# JavaScript Events

Amr A Khllaf

June 16, 2025

## Contents

# 1 Regex Revision, Flags and replace Method

## 1.1 Regex Revision

- A regular expression (regex) is a sequence of characters that forms a search pattern.
- It is used for string searching and manipulation.
- Common uses include validating input, searching for patterns, and replacing text.
- Regex can be used in various programming languages, including JavaScript, Python, and Java.
- Basic syntax includes:
  - Literal characters: `abc` matches "abc"
  - Metacharacters: `.`, `*`, `+`, `?`, `^`, `$`, `[]`, `()`, `{}`, `|`
  - Character classes: `[abc]` matches any of the characters a, b, or c
  - Quantifiers: `*` (zero or more), `+` (one or more), `?` (zero or one)
  - Anchors: `^` (start of string), `$` (end of string)

---

## 1.2 Regex Flags

- Flags are used to modify the behavior of a regex.
- Common flags include:
  - `g`: Global search (**find all matches**)
  - `i`: Case-insensitive search
  - `m`: Multiline search
- Flags can be combined, e.g., `/pattern/gi` for global and case-insensitive search.

## 1.3 Replace Method (String Method)

```
const str = "Hello World";
const newStr = str.replace(/World/, "JavaScript");
console.log(newStr); // "Hello JavaScript"
```

```
// Using flags with replace
const strWithFlags = "Hello World, hello world";
const newStr = strWithFlags.replace(/world/i, "JavaScript");
console.log(newStr); // "Hello World, JavaScript world"

// Using global flag to replace all occurrences
const strWithGlobal = "Hello World, hello world";
const newStrGlobal = strWithGlobal.replace(/world/gi, "JavaScript");
console.log(newStrGlobal); // "Hello JavaScript, JavaScript"

const strWithGlobal = "Hello World, hello world";
console.log(strWithGlobal.replace(/[a-z]{3}/gi, "JavaScript"));
// "JavaScriptlo JavaScriptld, JavaScriptlo JavaScriptld"
```

---

# 2 What is DOM and How to select multiple elements?

## 2.1 What is DOM?

- The Document Object Model (DOM) is a programming interface for web documents.
- Any HTML element can be represented as a node in the DOM tree.
- Any HTML element used in Js can be represented as Object.
- The DOM allows programming languages to manipulate the structure, style, and content of web pages.
- The DOM represents the document as a tree structure, where each node is an object representing a part of the document.

## 2.2 What is the difference between DOM and BOM?

- **DOM (Document Object Model)**: Represents the structure of the document (HTML or XML) as a tree of objects. It allows manipulation of the document's content and structure.
- **BOM (Browser Object Model)**: Represents the browser's environment and allows interaction with the browser itself (e.g., window, history, location). It provides information about the browser and the user's environment.

## 2.3 What is the Properties and Methods of DOM that can we used from HTML Elements?

- **Properties**:
  - `innerHTML`: Gets or sets the HTML content of an element.
  - `textContent`: Gets or sets the text content of an element.
  - `className`: Gets or sets the class attribute of an element.
  - `id`: Gets or sets the ID of an element.
  - `style`: Gets or sets the inline styles of an element.
- **Methods**:

- `getElementById(id)`: Returns the element with the specified ID.
- `getElementsByClassName(className)`: Returns a collection of elements with the specified class name.
- `getElementsByTagName(tagName)`: Returns a collection of elements with the specified tag name.
- `getElementsByName(name)`: Returns a collection of elements with the specified name attribute.
- `querySelector(selector)`: Returns the first element that matches the specified CSS selector.
- `querySelectorAll(selector)`: Returns a collection of all elements that match the specified CSS selector.
- `createElement(tagName)`: Creates a new element with the specified tag name.
- `appendChild(child)`: Adds a child element to an element.
- `removeChild(child)`: Removes a child element from an element.

## 2.4 How to select multiple elements from HTML?

- To select multiple elements, you can use **methods** like: `getElementsByClassName`, `getElementsByTagName`, `getElementBy Name` or `querySelectorAll`.

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
    ↪  />
    <title>Document</title>
  </head>
  <body>
    <div class="container">
      <input class="form-control" id="demo" />
      <input class="form-control" />
      <input type="radio" name="gender" />
      <input type="radio" name="gender" />
    </div>
  </body>
</html>
```

- **Using `getElementsByTagName`:**

```javascript
// Selecting multiple elements using getElementsByTagName
const inputs = document.getElementsByTagName("input");
console.log(inputs); // HTMLCollection of input elements
// output: HTMLCollection(4) [input#demo, input, input, input]

// Selecting multiple elements using getElementsByTagName
const inputs = document.getElementsByTagName("input");
console.log(inputs); // HTMLCollection of input elements
// output: HTMLCollection(4) [input#demo, input, input, input]
// Converting HTMLCollection to an array
const inputsArray = Array.from(inputs);
console.log(inputsArray); // Array of input elements

var allInputs = document.getElementsByTagName("input");
console.log(allInputs[0]);
// output: <input id="demo">
console.log(allInputs.length);
// output: 4
```

- HTMLCollection is a **live collection**, meaning it updates automatically when the document changes.
- Is HTMlCollection an Array?
- No, HTMLCollection is not an array. It is a collection of elements that can be accessed by index, but it does not have array methods like `forEach`, `map`, etc. To use array methods, you can convert it to an array using `Array.from()` or the spread operator (`[...collection]`).
- So HTMlCollection is not an Array, but it's like array because it's iterable and we can use index to access elements.

---

- **Using `getElementsByClassName`:**

```
// Selecting multiple elements using getElementsByClassName
const formControls = document.getElementsByClassName("form-control");
console.log(formControls); // HTMLCollection of elements with class
↪   "form-control"
// output: HTMLCollection(3) [input.form-control, input.form-control,
↪   input.form-control]

// If i want to select the first element with class "form-control"
const firstFormControl = formControls[0];
console.log(firstFormControl); // <input class="form-control" id="demo">
```

- **Using `getElementByName`:**

```
// Selecting multiple elements using getElementsByName
const genderInputs = document.getElementsByName("gender");
console.log(genderInputs); // "NodeList" of elements with name "gender"
// output: NodeList(2) [input[type=radio][name=gender],
↪   input[type=radio][name=gender]]

// If i want to select the first element with name "gender"
const firstGenderInput = genderInputs[0];
console.log(firstGenderInput); // <input type="radio" name="gender">
```

### 2.4.1 Difference between `HTMLCollection` and `NodeList`?

- `HTMLCollection` is a collection of elements that can be accessed by their **index** or by their **ID (.id)** or **Name(.name)**.
- It is a live collection, meaning it automatically updates when the document changes.

- `NodeList` is a collection of nodes (elements, text nodes, etc.) that can be accessed by **their index**.
- It is not a live collection, meaning it does not automatically update when the document changes.
- `NodeList` can be returned by methods like `querySelectorAll`, while `HTMLCollection` is returned by methods like `getElementsByTagName` and `getElementsByClassName`.
- Both `HTMLCollection` and `NodeList` are iterable, meaning you can use a `for...of` loop to iterate over them.
- You can convert both `HTMLCollection` and `NodeList` to an array using `Array.from()` or the spread operator (`[...collection]`) to use array methods like `forEach`, `map`, etc.

```javascript
const inputsArray = Array.from(inputs);
console.log(inputsArray); // Array of input elements

const genderInputsArray = Array.from(genderInputs);
console.log(genderInputsArray); // Array of gender input elements

const formControlsArray = Array.from(formControls);
console.log(formControlsArray); // Array of form control elements
```

---

## 3 QuerySelector and querySelectorAll & Some Shortcuts

### 3.1 What is `querySelector` and `querySelectorAll`?

- `querySelector` is a method that returns the first element that matches a specified CSS selector.
- `querySelectorAll` is a method that returns all elements that match a specified CSS selector as a NodeList.
- Both methods allow you to use CSS selectors to select elements from the DOM.
- `querySelector` returns a single element, while `querySelectorAll` returns a collection of elements.
- If no elements match the selector, `querySelector` **returns `null`**, while `querySelectorAll` **returns an empty NodeList**.
- In CSS, I'm able to select elements using selectors like:
  - `#id` for an element with a specific ID
  - `.class` for elements with a specific class
  - `tag` for elements of a specific tag
  - `tag.class` for elements of a specific tag with a specific class
  - `tag#id` for an element of a specific tag with a specific ID
  - `tag[attribute=value]` for elements of a specific tag with a specific attribute and value
- `tag1, tag2` for multiple elements of different tags

- `tag.class1.class2` for elements of a specific tag with multiple classes

## 3.2 How to use `querySelector` and `querySelectorAll`?

- They appear in ES5 and later versions of JavaScript.

### 3.2.1 `querySelector` Example

- To select a single element using `querySelector`, you can use the following syntax:

```javascript
// Selecting a single element
const singleElement = document.querySelector("#myId");
console.log(singleElement);

// Selecting an element by class
const elementByClass = document.querySelector(".myClass");
console.log(elementByClass);

// Selecting an element by tag name with nested class
const elementByTagAndClass = document.querySelector("div.myClass");
console.log(elementByTagAndClass);

// Selecting an element using nth-child
const nthChildElement = document.querySelector("ul li:nth-child(2)");
console.log(nthChildElement);

// Selecting an element using attribute selector
const elementByAttribute = document.querySelector("input[name='gender']");
const elementByAttribute = document.querySelector("[name='gender']");
console.log(elementByAttribute);
```

---

### 3.2.2 `querySelectorAll` Example

- To select multiple elements using `querySelectorAll`, you can use the following syntax:

```javascript
// Selecting multiple elements
const multipleElements = document.querySelectorAll(".myClass");
console.log(multipleElements); // NodeList of elements with class
↪ "myClass"
```

## 3.3 Some Shortcuts for `querySelector` and `querySelectorAll`

1. document.body
    - Selects the `<body>` element of the document.
2. document.head
    - Selects the `<head>` element of the document.
3. document.forms
    - Selects all `<form>` elements in the document.

4. document.images
    - Selects all `<img>` elements in the document.
5. document.links
    - Selects all `<a>` elements with an `href` attribute in the document.
6. document.scripts
    - Selects all `<script>` elements in the document.

---

## 4 Add Event Listener Method

### 4.1 What is an Event?

- An event is an action or occurrence that happens in the browser, such as a user clicking a button, submitting a form, or pressing a key.
- Events can be triggered by user interactions or by the browser itself.

### 4.2 What is an Event Listener?

- It's a DOM method that allows you to listen for specific events on HTML elements.
- An event listener is a function that waits for a specific event to occur and executes code in response to that event.
- It allows you to define how your application should respond to user interactions or other events.

### 4.3 How to add an Event Listener?

- To Add Event(Action) on Any HTML Element:
    1. Select the element you want to add the event on it.
    2. Add Event on it.
- You can add an event listener to an element using the `addEventListener` method.
- It took two parameters:
    - The first parameter is the type of event you want to listen for (e.g., "click", "mouseover", "keydown").
    - The second parameter is the function that will be executed when the event occurs.
    - If You want to send a function as a parameter, send the name of the function without parentheses (`functionName`) ==> **Reference**, not `functionName()` ==> **Call or Invoke**.

```html
<button id="myButton">Click me</button>

<script>
  const button = document.querySelector("#myButton");
  button.addEventListener("click", function () {
    alert("Button clicked!");
  });
</script>
```

## 4.4 What is the difference between `onclick` and `addEventListener`?

- `onclick` is a property that can be set to a function to handle click events on an element. It can only handle one event at a time, meaning if you set it again, it will overwrite the previous function.

```
const button = document.querySelector("#myButton");
button.onclick = function () {
  alert("Button clicked!");
};
// If you set it again, it will overwrite the previous function
button.onclick = function () {
  alert("Button clicked again!");
};
```

- `addEventListener` is a method that allows you to add multiple event listeners to an element for the same event type. It does not overwrite previous listeners, allowing you to handle multiple events.

```
const button = document.querySelector("#myButton");
button.addEventListener("click", function () {
  alert("Button clicked!");
});
// You can add another event listener without overwriting the previous
↪   one
button.addEventListener("click", function () {
  alert("Button clicked again!");
});
```

### 4.4.1 Example on String of toUppercase and toLowercase and charAt String Methods

- if i have a string and i want to convert it to uppercase or lowercase, i can use the `toUpperCase()` and `toLowerCase()` methods.
- i want to convert the first letter of a string to uppercase and the rest to lowercase, i can use the `charAt()` method to get the first character and then use `toUpperCase()` and `toLowerCase()` methods.
- Explain slice Method:
  - The `slice()` method is used to extract a section of a string and return it as a new string.
  - It takes two parameters: the starting index and the ending index (optional).
  - If the ending index is not provided, it extracts from the starting index to the end of the string.
  - It does not modify the original string.
- Explain charAt Method:

- The `charAt()` method is used to get the character at a specific index in a string.
- It takes one parameter: the index of the character you want to retrieve.
- If the index is out of bounds, it returns an empty string.
- It does not modify the original string.

```
const str = "hello world";
const upperStr = str.charAt(0).toUpperCase() + str.slice(1).toLowerCase();
console.log(upperStr); // "Hello world"

const str2 = "HELLO WORLD";
const lowerStr = str2.charAt(0).toLowerCase() +
↪  str2.slice(1).toUpperCase();
console.log(lowerStr); // "hELLO WORLD"
```

---

## 5 What if my function takes a parameter?

- If your function takes a parameter, you can pass the parameter to the event listener function when the event occurs.
- Two Parameters for the Add Event Listener Method:
  - The first parameter is the type of event you want to listen for (e.g., “click”, “mouseover”, “keydown”).
  - The second parameter is the function (**Without any name**) that will be executed when the event occurs.

```
function sayHello(name) {
  console.log(`Hello, ${name}!`);
}

const button = document.querySelector("#myButton");
button.addEventListener("click", function () {
  sayHello("John"); // This is happening immediately when the event occurs
});
```

- In this example, when the button is clicked, the `sayHello` function is called with the parameter “John”.
- If you want to pass a parameter to the event listener function, you can use an arrow function or an anonymous function to call your function with the desired parameter.

```
const button = document.querySelector("#myButton");
button.addEventListener("click", () => {
  sayHello("John"); // This is happening immediately when the event occurs
});
```

---

## 6 Many Events on One Element

- You can add multiple event listeners to the same element for different events or even the same event type.

```
let h2 = document.querySelector("h2");
h2.addEventListener("click", () => {
  console.log("H2 clicked!");
  h2.style.color = "blue";
  h2.addEventListener("mouseover", () => {
    h2.style.color = "red";
  });
  h2.addEventListener("dblclick", () => {
    console.log("H2 double-clicked!");
    h2.style.color = "green";
  });
});
```

- Another Example:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
    ↪  />
    <title>Week 10</title>
    <link rel="stylesheet" href="css/all.min.css" />
    <link rel="stylesheet" href="css/bootstrap.min.css" />
    <link rel="stylesheet" href="css/style.css" />
  </head>
  <body>
    <div class="test"></div>
    <script src="js/all.min.js"></script>
    <script src="js/bootstrap.bundle.min.js"></script>
    <script src="js/main.js"></script>
  </body>
</html>
```

```
.test {
  height: 400px;
  width: 400px;
  background-color: orange;
}
```

### 6.1 We have 3 Events We can Use by the mouse :

1. **mouseenter**: This event is triggered when the mouse pointer enters the element.
2. **mouseleave**: This event is triggered when the mouse pointer leaves the element.
3. **mousemove**: This event is triggered when the mouse pointer is moved within the element.

- Note : the Hove can be done in two ways:

1. Using `mouseenter` and `mouseleave` events to change the background color when the mouse enters or leaves the element ( Best Practice).
2. Using `mousemove` event to change the background color when the mouse moves over the element.

```
let myDiv = document.querySelector("div.test");
// Selecting the div with class "test"

myDiv.addEventListener("mouseenter", () => {
  myDiv.style.backgroundColor = "blue";
});

myDiv.addEventListener("mouseleave", () => {
  myDiv.style.backgroundColor = "red";
});

myDiv.addEventListener("mousemove", () => {
  console.log("Mouse is moving over the div"); // Log message when mouse
   ↪  moves over the div
  myDiv.style.backgroundColor = "green";
});
```

---

## 7 Setting One Event On Many Elements

- You can set one event on many elements by using a loop to iterate over the elements and add the event listener to each one.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
     ↪  />
    <title>Week 10</title>
    <link rel="stylesheet" href="css/all.min.css" />
    <link rel="stylesheet" href="css/bootstrap.min.css" />
    <link rel="stylesheet" href="css/style.css" />
  </head>
  <body>
    <h2>JavaScript Events</h2>
    <h2>JavaScript Events</h2>
    <h2>JavaScript Events</h2>
    <h2>JavaScript Events</h2>
    <h2>JavaScript Events</h2>
    <h2>JavaScript Events</h2>
    <h2>JavaScript Events</h2>
    <h2>JavaScript Events</h2>

    <script src="js/all.min.js"></script>
    <script src="js/bootstrap.bundle.min.js"></script>
    <script src="js/main.js"></script>
  </body>
</html>
```

```
let h2Elements = document.querySelectorAll("h2"); // NodeList of all h2
↪   elements

// Using a for loop to add event listener to each h2 element
for (let i = 0; i < h2Elements.length; i++) {
  h2Elements[i].addEventListener("click", () => {
    console.log(`H2 element ${i + 1} clicked!`); // Log message when an h2
      ↪   element is clicked
    h2Elements[i].style.color = "blue"; // Change the color of the clicked
↪   h2 element to blue
  });
}

---



// Using forEach to add event listener to each h2 element
h2Elements.forEach((h2, index) => {
  h2.addEventListener("click", () => {
    console.log(`H2 element ${index + 1} clicked!`); // Log message when an
      ↪   h2 element is clicked
    h2.style.color = "blue"; // Change the color of the clicked h2 element
↪   to blue
  });
});
```

```
// If i click on the even tags it will print hi & if it click on the odd
↪   tags it will print bye


// Using a for loop to add event listener to each h2 element
let h2Elements = document.querySelectorAll("h2"); // NodeList of all h2
↪   elements

for (let i = 0; i < h2Elements.length; i++) {
  h2Elements[i].addEventListener("click", () => {
    if (i % 2 === 0) {
      console.log("Hi");
    } else {
      console.log("Bye");
    }
  });
}

---
```

```javascript
// Using forEach to add event listener to each h2 element
h2Elements.forEach((h2, index) => {
  h2.addEventListener("click", () => {
    if (index % 2 === 0) {
      console.log("Hi");
    } else {
      console.log("Bye");
    }
  });
});
```

---

## 8 Styling Elements with JavaScript

- Use the `.style` property to set inline styles.
- CSS properties are camelCase in JavaScript (e.g., `backgroundColor`, `fontSize`).
- Values must be strings.

**Example:**

```javascript
let myDiv = document.querySelector("div.test");
myDiv.addEventListener("mouseenter", () => {
  myDiv.style.backgroundColor = "blue";
});
myDiv.addEventListener("mouseleave", () => {
  myDiv.style.backgroundColor = "red";
});
myDiv.addEventListener("mousemove", () => {
  myDiv.style.backgroundColor = "green";
});
```

---

## 9 Event Object Information

- When an event occurs, the browser creates an **event object** with information about the event.
- The event object is passed as an argument to the handler.

**Example:**

```javascript
document.querySelector("div").addEventListener("click", (event) => {
  console.log(event); // The event object
  console.log(event.target); // The element that was clicked
  console.log(event.type); // Event type (e.g., "click")
  console.log(event.clientX, event.clientY); // Mouse coordinates
  console.log(event.currentTarget); // The element the listener is
   ↪ attached to
});
```

- `event.target`: The element that triggered the event.

- `event.currentTarget`: The element the listener is attached to.
- `event.clientX`/`event.clientY`: Mouse coordinates relative to the viewport.

**Hide clicked element:**

```
document.querySelector("div").addEventListener("click", (event) => {
  event.target.style.display = "none";
});
```

---

## 10 Event Types and Event Information

- **Mouse Events:** `click`, `dblclick`, `mousemove`, `mouseenter`, `mouseleave`
- **Keyboard Events:** `keydown`, `keyup`, `keypress`
- **Form Events:** `focus`, `blur`, `change`

**Form Example:**

```
document.querySelector("input").addEventListener("focus", (event) => {
  console.log("Input focused:", event.target);
});
document.querySelector("input").addEventListener("blur", (event) => {
  console.log("Input blurred:", event.target);
});
document.querySelector("input").addEventListener("change", (event) => {
  console.log("Input changed:", event.target);
});
```

**Keyboard Example:**

```
document.addEventListener("keydown", (event) => {
  console.log("Key down:", event.key, event.code);
  if (event.key === "Enter") {
    console.log("Enter key pressed");
  }
  if (event.code === "Escape") {
    console.log("Escape key pressed");
  }
  if (event.key === "a") {
    console.log("A key pressed");
  }
});
```

- `event.key`: The value of the key pressed (e.g., “Enter”, “a”).
- `event.code`: The physical key (e.g., “KeyA”, “Enter”).
- `event.keyCode`: Deprecated; use `event.key` or `event.code`.

---

## 11 Random Background Color Task

- The **Math** object provides methods for generating random numbers.

- `Math.random()`: Returns a random float between 0 (inclusive) and 1 (exclusive).
- To generate a random color, use `Math.random()` for red, green, and blue values (0–255).

**Useful Math methods:**

- `Math.random()`: It returns a random float between 0 (inclusive) and 1 (exclusive).
- `Math.floor()`: It returns the largest integer less than or equal to a given number.
- `Math.ceil()`: It returns the smallest integer greater than or equal to a given number.
- `Math.round()`: It returns the value of a number rounded to the nearest integer.
- `Math.max()`: It returns the largest of zero or more numbers.
- `Math.min()`: It returns the smallest of zero or more numbers.
- To get a random integer from 0 to 255: `Math.floor(Math.random() * 256)`
- **Example:** on the methods:

```
console.log(Math.random()); // Random float between 0 and 1
console.log(Math.floor(Math.random() * 256)); // Random integer between 0
↪  and 255
console.log(Math.ceil(4.2)); // 5
console.log(Math.round(4.5)); // 5
console.log(Math.floor(4.9)); // 4
console.log(Math.max(1, 2, 3)); // 3
console.log(Math.min(1, 2, 3)); // 1
```

- **Random Background Color Example:**

```
document.addEventListener("keydown", (event) => {
  console.log("Key down:", event.key);
  if (event.code == "Space") {
    let r = Math.floor(Math.random() * 256);
    let g = Math.floor(Math.random() * 256);
    let b = Math.floor(Math.random() * 256);
    document.body.style.backgroundColor = `rgb(${r}, ${g}, ${b})`;
    console.log(`Background color changed to rgb(${r}, ${g}, ${b})`);
  }
});
```

---

## 12 Class List

- The `classList` property provides methods to manipulate the classes of an element.
- It allows you to add, remove, toggle, and check for classes without affecting other classes.
- Common methods include:
  - `classList.add(className)`: Adds a class to the element.
  - `classList.remove(className)`: Removes a class from the element.
  - `classList.toggle(className)`: Toggles a class on or off.
  - `classList.contains(className)`: Checks if the element has a specific class.

- `classList.replace(oldClass, newClass)`: Replaces an old class with a new class.
- `classList.value`: Returns a string of all classes.
- `classList.length`: Returns the number of classes.
- `classList.item(index)`: Returns the class at the specified index.
- `classList.forEach(callback)`: Executes a callback function for each class.
- **Note**: `this` keyword is refer to the element that the event listener is attached to.

```
document.querySelector("div").addEventListener("click", function () {
  this.classList.toggle("active"); // Toggle the "active" class on the
  ↪  clicked div
  console.log(this.classList); // Log the class list of the clicked div
  console.log(this.classList.contains("active")); // Check if the "active"
  ↪  class is present
  console.log(this.classList.value); // Get all classes as a string
  console.log(this.classList.length); // Get the number of classes
});
```

## 12.1 What is the class and what is the classlist

- **Class**: A class is a name given to a group of elements in HTML. It is defined in the HTML document using the `class` attribute. Classes are used to apply CSS styles and to select elements in JavaScript.
- **classList**: The `classList` property is a read-only property that returns a live `DOMTokenList` collection of the class attributes of the element. It provides methods to manipulate the classes of an element without affecting other classes.
- The `classList` property allows you to add, remove, toggle, and check for classes without affecting other classes. It is a convenient way to work with classes in JavaScript.
- The `classList` property is available on all HTML elements and provides a simple way to manage classes.
- The `classList` property is a part of the DOM API and is supported in all modern browsers.
- The `classList` property is useful for dynamically changing the appearance of elements based on user interactions or other events.
- **Example of class and classList**:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
    ↪  />
    <title>Class and ClassList Example</title>
    <style>
      .active {
        background-color: yellow;
      }
    </style>
  </head>
  <body>
    <div class="box">Click me to toggle class</div>

    <script>
      const box = document.querySelector(".box");
      box.addEventListener("click", function () {
        this.classList.toggle("active"); // Toggle the "active" class
        console.log(this.classList); // Log the class list
      });
    </script>
  </body>
</html>
```

- if i want to add many classes once :

```
this.classList.add("class1", "class2", "class3");
```

- if i want to remove many classes once :

```
this.classList.remove("class1", "class2", "class3");
```

---

## 13 getAttribute and setAttribute Methods & this Vs. e.target

### 13.1 What is getAttribute and setAttribute?

- They are DOM Methods
- `getAttribute(attributeName)`: This method retrieves the value of the specified attribute from an element.
- `setAttribute(attributeName, value)`: This method sets the value of the specified attribute on an element.
- Both methods are used to manipulate attributes of HTML elements in JavaScript.
- They are useful for dynamically changing attributes like `src`, `href`, `class`, etc.
- `getAttribute` returns the value of the attribute as a string, while `setAttribute` sets the value of the attribute to the specified value.
- If the attribute does not exist, `getAttribute` returns `null`, and `setAttribute` creates the attribute with the specified value.

```
let img = document.querySelector("img");
img.addEventListener("click", function () {
  console.log(this.getAttribute("src")); // Get the value of the "src"
    ↪ attribute
  this.setAttribute("src", "new-image.jpg"); // Set a new value for the
    ↪ "src" attribute
});
```

### 13.1.1 Difference between this.src & getAttribute("src")

- `this.src`: This property directly accesses the `src` attribute of the element. It returns the full URL of the image.
- `getAttribute("src")`: This method retrieves the value of the `src` attribute as it is defined in the HTML. It returns the relative path or the full URL depending on how it was set.

```
let img = document.querySelector("img");
img.addEventListener("click", function () {
  console.log(this.src); // Full URL of the image (Absolute Path)
  console.log(this.getAttribute("src")); // Full URL as defined in HTML
   ↪  (Relative path)
});
```

- **Example**: If i want to make a lot of images and when i click on the image, the image will change

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
    ↪  />
    <title>Image Change Example</title>
  </head>
  <body>
    <img src="image1.jpg" alt="Image 1" />
    <img src="image2.jpg" alt="Image 2" />
    <img src="image3.jpg" alt="Image 3" />
  </body>
</html>
```

```
let images = document.querySelectorAll("img"); // Select all images
// This is by forEach method
images.forEach((img) => {
  img.addEventListener("click", function () {
    this.setAttribute("src", "new-image.jpg");
  });

  // This is by for loop method
  for (let i = 0; i < images.length; i++) {
    console.log(images[i]); // Log each image element = <img
    ↪  src="image1.jpg" alt="Image 1" />
    console.log(images[i].getAttribute("src")); // Log the current src
    ↪  attribute value = "image1.jpg"
    images[i].addEventListener("click", function () {
      this.setAttribute("src", "new-image.jpg");
    });
  }
});
```

## 13.2 What is the difference between `this` and `e.target`?

- `this`: Refers to the **selected element** that the event listener is attached to. It is used within the context of the event handler function.
- `e.target`: Refers to the element that triggered the event. It is used to access the specific element that caused the event to fire.
- `this` is used in the context of the event listener, while `e.target` is used to refer to the element that triggered the event, which may be different if the event bubbles up from a child element.

```
document.querySelector("div").addEventListener("click", function (e) {
  console.log(this); // Refers to **the div element** that the event
  ↪  listener is attached to
  console.log(e.target); // Refers to the element that triggered the event
  ↪  (could be a child element)
  console.log(this === e.target); // true if the clicked element is the
  ↪  same as the div
});
```

---

# 14 Summary

- The DOM (Document Object Model) is a programming interface for web documents that represents the structure of the document as a tree of objects.
- The BOM (Browser Object Model) represents the browser's environment and allows interaction with the browser itself.
- The `addEventListener` method is used to attach event handlers to HTML elements, allowing you to respond to user interactions.
- The `querySelector` and `querySelectorAll` methods allow you to select elements using CSS selectors, with `querySelector` returning the first match and `querySelectorAll` returning all matches.
- The `classList` property provides methods to manipulate the classes of an element, allowing you to add, remove, toggle, and check for classes.
- The `getAttribute` and `setAttribute` methods are used to retrieve and set attributes of HTML elements.
- The `this` keyword refers to the element that the event listener is attached to, while `e.target` refers to the element that triggered the event.
- The `Math` object provides methods for generating random numbers, which can be used to create dynamic effects like changing background colors.
- Event objects provide information about the event, such as the type of event, the target element, and mouse coordinates.