Tutorial : Document Object Model

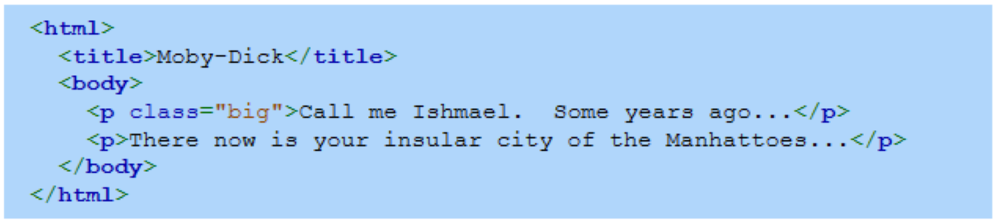
Document Object Model (DOM)

DOM structure

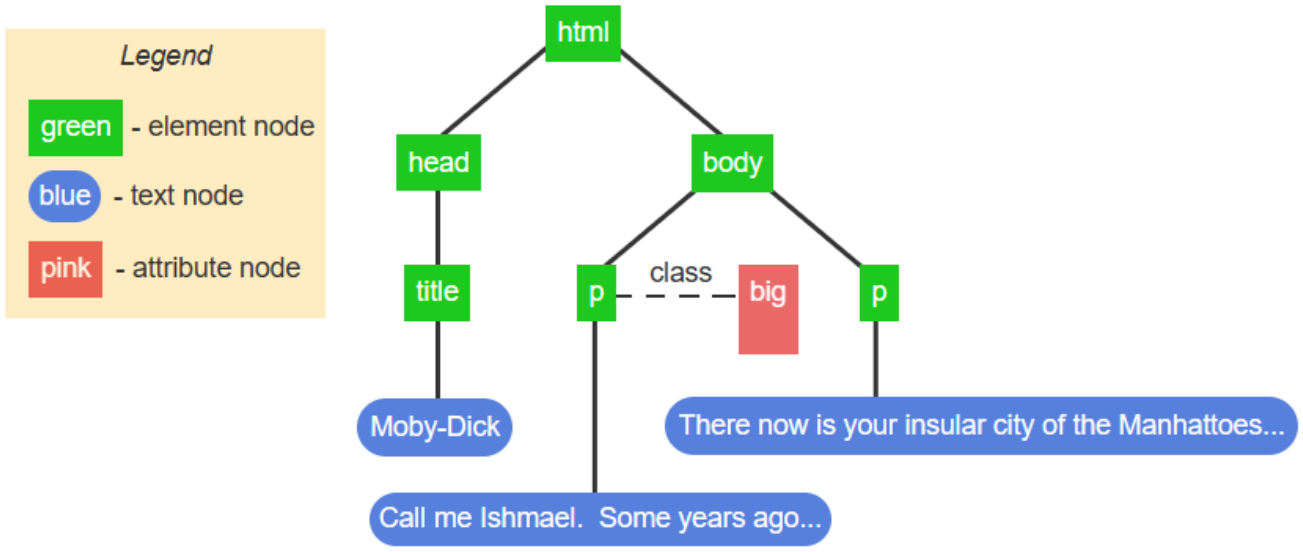
The Document Object Model (DOM) is a data structure corresponding to the HTML document displayed in a web browser. A DOM tree is a visualization of the DOM data structure. A node is an individual object in the DOM tree. Nodes are created for each element, the text between an element's tags, and the element's attributes.

- The root node is the node at the top of the DOM.
- A child node is the node directly under another node. A node can have zero, one, or more child nodes (children).
- A parent node is the node directly above another node. All nodes, except the root node, have one parent node.

Creating the DOM from HTML.



```
<html>
  <title>Moby-Dick</title>
  <body>
    <p class="big">Call me Ishmael.  Some years ago...</p>
    <p>There now is your insular city of the Manhattoes...</p>
  </body>
</html>
```



Explanation :

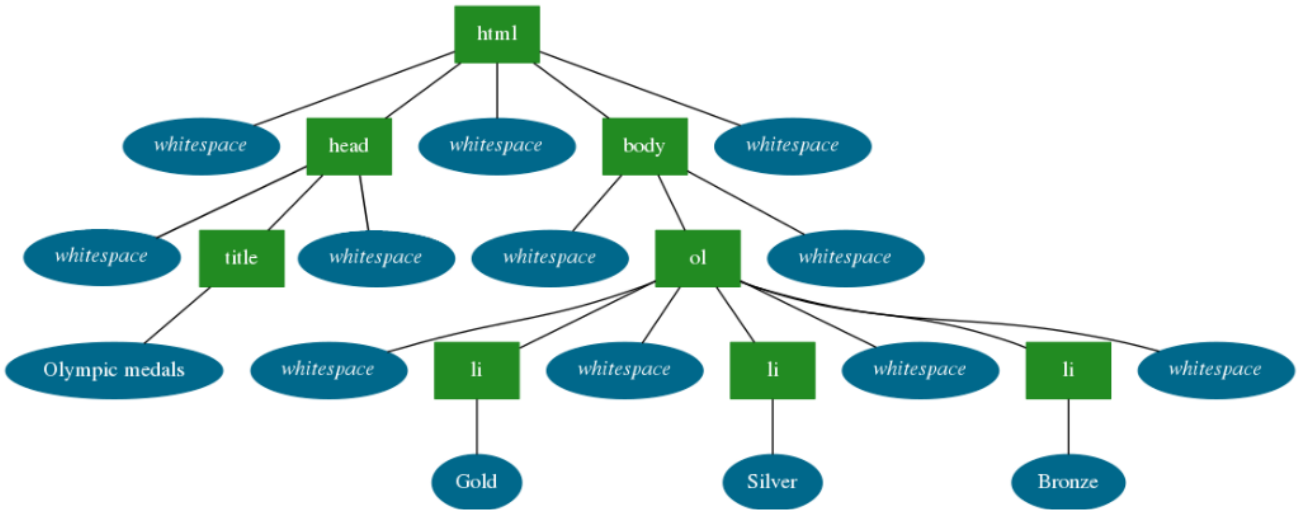
1. The web browser reads the HTML and creates the DOM's root node from the html element.
2. Although no head element exists in the HTML, a head node is created as a child of the html node. The title node is added as a child of the head node.
3. A text node is created for the title element's text content.
4. The body node is a child of the root node. The p element is contained within the body element, so the p node is a child of the body node.
5. An attribute node is created for the p element's class attribute. Attribute nodes are always connected to element nodes and are not considered children.
6. The browser continues reading the HTML and creating DOM nodes.

n idealized representation of the DOM tree excludes text nodes that only contain whitespace. However, a web developer occasionally needs to know the complete DOM tree, which includes whitespace as shown in the example below.

Figure 1: Complete DOM tree visualization with whitespace text nodes.

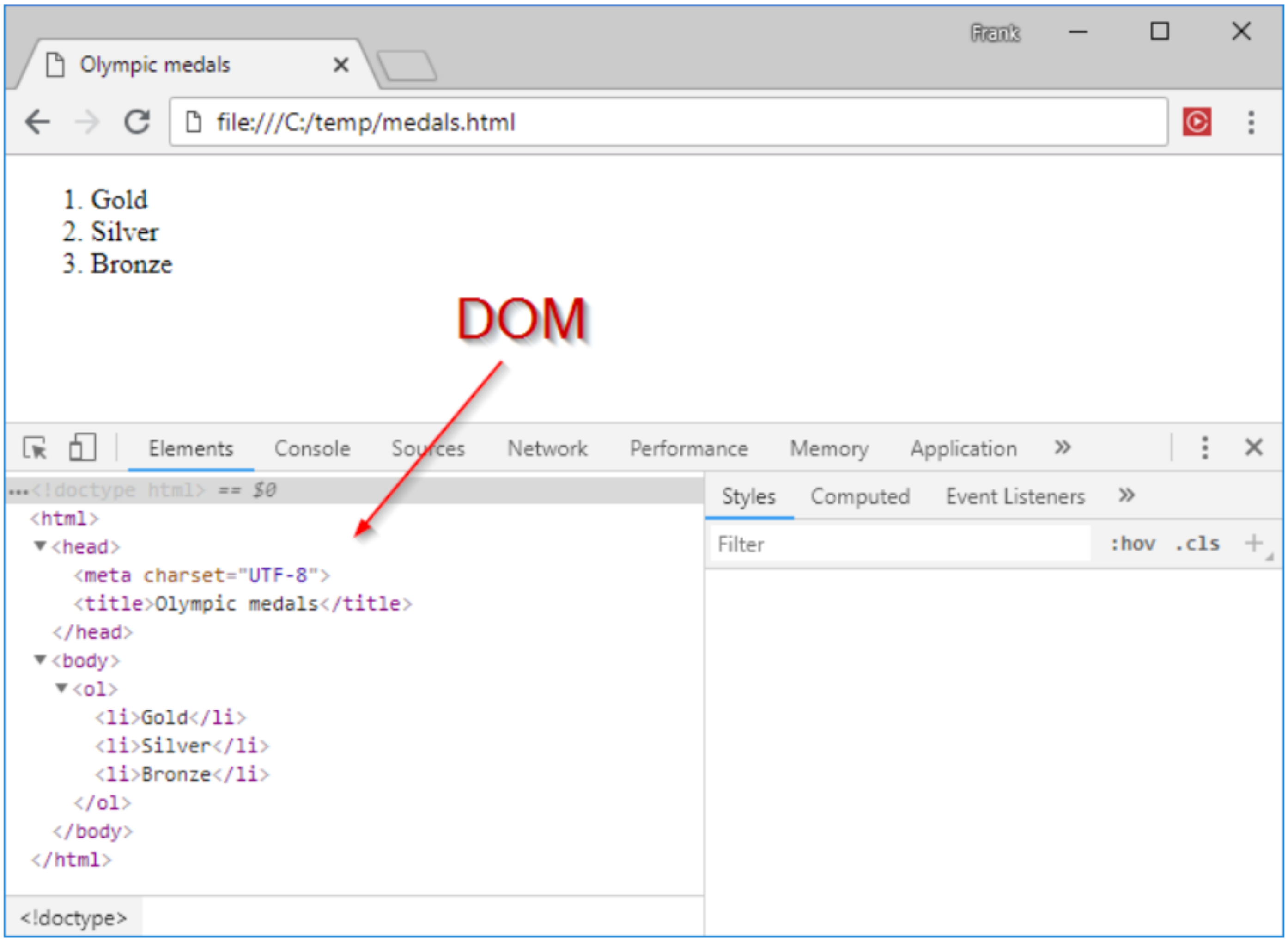
```
<!DOCTYPE html>
<html>
   <head>
      <title>Olympic medals</title>
   </head>
   <body>
```

```
        <ol>
           <li>Gold</li>
           <li>Silver</li>
           <li>Bronze</li>
        </ol>
     </body>
  </html>
```



Viewing the DOM in Chrome

The Chrome DevTools can display an HTML document's DOM by pressing Ctrl+Shift+C on Windows or Ctrl+Option+C on a Mac. The DOM may differ from the HTML. Ex: The `<head>` tag may be missing from the HTML file but is visible in the DOM below because `<meta>` and `<title>` elements are always placed in the `<head>` element.



```
...<!doctype html> == $0
  <html>
  ▼<head>
     <meta charset="UTF-8">
     <title>Olympic medals</title>
   </head>
  ▼<body>
   ▼<ol>
       <li>Gold</li>
       <li>Silver</li>
       <li>Bronze</li>
     </ol>
   </body>
  </html>
```

Searching the DOM

JavaScript is commonly used to search the DOM for a specific node or set of nodes and then change the nodes' attributes or content. Ex: In an email application, the user may click a Delete button to delete an email. The JavaScript must search the DOM for the node containing the email's contents and then change the contents to read "Email deleted".

The `document` object provides five primary methods that search the DOM for specific nodes:

1. The document.getElementById() method returns the DOM node whose `id` attribute is the same as the method's parameter.
   Ex: `document.getElementById("early_languages")` returns the p node in the HTML below.

2. The document.getElementsByTagName() method returns an array of all the DOM nodes whose type is the same as the method's parameter.
   Ex: `document.getElementsByTagName("li")` returns an array containing the four li nodes from in the HTML below.

3. The document.getElementsByClassName() method returns an array containing all the DOM nodes whose `class` attribute matches the method's parameter.
   Ex: `document.getElementsByClassName("traditional")` returns an array containing the `ol` node with the `class` attribute matching the word traditional.
4. The document.querySelectorAll() method returns an array containing all the DOM nodes that match the CSS selector passed as the method's parameter.
   Ex: `document.querySelectorAll("li a")` returns an array containing the two anchor nodes in the HTML below.
5. The document.querySelector() method returns the first element found in the DOM that matches the CSS selector passed as the method's parameter. `querySelector()` expects the same types of parameters as `querySelectorAll()` but only returns the first element found while navigating the DOM tree in a depth-first traversal.
   Ex: `document.querySelector("li")` returns the li node about Fortran.

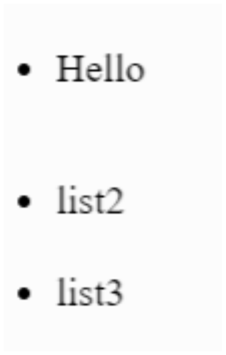
A DOM search method name indicates whether the method returns one node or an array of nodes. If the method name starts with "getElements" or ends in "All", then the method returns an array, even if the array contains one node or is empty. `getElementById()` and `querySelector()` either return a single node or null if no node matches the method arguments.

```
Example : Select Element by Id :
```

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width">
 <title>replit</title>
 <link href="style.css" rel="stylesheet" type="text/css" />
</head>

<body>
<body>
   <ul>
      <li id="myList">list1</li>
      <li>list2</li>
      <li>list3</li>
   </ul>
</body>
<script>
      let elm=document.getElementById("myList")
```

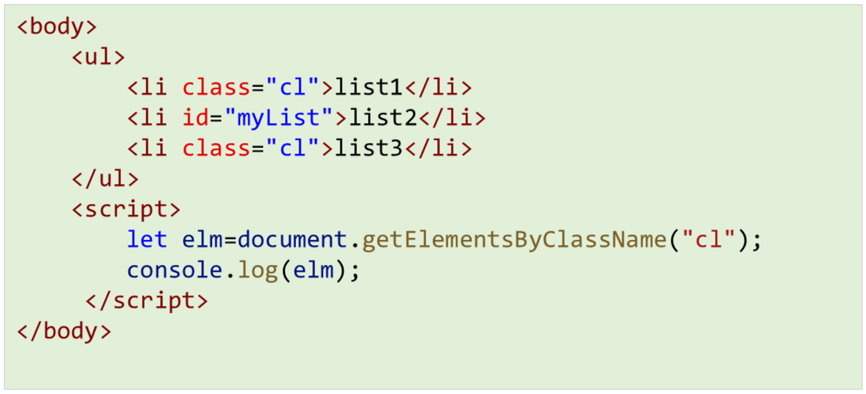
```
        elm.innerHTML="<p>Hello</p>"; //changes the text list1 to hello
        console.log(elm);
</script>
</body>
</html>
```

Output on browser :

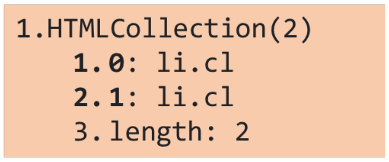


- Hello
- list2
- list3

Example 2 : Select element by class name

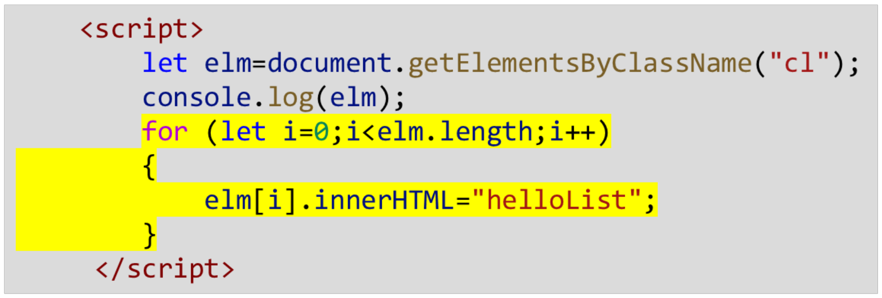


```
<body>
    <ul>
        <li class="cl">list1</li>
        <li id="myList">list2</li>
        <li class="cl">list3</li>
    </ul>
    <script>
        let elm=document.getElementsByClassName("cl");
        console.log(elm);
     </script>
</body>
```

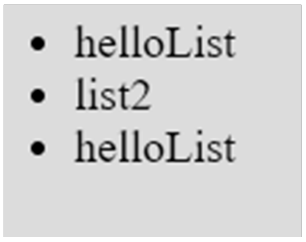


```
1.HTMLCollection(2)
    1.0: li.cl
    2.1: li.cl
    3.length: 2
```

NOTE : It is an array

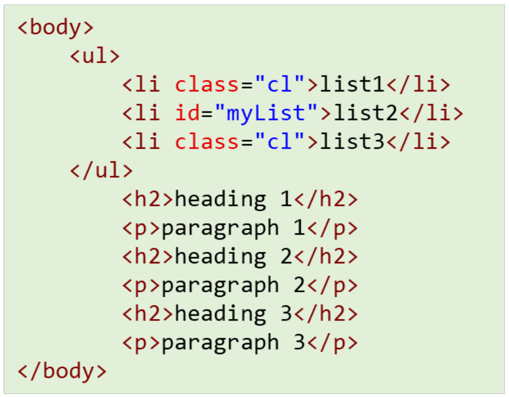


```
    <script>
        let elm=document.getElementsByClassName("cl");
        console.log(elm);
        for (let i=0;i<elm.length;i++)
        {
            elm[i].innerHTML="helloList";
        }
     </script>
```

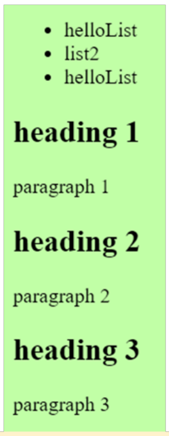


- helloList
- list2
- helloList

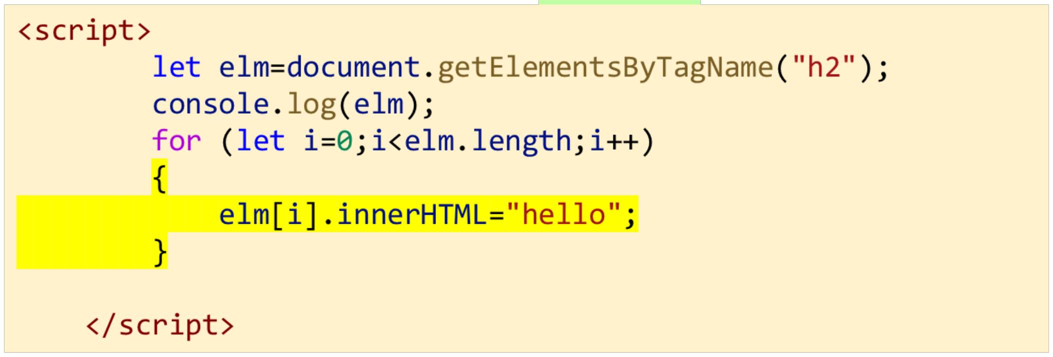
Example 3 : Element by tag



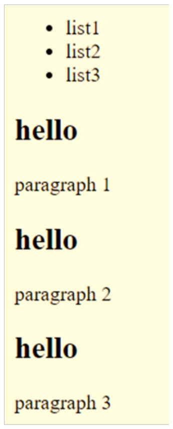
```
<body>
    <ul>
        <li class="cl">list1</li>
        <li id="myList">list2</li>
        <li class="cl">list3</li>
    </ul>
        <h2>heading 1</h2>
        <p>paragraph 1</p>
        <h2>heading 2</h2>
        <p>paragraph 2</p>
        <h2>heading 3</h2>
        <p>paragraph 3</p>
</body>
```



```
• helloList
• list2
• helloList

heading 1

paragraph 1

heading 2

paragraph 2

heading 3

paragraph 3
```

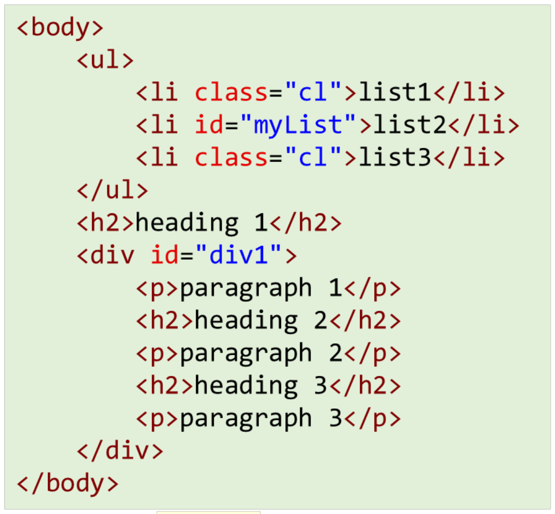


```
<script>
        let elm=document.getElementsByTagName("h2");
        console.log(elm);
        for (let i=0;i<elm.length;i++)
        {
            elm[i].innerHTML="hello";
        }

    </script>
```

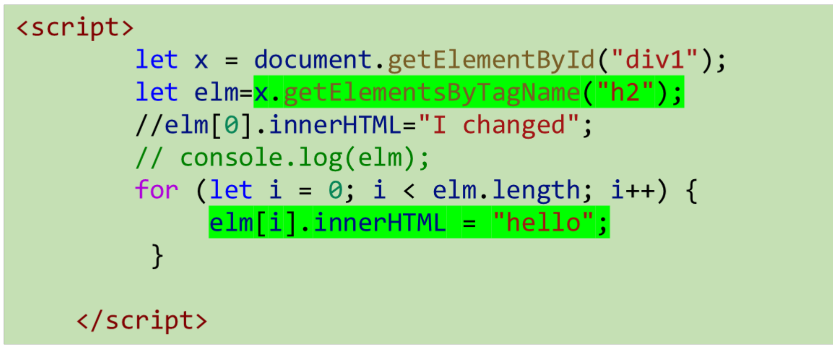


```
• list1
• list2
• list3

hello

paragraph 1

hello

paragraph 2

hello

paragraph 3
```

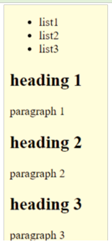
Example 4 : getElementByID and getElementByTagName :



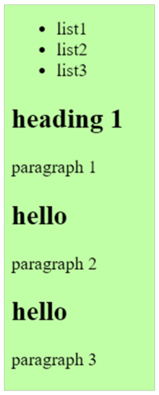
```
<body>
    <ul>
        <li class="cl">list1</li>
        <li id="myList">list2</li>
        <li class="cl">list3</li>
    </ul>
    <h2>heading 1</h2>
    <div id="div1">
        <p>paragraph 1</p>
        <h2>heading 2</h2>
        <p>paragraph 2</p>
        <h2>heading 3</h2>
        <p>paragraph 3</p>
    </div>
</body>
```



```
<script>
        let x = document.getElementById("div1");
        let elm=x.getElementsByTagName("h2");
        //elm[0].innerHTML="I changed";
        // console.log(elm);
        for (let i = 0; i < elm.length; i++) {
            elm[i].innerHTML = "hello";
         }

    </script>
```

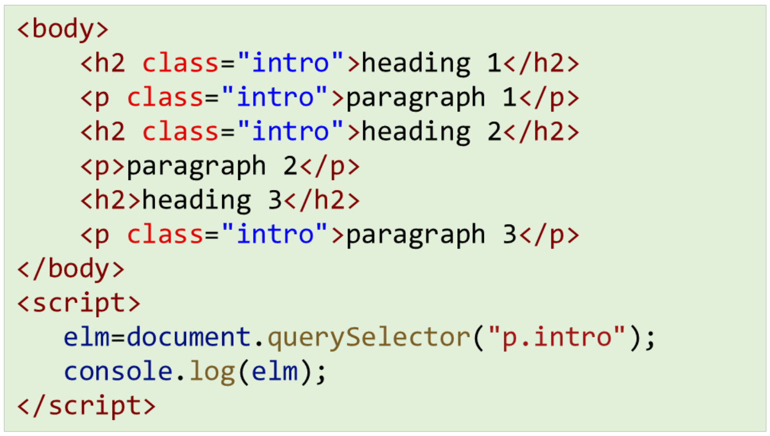


```
• list1
• list2
• list3

heading 1

paragraph 1

heading 2

paragraph 2

heading 3

paragraph 3
```

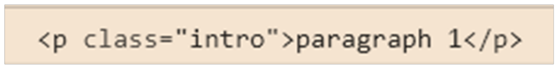


```
• list1
• list2
• list3

heading 1

paragraph 1

hello

paragraph 2

hello

paragraph 3
```

Example 5 : Query Selector

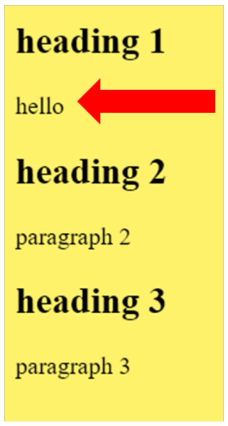


```
<body>
    <h2 class="intro">heading 1</h2>
    <p class="intro">paragraph 1</p>
    <h2 class="intro">heading 2</h2>
    <p>paragraph 2</p>
    <h2>heading 3</h2>
    <p class="intro">paragraph 3</p>
</body>
<script>
   elm=document.querySelector("p.intro");
   console.log(elm);
</script>
```

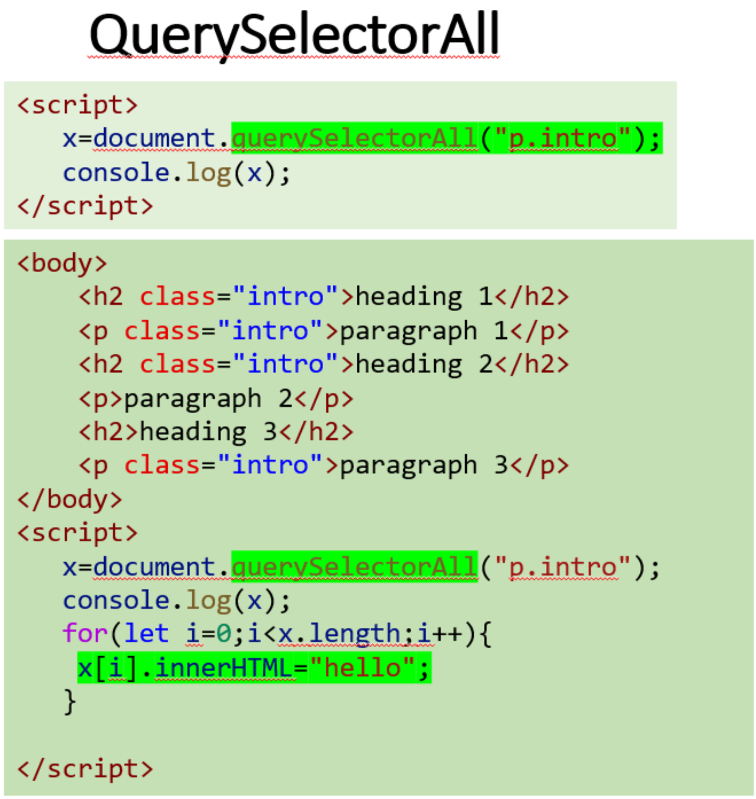


```
<p class="intro">paragraph 1</p>
```

It selected only first matching element .



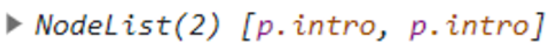
Example 6 : Query Selector All



QuerySelectorAll

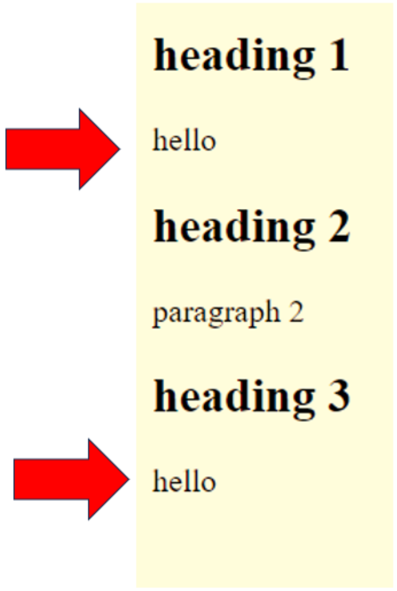
```
<script>
    x=document.querySelectorAll("p.intro");
    console.log(x);
</script>
```

```
<body>
    <h2 class="intro">heading 1</h2>
    <p class="intro">paragraph 1</p>
    <h2 class="intro">heading 2</h2>
    <p>paragraph 2</p>
    <h2>heading 3</h2>
    <p class="intro">paragraph 3</p>
</body>
<script>
    x=document.querySelectorAll("p.intro");
    console.log(x);
    for(let i=0;i<x.length;i++){
     x[i].innerHTML="hello";
    }

</script>
```



```
▸ NodeList(2) [p.intro, p.intro]
```

It selected all matching element .



HTMLCollection and NodeList

Technically, `getElementsByTagName()` and `getElementsByClassName()` return an HTMLCollection, and `querySelectorAll()` returns a NodeList. HTMLCollection is an interface representing a generic collection of elements. A NodeList is an object with a collection of nodes. HTMLCollection and NodeList both act like an array. Both have a `length` property, and elements can be accessed with braces. Ex: `elementList[0]` is the first element in an HTMLCollection or NodeList.

Modifying DOM node attributes

After searching the DOM for an element, JavaScript may be used to examine the element's attributes or to change the attributes. By modifying attributes, JavaScript programs can perform actions including:

- Change which image is displayed by modifying an img element's `src` attribute.
- Determine which image is currently displayed by reading the img element's `src` attribute.
- Change an element's CSS styling by modifying an element's `style` attribute.

Every attribute for an HTML element has an identically named property in the element's DOM node. Ex: `<a href="https://www.nasa.gov/" id="nasa_link">NASA</a>` has a corresponding DOM node with properties named `href` and `id`. Each attribute property name acts as both a getter and a setter.

- Getter: Using the property name to read the value allows a program to examine the attribute's value. Ex: `nasaUrl = document.getElementById("nasa_link").href` assigns `nasaUrl` the string `"https://www.nasa.gov/"` from the anchor element's `href` attribute.

- Setter: Writing to a property allows a program to modify the attribute, which is reflected in the rendered webpage. Ex: `document.getElementById("nasa_link").href = "https://www.spacex.com/"` changes the element's hyperlink to the SpaceX URL.

An element's attribute can be removed using the element method removeAttribute().
Ex: `document.getElementById("nasa_link").removeAttribute("href")` removes the link from the anchor element so that clicking on the HTML element no longer performs an action.

Example : Modifying DOM node attributes.

1. The webpage shows a photo of a lake with a sentence underneath.

```html
<body>
   <img src="lake.jpg"
        alt="Lake photo">
   <p>
      I love the outdoors!
   </p>
</body>
```

I love the outdoors!

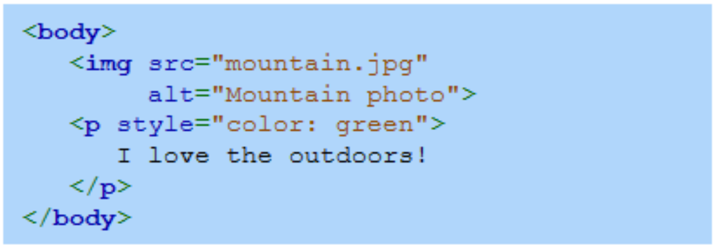
2. Changing the img element's src attribute causes the webpage to replace lake.jpg with mountain.jpg.

```html
<body>
   <img src="mountain.jpg"
        alt="Mountain photo">
   <p>
      I love the outdoors!
   </p>
</body>
```

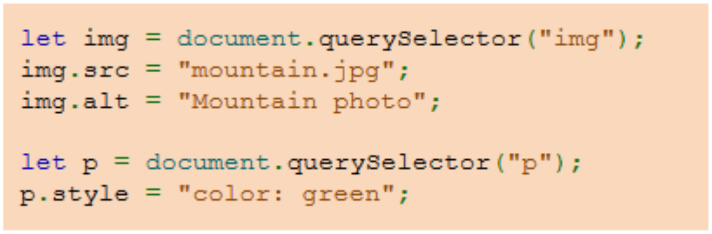
```javascript
let img = document.querySelector("img");
img.src = "mountain.jpg";
img.alt = "Mountain photo";

let p = document.querySelector("p");
p.style = "color: green";
```

I love the outdoors!

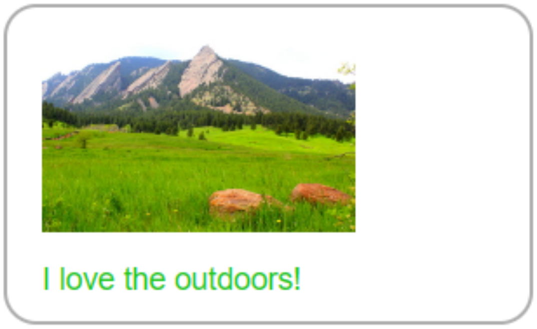
3. Setting the p element's style attribute changes the paragraph's CSS styling, making the text green.



```
<body>
   <img src="mountain.jpg"
        alt="Mountain photo">
   <p style="color: green">
      I love the outdoors!
   </p>
</body>
```



```
let img = document.querySelector("img");
img.src = "mountain.jpg";
img.alt = "Mountain photo";

let p = document.querySelector("p");
p.style = "color: green";
```



I love the outdoors!

Modifying DOM node content

After searching the DOM for an element, JavaScript may be used to examine or change the element's content.

Two common properties are used to get or set an element's content:

1. The textContent property gets or sets a DOM node's text content.
   Ex: `document.querySelector("p").textContent = "$25.99";` changes the paragraph to `<p>$25.99</p>`.
2. The innerHTML property gets or sets a DOM node's content, including all of the node's children, using an HTML-formatted string. Ex: `document.querySelector("p").innerHTML = "<strong>$25.99</strong>";` changes the paragraph to `<p><strong>$25.99</strong></p>`.

The `innerHTML` property uses an internal parser to create any new DOM nodes. `textContent`, however, only creates or changes a single text node. For setting an element's text, `textContent` is somewhat faster than `innerHTML` because no HTML parsing is performed.

Example :

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
  <p>
```

```
      NAMES
    </p>
    <ol>
      <li>A</li>
      <li>B</li>
      <li>C</li>
    </ol>
   <script >
     let p=document.querySelector("p");
     p.textcontent="Most common girls name in 2020";
     let ol=document.querySelector("ol");
     ol.innerHTML="<li>Emma</li><li>Olivia</li><li>Ava</li>";
   </script>
</body>
</html>
```

Download the following link to run the code :
https://replit.com/join/bzzfpsczkv-pragaticoder (https://replit.com/join/bzzfpsczkv-pragaticoder)

Less common ways to change node content

The nodeValue property gets or sets the value of text nodes. As the DOM tree represents textual content separately from HTML elements, the textual content of an HTML element is the first child node of the HTML element's node. So, to access the textual content of an HTML element within the DOM, `firstChild.nodeValue` is used to access the value of the HTML's element's first child.

Ex: `document.getElementById("saleprice").firstChild.nodeValue = "$25.99"`:

1. Gets the DOM node for the element with id "saleprice".
2. Uses `firstChild` to access the textual content node for the element.
3. Uses `nodeValue` to update the content.

The innerText property gets or sets a DOM node's rendered content. `innerText` is similar to `textContent`, but `innerText` is aware of how text is rendered in the browser. Ex: In `<p>Testing one</p>`, `textContent` returns "Testing   one" with spaces, but `innerText` returns "Testing one" with the spaces collapsed into a single space.