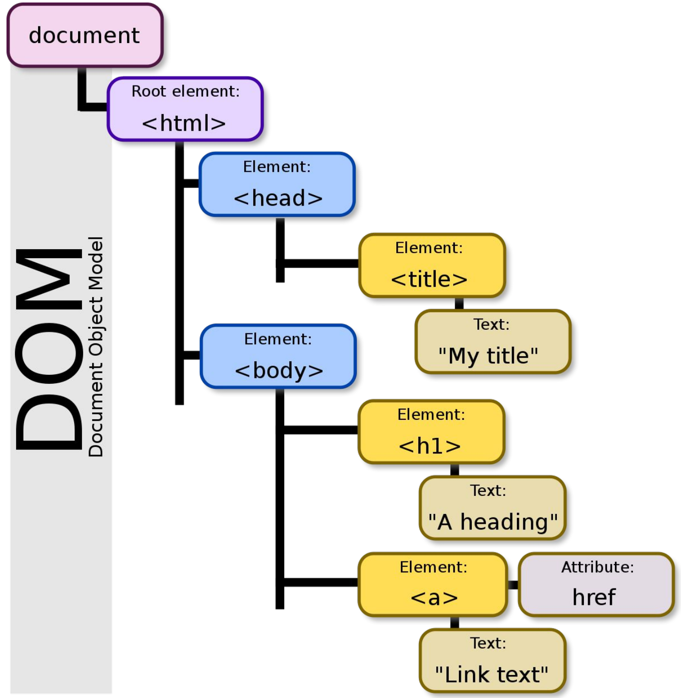
Web development basics

DOM manipulations + form validation

Introduction to DOM Manipulation

The Document Object Model (**DOM**) is a programming interface for **HTML** and **XML** documents. It represents the structure of a document as a tree-like hierarchy, with the document itself at the top (the "**root**" of the tree), and each element, attribute, and piece of text inside the document as a node in the tree.

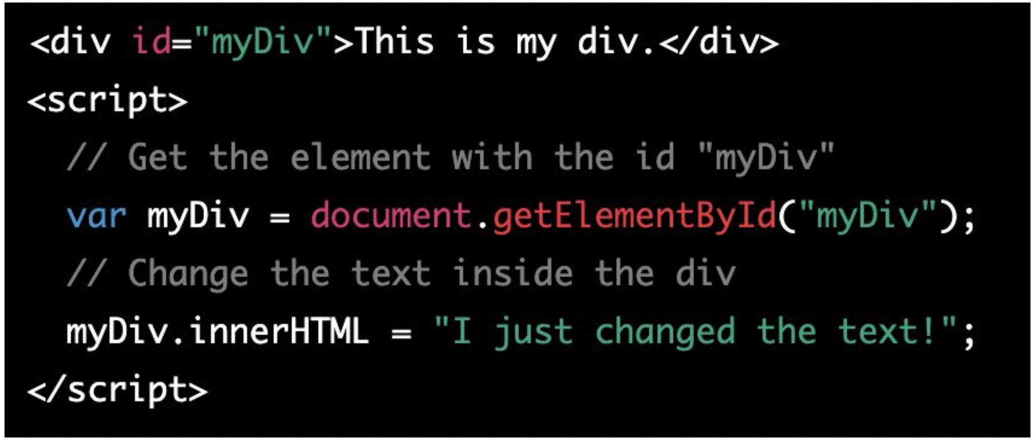
JavaScript can be used to manipulate the DOM, which allows developers to change the structure and content of a web page in real time. For example, you can use JavaScript to add or **remove** elements from the page, change the **text** or **styles** of elements, or respond to user actions like **clicks** and **hover** events.



DOM and JS

The **DOM** can be accessed and manipulated using JavaScript methods and properties, such as **getElementById**, **querySelector**, and **innerHTML**.

This will change the text inside the div element with the id "myDiv" to "I just changed the text!".



```
<div id="myDiv">This is my div.</div>
<script>
  // Get the element with the id "myDiv"
  var myDiv = document.getElementById("myDiv");
  // Change the text inside the div
  myDiv.innerHTML = "I just changed the text!";
</script>
```

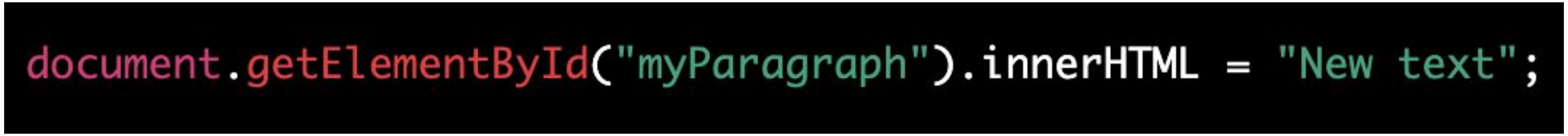
DOM Access Methods

- `getElementById` – search element by element id
- `getElementsByTagName` – search element by tag name (e.g., span, div)
- `getElementsByClassName` – search element by class name
- `getElementsByName` – search element by name attribute
- `querySelector` – returns the first element that matches the specified selector
- `querySelectorAll` – returns elements that match the specified selector

Manipulating DOM Elements

There are several ways to manipulate DOM elements in JavaScript, some of the most common are:

1) Changing the **content** of an element: You can change the text or HTML inside an element by using the **innerHTML** property. For example, to change the text inside a <p> element with the id "myParagraph"



```
document.getElementById("myParagraph").innerHTML = "New text";
```

Manipulating DOM Elements 2

2) Changing the **styles** of an element: You can change the CSS styles of an element by using the style property. For example, to change the background color of a <div> element with the id "myDiv" to red

```
document.getElementById("myDiv").style.backgroundColor = "red";
```

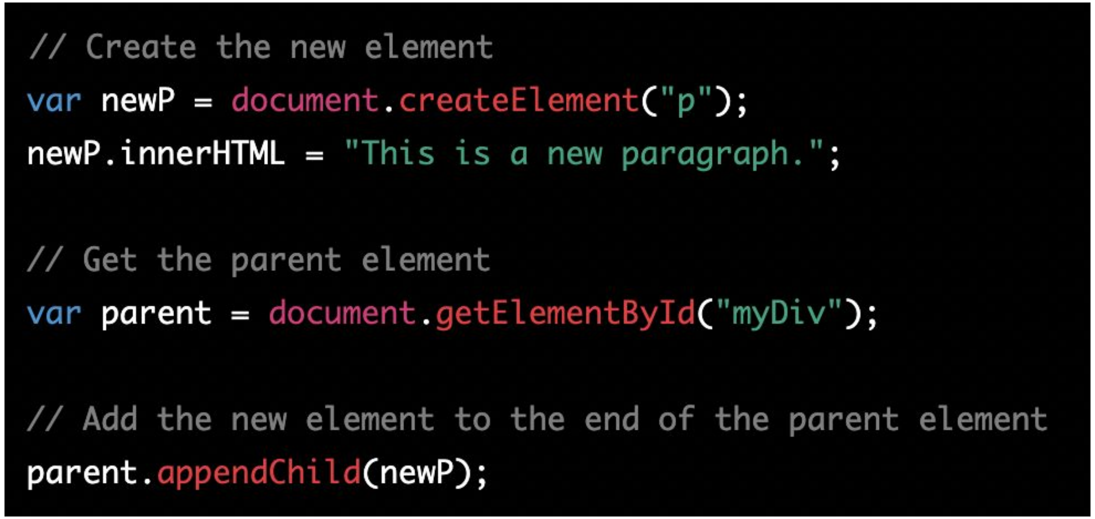
3) **Adding** and **Removing classes**: You can add or remove classes from an element by using the classList property. For example, to add a class "highlight" to a <p> element with the id "myParagraph"

```
document.getElementById("myParagraph").classList.add("highlight");
```

```
document.getElementById("myParagraph").classList.remove("highlight");
```

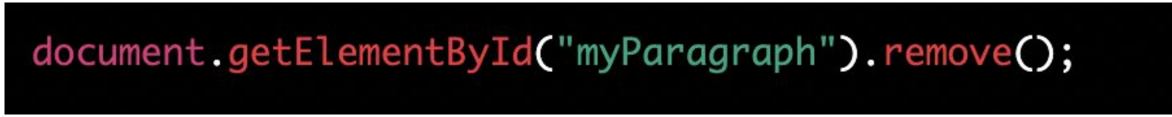
Creating Elements

In JavaScript, you can create new elements in the DOM using the document.createElement() method, and then add them to the page using the appendChild() or insertBefore() methods.



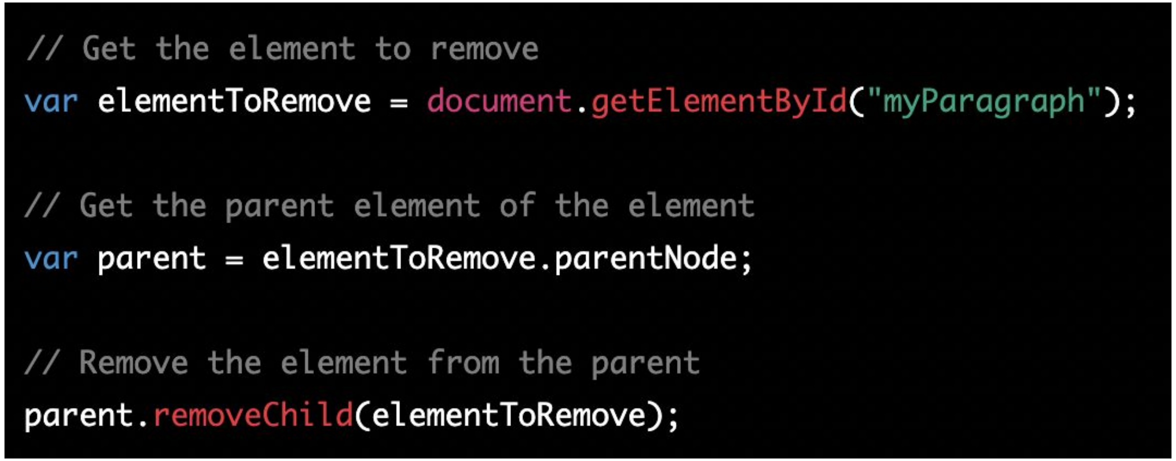
```
// Create the new element
var newP = document.createElement("p");
newP.innerHTML = "This is a new paragraph.";

// Get the parent element
var parent = document.getElementById("myDiv");

// Add the new element to the end of the parent element
parent.appendChild(newP);
```

Removing Elements



```
document.getElementById("myParagraph").remove();
```

To remove an existing element from the DOM, you can use the **removeChild**() method or the remove() method if the browser support it.

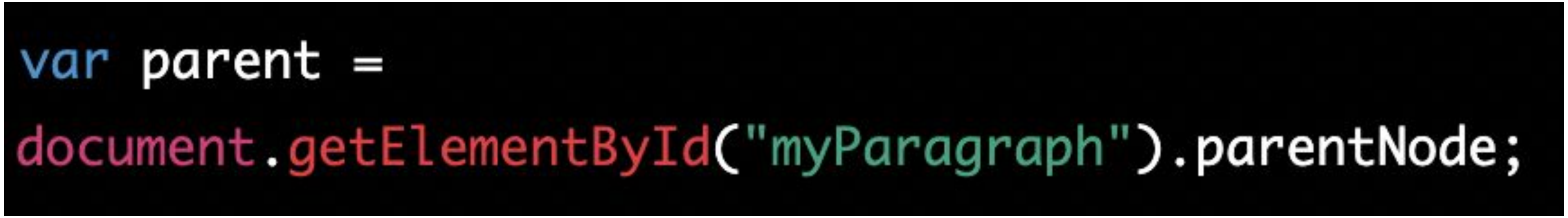


```
// Get the element to remove
var elementToRemove = document.getElementById("myParagraph");

// Get the parent element of the element
var parent = elementToRemove.parentNode;

// Remove the element from the parent
parent.removeChild(elementToRemove);
```

Traversing the DOM

In JavaScript, you can traverse the DOM to access parent, child, and sibling elements using the following properties and methods:

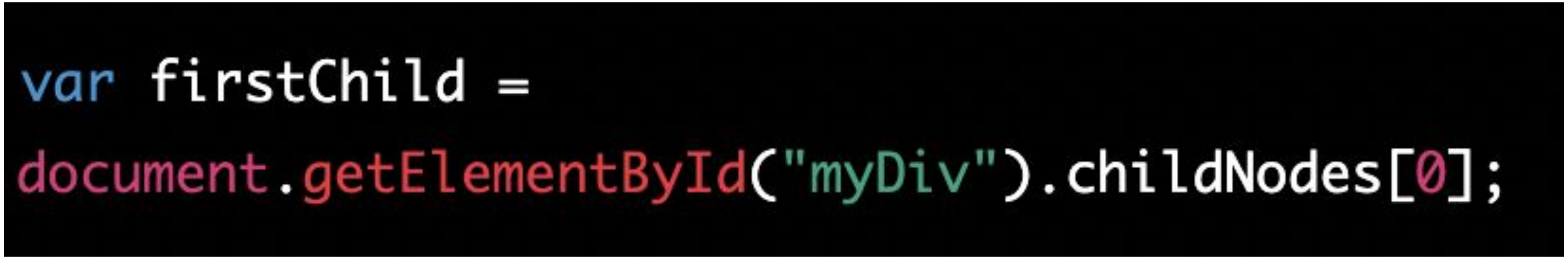
1) **parentNode**: This property returns the parent node of an element. For example, to get the parent element of a <p> element with the id "myParagraph", you can use this code



```
var parent =
document.getElementById("myParagraph").parentNode;
```

Traversing the DOM 2

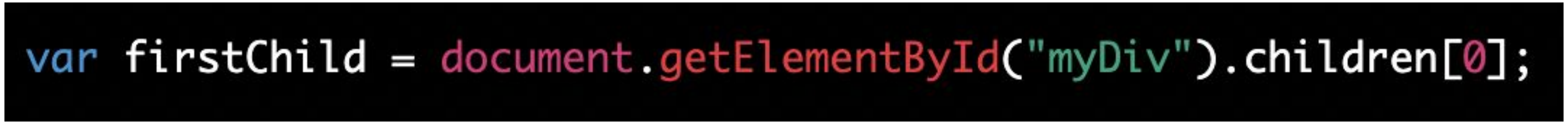
2) **childNodes**: This property returns a NodeList of all the child nodes of an element, including text and comment nodes. To access a specific child element, you can use the childNodes property in combination with the [index] notation. For example, to get the first child element of a <div> element with the id "myDiv", you can use this code



```
var firstChild =
document.getElementById("myDiv").childNodes[0];
```

Traversing the DOM 3

3) **children**: This property returns an HTMLCollection of all the child elements of an element, ignoring text and comment nodes. It's similar to childNodes but it's more efficient. For example, to get the first child element of a <div> element with the id "myDiv", you can use this code

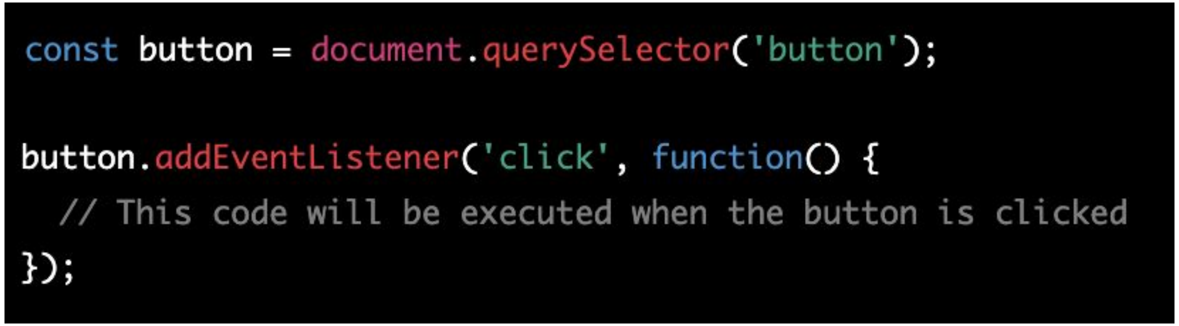


```
var firstChild = document.getElementById("myDiv").children[0];
```

Additional reading materials - https://zellwk.com/blog/dom-traversals/

Event listeners

In JavaScript, events are actions or occurrences that happen in a web page or web application, such as a button being clicked, a mouse being moved over an element, or a page finishing loading. To respond to events in JavaScript, you can use event handlers. An event handler is a function that gets executed when a specific event occurs. You can attach an event handler to an element in your web page using the addEventListener method, like this:



```javascript
const button = document.querySelector('button');

button.addEventListener('click', function() {
  // This code will be executed when the button is clicked
});
```

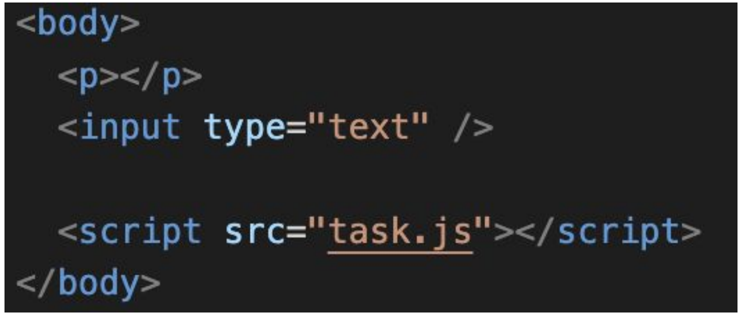
Classwork about dom manipulation

Step 1: Create a simple HTML webpage with a <div> element and a input field. Give the <div> element an id so that you can target it with JavaScript.

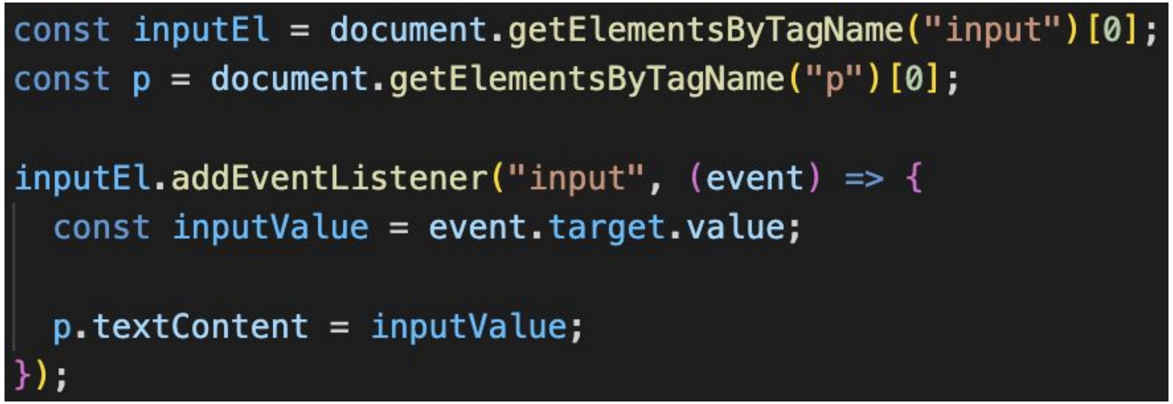
Step 2: Create a JavaScript file and link it to your HTML file. In the JavaScript file, use the getElementById() method to target the <div> element and the input.

Step 3: Create event listener that listens for input field to be changed, then on input change, print input value in some other paragraph element

Classwork about dom manipulation - solution



```html
<body>
  <p></p>
  <input type="text" />

  <script src="task.js"></script>
</body>
```



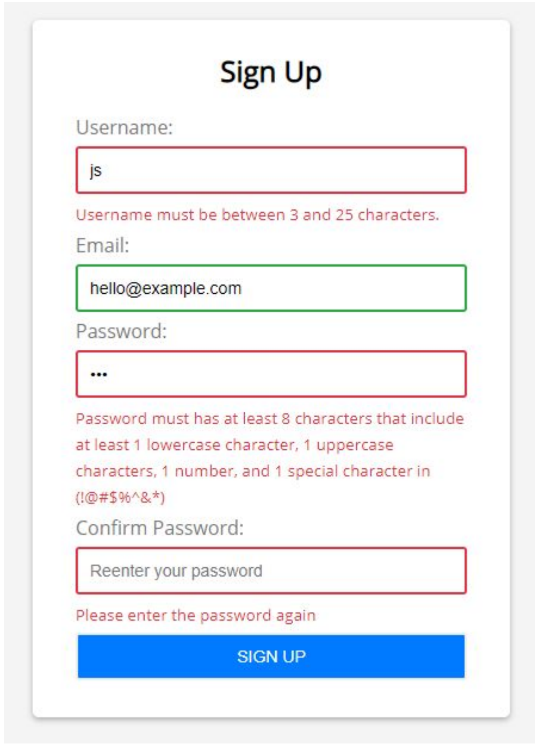
```javascript
const inputEl = document.getElementsByTagName("input")[0];
const p = document.getElementsByTagName("p")[0];

inputEl.addEventListener("input", (event) => {
  const inputValue = event.target.value;

  p.textContent = inputValue;
});
```

Introduction to Form Validation

HTML form validation is a process of checking that the user input in a form is complete, accurate, and secure before it is submitted to the server. It is typically done using JavaScript, and it helps to ensure that the form is filled out correctly and that all required fields are completed.

There are several reasons why HTML form validation is important for web development:

1) **User experience:** Form validation improves the user experience by providing immediate feedback on errors and ensuring that the form is filled out correctly before it is submitted.



Sign Up

Username: js

Username must be between 3 and 25 characters.

Email: hello@example.com

Password: •••

Password must has at least 8 characters that include at least 1 lowercase character, 1 uppercase characters, 1 number, and 1 special character in (!@#$%^&*)

Confirm Password: Reenter your password

Please enter the password again

Introduction to Form Validation

2) **Data integrity**: By validating user input, you can ensure that the data entered into the form is accurate and complete, which helps to maintain the integrity of the data stored on the server.

3) **Security**: Form validation can also help to prevent security risks by detecting and blocking malicious input, such as cross-site scripting (XSS) attacks.

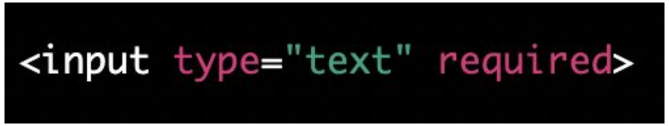
4) **Server load**: By catching errors on the client-side, you can reduce the number of invalid submissions and save server resources.

5) **Accessibility**: Form validation can also help to make forms more accessible by providing clear error messages and highlighting invalid form fields.

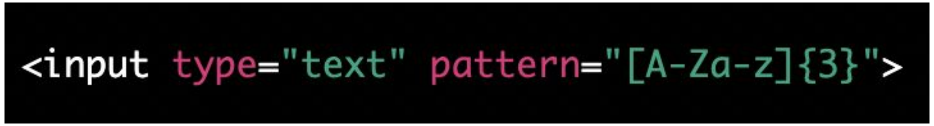
Basic Form Validation

Basic form validation using HTML can be done using the following methods:

1) **required** attribute: This attribute is used on form elements such as <input> and <textarea> to make the field required. If the user leaves the field blank and attempts to submit the form, an error message will be displayed.
2) **pattern** attribute: This attribute is used to specify a regular expression that the input must match. If the input doesn't match the pattern, an error message will be displayed.



```
<input type="text" required>
```

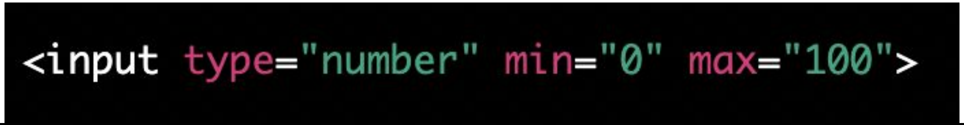


```
<input type="text" pattern="[A-Za-z]{3}">
```

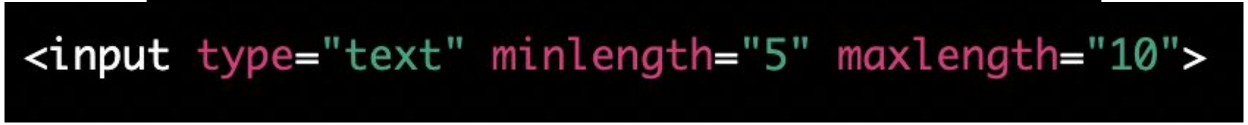
Basic Form Validation 2

3) **min** and **max** attribute: This attribute is used to specify the minimum and maximum value that the input can have. If the input is outside of the specified range, an error message will be displayed.

4) **minlength** and **maxlength** attribute: This attribute is used to specify the minimum and maximum number of characters that the input can have. If the input is outside of the specified range, an error message will be displayed.



```
<input type="number" min="0" max="100">
```

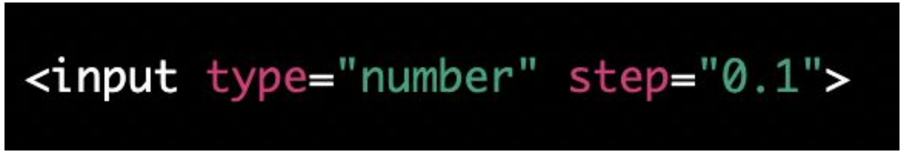


```
<input type="text" minlength="5" maxlength="10">
```

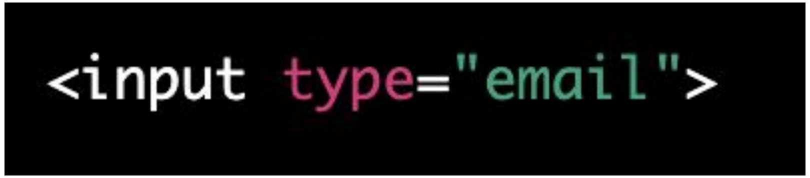
Basic Form Validation 3

5) **type** attribute: This attribute is used to specify the type of input, such as text, email, number, etc. If the input does not match the specified type, an error message will be displayed.

6) **step** attribute: This attribute is used to specify the legal number intervals for an input field of type number. If the input is outside of the specified range, an error message will be displayed.



```
<input type="number" step="0.1">
```



```
<input type="email">
```

Custom Form Validation

```
var form = document.getElementById("myForm");
```

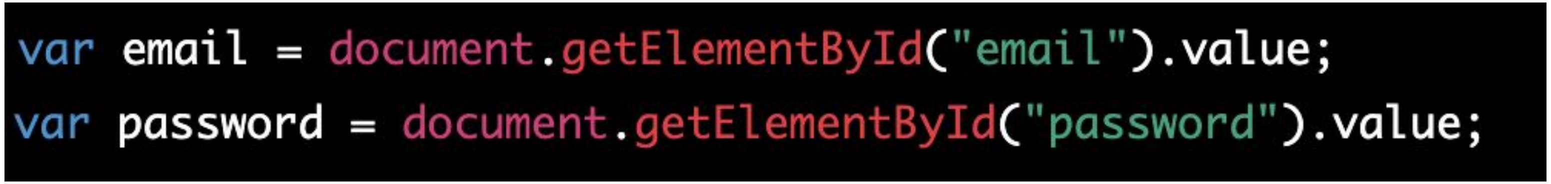
For custom form validation using javascript, you need to prevent default form submit behaviour and apply custom validation. For that you need to:

1) Select the form element using JavaScript. This can be done using the getElementById() method or querySelector() method.
2) Next, you need to add an event listener to the form's submit event. This event listener will be called when the form is submitted, and it will be responsible for performing the validation.

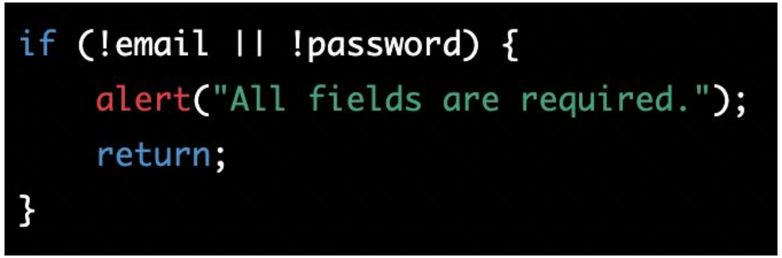
```
form.addEventListener("submit", function(event) {
    event.preventDefault(); // prevent form submission
    // perform validation
});
```

Custom Form Validation 2

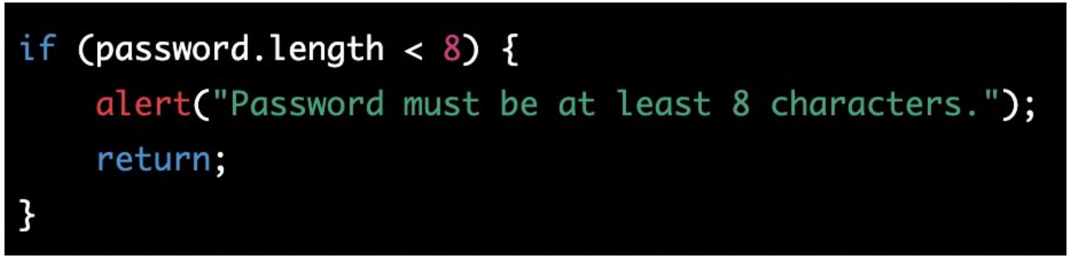
3) Inside the event listener, you can use JavaScript to check the values of the form fields and determine if they are valid. For example, you can check if a required field is filled out, if an email address is in the correct format, or if a password meets a certain complexity requirement.



```
var email = document.getElementById("email").value;
var password = document.getElementById("password").value;
```



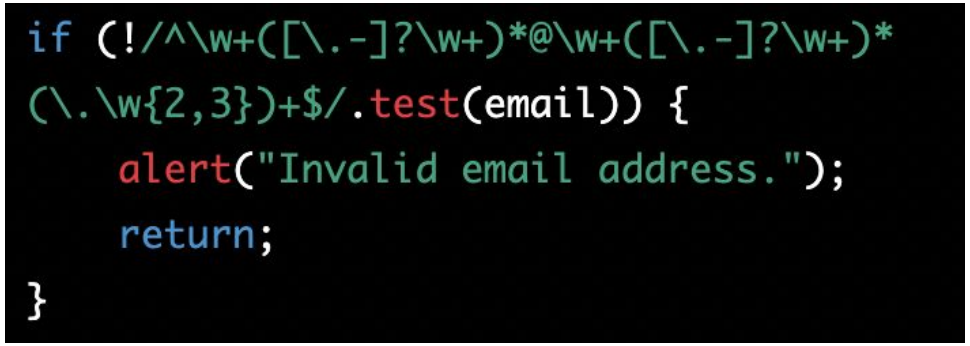
```
if (!email || !password) {
    alert("All fields are required.");
    return;
}
```



```
if (password.length < 8) {
    alert("Password must be at least 8 characters.");
    return;
}
```

Using Regular Expressions

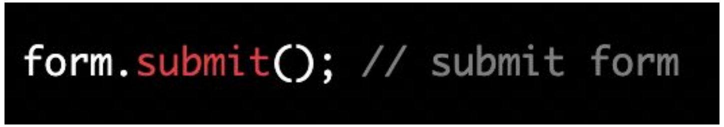
Regex, or regular expressions, are a powerful tool for matching patterns in text. In JavaScript, regular expressions can be used for form validation by testing whether a string of characters entered by a user matches a specific pattern. For example, a regular expression could be used to check that a user has entered a valid email address, phone number, or credit card number. The JavaScript method test() can be used to test whether a string matches a regular expression. If the string matches the pattern defined by the regular expression, test() returns true, otherwise it returns false.



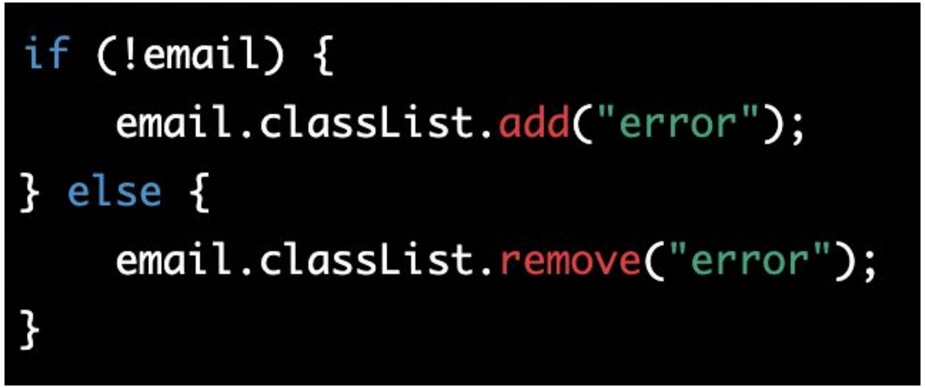
```
if (!/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*
(\.\w{2,3})+$/.test(email)) {
    alert("Invalid email address.");
    return;
}
```

Form Submission

If the validation is successful, you can submit the form, otherwise you can display a message to the user indicating what went wrong. You can also change field styling to visually show to user, what went wrong



```
form.submit(); // submit form
```



```
if (!email) {
    email.classList.add("error");
} else {
    email.classList.remove("error");
}
```

Client-side vs Server-side Validation

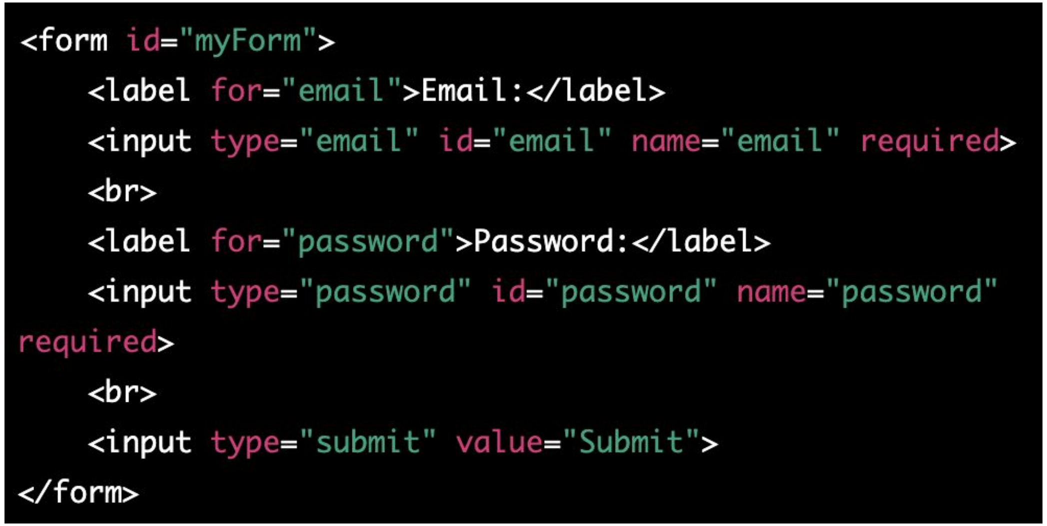
Client-side validation is the process of validating user input on the client (browser) using JavaScript before it is sent to the server. The main advantage of client-side validation is that it provides **immediate feedback** to the user, allowing them to correct errors before they submit the form. This can improve the user experience by reducing the number of round trips to the server. Additionally, it can **reduce the load on the server** by preventing invalid data from being sent in the first place.

Server-side validation, on the other hand, is the process of validating user input on the server using a server-side scripting language. The main advantage of server-side validation is that it can provide a **higher level of security**, as it is not possible for malicious users to bypass the validation by **disabling JavaScript** or **manipulating the client-side code**. Additionally, server-side validation can check for additional constraints that may not be possible to check on the client side, such as checking if a **username is already taken** or if a **credit card number is valid**.

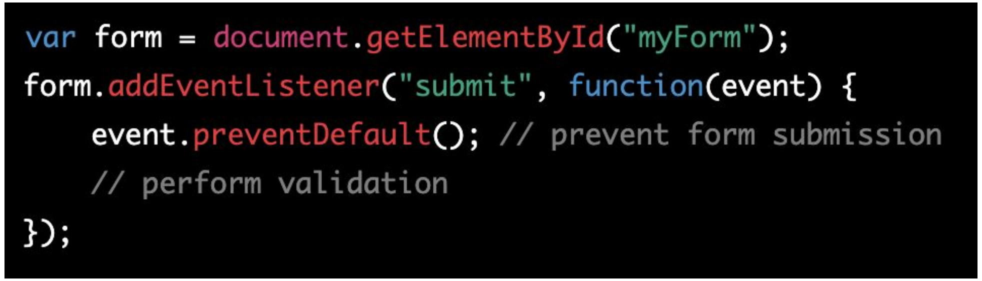
Classwork about form validation

1) Create an HTML form Create an HTML form with two input fields: "email" and "password". Both fields should be required.
2) Add a JavaScript file to your project and include it in your HTML file using a script tag.
3) Inside the JavaScript file, select the form element using the getElementById() method and add an event listener to the form's submit event.
4) Inside the event listener, use JavaScript to check the values of the form fields and determine if they are valid. For example, you can check if the email field is in the correct format using a regular expression and if the password is at least 8 characters long.
5) If the validation is successful, submit the form, otherwise display a message to the user indicating what went wrong.

Classwork about form validation - solution

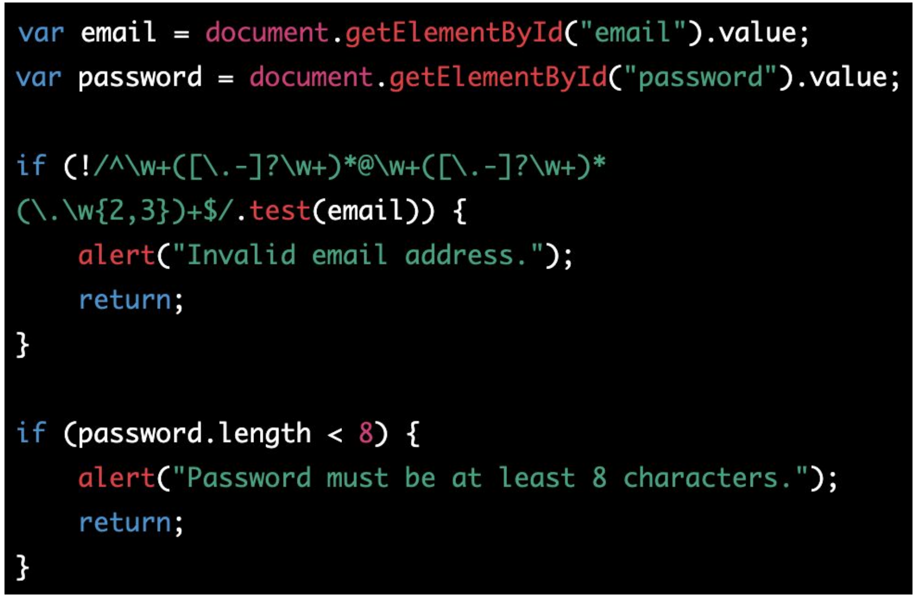


```
<form id="myForm">
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required>
    <br>
    <label for="password">Password:</label>
    <input type="password" id="password" name="password"
required>
    <br>
    <input type="submit" value="Submit">
</form>
```

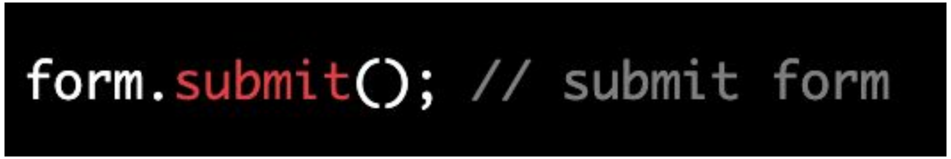


```
var form = document.getElementById("myForm");
form.addEventListener("submit", function(event) {
    event.preventDefault(); // prevent form submission
    // perform validation
});
```

Classwork about form validation - solution 2



```
var email = document.getElementById("email").value;
var password = document.getElementById("password").value;

if (!/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*
(\.\w{2,3})+$/.test(email)) {
    alert("Invalid email address.");
    return;
}

if (password.length < 8) {
    alert("Password must be at least 8 characters.");
    return;
}
```



```
form.submit(); // submit form
```

Final project

Create application using HTML, CSS and JS

Ideas:

1) Calculator application
2) Task manager (form to add new task, list with tasks, task complete and delete functionality)
3) Tic tac toe game
4) Simple blog. The blog should allow users to create new posts and delete posts. The blog should display a list of all posts and allow users to click on a post to read the full content.
5) Create shopping cart. The shopping cart should allow users to add and remove items, adjust quantities, and calculate the total price of the items.