNode; // reference to
the parent element
</script>
```

## The JavaScript document.createElement() Method

The `document.createElement()` method creates and returns a reference to a new Element Node with the specified tag name.

`document.createElement()` does not actually add the new element to the DOM, it must be attached with a method such as `element.appendChild()`.

```
const newButton
= document.createElement("button");
```

The `element.innerHTML` property can be used to access the HTML markup that makes up an element's contents.

`element.innerHTML` can be used to access the current value of an element's contents or to reassign them.

```
<box>
  <p>Hello there!</p>
</box>

<script>
  const box = document.querySelector('p');
  // Outputs '<p>Hello there!</p>':
  console.log(box.innerHTML)
  // Reassigns the value:
  box.innerHTML = '<p>Goodbye</p>'
</script>
```

## JavaScript document.getElementById() Method

The `document.getElementById()` method returns the element that has the `id` attribute with the specified value.

`document.getElementById()` returns `null` if no elements with the specified ID exists.

An ID should be unique within a page. However, if more than one element with the specified ID exists, the

`.getElementById()` method returns the first element in the source code.

```
// To save a reference to the element with
id 'demo':
const demoElement
= document.getElementById('demo');
```

## The JavaScript .querySelector() Method

The `.querySelector()` method selects the first child/descendant element that matches its selector argument.

It can be invoked on the `document` object to search the entire document or on a single element instance to search that element's descendants.

```
// Select the first <div>
const firstDiv
= document.querySelector('div');

// Select the first .button element inside
.main-navigation
const navMenu
= document.getElementById('main-
navigation');
const firstButtonChild
= navMenu.querySelector('.button');
```

## The document.body Object

`document.body` returns a reference to the contents of the `<body>` HTML element of a document/HTML page. The `<body>` element contains all the visible contents of the page.

## JavaScript `element.onclick` property

The `element.onclick` property can be used to set a function to run when an element is clicked. For instance, the given code block will add an `<li>` element each time the element with ID `addItem` is clicked by the user.

```
let element
= document.getElementById('addItem');
element.onclick = function() {
  let newElement
= document.createElement('li');
  document.getElementById('list').appendCh
ild(newElement);
};
```

## JavaScript `element.appendChild()` method.

The `parentElement.appendChild(childElement)` method appends an element as the last child of the parent. In the given code block, a newly created `<li>` element will be appended as the last child of the HTML element with the ID `list`.

```
var node1 = document.createElement('li');
document.getElementById('list').appendChil
d(node1);
```

## Changing CSS in Javascript with `element.style`.

The `element.style` property can be used to access or set the CSS style rules of an element. To do so, values are assigned to the attributes of `element.style`. In the example code, `blueElement` contains the HTML elements with the ID `colorful-element`. By setting the `backgroundColor` attribute of the `style` property to blue, the CSS property `background-color` becomes blue. Also note that, if the CSS property contains a hyphen, such as `font-family` or `background-color`, Camel Case notation is used in Javascript for the attribute name, so `background-color` becomes `backgroundColor`.

```
let blueElement
= document.getElementById('colorful-
element');
blueElement.style.backgroundColor
= 'blue';
```