DOM
(DOCUMENT OBJECT MODEL)

by : PNA & YK

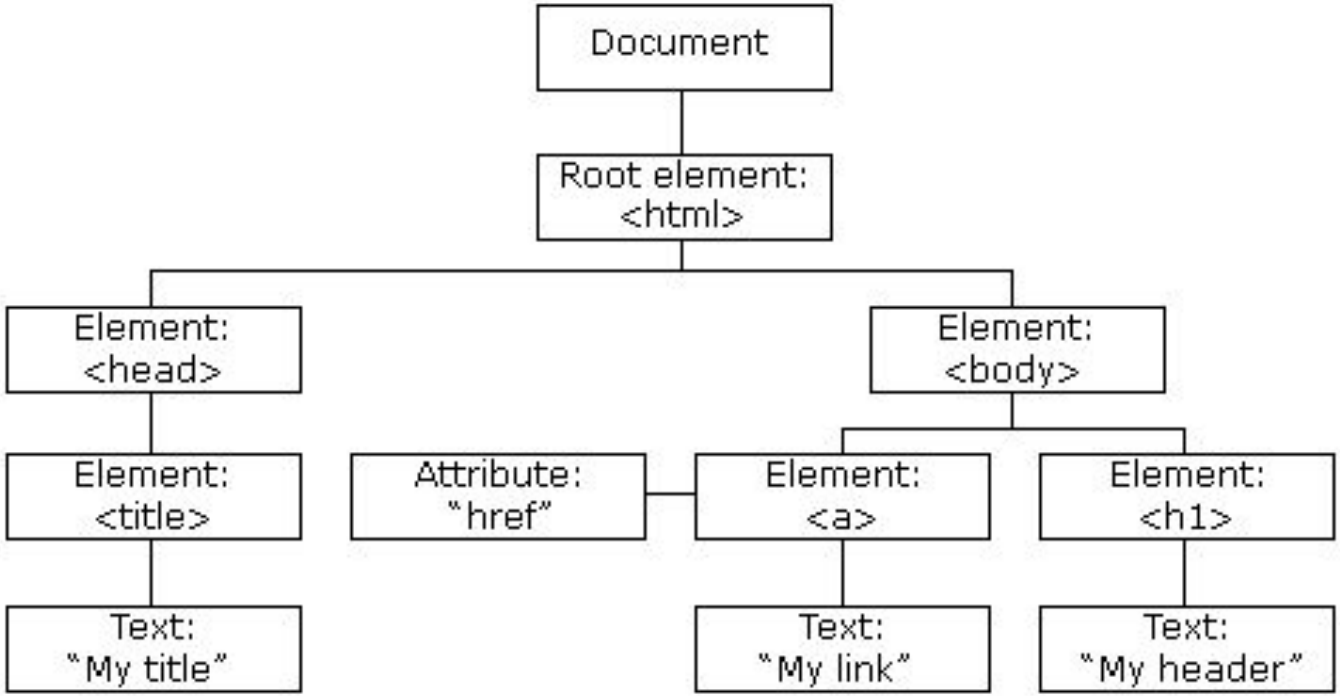
Batam, 27 January 2018

WHAT IS DOM ?

From CDN

The Document Object Model (DOM) is a programming interface for HTML, XML and SVG documents. It provides a structured representation of the document as a tree. The DOM defines methods that allow access to the tree, so that they can change the document structure, style and content. The DOM provides a representation of the document as a structured group of nodes and objects, possessing various properties and methods. Nodes can also have event handlers attached to them, and once an event is triggered, the event handlers get executed. **Essentially, it connects web pages to scripts or programming languages**.

DOM TREE


Document
Root element:
<html>
Element:
<head>
Element:
<body>
Element:
<title>
Attribute:
"href"
Element:
<a>
Element:
<h1>
Text:
"My title"
Text:
"My link"
Text:
"My header"

NODE RELATIONSHIP

The terms parent, child, and sibling are used to describe the relationships.

- In a node tree, the top node is called the root (or root node)
- Every node has exactly one parent, except the root (which has no parent)
- A node can have a number of children
- Siblings (brothers or sisters) are nodes with the same parent

EXAMPLE

```
<html>
  <head>
    <title>DOM Tutorial</title>
  </head>
  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>
</html>
```

- `<html>` is the root node
- `<html>` has no parents
- `<html>` is the parent of `<head>` and `<body>`
- `<head>` is the first child of `<html>`
- `<body>` is the last child of `<html>`

- `<head>` has one child: `<title>`
- `<title>` has one child (a text node): "DOM Tutorial"
- `<body>` has two children: `<h1>` and `<p>`
- `<h1>` has one child: "DOM Lesson one"
- `<p>` has one child: "Hello world!"
- `<h1>` and `<p>` are siblings

DOM SELECTION

1. getElementbyId() ---> element
2. getElementsbyTagname () ---> html collection
3. getElementsbyClassName () ---> html collection
4. querySelector () ---> element
5. querySelectorAll () ----> node list

GET ELEMENT BY ID

The getElementById() method returns the element that has the ID attribute with the specified value.

This method is one of the most common methods in the HTML DOM, and is used almost every time you want to manipulate, or get info from, an element on your document.

EXAMPLE --->

```
var container1 =
document.getElementById("container")
container1.style.backgroundColor = "red";
```

```
<h1 id="judul">Hello World</h1>
 <div id="container">
     <section id="a">
         <p class="p1">paragraf 1</p>
         <a
href="http://instagram.com/glints">Instagram
Glints</a>
         <p class="p2">paragraf 2</p>
         <p class="p3">paragraf 3</p>
         <ul>
           <li>ini item di section a </li>
         </ul>
     </section>
     <section id="b">
         <p>paragraf 4</p>
         <ul>
             <li>item 1</li>
             <li>item 2</li>
             <li>item 3</li>
         </ul>
     </section>
```

GET ELEMENT BY TAG NAME

The **`Element.getElementsByTagName()`** method returns a live `HTMLCollection` of elements with the given tag name. All descendants of the specified element are searched, but not the element itself.

```
var paragraf = document.getElementsByTagName("p")
//select all paragraf

var paragraf = document.getElementsByTagName("p")
var li = document.getElementsByTagName("li")
var div = document.getElementsByTagName("div")
var section = document.getElementsByTagName
("section")
```

GET ELEMENT BY CLASS

The getElementsByClassName() method returns a collection of all elements in the document with the specified class name, as a NodeList object.

```
var paragraf =
document.getElementsByClassName("p1");
var paragraf2 =
document.getElementsByClassName("p2");
paragraf[0].style.color ="blue";
```

QUERY SELECTOR

method **`querySelector()`** returns the first `Element` within the document that matches the specified selector, or group of selectors. If no matches are found, `null` is returned.

```
var pfourth = document.querySelector('#b p');
```

query selector All

To select multiple elements, we can use `getElementsByTagName` or `getElementsByClassName`, or we can use `querySelectorAll` and pass in a `CSS`selector. These will return what appear to be arrays (they are not **exactly** arrays, but for right now, that is not a problem).

```
var divs = document.querySelectorAll("div");
```

MODIFYING PROPERTIES AND ATTRIBUTES AND ELEMENTS IN DOM

We can change the text of an element through the `innerHTML` property.

```
var firstDiv = document.getElementsByTagName("div")[0];

firstDiv.innerHTML = "Just changed!";
```

This can also be done using the `innerText` property.

```
var secondDiv = document.getElementsByTagName("div")[1];

secondDiv.innerText = "Just changed Again!";
```

We can also directly manipulate the `CSS` properties for elements (through inline styling) with the `style` property.

```
var firstDiv = document.getElementsByTagName("div")[0];

firstDiv.style.color = "red";
firstDiv.style.backgroundColor = "teal";
```

Notice that if you're accessing CSS properties using dot notation, you need to camelCase those property names, since `firstDiv.style.background-color` is invalid JavaScript. (Bonus question: what do you think will happen if you try typing this in the console?) However, if you use brackets, you can write the properties the same way as you would in a stylesheet:

```
firstDiv.style["background-color"] = "purple"; // this works too
```

If we want to access/modify attributes on elements, we can do that with `getAttribute` and `setAttribute`:

```
var body = document.getElementById("container");

body.getAttribute("id"); // "container"
body.setAttribute("id", "new_container");
body.getAttribute("id"); // "new_container"
```

add and remove classes to elements using `classList`

```
var secondDiv =
document.getElementsByTagName("div")[1];

secondDiv.classList; // ["hello"]
secondDiv.classList.add("another_class");
secondDiv.classList; // ["hello",
"another_class"]
secondDiv.classList.remove("hello");
secondDiv.classList; // [another_class"]
```

CREATING ELEMENTS

Creating elements

To create elements we use the `.createElement` function on the `document` object and pass in a string with the name of the element that we would like to create. This will just return a new HTML element without any text/attributes or placement on the page!

```
var newDiv = document.createElement("div");
```

APPENDING ELEMENTS

So now that we created this element, how do we place it on the page?

Appending elements

```
var button = document.createElement("button");
button.innerText = "I am a button created with JavaScript!";

var container = document.getElementById("container");
container.appendChild(button);
```

given the following html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <div class="header">
    </div>
    <section id="container">
        <ul>
            <li class="first">one</li>
            <li class="second">two</li>
            <li class="third">three</li>
        </ul>
        <ol>
            <li class="first">one</li>
            <li class="second">two</li>
            <li class="third">three</li>
        </ol>
    </section>
    <div class="footer">
    </div>
</body>
</html>
```

Write the code necessary to do the following:

1. Select the `section` with an id of `container` without using `querySelector`.
2. Select the `section` with an id of `container` using `querySelector`.
3. Select all of the list items with a class of "second".
4. Select a list item with a class of third, but only the list item inside of the `ol` tag.
5. Add the class `main` to the `div` with a class of `footer`.
6. Remove the class `main` on the `div` with a class of `footer`.
7. Create a new `li` element.
8. Give the `li` the text "four".
9. Append the `li` to the `ul` element