

Web Programming 2X1

Intro to JavaScript

DOM

Outcomes

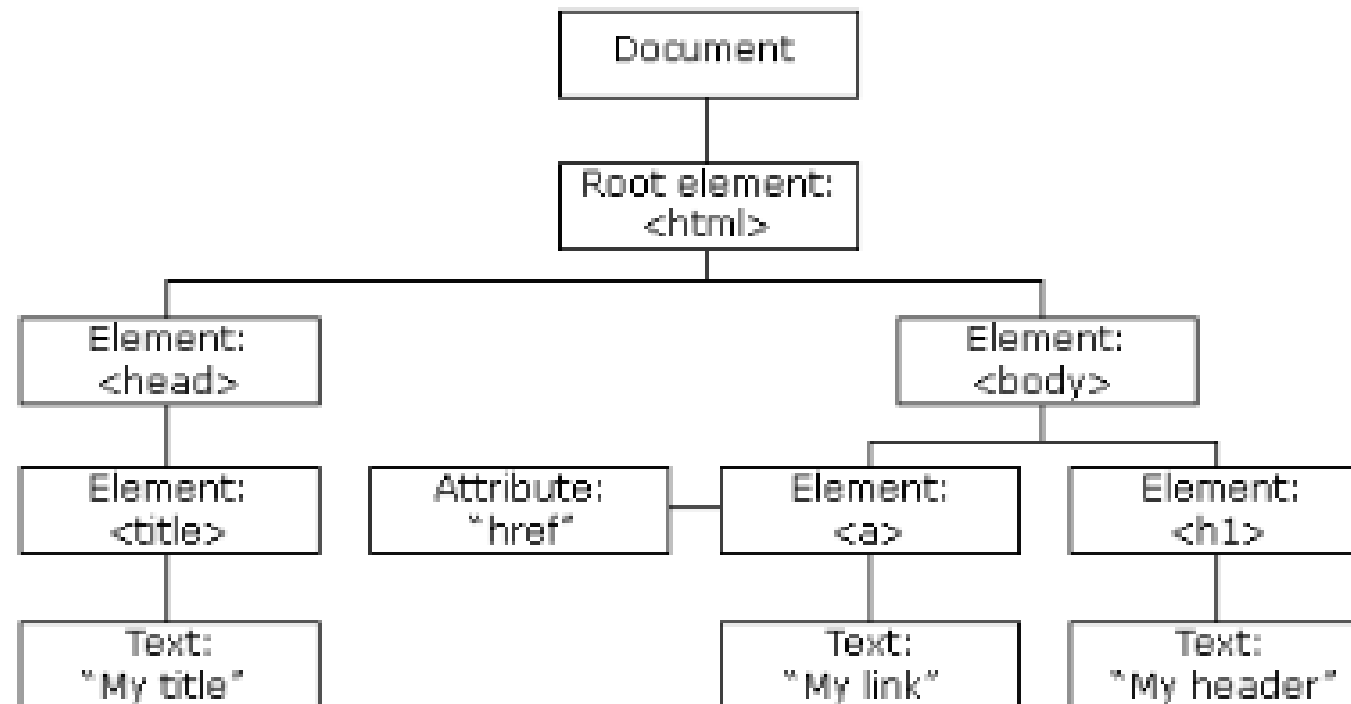
Students should understand the following outcomes, upon successful completion of this module:

- Intro to the DOM
- Searching for an element
 - `getElementById()`,
 - `getElementsByClassName()`
 - `getElementsByTagName()`
 - `querySelector()`
 - Selecting Within Elements
- Dynamically referencing DOM element using `textContent`

Intro to DOM

When a web page is loaded, the browser creates a **Document Object Model** (DOM) of the page.

***DOM**, represents *all page content* as *objects* that can be *modified*.*



Intro to DOM

- The *DOM* is the tree of HTML nodes.
- It is a representation of what is rendered in the web browser.
- We can change or create anything on the page using it.
- The *DOM* specification explains the structure of a document and provides objects to manipulate it.

index.html

```
1  <!DOCTYPE HTML>
2  <html>
3  <head>
4  |   <title>About elk</title>
5  </head>
6  <body>
7  |   <p>Hello</p>
8  |   <li>Mom</li>
9  |   <li>and</li>
10 |   <li>Dad</li>
11 </body>
12 </html>
```

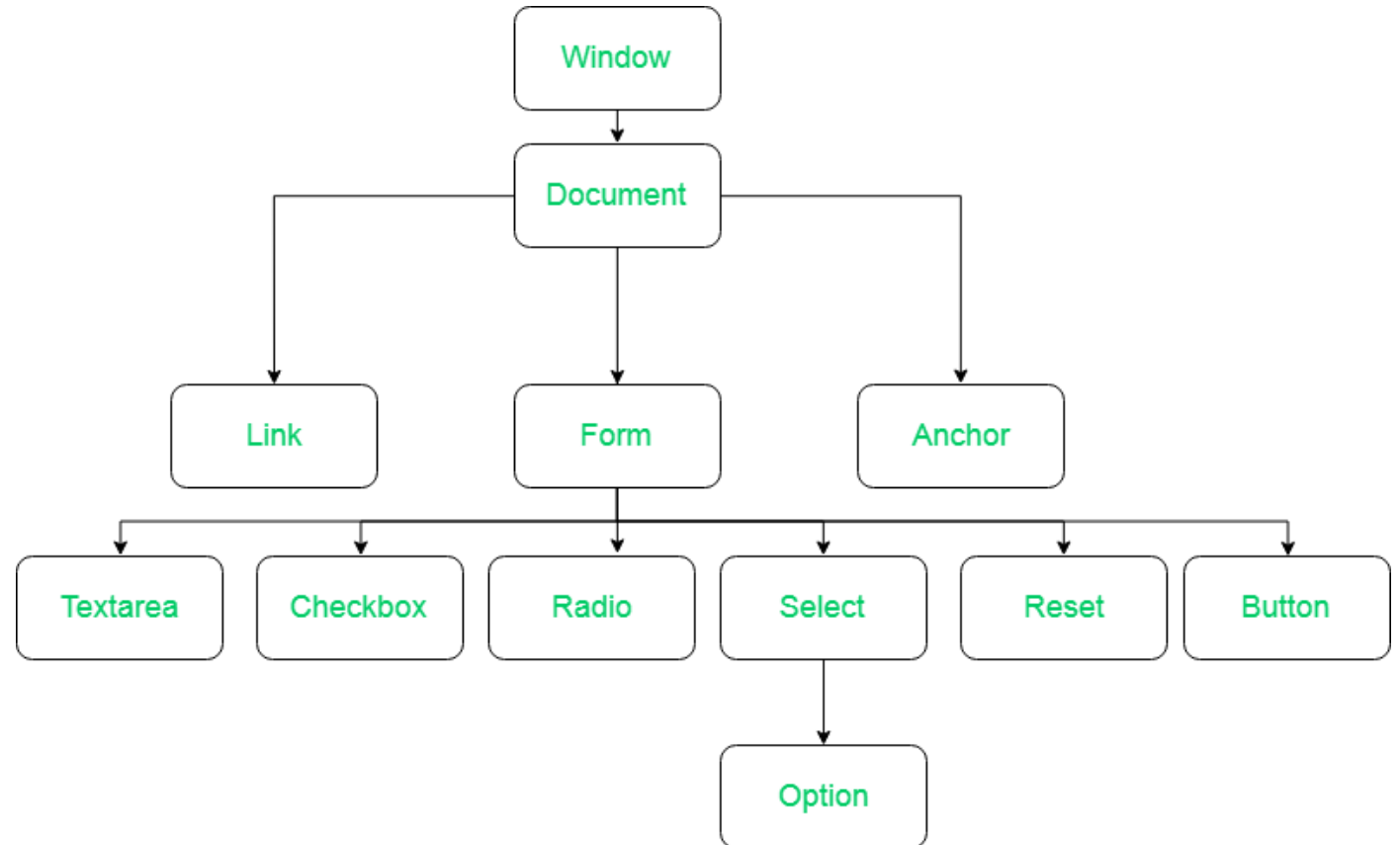
DOM tree

The backbone of an HTML document is *tags*.

According to the DOM, every *HTML tag* is an *object*.

Nested tags are “*children*” of the enclosing tag.

The *text* inside a tag is an *object* as well.

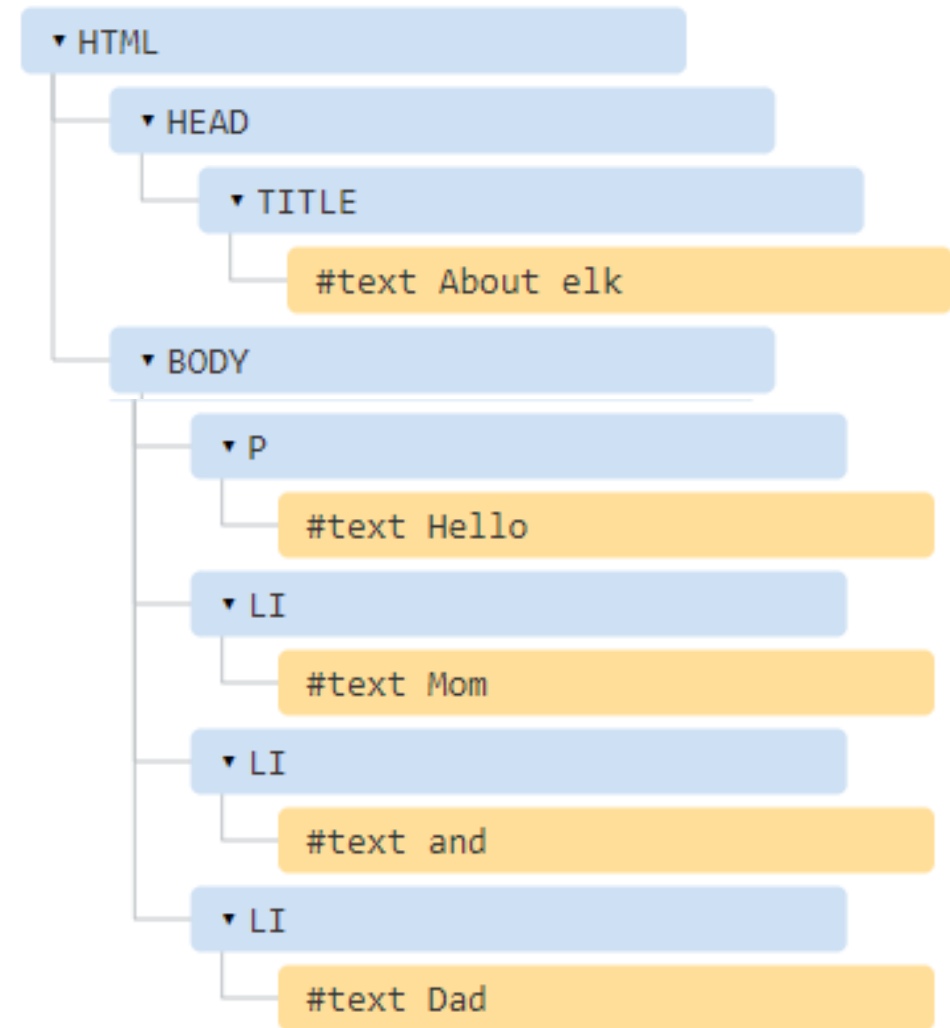


All these objects are accessible using JavaScript, and we can use them to modify the page.

DOM tree

index.html

```
1  <!DOCTYPE HTML>
2  <html>
3  <head>
4  |   <title>About elk</title>
5  </head>
6  <body>
7  |   <p>Hello</p>
8  |   <li>Mom</li>
9  |   <li>and</li>
10 |   <li>Dad</li>
11 </body>
12 </html>
```



DOM structure

DOM tree

Nodes of DOM

- Every *tree node* is an *object*.
- *Tags* are element nodes (or just elements) and form the tree structure:
 - `<html>` is at the root node, then `<head>` and `<body>` are its children, etc.
- The *text* inside elements forms *text nodes*, labelled as *#text*.
- A text node contains *only a string*. It may *not* have children and is always a *leaf* of the *tree*.
- *Spaces* and *newlines* are totally *valid* characters, like letters and digits.
- They form *text nodes* and become a part of the DOM.

Core Concepts of DOM

- **Node:** Every part of the document is a node, including elements, attributes, and text. Nodes can be of various types like elements, text nodes, attribute nodes, etc.
- **Element:** Elements are the most common type of nodes and represent the structure and content of the document. They can have attributes, child nodes, and content.
- **Attributes:** Attributes provide additional information about elements and are represented as node properties.
- **Methods for Access and Manipulation:** The DOM provides methods and properties to dynamically update web pages and creating interactive web applications.

Intro to DOM

For instance:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>My We</title>
  <script src="app.js"></script>
</head>
<body>
  <center>
    <button onclick="changeBody()">
      Ok
    </button>
  </center>
</body>
</html>
```

Index.html

```
// change the background color to black after clicking button
```

```
function changeBody () {
  document.body.style.background = "black";

  setTimeout(() => document.body.style.background = "", 3000);
}
```

app.js

Why DOM???

Interacting with the DOM

- Change / Remove HTML elements in the DOM / on the page
- Change & add CSS styles to elements
- Read & change element attributes (href, src, alt, custom)
- Create new HTML elements and insert them into the DOM / the page
- Attach event listeners to elements (click, keypress, submit)

Children: childNodes, firstChild, lastChild

There are two terms that we'll use from now on:

- **Child nodes (or children)** – elements that are direct children. In other words, they are nested exactly in the given one. For instance, `<head>` and `<body>` are children of `<html>` element.
- **Descendants** – all elements that are nested in the given one, including children, their children and so on.

```
1 <html>
2 <body>
3   <div>Begin</div>
4
5   <ul>
6     <li>
7       <b>Information</b>
8     </li>
9   </ul>
10 </body>
11 </html>
```

For instance, here `<body>` has **children** `<div>` and `` (and few blank text nodes)

...And **descendants** of `<body>` are not only direct children `<div>`, `` but also more deeply nested elements, such as `` (a child of ``) and `` (a child of ``) – the entire subtree.

Searching (Querying the DOM)

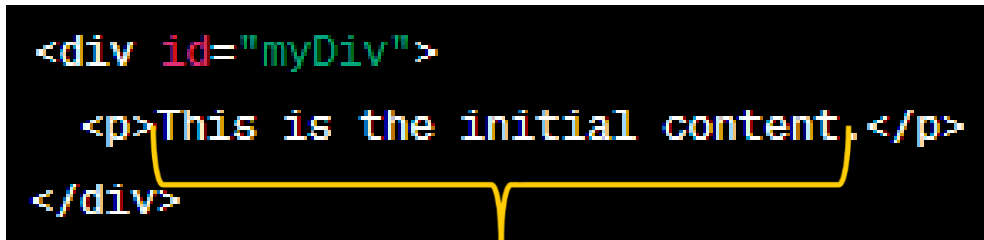
There are also other methods to look for nodes by a *tag*, *class*, *name* or the use of *querySelector*.

- a) *document.getElementById('id')* - returns elements that have the given CSS class name.
- b) *document.getElementsByClassName('className')* - returns elements that have the given CSS class name.
- c) *document.getElementsByTagName('tag')* - looks for elements with the given tag and returns the *collection* of them.
- d) *document.getElementsByName('name')* - returns elements with the given name attribute, document-wide. Very rarely used.
- e) *document.querySelector('css')* - returns the *first* element inside the document matching the given CSS selector.

innerHTML Property in DOM

- *innerHTML* is a property in JavaScript that allows you to access or modify the HTML content inside an element:
- *innerHTML property* represents the *content* between the opening and closing tags of an HTML *element* as a string.

```
<div id="myDiv">
  <p>This is the initial content.</p>
</div>
```

A yellow bracket is drawn under the text "This is the initial content." in the code snippet, indicating that the innerHTML property would retrieve this specific content from the HTML element.

innerHTML gets the content in-between a targeted HTML element

- This property can be used to dynamically update or retrieve the content of an element on a webpage.
- It is easiest way to get the content of an element by getting or replacing the content of HTML elements.

innerHTML Property in DOM

- *Say we have the following website:*

WPR271

PRG271

PRJ271

[Click Here](#)

Output Using InnerHTML

WPR271

PRG271

PRJ271

getElementById('id')

Example:

```
<body>
  <div id="input">
    <p>WPR271</p>
    <p>PRG271</p>
    <p>PRJ271</p>
  </div>

  <button onclick = "myFunction()" id="btn">
    Click Here
  </button>

  <p id="output"> </p>
</body>
```

Index.html

```
// get content from HTML document, under tag with "input" ID
```

```
function myFunction(){
  let text = document.getElementById("input").innerHTML;
  document.getElementById("output").innerHTML = text;
};
```

app.js

- ID should be unique and can only be utilized once.

getElementByTagName("tag")

- *Say we have the following website:*

WPR271

PRG271

PRJ271

Click Here

Output:

WPR271 PRG271 PRJ271

- We want to get all text from the form input elements.
- The text in the form input elements should then be manipulated using `getElementByTagName("tag")` method, and display as shown:

getElementsByTagName("tag")

- ***document.getElementsByTagName("tag")*** method is to retrieve a collection of tag elements on the page.

```
<body>
  <div id="input">
    <p>WPR271</p>
    <p>PRG271</p>
    <p>PRJ271</p>
  </div>

  <button onclick = "myFunction()" id="btn">
    Click Here
  </button>

  <p id="output"></p>

</body>
```

Index.html

```
function myFunction(){
  // Accessing and modifying innerHTML using
  //document.getElementsByTagName
  let divElements = document.getElementsByTagName("p");
  let content = "";

  for (var i = 0; i < 3; i++) {
    content += divElements[i].textContent + " ";
  }
  document.getElementById("output").innerHTML = content;
};
```

app.js

getElementByTagName("tag")

- 1) The function retrieves all the elements within the div tag, using `getElementsByTagName("p")`.
- 2) It loops through each `<p>` element, extracts its text content using the `textContent` property, and stores it in the `divElements[]` array.
- 3) Finally, it stores each text value from the `<p>` in a string called `content`, after concatenating it with an empty string.
- 4) Finally, the `innerHTML` of the "output" `<p>` element is set to the concatenated content of the elements.

getElementByClassName("name")

- *Say we have the following website:*

WPR271

PRG271

PRJ271

Click Here

Output using getElementByClassName()

WPR271 PRG271 PRJ271

- We want to get all text from the form input elements.
- The text in the form input elements should then be manipulated using the `getElementByClassName("classname")` method, and display as shown:

getElementByClassName("class")

- ***document.getElementsByClassName("tag")*** method is to retrieve a collection of tag elements on the page.

```
<body>
  <div class="myText">
    <p>WPR271</p>
    <p>PRG271</p>
    <p>PRJ271</p>
  </div>

  <button id="outputButton" onclick="useClassName()">
    Click Here
  </button>

  <p id="output"></p>

</body>
```

index.html:

```
function useClassName() {
  var myDiv = document.getElementsByClassName("myText");
  let paragraphText = "";

  for (var i = 0; i < myDiv.length; i++) {
    paragraphText += myDiv[i].textContent + " ";
  }

  document.getElementById("output").innerHTML = paragraphText;
}
```

app.js:

getElementByClassName("class")

1. We utilized the useClassName() function, which is triggered when the "Click Here" button is clicked.
2. The function uses document.getElementsByClassName("myText") to retrieve all elements with the class name "myText".
3. It loops through each element, extracts the text content, and stores it in the myDiv[], which is an array.
4. We then use the textContent property to get or set the text content of an HTML element.
5. textContent returns a string that includes all the textual content within the specified element.
6. Finally, the innerHTML of the "output" <p> element is set to the concatenated content of the elements in the "myText" class.

getElementByName("name")

- *Say we have the following website:*

Enter Name

Enter Surname

Enter Course

Simba

Zengeni

IT

- We want to get all test from the form input elements.
- The text in the form input elements should then be manipulated using `getElementByName("name")` method, and display as shown:

getElementByName("name")

- The `getElementsByName()` method is primarily used to select elements based on their name attribute, which is more commonly associated with form elements like input fields, radio buttons, and checkboxes.

```
<body>
  <div id="input">
    <label for="">Enter Name</label> <br>
    <input type="text" name="details" > <br> <br>
    <label for="">Enter Surname</label> <br>
    <input type="text" name="details" > <br> <br>
    <label for="">Enter Course</label> <br>
    <input type="text" name="details"> <br> <br>
  </div>

  <button onclick="myFunction()" id="btn">
    Display
  </button>

  <p id="output"></p>
</body>
```

Index.html

```
function myFunction() {
  var input = document.getElementsByName("details");
  var inputArray = [];

  for (var i = 0; i < input.length; i++) {
    var inputValue = input[i].value;
    inputArray.push(inputValue);
  }

  var output = document.getElementById("output");
  output.innerHTML = inputArray.join("<br>");
}
```

app.js

getElementByName("name")

1. We utilized *myFunction()*, which is triggered when the "Display" button is clicked.
2. The function uses *document.getElementsByName("details")* to retrieve elements from the input tags, using the attribute name, with name *value: "details"*.
3. It loops through each element, extracts the text content in the input elements, and stores it in the *inputArray[]* array.
4. We then use the *.value property* returns the current value of an HTML input element. The *value* property is used to access or modify the content of an input element, such as text entered into a text field, the selected option in a dropdown, or the value of a radio button or checkbox.
5. *inputValue* returns a string that includes all values from *input*, and pushes it into an array, which uses *join("
")* to concatenate each value.
6. Finally, the innerHTML of the "output" `<p>` element is set to the concatenated content of *inputArray[]*.

querySelector()

- `querySelector()` is a JavaScript method that allows you to select elements in the DOM using CSS-like selectors.
- It can select elements based on various criteria such as *tag names*, *class names*, *IDs*, *attribute* values, and more.
- The basic syntax of `querySelector`:

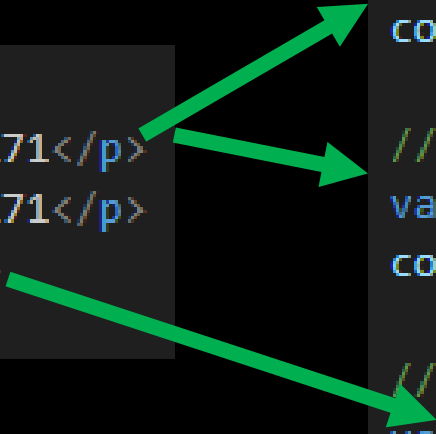
```
document.querySelector(selector);
```

- i. `document` refers to the root of the DOM, i.e., the entire HTML document.
- ii. `selector` is a string representing the CSS selector you want to use to select elements.

querySelector()

- Examples of querySelector usage:

```
<div>  
  <p class="classPara">WPR271</p>  
  <p class="classPara">PRG271</p>  
  <p id="idPara">WPR271</p>  
</div>
```



```
// Select the first element with class "classPara"  
let element1 = document.querySelector(".classPara");  
console.log(element1);
```

```
// Select the first <p> element with class "classPara"  
var element2 = document.querySelector("p.classPara");  
console.log(element2);
```

```
// Select the first element with ID "my-element"  
var element3 = document.querySelector("#idPara");  
console.log(element2);
```

querySelector [ID]

Create the following form:

Username:

Password:

Log In

Logged in as: Simba123 (Password: Paswerd123456)

querySelector [ID]

Functionality:

- 1) A user should enter their username and password.
- 1) The "Log In" button has an onclick attribute that calls the login() function when clicked.
- 2) Inside the login() function, we must use querySelector() to select the input fields by their IDs ('#username' and '#password').
- 3) The values of the input fields must be retrieved using the value property.
- 4) The retrieved values are then stored in an object called user.
- 5) Finally, the concatenated result is displayed in the <p> element with the ID 'output'.

querySelector [ID]

```
<body>
  <form id="loginForm">
    <div class="input-container">
      <label for="username">Username:</label>
      <input type="text" id="username" name="username">
    </div>
    <div class="input-container">
      <label for="password">Password:</label>
      <input type="password" id="password" name="password">
    </div>
    <button type="button" onclick="login()" id="loginButton">
      Log In
    </button>
  </form>

  <p id="output"></p>

</body>
```

Styles.css file is on Moodle

querySelector()

```
function login() {  
  var usernameInput = document.querySelector('#username');  
  var passwordInput = document.querySelector('#password');  
  
  var user = {  
    username:  
    usernameInput.value,  
    password: passwordInput.value  
  };  
  
  var output = document.querySelector('#output');  
  output.textContent = "Logged in as: " + user.username + " (Password: " + user.password + ")";  
}
```

app.js

querySelector [Class]

```
<body>
  <form class="login-form">
    <div class="input-container">
      <label for="username">Username:</label>
      <input type="text" name="username">
    </div>

    <div class="input-container">
      <label for="password">Password:</label>
      <input type="password" name="password">
    </div>

    <button type="button" onclick="login()" class="login-button"> Log In
  </button>
</form>

  <p class="output"></p>
</body>
```

The classes *'login-form'*, *'input-container'*, *'login-button'*, and *'output'* have been applied to the corresponding elements in the HTML.

Styles.css file is on Teams

querySelector [ID]

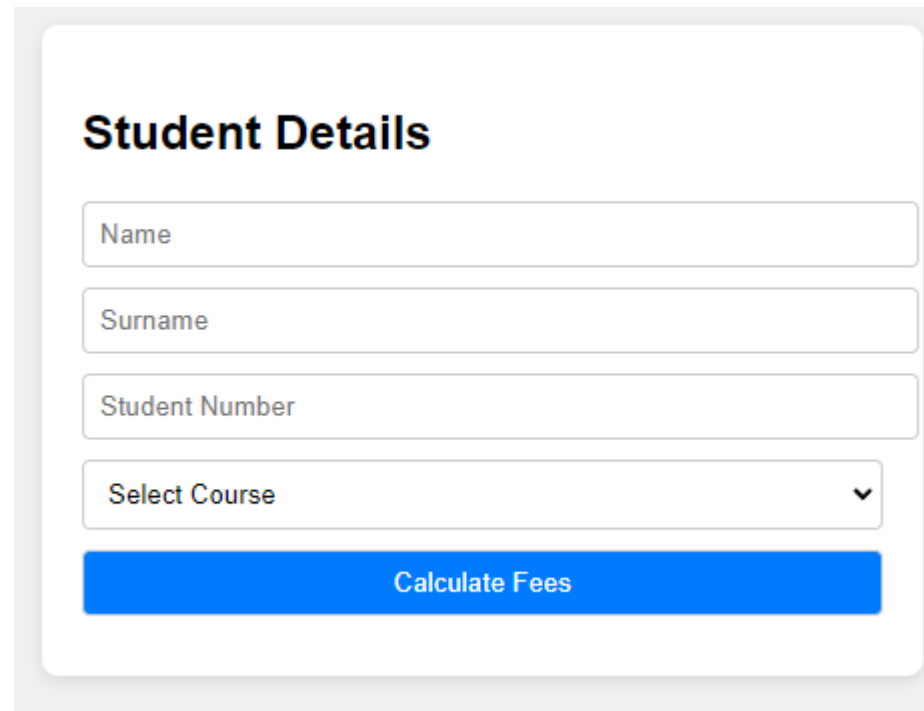
```
function login() {  
  var form = document.querySelector('.login-form');  
  var usernameInput = form.querySelector('[name="username"]');  
  var passwordInput = form.querySelector('[name="password"]');  
  
  var user = {  
    username: usernameInput.value,  
    password: passwordInput.value  
  };  
  
  var output = document.querySelector('.output');  
  output.textContent = "Logged in as: " + user.username + " (Password: " + user.password + ")";  
}
```

app.js

Exercise

Use the addendum (html and css files provided) to create the following application.

A student can either be in diploma or degree.



Student Details

Name

Surname

Student Number

Select Course ▼

Calculate Fees

Exercise

If the student is diploma, then fill in their details and its displayed

Student Details

Simba

Zengeni

4444

Diploma

Calculate Fees

Total Fees for Simba Zengeni: \$5000

If the student is degree, then fill in their details and its displayed

Student Details

Ryan

Kundayi

5555

Degree

Calculate Fees

Total Fees for Ryan Kundayi: \$6500

Validate, in JavaScript, all entries made before displaying.

Thank You!

The End

info@belgiumcampus.ac.za

+27 10 593 53 68



/belgiumcampusSA



#Belgium Campus



/belgiumcampus