While folks are joining

Get your laptops ready and keep the Sandbox opened. We will be coding away in the session!

Crio Fullstack Sprint: FE-2

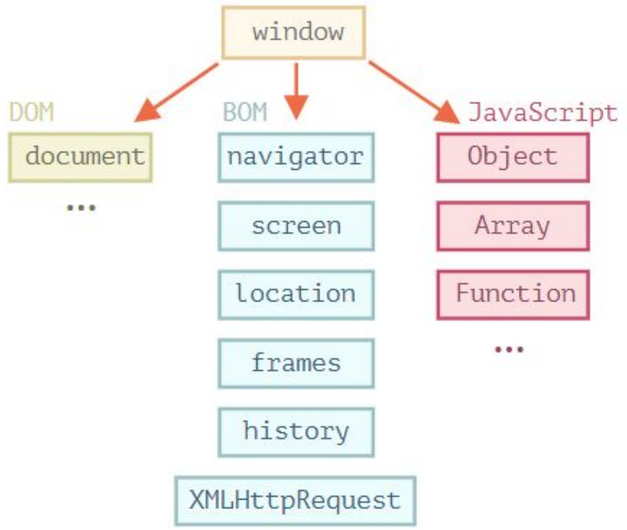
Session 4

In this session

- JavaScript inside Browsers
- JS with HTML
- The Document Object Model
- Nodes in DOM and their properties
- Selecting DOM Elements
- Creating and Inserting elements to DOM

JavaScript inside Browsers

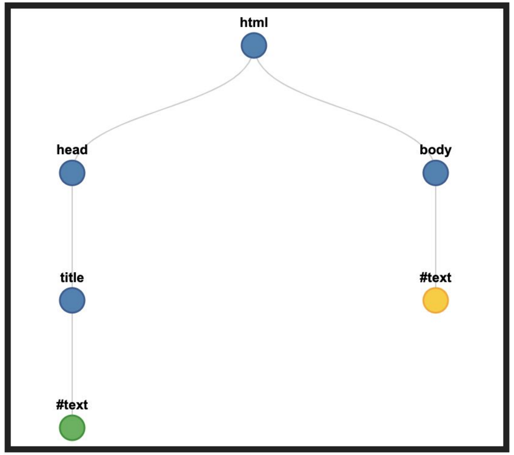
- JS → initially created for the web → Later evolved to be used on many platforms.
- Most commonly used inside a browser.
- The browser gives us additional powers to do more with web pages which load inside it.
- This "window" object represents the browser tab.
    - It is a global object, i.e. its properties can be accessed directly throughout our scripts.



The Document Object Model (DOM)

- DOM is the representation of the HTML structure
  - Makes it possible to manipulate a web page programmatically
  - The `document` object represents a web page loaded on the browser tab.
- `document` object is part of the global `window` object
  - Accessed by `window.document` or `document`
- In DOM,
  - Every HTML tag is an object
  - Nested tags are “children” of the enclosing one
  - The text inside a tag is an object as well.

```html
<!DOCTYPE html>
<html>
 <head>
   <title>About elk</title>
 </head>
 <body>
   The truth about elk.
 </body>
</html>
```

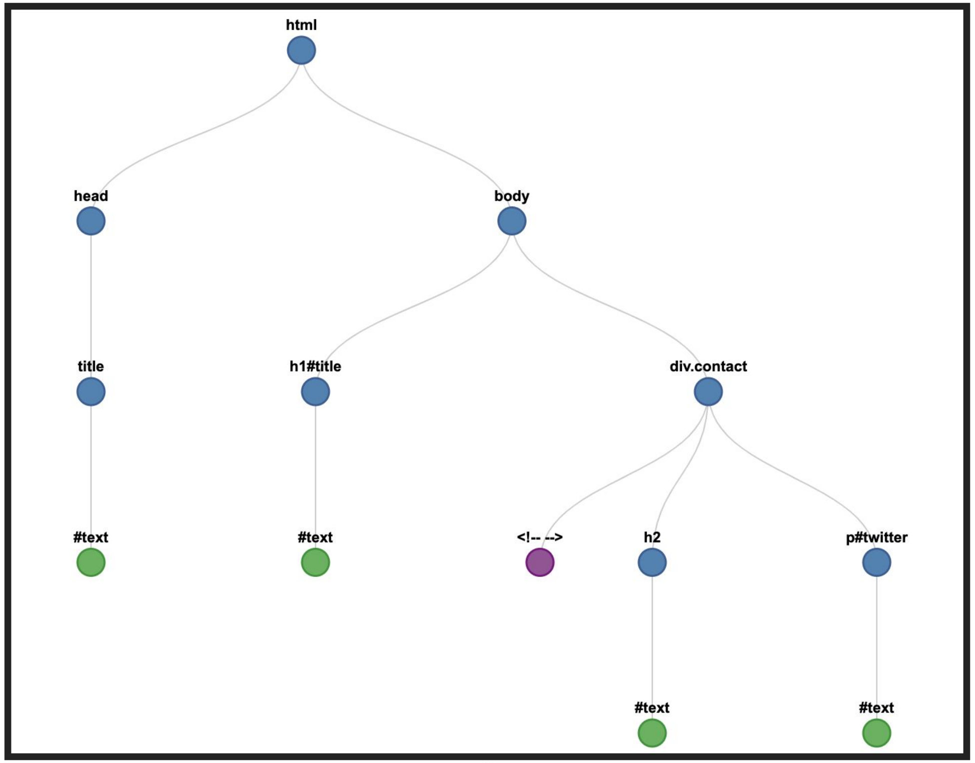


Another example

```
<!DOCTYPE html>
<html>
  <head>
    <title>DOM Visualizer</title>
  </head>
  <body>
    <h1 id="title">DOM Visualizer</h1>
    <div class="contact">
      <!-- This is a comment -->
      <h2>Romain Bohdanowicz</h2>
      <p id="twitter">Twitter : @bioub</p>
    </div>
  </body>
</html>
```

Notes

- Text inside elements → text nodes
  - Spaces(␣) and newlines (↵) forms text nodes
- HTML comments → comment nodes



JS with HTML

- HTML has a `<script>` tag for adding JavaScript to our web page document.
- There are 3 places where we can specify the script

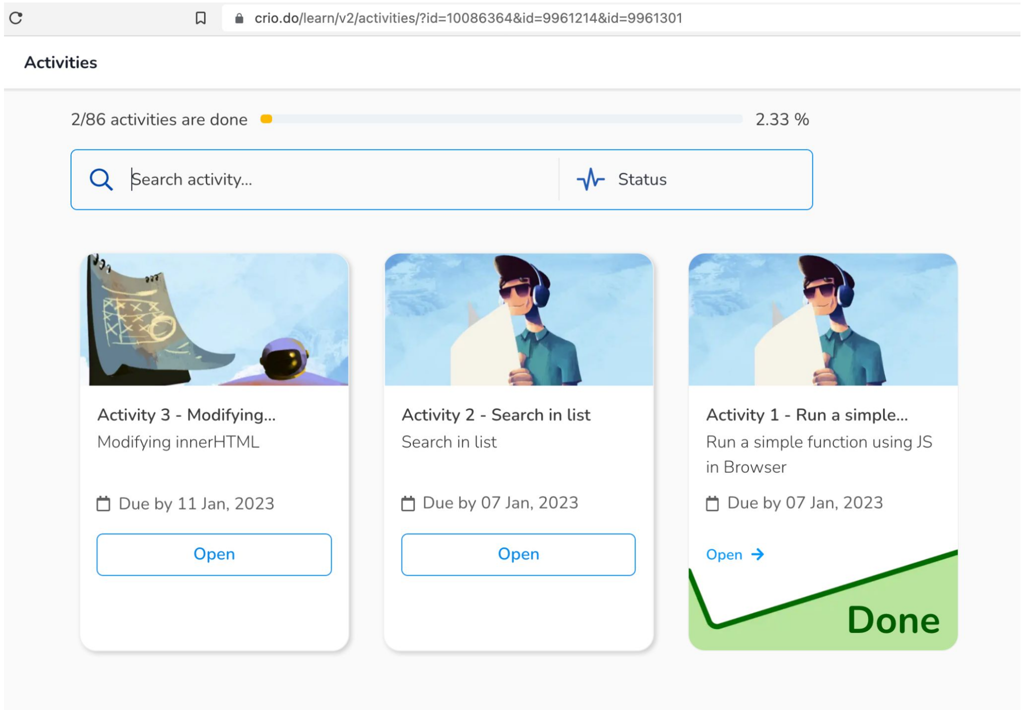
  1. Inside `<head>`
  2. Inside `<body>`
  3. Or Externally in a file **(recommended)** - Improves speed, makes HTML easier to read

💡 Placing scripts at the bottom of the <body> element **improves the display speed**, because script interpretation slows down the display.

Session Activities on Crio Platform

[Session 3 Activities Link]

- View all activities related to a session directly on the platform
- Mark as completed once the requirements are implemented



crio.do/learn/v2/activities/?id=10086364&id=9961214&id=9961301

Activities

2/86 activities are done 2.33 %

Activity 3 - Modifying...
Modifying innerHTML
Due by 11 Jan, 2023

Activity 2 - Search in list
Search in list
Due by 07 Jan, 2023

Activity 1 - Run a simple...
Run a simple function using JS in Browser
Due by 07 Jan, 2023

Activity 1 - Run a simple function using JS in Browser

- Create “activity1.html”. In it, type “!” and press Enter to add the HTML starter code
- Create a script inside the body and write a function `init()`
- This function prints "Page Loaded" on the console.
- Call this function inside this script tag.

If the message is displayed on the console, it means JS is working!

Curious Cats 🐈

Can you tell what is the difference between putting the `<script>` tag inside `<body>` and `<head>`

Try to console log with `document.body` from `<script>` inside `<head>` and `<body>` see the difference, can you reason why?

Nodes in DOM

- Everything inside the DOM including the `document` object is a DOM node
- In practice, we will be using 3 of these
    - `document` – the “entry point” into DOM.
    - element nodes – HTML-tags, the tree building blocks.
    - text nodes – contain text.
- Everything inside the DOM including the `document` is a DOM node.
- DOM nodes are objects, they have
    - methods (or functions)
    - properties (whose values we can change)
- We can use JS to call these properties and methods to modify or access these nodes

Selecting Elements - Using HTML

- Like we used CSS Selectors in CSS, here also we can select elements by
  - Tag names - `document.getElementbyTagName(HTML_TAG_NAME)`
  - Class names - `document.getElementsByClassName(ELEMENT_CLASS_NAME)`
  - ID - `document.getElementById(ID)`
- With Tag names and class names we get → An array like list of elements
  - Known as “HTML Collection”
- With id, we get a specific DOM element.

Quick Note - HTML Collection

- An HTML Collection is an *array-like list* which we can convert to Array using the Array.from() function of Array class in JS.

```
const headingCollection = document.getElementsByClassName("heading");
const arr = Array.from(headingCollection);
```

Node Properties

- There are two popular properties attached to Nodes which we use to **get or set** the inner elements of any element or Node.
    - `innerHTML` - For the HTML inside the element as a string.
    - `textContent` - For the text inside an element.

```
<div id="news">
 <h1>Headline!</h1>
 <p>Martians attack people!</p>
</div>

<script>
  // Headline! Martians attack people!
  console.log(document.getElementById("news").textContent);
  console.log(document.getElementById("news").innerHTML);


  document.getElementById("news").innerHTML =
    “<h1>Breaking news!</h1><p>innerHTML makes it super easy  to set HTML for an element</p>”;
</script>
```

Activity 2 - Search in list

- Given the below starter HTML

```html
<ul id="names">
  <li>John</li>
  <li>Mary</li>
  <li>Dora</li>
  <li>Donny</li>
</ul>
```

- Write a function searchPerson(elemId, name) which takes in a DOM element's id (here "names") and searches the name in the list.
  - If name is present, log “Found”, else log “Not Found”
  - Eg: searchPerson(“names”, “John”) // logs “Found”
  - Eg: searchPerson(“names”, “Crio.Do”) // logs “Not Found”

Hints

- Select the parent element (ul) using the Id.
- Create an Array of list items of this parent element.
- Extract the text out of each element and then check if the name is present in this array or not.

5 mins break

Selecting Elements - Using CSS Queries

- To select elements based on CSS queries like `div li` or `div > p` we can use `querySelector` to find the first element which matches the query. It **returns a single HTML element**
- To find all elements we use `querySelectorAll`. **It returns a Node List** which is again an Array-like iterable.

```
<div id="one">
   <h1>Headline!</h1>
   <p>Martians attack people!</p>
</div>
<script>
   console.log(document.querySelector("div > p"));
</script>
```

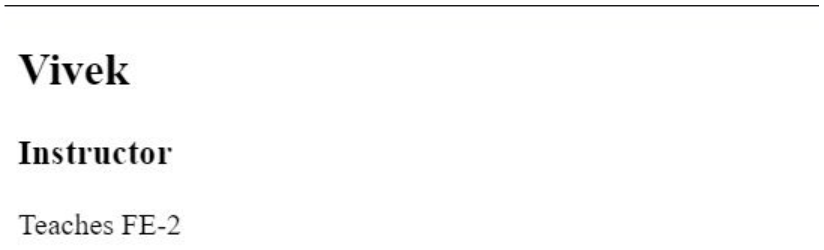
Summary: Selecting DOM Elements

Method	Input	Returns
`getElementsbyTagName`	HTML Tag name	HTML Collection
`getElementsByClassName`	Element Class Name	HTML Collection
`getElementById`	Element's Unique ID	DOM Node
`querySelector`	CSS Selector	DOM Node
`querySelectorAll`	CSS Selector	Node List

Activity 3 - Modifying innerHTML

- Use the starter HTML here and write a function `updateCard(elem, name, role, bio)` which takes in an element to update, name, role, and bio as strings and changes the content on the screen.

```html
<section id="card">
  <h1>New Student Name</h1>
  <h3>Role</h3>
  <p>Bio Text Goes Here</p>
</section>
```



Vivek

Instructor

Teaches FE-2

- For example the below code should render

```javascript
const cardElement = document.getElementById("card");
updateCard(cardElement, "Vivek", "Instructor", "Teaches FE-2");
```

Adding new elements to DOM

Method 1: Appending to innerHTML

```
<ul id="myList">

  <li>List item 1</li>

  <li>List item 2</li>

  <li>List item 3</li>

  <li>List item 4</li>

  <li>List item 5</li>

</ul>
```

Step 1: Get existing element

```
let existingDOMNode = document.getElementById("myList");
```

Step 2: Directly update its innerHTML

```
existingDOMNode.innerHTML =  existingDOMNode.innerHTML + "<li>List item 6</li>";
```

```
<ul id="myList">

  <li>List item 1</li>

  <li>List item 2</li>

  <li>List item 3</li>

  <li>List item 4</li>

  <li>List item 5</li>

  <li>List item 6</li>

</ul>
```

Method 2: Appending a new element

```
<ul id="myList">

  <li>List item 1</li>

  <li>List item 2</li>

  <li>List item 3</li>

  <li>List item 4</li>

  <li>List item 5</li>

</ul>
```

Step 1: Get existing element

```
let existingDOMNode = document.getElementById("myList");
```

Step 2: Create a new element; Not yet on DOM

```
const newElement = document.createElement("li");
```

Step 3: Add content to the new element

```
newElement.textContent = "List item 6";
```

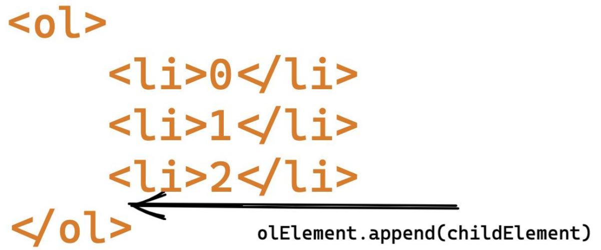
Step 4: ***Add new element as child of existing DOM Node***

```
existingDOMNode.append(newElement);
```

```
<ul id="myList">

  <li>List item 1</li>

  <li>List item 2</li>

  <li>List item 3</li>

  <li>List item 4</li>

  <li>List item 5</li>

  <li>List item 6</li>

</ul>
```

Summary

- `document.createElement(tag)` - Creates a new element node with the given tag
- .append() - Adds an element a child of a DOM element
- ***Pitfall***: New element doesn't show up on DOM unless it's added as a child of an existing DOM element



```
<ol>
    <li>0</li>
    <li>1</li>
    <li>2</li>
</ol>
```

```
olElement.append(childElement)
```

Activity 4 - Add to Lists

- Use the following list as starter code
- Write a function `appendToParent(parent, childTagType, text)` to add a new child to the parent with the given `childTagType` and the given `text` content for the child.

```html
<ol id="elem">
  <li>Hello</li>
  <li>World</li>
</ol>
```

```javascript
let parentElement = document.getElementById("elem");

appendToParent(parentElement, "li", "Newly added");
```

```html
<ol id="elem">
  <li>Hello</li>
  <li>World</li>
  <li>Newly added</li>
</ol>
```

Until next session 🍃

Thank you for joining in today, we’d love to hear your thoughts and feedback here -

https://forms.gle/K4WZBHCtfKac7a4o9

Thank you