What is JavaScript?

JavaScript (JS) is a high-level, interpreted programming language that enables interactive web pages. It's one of the core technologies of web development, alongside **HTML** (HyperText Markup Language) and **CSS** (Cascading Style Sheets). JavaScript allows you to create dynamic and interactive effects on web pages, such as animations, form validations, interactive maps, and more.

Where is JavaScript used?

- **Web Development**: Adding interactivity to websites (form validation, dynamic content, animations, etc.).
- **Backend Development**: With Node.js, JavaScript can also be used on the server-side to build web servers and APIs.
- **Mobile Apps**: Using frameworks like React Native, you can create mobile apps with JavaScript.
- **Game Development**: JavaScript can be used in game engines like Phaser to create browser-based games.

Basic JavaScript Syntax

1. Variables

Variables are used to store data.

```
var number = 100; // "var" values can be updated
let name = "Alice"; // "let" values can be updated
const age = 25; // "const" means this value cannot be changed
```

2. Data Types

Common data types include:

- **String**: Text, enclosed in quotes. (either in " " or ' ')

  ```
  let greeting = "Hello, World!";
  var message = "some message";
  ```

- **Number**: Integer or floating-point numbers.

  ```
  let count = 10;
  let price = 19.99;
  ```

- **Boolean**: True or false.

```
let isActive = true;
```

3. Functions

Functions are blocks of code that can be executed when called.

```
function greet(name) {
  console.log("Hello, " + name);
}

greet("UserName"); // Outputs: Hello, UserName
```

4. Control Flow (Conditionals)

if-else statements allow you to make decisions.

```
let num = 5;

if (num > 0) {
  console.log("Positive number");
} else {
  console.log("Non-positive number");
}
```

5. Loops

Loops allow you to repeat actions multiple times.

- **for loop**:

  ```
  for (let i = 0; i < 5; i++) {
    console.log(i); // Outputs 0, 1, 2, 3, 4
  }
  ```

- **while loop**:

  ```
  let count = 0;
  while (count < 5) {
    console.log(count); // Outputs 0, 1, 2, 3, 4
    count++;
  }
  ```

6. Objects

Objects are collections of key-value pairs.

```
let person = {
  name: "John",
  age: 30,
  greet: function () {
    console.log("Hello " + this.name);
  },
};

person.greet(); // Outputs: Hello John
```

7. Arrays

Arrays are ordered lists of values.

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

JavaScript in the Browser

JavaScript can be included in HTML files to add interactivity. Here's a basic example:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>JavaScript Example</title>
  </head>
  <body>
    <h1>My First JavaScript</h1>
    <button onclick="alert('Hello!')">Click Me</button>

    <script>
      // This is an inline JavaScript code
      console.log("Page Loaded!");
    </script>
  </body>
</html>
```

In this example:

- Clicking the button will trigger the `alert` function and show a pop-up.

- `console.log` prints a message to the browser’s developer console.

Console in JavaScript

The **console** is a built-in object in JavaScript that provides access to the browser's debugging console. It's primarily used for logging messages, inspecting values, and debugging code. The most common use of the console is for **logging** messages or data during the execution of your JavaScript code.

Common Console Methods

1. `console.log()`: Used to print general messages or the value of variables to the console.

   ```
   let name = "Alice";
   console.log("Hello, " + name); // Outputs: Hello, Alice
   ```

2. `console.error()`: Logs an error message, typically in red, to indicate a problem or bug.

   ```
   console.error("An error occurred!"); // Outputs: An error occurred! (in red)
   ```

3. `console.warn()`: Logs a warning message, often shown in yellow, to alert the developer of potential issues.

   ```
   console.warn("This is a warning!"); // Outputs: This is a warning! (in
   yellow)
   ```

4. `console.info()`: Logs informational messages, usually in blue.

   ```
   console.info("This is some information.");
   ```

5. `console.table()`: Logs data in a table format, which is useful for displaying arrays or objects in a more readable format.

   ```
   let users = [
     { name: "Alice", age: 30 },
     { name: "Bob", age: 25 },
   ];
   console.table(users);
   ```

6. **`console.group()` and `console.groupEnd()`**: Used to group related messages together, making it easier to organize and visualize the console output.

```
console.group("User Info");
console.log("Name: Alice");
console.log("Age: 30");
console.groupEnd(); // Ends the group
```

7. `console.time()` **and** `console.timeEnd()`: Used to measure the time taken for certain operations or code execution.

```
console.time("loop");
for (let i = 0; i < 1000; i++) {}
console.timeEnd("loop"); // Outputs the time taken to execute the loop
```

The **DOM**: Document Object Model

The **DOM** (Document Object Model) is a programming interface for web documents. It represents the structure of a web page as a tree of objects, where each object corresponds to a part of the page. JavaScript can interact with this model to dynamically change the content and structure of a web page.

The DOM allows you to manipulate the HTML or XML document as an object-oriented structure, meaning you can access, modify, delete, or create elements in the document using JavaScript.

How the DOM Works

The DOM treats an HTML document as a tree of nodes, where:

- Each **element** in the document is represented as a node (e.g., `<div>`, `<p>`, `<a>`, etc.).
- Text inside an element is also a node.
- Attributes (like `class`, `id`, etc.) are also part of the node tree.

Example:

Consider this simple HTML:

```
<!DOCTYPE html>
<html>
  <head>
    <title>DOM Example</title>
  </head>
  <body>
    <h1 id="header">Welcome to the DOM</h1>
    <p class="description">This is an example of the DOM.</p>
    <button onclick="changeText()">Change Text</button>

    <script>
      // JavaScript to manipulate the DOM
```

```
        function changeText() {
          const header = document.getElementById("header");
          header.textContent = "Text Changed!";
        }
      </script>
    </body>
  </html>
```

In this example, the DOM structure might look like this:

```
Document
├── html
│   ├── head
│   │   └── title
│   └── body
│       ├── h1
│       └── p
│       └── button
```

Main Features of the DOM

1. **Node Types**: The DOM has different types of nodes:
   - **Element nodes**: Represent HTML tags (e.g., `<div>`, `<span>`).
   - **Text nodes**: Represent the text inside the elements.
   - **Attribute nodes**: Represent the attributes of the elements (e.g., `class`, `id`, `href`).
   - **Document node**: Represents the entire document.
2. **DOM Tree Structure**: The DOM represents the document as a tree-like structure. The root of this tree is the `document` object, and it contains child nodes representing each element, text, and attribute within the HTML structure.
3. **Manipulation**: JavaScript can modify the DOM by adding, removing, or changing elements, attributes, and text. This allows dynamic changes to web pages without requiring a full page reload.

Accessing and Manipulating the DOM with JavaScript

JavaScript provides a set of methods to interact with the DOM. Here are some key methods:

1. Accessing Elements

You can access elements in the DOM using various methods:

- **By ID**: `document.getElementById()`

  ```
  let element = document.getElementById("header");
  ```

- **By Class Name**: `document.getElementsByClassName()`

  ```
  let paragraphs = document.getElementsByClassName("description");
  ```

- **By Tag Name**: `document.getElementsByTagName()`

  ```
  let divs = document.getElementsByTagName("div");
  ```

- **By Query Selector**: `document.querySelector()` (returns the first matching element)

  ```
  let firstParagraph = document.querySelector("p");
  ```

- **By Query Selector All**: `document.querySelectorAll()` (returns all matching elements)

  ```
  let allParagraphs = document.querySelectorAll("p");
  ```

2. Manipulating Elements

Once you've selected an element, you can modify its properties, styles, and content.

- **Changing text content**: Use `textContent` or `innerText`.

  ```
  element.textContent = "New text content!";
  // or
  element.innerText = "New text content!";
  ```

- **Changing HTML content**: Use `innerHTML`.

  ```
  element.innerHTML = "<strong>Bold Text</strong>";
  ```

- **Changing attributes**: Use `setAttribute()` or `getAttribute()`.

  ```
  element.setAttribute("class", "new-class");
  let classValue = element.getAttribute("class");
  ```

- **Changing styles**: Use `style` property.

```
element.style.color = "blue";
```

3. Creating and Inserting Elements

You can also create new elements and add them to the DOM.

- **Creating new elements**: Use `document.createElement()`.

  ```
  let newDiv = document.createElement("div");
  ```

- **Appending elements**: Use `appendChild()`, `insertBefore()`, or `append()` to add new elements.

  ```
  document.body.appendChild(newDiv);
  ```

- **Inserting HTML**: Use `insertAdjacentHTML()` to insert HTML into an element.

  ```
  element.insertAdjacentHTML("beforeend", "<p>New paragraph</p>");
  ```

4. Removing Elements

You can remove elements from the DOM using `removeChild()` or `remove()`.

```
let elementToRemove = document.getElementById("header");
elementToRemove.remove(); // Removes the header element from the DOM
```

Event Handling in the DOM

The DOM allows you to add interactivity to your webpage through events. Events represent user interactions or system changes (e.g., clicking a button, submitting a form, etc.).

1. Adding Event Listeners

You can use `addEventListener()` to attach an event to an element:

```
const button = document.querySelector("button");

button.addEventListener("click", function () {
  alert("Button clicked!");
});
```

2. Common Event Types

- **click**: Triggered when an element is clicked.
- **mouseover**: Triggered when the mouse pointer hovers over an element.
- **keydown/keyup**: Triggered when a key is pressed or released.
- **submit**: Triggered when a form is submitted.
- **change**: Triggered when an input value changes.

Example: DOM Manipulation in Action

Here’s an example that demonstrates accessing and modifying the DOM dynamically:

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>DOM Example</title>
    <style>
      .hidden {
        display: none;
      }
    </style>
  </head>
  <body>
    <h1 id="title">Welcome!</h1>
    <button id="changeButton">Change Text</button>
    <button id="hideButton">Hide Heading</button>

    <script>
      // Accessing elements
      const title = document.getElementById("title");
      const changeButton = document.getElementById("changeButton");
      const hideButton = document.getElementById("hideButton");

      // Changing text content
      changeButton.addEventListener("click", function () {
        title.textContent = "Text has been changed!";
      });

      // Hiding element
      hideButton.addEventListener("click", function () {
        title.classList.toggle("hidden");
      });
    </script>
  </body>
</html>
```

In this example:

- Clicking the **"Change Text"** button changes the content of the heading (`<h1>`).
- Clicking the **"Hide Heading"** button toggles the visibility of the heading using a CSS class (`.hidden`).
