JavaScript Summary: Mastering the DOM, BOM, and Events

This summary covers the essentials of interacting with web pages using JavaScript, focusing on the Document Object Model (DOM), handling events, and an introduction to the Browser Object Model (BOM).

1. Introduction to the Document Object Model (DOM)

The **DOM** is a programming interface for web documents. It represents the page so that programs can change the document structure, style, and content. Essentially, it's a tree-like representation of your HTML document, where each HTML element is a node (object) that JavaScript can access and manipulate.

---

2. How to Select Elements from the DOM

Selecting the right element is the first step to manipulating it. JavaScript provides several methods for this:

- **`document.getElementById("idName")`**:
  - Selects a single element by its unique `id` attribute.
  - Returns the element object or `null` if not found.
  - **Example:** `let paragraph = document.getElementById("f-p");`
- **`document.getElementsByClassName("className")`**:
  - Selects all elements with a specific `class` name.
  - Returns an HTMLCollection (array-like object). You often need to loop through it.
  - **Example:**

    JavaScript

    ```
    let paragraphs = document.getElementsByClassName('prag');
    for (let i = 0; i < paragraphs.length; i++) {
        // console.log(paragraphs[i]);
    }
    ```

- **`document.getElementsByTagName("tagName")`**:
  - Selects all elements with a specific HTML tag name (e.g., `div`, `p`, `img`).
  - Returns an HTMLCollection.
  - **Example:** `let divs = document.getElementsByTagName("div");`
- **`document.querySelector("cssSelector")`**:
  - The most versatile method. Selects the **first** element that matches the specified CSS selector.

    - Can use any valid CSS selector (e.g., tag, id, class, descendant, child, attribute selectors).
    - **Example:** `let firstParagraphInDiv = document.querySelector("div p.prag");`
- **`document.querySelectorAll("cssSelector")`**:
    - Selects **all** elements that match the specified CSS selector.
    - Returns a NodeList (array-like object) which can be iterated using `for...of` or `forEach`.
    - **Important:** You cannot directly assign a value to `innerHTML` or other properties on a NodeList. You must loop through its elements.
    - **Example:**

      JavaScript

      ```
      let elements = document.querySelectorAll("div p.p");
      for (let element of elements) {
          element.innerHTML = "eman"; // Correct way to change content for multiple elements
      }
      ```

---

3. How to Change Element Content

Once you've selected an element, you can modify its content:

- **`element.innerHTML`**:
    - Gets or sets the HTML content (including tags) inside an element.
    - **Example (get):** `let content = pragraph.innerHTML;`
    - **Example (set):** `pragraph.innerHTML = "<span style='color:green'>span</span>";`
- **`element.innerText`**:
    - Gets or sets the visible text content of an element, ignoring HTML tags.
    - **Example:** `pragraph.innerText = "defr";`
- **`element.textContent`**:
    - Similar to `innerText` but includes the content of `<script>` and `<style>` elements and maintains spacing. (Good to be aware of the subtle difference).
- **`document.title`**:
    - Directly sets the title of the HTML document (visible in the browser tab).
    - **Example:** `document.title = "Dom";`
- **`document.links`**:

- An HTMLCollection of all `<a>` (anchor) elements in the document.

---

4. How to Change Element Style

You can modify an element's CSS styles directly through JavaScript:

- **`element.style`**:
  - An object that represents the inline style of an element.
  - CSS properties are accessed using **camelCase** (e.g., `backgroundColor` instead of `background-color`).
  - **Example:**

    JavaScript

    ```
    pragraph.style.backgroundColor = "black";
    pragraph.style.color = "green";
    pragraph.style.fontSize = "10px";
    ```

  - You can also set multiple styles at once by directly assigning a CSS string, but this overwrites any existing inline styles: `pragraph.style = "color:blue; font-size: 30px";`

---

5. Dealing with Attributes

Attributes provide additional information about HTML elements.

- **Direct Property Access (for common attributes):**
  - `element.className` (for `class`)
  - `element.id` (for `id`)
  - `element.src` (for `img` `src`)
  - **Example:** `console.log(pragraph.className);`
- **`element.getAttribute("attributeName")`**:
  - Retrieves the value of a specified attribute.
  - **Example:** `console.log(pragraph.getAttribute("class"));`
- **`element.setAttribute("attributeName", "value")`**:
  - Sets or changes the value of a specified attribute.
  - **Example:** `pragraph.setAttribute("class", 'newAtt');`
- **`element.hasAttribute("attributeName")`**:
  - Checks if an element has a specific attribute (returns `true` or `false`).

    - **Example:** `console.log(pragraph.hasAttribute('class'));`
- **`element.removeAttribute("attributeName")`**:
    - Removes a specified attribute from an element.
    - **Example:** `pragraph.removeAttribute('class');`

`classList` Property (for CSS Classes)

The `classList` property is a powerful way to manage an element's CSS classes. It's preferred over directly manipulating `className` strings.

- **`element.classList.add("class1", "class2", ...)`**: Adds one or more classes.
    - **Example:** `pragraph.classList.add("newatt", "test");`
    - Can use spread syntax for an array of classes: `pragraph.classList.add(...classNames);`
- **`element.classList.remove("class1", "class2", ...)`**: Removes one or more classes.
    - **Example:** `pragraph.classList.remove('newatt', 'test');`
- **`element.classList.contains("className")`**: Checks if an element has a specific class.
    - **Example:** `console.log(pragraph.classList.contains('prag'));`
- **`element.classList.toggle("className")`**: Adds the class if it's not present, and removes it if it is. Useful for toggling states.
    - **Example:** `pragraph.classList.toggle('className');`

---

6. Create New Elements from JavaScript

You can dynamically create new HTML elements and add them to the document.

1. **Select the Parent Element**: Decide where the new element will be added.
    - **Example:** `let section = document.getElementById('section');`
2. **Create the New Element**:
    - **`document.createElement('tagName')`**: Creates a new HTML element node.
    - **Example:** `let heading = document.createElement('h1');`
3. **Add Content and Attributes to the New Element**:
    - **Example:** `heading.innerText = "this is heading";`
4. **Append the New Element to a Parent**:

    - **`parentElement.appendChild(childElement)`**: Appends the new element as the last child of the parent.
    - **Example:** `section.appendChild(heading);`
- **Creating Text Nodes:**
    - **`document.createTextNode("some text")`**: Creates a text node. Often appended to elements using `appendChild`.
    - **Example:** `let content = document.createTextNode("this is heading"); heading.appendChild(content);`
- **Creating Comments:**
    - **`document.createComment("some comment")`**: Creates an HTML comment node.
    - **Example:** `let comment = document.createComment("this is comment");`
- **Inserting Elements at Specific Positions:**
    - **`parentElement.insertBefore(newElement, referenceElement)`**: Inserts `newElement` immediately before `referenceElement` within `parentElement`.
    - **Example:** `section.insertBefore(comment, p);`
- **Dynamic Card Creation Example:** The provided code demonstrates a `createCard` function that takes product data and dynamically constructs a complete HTML card (`div` with `img`, `h5`, `p`, `a`) using the above methods. This is a common pattern for rendering data from arrays or APIs.

    JavaScript

```
function createCard(data){
    let cardDiv = document.createElement('div');
    let img = document.createElement('img');
    // ... (rest of the card elements and their attribute/class
assignments)

    // Append children to build the card structure
    cardDiv.appendChild(img);
    cardDiv.appendChild(cardBody);
    cardBody.appendChild(heading);
    // ...

    // Set content
    heading.innerHTML = data.title;
    // ...

    return cardDiv;
}

// Usage example:
// products.forEach((product) => {
//     let card = createCard(product);
```

```
//     section.appendChild(card);
// });
```

---

7. Events

Events are actions or occurrences that happen in the system you are programming, which the system tells you about so you can respond to them. Examples include user clicks, keyboard presses, form submissions, page loading, etc.

You can make your web page interactive by "listening" for these events and executing JavaScript code in response.

- **Event Handlers (Traditional Approach):** You can assign a function directly to an element's event property. These are often named `onEventName`.
    - **`element.onclick`**: Fires when the element is clicked.

      JavaScript

      ```
      btn.onclick = function(event) {
          btn.style.backgroundColor = "red";
          // event.preventDefault(); // Prevents default browser behavior
      }
      ```

    - **`element.ondblclick`**: Fires when the element is double-clicked.

      JavaScript

      ```
      btn.ondblclick = function(event) {
          btn.style.backgroundColor = "green";
      }
      ```

    - **`element.onfocus`**: Fires when an element (like an input field) gains focus.

      JavaScript

      ```
      userName.onfocus = function(event) {
          userName.style.backgroundColor = 'green';
          userName.style.outline = 'red 2px solid';
      }
      ```

    - **`element.onblur`**: Fires when an element loses focus.

      JavaScript

      ```
      userName.onblur = function(event) {
          userName.style.backgroundColor = 'white';
      }
      ```

    - **`element.onkeyup`**: Fires when a keyboard key is released while the element has focus. Useful for real-time input handling.

      JavaScript

      ```
      function printInput(event) {
          console.log(event.target.value);
      ```

```
    result.innerHTML += event.target.value; // Append character by
character
}
userName.onkeyup = printInput;
userEmail.onkeyup = printInput;
```

- **`element.onkeydown`**: Fires when a keyboard key is pressed down.
- **`form.onsubmit`**: Fires when a form is submitted. You often use `event.preventDefault()` here to handle submission with JavaScript (e.g., validation, AJAX).

  JavaScript

  ```
  form.onsubmit = function(e) {
      alert("form submit");
      e.preventDefault(); // Prevent actual form submission
      if (userName.value === "") { // Note: Corrected from userName=""
          alert('error: username is empty');
      } else {
          // form.submit(); // If you want to submit programmatically
  after checks
      }
  }
  ```

- **`window.onload`**: Fires when the entire page (including all images, scripts, etc.) has finished loading.

  JavaScript

  ```
  // window.onload = function() {
  //     alert("page is loaded");
  // }
  ```

- **`event.preventDefault()`**: An important method of the `Event` object. It stops the browser's default action for a particular event (e.g., preventing a link from navigating, preventing a form from submitting).
- **`event.target`**: Refers to the DOM element that triggered the event. `event.target.value` is commonly used to get the current value of an input field.
- **Note on `addEventListener`**: While the above methods work, `addEventListener()` is the **modern and preferred way** to handle events. It allows multiple event handlers for the same event on the same element, and provides more control. It's recommended to research this further for more robust event handling.

---

8. JavaScript Timing

JavaScript provides functions to schedule code execution after a delay or at regular intervals.

- **`setInterval(callback, delay)`**:
  - Executes a `callback` function repeatedly after every `delay` milliseconds.
  - Returns an interval ID that can be used to stop it.

- **Example:**

  JavaScript

  ```
  let count = 0;
  let interval = setInterval(() => {
      console.log(++count);
      if (count >= 10) {
          clearInterval(interval); // Stop the interval
      }
  }, 1000); // Runs every 1 second
  ```

- **`clearInterval(intervalId)`**:
  - Stops the execution of a `setInterval` timer.
- **`setTimeout(callback, delay)`**:
  - Executes a `callback` function **once** after a specified `delay` milliseconds.
  - **Example:**

    JavaScript

    ```
    setTimeout(() => {
        alert("welcome");
    }, 2000); // Runs after 2 seconds
    ```

- **`clearTimeout(timeoutId)`**:
  - Stops the execution of a `setTimeout` timer (if it hasn't run yet).

---

9. Browser Object Model (BOM)

The **BOM** (Browser Object Model) allows JavaScript to interact with the browser itself, not just the document content. The **`window`** object is the global object and represents the browser window.

- **`window` object:**
  - The top-level object in the BOM. All global JavaScript objects, functions, and variables automatically become members of the `window` object.
  - **`window.innerHeight`** / **`window.innerWidth`**: Get the inner dimensions (height/width) of the browser's content area (viewport).
  - **`window.scrollX`** / **`window.scrollY`**: Get the number of pixels the document is currently scrolled horizontally/vertically.
  - **`window.open(url, target, features)`**: Opens a new browser window or tab.
    - **Example:** `let newWindow = window.open("https://github.com/", "_blank", "width=2000,height=200");`

    - **`window.close()`**: Closes the current window (usually only works for windows opened by `window.open`).
        - **Example:** `newWindow.close();` (within a timeout, as shown in your code).
- **`location` object:**
    - Part of the `window` object, it allows you to get information about the current page's URL and redirect the browser to a new URL.
    - **`location.assign(url)`**: Loads a new document. The current page is added to the browser's history, so the user can use the "back" button.
    - **`location.replace(url)`**: Loads a new document, but it *replaces* the current page in the browser's history. The user cannot use the "back" button to return to the previous page.
    - **`location.href`**: Sets or returns the entire URL. Assigning a new URL to `location.href` is equivalent to `location.assign()`.
    - **Example (all redirect):**

      JavaScript

      ```
      setTimeout(() => {

      location.assign("https://developer.mozilla.org/en-US/docs/Web/API/
      Window/location");
          // location.replace(...)
          // location.href = ...
      }, 2000);
      ```
- **`history` object:**
    - Part of the `window` object, it allows you to navigate the browser's history.
    - **`history.go(delta)`**: Navigates through the history.
        - `history.go(-1)`: Goes back to the previous page.
        - `history.go(1)`: Goes forward to the next page.
- **Other important BOM objects (brief mention):**
    - **`document`**: (Already covered in DOM section).
    - **`screen`**: Provides information about the user's screen.
    - **`navigator`**: Provides information about the browser (user agent, plugins, etc.).