Web-Programming-Series-1-Important-Topics

For more notes visit

https://rtpnotes.vercel.app

1. Basic HTML Tags and Their Examples (Any 4)

HTML (Hypertext Markup Language) uses **tags** to structure content on a webpage. Each tag tells the browser how to display the content.

1. html Tag

- What It Does: Defines the beginning and end of an HTML document.
- Use: Wraps all the content on the page.

Syntax

```
<html>
  <!-- All content of the webpage goes here -->
</html>
```

2. head Tag

- **What It Does**: Contains meta-information about the webpage, like its title and links to stylesheets or scripts.
- **Use**: It doesn't display content directly on the page, but it helps set up the page's structure.

Example:

```
<head>
        <title>My Webpage</title>
```

```
        <link rel="stylesheet" href="styles.css">
</head>
```

3. title Tag

- **What It Does**: Defines the title of the webpage that appears in the browser tab.

Example

```
<title>My First Webpage</title>
```

4. body Tag

- **What It Does**: Contains the visible content of the webpage. Everything inside the `<body>` tag appears on the screen.

Example:

```
<body>
        <h1>Welcome to My Webpage</h1>
        <p>This is a paragraph of text.</p>
</body>
```

5. h1 to h6 Tags

- **What It Does**: Headings that define the title or section headers. `<h1>` is the largest (most important), and `<h6>` is the smallest.

Example:

```
<h1>Main Heading</h1>
<h2>Subheading</h2>
<h3>Smaller Subheading</h3>
```

6. p Tag

- **What It Does**: Defines a paragraph of text.

Example:

```
<p>This is a simple paragraph of text.</p>
```

7. a (Anchor) Tag

- **What It Does**: Creates a hyperlink to another webpage or location.

Example:

```
<a href="https://www.example.com">Click here to visit Example.com</a>
```

`href`: The URL to which the link points.

8. img Tag

- **What It Does**: Embeds an image in the webpage.

Example

```
<img src="image.jpg" alt="A beautiful scenery" width="400" height="300">
```

- `src`: The source (file path) of the image.
- `alt`: Alternative text for the image (important for accessibility).
- `width` **and** `height`: Dimensions of the image.

9. ul, ol, and li Tags

- **What It Does**: Creates lists. `<ul>` is for unordered (bullet) lists, and `<ol>` is for ordered (numbered) lists. `<li>` represents each list item.

Unordered List (bullets):

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```

Ordered List (numbers):

```
<ol>
  <li>First Item</li>
  <li>Second Item</li>
  <li>Third Item</li>
</ol>
```

10. div Tag

- **What It Does**: A container used to group HTML elements together, often for styling purposes with CSS.

```
<div>
  <h2>Section Title</h2>
  <p>This is some content inside a div.</p>
</div>
```

11. span Tag

- **What It Does**: A smaller container for inline elements (like text), often used to style or manipulate a specific part of the text.
  Example

```
<p>This is <span style="color:red;">important</span> text.</p>
```

This is important text.

12. form Tag

- **What It Does**: Creates a form to collect user input, such as text fields, checkboxes, and buttons.
  Example

```
<form action="/submit" method="POST">
  <label for="name">Name:</label>
  <input type="text" id="name" name="name">
  <input type="submit" value="Submit">
</form>
```

Name: Submit

13. br Tag

- **What It Does**: Inserts a line break (used to move text to the next line).

Example

```
<p>This is the first line.<br>This is the second line.</p>
```

This is the first line.
This is the second line.

14. strong and em Tags

- **What It Does**: These tags are used for emphasizing text. `<strong>` makes text bold, and `<em>` makes text italic.

Example

```
<p>This is <strong>bold text</strong> and this is <em>italic text</em>.</p>
```

This is **bold text** and this is *italic text*.

15. button Tag

- **What It Does**: Creates a clickable button.

Example

```
<button type="button">Click Me</button>
```

Click Me

2. Url, Parts of URL

Url

A URL is the address of a resource on the web. It points to a specific location on the internet, like a webpage or a file.

Parts of URL

- **Scheme/Protocol** (`https://` or `http://`):
  - Defines how data is transferred between the browser and the server.
  - Common protocols:
    - `https://` : Secure communication (encrypted)
    - `http://` : Insecure communication (not encrypted)
  - Other protocols: `ftp://` (for files), `mailto:` (for emails)
- **Domain Name** (`www.example.com`):
  - The human-readable name of the website.
  - It translates into an IP address that computers use to locate servers.
  - Structure:
    - `www` : Subdomain (optional)
    - `example` : Domain name (company or website name)
    - `.com` : Top-Level Domain (TLD), examples: `.org` , `.net` , `.edu`
- **Port** (`:8080` or `:443`):
  - Specifies the server port to connect to.

    - Common ports:
        - `:80`: Default for `http`
        - `:443`: Default for `https`
    - If omitted, defaults are used based on the protocol.
- **Path** (`/about-us`):
    - Specifies the exact resource (page or file) on the server.
    - Can represent directories and files (e.g., `/products/shoes`).
    - If no path is provided, the default page (often `index.html`) is loaded.
- **Query String** (`?search=shoes&sort=price`):
    - Starts with a `?`, followed by key-value pairs (`key=value`).
    - Multiple parameters are separated by `&`.
    - Used to send additional information to the server, like search terms or filters.
- **Fragment/Anchor** (`#section2`):
    - Starts with `#`, and points to a specific part of a webpage.
    - Allows the browser to jump to a section (like a heading or paragraph) without reloading the page.

Example

URL: `https://www.example.com:8080/products/shoes?search=running#details`

- **Scheme/Protocol**: `https://` (secure)
- **Domain**: `www.example.com`
- **Port**: `:8080`
- **Path**: `/products/shoes`
- **Query String**: `?search=running`
- **Fragment**: `#details`

3. HTTP-Methods

HTTP methods are ways in which a client (like a web browser) communicates with a server.

1. HEAD Method

- **Purpose**: Retrieve the headers of a resource, but not the actual content (like a preview).
- **How It Works**: When you use the HEAD method, the server sends back the headers (like metadata: file size, last modified date) but **doesn’t send the content** (the body of the file or webpage).
- **Use Case**: Check if a file exists or get information about a webpage without downloading the entire content. For example, a browser might check if a page has been updated without loading the whole page.

Example:
You want to see if an image file exists on a server, but you don't want to download the image itself. You use a HEAD request to just check its metadata.

2. PUT Method

- **Purpose**: Upload or **replace** a resource on the server.
- **How It Works**: PUT is used when you want to upload a file or data to a specific location (URL) on the server. If the resource already exists, PUT will **replace it** with the new data. If it doesn't exist, PUT can create it.
- **Use Case**: Useful for file uploads or updating a resource at a specific location.

Example:
If you’re updating a document or file on the server (say you want to upload a new version of your profile picture), you'd use a PUT request to replace the old one with the new one.

3. POST Method

- **Purpose**: Send data to the server to **create** or **process** a resource.
- **How It Works**: POST sends data to the server, and the server processes it (e.g., creating a new resource, submitting a form, or saving information). Unlike PUT, POST **doesn't replace** a resource but often creates a new one or triggers an action on the server.
- **Use Case**: Used for submitting forms, uploading data, or interacting with the server in a way that doesn't involve replacing an entire file.

Example:
When you sign up for a website or submit a form (like login credentials), you're using a POST request to send your data to the server for processing.

4. Integrate CSS with HTML

- CSS (Cascading Style Sheets) is used to control the appearance and layout of a webpage written in HTML. While HTML structures the content, CSS styles it.
- There are three main ways to integrate CSS with HTML: **Inline CSS**, **Internal CSS**, and **External CSS**.

1. Inline CSS

- **What It Is**: CSS styles are applied directly to HTML elements using the `style` attribute.
- **When to Use**: Useful for small, specific style changes to a single element.

Example:

```
<p style="color: blue; font-size: 18px;">This is a blue paragraph.</p>
```

In this example, the paragraph text will be blue and have a font size of 18px.

This is a blue paragraph.

- Not efficient for styling multiple elements.
- Can make the HTML code messy and hard to maintain.

2. Internal CSS

- **What It Is**: CSS rules are written inside the `<style>` tag, within the `<head>` section of the HTML file.
- **When to Use**: Useful for styling a single page, especially if the styles are unique to that page.

Example:

```
<html>
<head>
  <style>
    p {
      color: green;
      font-size: 16px;
    }
    h1 {
      text-align: center;
    }
  </style>
</head>
<body>
  <h1>Welcome to My Webpage</h1>
  <p>This is a green paragraph.</p>
</body>
</html>
```

In this example, the CSS styles are applied to all `<p>` and `<h1>` tags on the page.

Pros:

- Easier to manage than inline styles.
- Keeps styles separate from content but still in the same file.

Cons:

- Limited to the specific HTML file.
- Not reusable across multiple pages.

3. External CSS

- **What It Is**: CSS is stored in a separate file (with a `.css` extension), and the HTML links to this file using the `<link>` tag in the `<head>` section.
- **When to Use**: Ideal for larger websites where styles are shared across multiple pages.

Example (HTML):

```html
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <h1>Welcome</h1>
  <p>This is a styled paragraph.</p>
</body>
</html>
```

Example (styles.css)

```css
h1 {
  color: red;
  font-size: 24px;
}
p {
  color: black;
  font-size: 14px;
}
```

Here, the CSS is stored in `styles.css`, and it's applied to all HTML files that link to this stylesheet.

Pros:

- Keeps HTML clean and focused on structure.
- One CSS file can style multiple pages.
- Easier to maintain and update styles across the website.

Cons:

- Requires an extra file, meaning the browser has to load another resource.

CSS Integration Summary

Method	Where Styles Are Written	Use Cases	Pros	Cons
Inline CSS	Inside HTML elements (`style` attribute)	Quick, one-off styling for specific elements	Easy to apply	Messy, hard to maintain
Internal CSS	In the `<style>` tag inside `<head>`	Styling specific to a single page	Centralized in one file	Can't reuse across pages
External CSS	In a separate `.css` file	Best for styling multiple pages	Reusable, clean HTML	Requires an additional file

5. HTML ID and class attributes

In HTML, the **ID** and **Class** attributes help you identify and style elements on a webpage. Think of them as "labels" you can attach to parts of your page to make them easier to work with.

ID Attribute

- **What It Is**: A unique identifier for a specific element on your webpage.
- **How It Works**: An ID is like a **name tag** for an element. It's meant to be **unique**, meaning that only one element on the page should have a particular ID.
- **How to Use**: You use the `id` attribute to single out an element that you might want to style or interact with using JavaScript.
- **Syntax**: `<element id="unique-name">`

Why Use It:

- **Target with CSS**: You can style this specific element with CSS.
- **JavaScript Access**: You can use JavaScript to find or manipulate the element.

Example

```
<p id="intro">This is the introduction paragraph.</p>
```

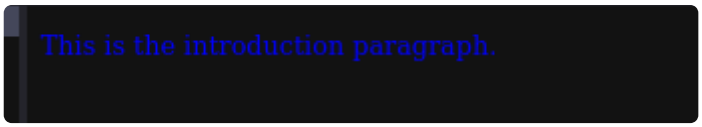
- Here the `<p>` element has an id called "intro"

- We can use this id to style using CSS

```
#intro { color: blue; }
```

- To refer elements by ID in css, we use `#` and the id

Output



Class Attribute

- **What It Is**: A way to group similar elements together.
- **How It Works**: A class is like a **category** or **group** name. Multiple elements on a webpage can share the same class, which makes it easier to apply the same styling to all of them.
- **How to Use**: You use the `class` attribute to style multiple elements or give them similar behavior.
- **Syntax**: `<element class="group-name">`

Why Use It:

- **Target with CSS**: You can apply the same style to all elements with the same class.
- **Reuse Classes**: You can use the same class for many elements.

Example

```
<p class="highlight">This text will be highlighted.</p>
<p class="highlight">This text will also be highlighted.</p>
```

```
.highlight { background-color: yellow; }
```

- In CSS, the `.` symbol targets all elements with the `"highlight"` class.
- So The background color of elements with "highlight" class will be yellow

Output

6. Tables in HTML

Tables in HTML are used to display data in a structured format of rows and columns. They are created using the `<table>` element, along with various child elements such as:

- `<tr>` (table row): Defines a row in the table.
- `<th>` (table header): Defines a header cell, typically used for labels.
- `<td>` (table data): Defines a standard data cell.
  Here's an example of a simple HTML table:

```
<table>
  <tr>
    <th>Header 1</th>
    <th>Header 2</th>
  </tr>
  <tr>
    <td>Data 1</td>
    <td>Data 2</td>
  </tr>
</table>
```

Header 1	Header 2
Data 1	Data 2

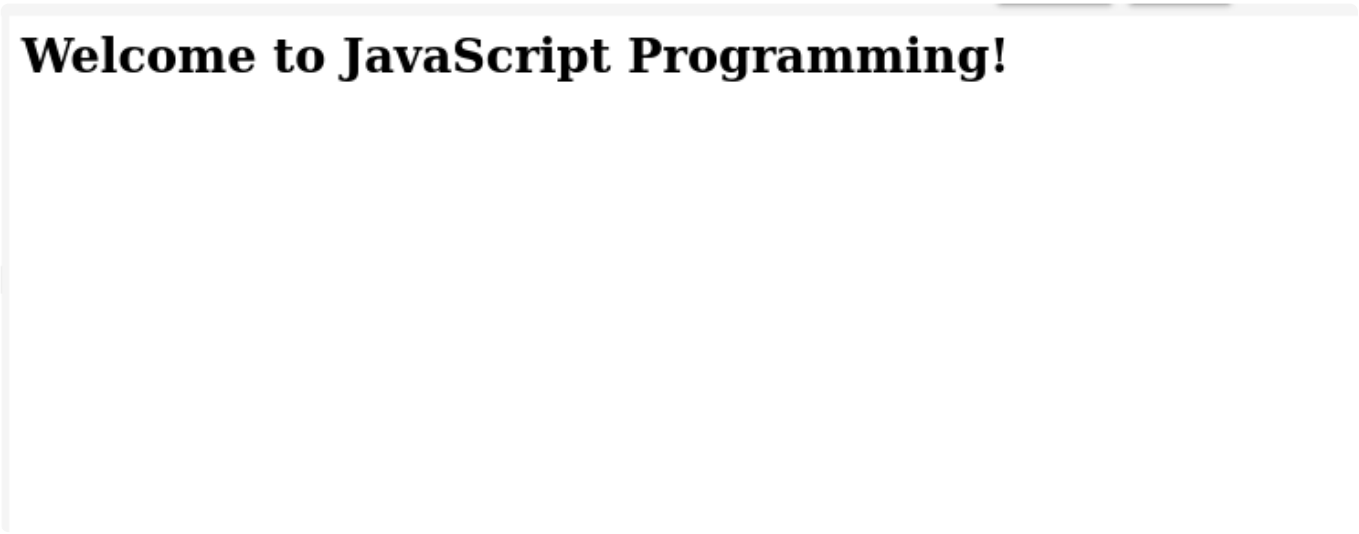
7. Javascript- Basic Programs

JavaScript is a client-side scripting language used to create dynamic and interactive web applications. It is highly portable and supported by all major web browsers. JavaScript interacts with HTML through the Document Object Model (DOM), enabling developers to manipulate elements on a web page.

Examples

1. Inserting html tag using Javascript

```
<html>
        <head>
                <title> A First Program in Javascript</title>
                <script type="text/javascript">
                        document.writeln("<h1>Welcome to JavaScript
Programming!</h1>");
                </script>
        </head>
        <body>
        </body>
</html>
```



2. Displaying a line of colored text

```
<html>
        <head>
                <title> Printing a line with Multiple Statements</title>
```

```
                <script type="text/javascript">
                        document.write("<h1 style = 'color:magenta'>");
                        document.write("Welcome to JavaScript " +
"Programming!</h1>");
                </script>
        </head>
        <body>
        </body>
</html>
```

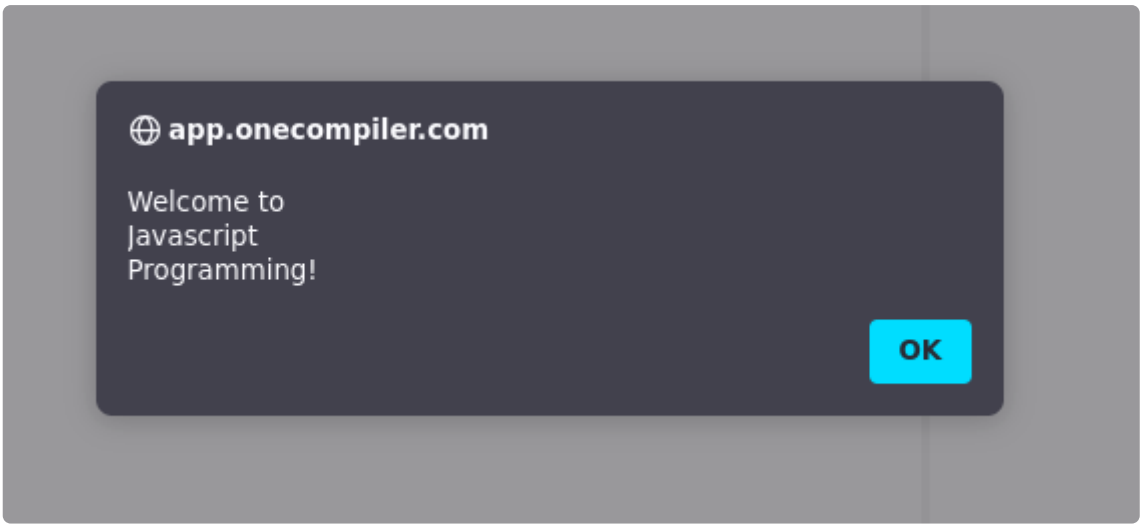
3. Displaying text in alert dialog

Example: Displaying Text in an Alert Dialog

```
<html>
   <head>
      <meta charset = "utf-8">
      <title>Printing Multiple Lines in a Dialog Box</title>
      <script type = "text/javascript">
         <!--
         window.alert( "Welcome to\nJavaScript\nProgramming!" );
         // -->
      </script>
   </head>
   <body>
      <p>Click Refresh (or Reload) to run this script again.</p>
   </body>
</html>
```

```
<html>
        <head>
```

```
                <title> Printing Multiple Lines in a Dialog Box</title>
                <script type="text/javascript">
                        window.alert("Welcome to\nJavascript\nProgramming!")
                </script>
        </head>
        <body>
                <p> Click Refresh (or Reload) to run this script again. </p>
        </body>
</html>
```



8. Javascript-Keywords

In JavaScript, **keywords** are reserved words that have a predefined meaning in the language and cannot be used as identifiers (like variable names, function names, etc.). They serve specific purposes in controlling the flow, structure, and behavior of JavaScript code.

Control Flow Keywords:

- `if`, `else`: Conditional statements.
- `switch`, `case`, `default`: Used for multiple condition checks.
- `for`, `while`, `do`: Looping statements.
- `break`, `continue`: Control loop execution.

Variable and Function Declaration:

- `var`, `let`, `const`: Declare variables.
- `function`: Define a function.
- `return`: Return a value from a function.

Class and Object Keywords:

- `class`: Define a class.
- `extends`: Inherit a class.
- `super`: Access a parent class.
- `this`: Refers to the current object.
- `new`: Create an instance of an object or class.

Error Handling:

- `try`, `catch`, `finally`: Handle exceptions.
- `throw`: Throw an exception.

Asynchronous Keywords:

- `async`, `await`: Used for handling asynchronous operations.
- `promise`: Though not a keyword, related to async programming.

Miscellaneous:

- `null`: Represents a null value.
- `undefined`: Represents an undefined value.
- `typeof`: Returns the type of a variable.
- `instanceof`: Checks if an object is an instance of a class.
- `delete`: Deletes a property from an object.
- `void`: Evaluates an expression without returning a value.

9. Javascript-Control Statements

`if` statement: Executes a block of code if a specified condition evaluates to `true`.

```
if (condition) {
  // Code to execute if the condition is true
}
```

`if...else` statement: Executes one block of code if the condition is `true`, and another block if the condition is `false`.

```
if (condition) {
  // Code to execute if the condition is true
} else {
  // Code to execute if the condition is false
}
```

`else if` statement: Adds more conditions to the `if...else` structure, allowing for multiple conditions to be checked.

```
if (condition1) {
  // Code to execute if condition1 is true
} else if (condition2) {
  // Code to execute if condition2 is true
} else {
  // Code to execute if none of the conditions are true
}
```

`switch` statement: Evaluates an expression and executes code blocks based on different matching cases. It is a more efficient alternative to using multiple `if...else if` statements when checking for multiple specific values.

```
switch (expression) {
  case value1:
    // Code to execute if expression === value1
    break;
  case value2:
    // Code to execute if expression === value2
```

```
    break;
  default:
    // Code to execute if no cases match
}
```

10. Javascript-Break-and-Continue

In JavaScript, the `break` and `continue` statements are used to control the flow of loops. They provide ways to terminate or skip iterations based on specific conditions.

1. break Statement

- **Purpose**: The `break` statement is used to exit a loop (such as `for`, `while`, or `do...while`) or a `switch` statement immediately.
- **Usage**: When encountered, it terminates the nearest enclosing loop or `switch`, and the control moves to the next statement following the loop or `switch`.

Example

```
for (let i = 0; i < 10; i++) {
  if (i === 5) {
    break; // Exits the loop when i is 5
  }
  console.log(i); // Outputs 0, 1, 2, 3, 4
}
```

2. continue Statement

- **Purpose**: The `continue` statement is used to skip the current iteration of a loop and proceed to the next iteration.
- **Usage**: When encountered, it skips the remaining code in the current loop iteration and jumps to the next iteration based on the loop's condition.

```
for (let i = 0; i < 10; i++) {
  if (i % 2 === 0) {
    continue; // Skips the even numbers
```

```
  }
  console.log(i); // Outputs 1, 3, 5, 7, 9
}
```

11. Javascript-Functions

JavaScript functions are reusable blocks of code designed to perform a specific task. They allow developers to organize and structure their code more effectively, promoting reusability and maintainability.

```
function functionName(parameters) {
  // Code to be executed
}
```

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

greet('Alice'); // Outputs: Hello, Alice!
```

Parameters and Arguments

- **Parameters**: Variables listed in the function definition.
- **Arguments**: Values passed to the function when called.

```
function greet(name = 'Guest') {
  console.log(`Hello, ${name}!`);
}

greet(); // Outputs: Hello, Guest!
```

Return Statement

The `return` statement is used to return a value from a function. Once the `return` statement is executed, the function stops executing further code.

```
function add(a, b) {
  return a + b; // Returns the sum of a and b
}

let result = add(5, 3); // result is 8
```

12. Javascript-Arrays

JavaScript arrays are a special type of object that allows you to store multiple values in a single variable. They are versatile and can hold various data types, including numbers, strings, objects, and even other arrays. Arrays are zero-indexed, meaning the first element has an index of 0.

```
const fruits = ['apple', 'banana', 'cherry'];
```

Accessing Array Elements

```
const fruits = ['apple', 'banana', 'cherry'];
console.log(fruits[0]); // Outputs: apple
```

Modifying Arrays

```
fruits[1] = 'orange'; // Changes 'banana' to 'orange'
```

Adding Elements

```
fruits.push('date'); // fruits is now ['apple', 'orange', 'cherry', 'date']
```

Popping Elements

```
fruits.pop(); // fruits is now ['kiwi', 'apple', 'orange', 'cherry']
```

Length of array

```
console.log(fruits.length); // Outputs: 3
```

13. Javascript-DOM

The **Document Object Model (DOM)** represents the structure of an HTML document as a tree of objects. Each element, attribute, and piece of text in the HTML document is represented as a node in the DOM tree, enabling developers to manipulate the document's content and structure programmatically.

Accessing the DOM: JavaScript provides several methods to access and manipulate DOM elements:

- `document.getElementById(id)` : Selects a single element with the specified ID.
- `document.getElementsByClassName(className)` : Selects all elements with the specified class name.
- `document.getElementsByTagName(tagName)` : Selects all elements with the specified tag name.
- `document.querySelector(selector)` : Selects the first element that matches a specified CSS selector.
- `document.querySelectorAll(selector)` : Selects all elements that match a specified CSS selector.

Examples

Changing content

```
const header = document.getElementById('header');
header.textContent = 'New Header Text';
```

```
const element = document.querySelector('.item');
element.classList.add('active'); // Adds class
element.classList.remove('active'); // Removes class
```

```
const newDiv = document.createElement('div');
newDiv.textContent = 'Hello, World!';
document.body.appendChild(newDiv); // Appends to the body
```

Events

JavaScript allows developers to listen for events and execute specific functions when these events occur.

When an event occurs, an event object is created that contains information about the event, such as the target element and the type of event.

```
const button = document.getElementById('myButton');
button.addEventListener('click', function() {
  alert('Button clicked!');
});
```

Removing Event listeners

```
function handleClick() {
  alert('Button clicked!');
}

button.addEventListener('click', handleClick);
button.removeEventListener('click', handleClick);
```