DOM Kiválasztó és Navigációs Metódusok Összefoglaló

getElementById

A `getElementById` metódus egy olyan elemet keres a DOM-ban, amelynek az `id` attribútuma megegyezik a megadott sztringgel.

Szintaxis:

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

Jellemzők:

- Csak egy elemet ad vissza, mivel az `id` attribútumnak egyedinek kell lennie az oldalon.
- Ha nincs ilyen `id` attribútumú elem, `null` értéket ad vissza.

querySelectorAll

A `querySelectorAll` metódus egy NodeList-et ad vissza azokról az elemekről, amelyek megfelelnek a megadott CSS szelektornak.

Szintaxis:

```
let elements = document.querySelectorAll(".myClass");
```

Jellemzők:

- NodeList-et ad vissza, amely statikus gyűjtemény, tehát nem frissül automatikusan, ha a DOM változik.
- Tetszőleges CSS szelektort használhatunk.

querySelector

A `querySelector` metódus az első olyan elemet adja vissza, amely megfelel a megadott CSS szelektornak.

Szintaxis:

```
let element = document.querySelector(".myClass");
```

Jellemzők:

- Csak az első megfelelő elemet adja vissza.
- Tetszőleges CSS szelektort használhatunk.

matches

A `matches` metódus ellenőrzi, hogy egy adott elem megfelel-e a megadott CSS szelektornak.

Szintaxis:

```
let isMatch = element.matches(".myClass");
```

Jellemzők:

- `true` értéket ad vissza, ha az elem megfelel a szelektornak, különben `false`.

closest

A `closest` metódus az aktuális elemből kiindulva keres felfelé a DOM fa hierarchiájában, és az első olyan elemet adja vissza, amely megfelel a megadott CSS szelektornak.

Szintaxis:

```
let closestElement = element.closest(".myClass");
```

Jellemzők:

- Ha nincs megfelelő elem, `null` értéket ad vissza.

getElementsByClassName

A `getElementsByClassName` metódus egy élő HTMLCollection-t ad vissza azokról az elemekről, amelyek az adott osztálynévvel rendelkeznek.

Szintaxis:

```
let elements = document.getElementsByClassName("myClass");
```

Jellemzők:

- Élő gyűjteményt ad vissza, amely automatikusan frissül, ha a DOM változik.
- Több osztálynevet is megadhatunk szóközzel elválasztva.

getElementsByTagName

A `getElementsByTagName` metódus egy élő HTMLCollection-t ad vissza az adott címkével rendelkező összes elemmel.

Szintaxis:

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

Jellemzők:

- Élő gyűjteményt ad vissza, amely automatikusan frissül, ha a DOM változik.

getElementsByName

A `getElementsByName` metódus egy élő NodeList-et ad vissza azokról az elemekről, amelyek az adott névvel rendelkeznek.

Szintaxis:

```
let elements = document.getElementsByName("myName");
```

Jellemzők:

- Különösen hasznos form elemek esetén, ahol a `name` attribútum gyakran használt.

Live Collections (Élő Gyűjtemények)

Az élő gyűjtemények automatikusan frissülnek, amikor a DOM változik. Ez azt jelenti, hogy ha elemeket hozzáadunk vagy eltávolítunk a DOM-ból, az élő gyűjtemények (mint a `getElementsByClassName`, `getElementsByTagName`, és `getElementsByName` által visszaadott gyűjtemények) azonnal tükrözik ezeket a változásokat.

Példa:

```
let elements = document.getElementsByTagName("div");
console.log(elements.length); // Kiírja a <div> elemek számát

// Hozzáadunk egy új <div> elemet
let newDiv = document.createElement("div");
document.body.appendChild(newDiv);

console.log(elements.length); // Az új <div> hozzáadása után is kiírja a <div> elemek frissített számát
```

Összegzés

Metódus	Visszatérési típus	Élő gyűjtemény?	Leírás
`getElementById`	Element	Nem	Az első elem, amelynek az `id` attribútuma megfelel.
`querySelectorAll`	NodeList	Nem	Az összes elem, amely megfelel a CSS szelektornak.
`querySelector`	Element	Nem	Az első elem, amely megfelel a CSS szelektornak.
`matches`	Boolean	N/A	Igaz, ha az elem megfelel a CSS szelektornak.
`closest`	Element	N/A	Az első felmenő elem, amely megfelel a CSS szelektornak.
`getElementsByClassName`	HTMLCollection	Igen	Az összes elem, amely az adott osztálynévvel rendelkezik.
`getElementsByTagName`	HTMLCollection	Igen	Az összes elem, amely az adott címkével rendelkezik.
`getElementsByName`	NodeList	Igen	Az összes elem, amely az adott névvel rendelkezik.

DOM Propertyk és HTML Attribútumok Összefoglaló

HTML Attribútumok

A HTML attribútumok további információkat adnak az elemekhez. Az attribútumok mindig a kezdő tagban vannak megadva, és általában kulcs-érték párokként szerepelnek.

Példák:

```
<img src="image.jpg" alt="Kép leírása" />
<a href="https://www.example.com" target="_blank">Link</a>
<input type="text" value="Alapértelmezett szöveg" />
```

DOM Propertyk

A DOM (Document Object Model) egy programozási interfész, amely lehetővé teszi a HTML és XML dokumentumok fára épülő reprezentációját. A DOM propertyk a JavaScript segítségével érhetők el és módosíthatók, hogy dinamikusan változtassuk a dokumentumot.

Különbségek a DOM Propertyk és HTML Attribútumok Között

- **HTML attribútumok**: Az elem alapértelmezett értékei, amelyek a HTML kódban vannak megadva.
- **DOM propertyk**: Az elem aktuális értékei, amelyek a JavaScript segítségével érhetők el és módosíthatók.

Példa: Eredeti Attribútum és DOM Property Értékek

```
<input id="myInput" type="text" value="Alapértelmezett érték" />
```

```
let input = document.getElementById("myInput");

// HTML attribútum érték lekérdezése
console.log(input.getAttribute("value")); // Output: "Alapértelmezett érték"

// DOM property érték lekérdezése
console.log(input.value); // Output: "Alapértelmezett érték"

// DOM property módosítása
input.value = "Új érték";

// HTML attribútum továbbra is az eredeti értéket mutatja
console.log(input.getAttribute("value")); // Output: "Alapértelmezett érték"
```

Gyakran Használt HTML Attribútumok és DOM Propertyk

`class` és `className`

- **HTML attribútum**: `class`
- **DOM property**: `className`

```
<div class="myClass"></div>
```

```
let div = document.querySelector("div");
console.log(div.className); // Output: "myClass"
div.className = "newClass";
console.log(div.className); // Output: "newClass"
```

`for` és `htmlFor`

- **HTML attribútum**: `for`
- **DOM property**: `htmlFor`

```
<label for="myInput">Input Label</label> <input id="myInput" type="text" />
```

```
let label = document.querySelector("label");
console.log(label.htmlFor); // Output: "myInput"
label.htmlFor = "newInput";
console.log(label.htmlFor); // Output: "newInput"
```

`style`

- **HTML attribútum**: Inline CSS stílusok.
- **DOM property**: `style` objektum, amely CSS stílus tulajdonságokat tartalmaz.

```
<div style="color: red; background-color: yellow;"></div>
```

```
let div = document.querySelector("div");
console.log(div.style.color); // Output: "red"
div.style.color = "blue";
console.log(div.style.color); // Output: "blue"
```

`id`

- **HTML attribútum**: `id`
- **DOM property**: `id`

```
<div id="myId"></div>
```

```
let div = document.querySelector("div");
console.log(div.id); // Output: "myId"
div.id = "newId";
console.log(div.id); // Output: "newId"
```

value

- **HTML attribútum**: `value`
- **DOM property**: `value`

```
<input type="text" value="Alapértelmezett érték" />
```

```
let input = document.querySelector("input");
console.log(input.value); // Output: "Alapértelmezett érték"
input.value = "Új érték";
console.log(input.value); // Output: "Új érték"
```

href

- **HTML attribútum**: `href`
- **DOM property**: `href`

```
<a href="https://www.example.com">Link</a>
```

```
let link = document.querySelector("a");
console.log(link.href); // Output: "https://www.example.com"
link.href = "https://www.newexample.com";
console.log(link.href); // Output: "https://www.newexample.com"
```

Speciális Esetek

Adat-Attribútumok

Az adat-attribútumok egyedi adatokat tárolnak a HTML elemekben, amelyek a `data-` prefix-szel kezdődnek. Ezek JavaScript-ben a `dataset` property segítségével érhetők el.

```
<div data-user-id="12345"></div>
```

```
let div = document.querySelector("div");
console.log(div.dataset.userId); // Output: "12345"
div.dataset.userId = "67890";
console.log(div.dataset.userId); // Output: "67890"
```

Összegzés

- **HTML Attribútumok**: Az elem alapértelmezett értékei, amelyeket a HTML kódban adunk meg. Ezek a `getAttribute` és `setAttribute` metódusokkal érhetők el és módosíthatók.

- **DOM Propertyk**: Az elem aktuális értékei, amelyeket JavaScript-ből érhetünk el és módosíthatunk. Ezek a propertyk közvetlenül az elem objektumán érhetők el.

A DOM propertyk és HTML attribútumok közötti különbségek megértése alapvető fontosságú a hatékony DOM manipulációhoz és a dinamikus weboldalak létrehozásához.

HTML Attribútumok Kezelése

A HTML attribútumok kezelése a DOM API segítségével számos metódust kínál, amelyekkel ellenőrizhetjük, lekérdezhetjük, beállíthatjuk és eltávolíthatjuk az attribútumokat az elemekről. Az alábbiakban részletesen bemutatom a `hasAttribute`, `getAttribute`, `setAttribute` és `removeAttribute` metódusokat.

hasAttribute

A `hasAttribute` metódus ellenőrzi, hogy egy adott elem rendelkezik-e a megadott attribútummal. Igaz értéket ad vissza, ha az attribútum létezik, és hamis értéket, ha nem.

Szintaxis:

```
element.hasAttribute(name);
```

- **name**: Az attribútum neve (string).

Példa:

```
<!DOCTYPE html>
<html lang="hu">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>hasAttribute Példa</title>
    </head>
    <body>
        <div id="myElement" class="myClass"></div>
        <script>
            let element = document.getElementById("myElement");
            console.log(element.hasAttribute("class")); // Output: true
            console.log(element.hasAttribute("style")); // Output: false
        </script>
    </body>
</html>
```

getAttribute

A `getAttribute` metódus visszaadja az adott attribútum értékét az elemről. Ha az attribútum nem létezik, `null` értéket ad vissza.

Szintaxis:

```
element.getAttribute(name);
```

- **name**: Az attribútum neve (string).

Példa:

```
<!DOCTYPE html>
<html lang="hu">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>getAttribute Példa</title>
    </head>
    <body>
        <img id="myImage" src="image.jpg" alt="Kép leírása" />
        <script>
            let image = document.getElementById("myImage");
            console.log(image.getAttribute("src")); // Output: "image.jpg"
            console.log(image.getAttribute("alt")); // Output: "Kép leírása"
            console.log(image.getAttribute("title")); // Output: null
        </script>
    </body>
</html>
```

setAttribute

A `setAttribute` metódus beállítja az adott attribútum értékét az elemre. Ha az attribútum már létezik, az értékét módosítja; ha nem létezik, új attribútumot hoz létre.

Szintaxis:

```
element.setAttribute(name, value);
```

- **name**: Az attribútum neve (string).
- **value**: Az attribútum értéke (string).

Példa:

```
<!DOCTYPE html>
<html lang="hu">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>setAttribute Példa</title>
    </head>
    <body>
        <a id="myLink" href="https://www.example.com">Link</a>
        <script>
            let link = document.getElementById("myLink");
            link.setAttribute("href", "https://www.newexample.com");
            link.setAttribute("title", "Új cím");
            console.log(link.getAttribute("href")); // Output: "https://www.newexample.com"
            console.log(link.getAttribute("title")); // Output: "Új cím"
        </script>
    </body>
</html>
```

removeAttribute

A `removeAttribute` metódus eltávolítja az adott attribútumot az elemről.

Szintaxis:

```
element.removeAttribute(name);
```

- **name**: Az attribútum neve (string).

Példa:

```
<!DOCTYPE html>
<html lang="hu">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>removeAttribute Példa</title>
    </head>
    <body>
        <button id="myButton" disabled>Gomb</button>
        <script>
            let button = document.getElementById("myButton");
            button.removeAttribute("disabled");
            console.log(button.hasAttribute("disabled")); // Output: false
        </script>
    </body>
</html>
```

Összegzés

A `hasAttribute`, `getAttribute`, `setAttribute` és `removeAttribute` metódusok lehetővé teszik a HTML attribútumok hatékony kezelését a JavaScript segítségével. Ezek a metódusok alapvető eszközök a dinamikus webfejlesztésben, és segítségükkel könnyedén módosíthatjuk az elemek viselkedését és megjelenését.

Metódus	Leírás
`hasAttribute(name)`	Ellenőrzi, hogy az elem rendelkezik-e a megadott attribútummal.
`getAttribute(name)`	Visszaadja az adott attribútum értékét az elemről.
`setAttribute(name, value)`	Beállítja az adott attribútum értékét az elemre.
`removeAttribute(name)`	Eltávolítja az adott attribútumot az elemről.

Ezekkel a metódusokkal rugalmasan és hatékonyan kezelhetjük a HTML attribútumokat, így dinamikus és interaktív webalkalmazásokat hozhatunk létre.

DOM Propertyk Magyarázata

nodeType

A `nodeType` property visszaadja a csomópont típusát. Ez egy numerikus érték, amely az alábbiak közül egy lehet:

- **1**: Element (elem csomópont)
- **3**: Text (szöveg csomópont)
- **8**: Comment (komment csomópont)
- **9**: Document (dokumentum csomópont)

Szintaxis:

```
let type = node.nodeType;
```

Példa:

```
<!DOCTYPE html>
<html lang="hu">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>nodeType Példa</title>
    </head>
    <body>
        <div id="myElement">Szöveg</div>
        <script>
            let element = document.getElementById("myElement");
            console.log(element.nodeType); // Output: 1 (Element)
            console.log(element.firstChild.nodeType); // Output: 3 (Text)
        </script>
    </body>
</html>
```

nodeName

A `nodeName` property visszaadja a csomópont nevét. Ez lehet a tag neve (pl. `DIV`, `SPAN`) vagy más típusú csomópontok esetén speciális érték (pl. `#text` szöveg csomópontok esetén).

Szintaxis:

```
let name = node.nodeName;
```

Példa:

```
<!DOCTYPE html>
<html lang="hu">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>nodeName Példa</title>
    </head>
    <body>
        <div id="myElement">Szöveg</div>
        <script>
            let element = document.getElementById("myElement");
            console.log(element.nodeName); // Output: "DIV"
            console.log(element.firstChild.nodeName); // Output: "#text"
        </script>
    </body>
</html>
```

tagName

A `tagName` property visszaadja az elem címkéjének nevét. Ez mindig nagybetűs formában van visszaadva.

Szintaxis:

```
let tag = element.tagName;
```

Példa:

```
<!DOCTYPE html>
<html lang="hu">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>tagName Példa</title>
    </head>
    <body>
        <div id="myElement">Szöveg</div>
        <script>
            let element = document.getElementById("myElement");
            console.log(element.tagName); // Output: "DIV"
        </script>
    </body>
</html>
```

innerHTML

Az `innerHTML` property lehetővé teszi az elem belső HTML tartalmának lekérdezését vagy beállítását. Ez az elem összes gyermekének HTML kódját tartalmazza.

Szintaxis:

```
let html = element.innerHTML;
element.innerHTML = "<p>Új tartalom</p>";
```

Példa:

```
<!DOCTYPE html>
<html lang="hu">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>innerHTML Példa</title>
    </head>
    <body>
        <div id="myElement">Eredeti <strong>tartalom</strong></div>
        <script>
            let element = document.getElementById("myElement");
            console.log(element.innerHTML); // Output: "Eredeti <strong>tartalom</strong>"
            element.innerHTML = "<p>Új tartalom</p>";
        </script>
    </body>
</html>
```

outerHTML

Az `outerHTML` property lehetővé teszi az elem és annak teljes tartalmának HTML kódjának lekérdezését vagy beállítását. Ez magában foglalja az elemet magát és annak minden gyermekét.

Szintaxis:

```
let html = element.outerHTML;
element.outerHTML = '<div id="newElement">Új tartalom</div>';
```

Példa:

```
<!DOCTYPE html>
<html lang="hu">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>outerHTML Példa</title>
    </head>
    <body>
        <div id="myElement">Eredeti <strong>tartalom</strong></div>
        <script>
            let element = document.getElementById("myElement");
            console.log(element.outerHTML); // Output: '<div id="myElement">Eredeti <strong>tartalom</strong></div>'
            element.outerHTML = '<div id="newElement">Új tartalom</div>';
        </script>
    </body>
</html>
```

textContent

A `textContent` property lehetővé teszi az elem szöveges tartalmának lekérdezését vagy beállítását. Ez az elem és annak összes gyermekének szövegét adja vissza, figyelmen kívül hagyva a HTML tageket.

Szintaxis:

```
let text = element.textContent;
element.textContent = "Új szöveg";
```

Példa:

```
<!DOCTYPE html>
<html lang="hu">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>textContent Példa</title>
    </head>
    <body>
        <div id="myElement">Eredeti <strong>tartalom</strong></div>
        <script>
            let element = document.getElementById("myElement");
            console.log(element.textContent); // Output: "Eredeti tartalom"
            element.textContent = "Új szöveg";
        </script>
    </body>
</html>
```

hidden

A `hidden` property egy boolean érték, amely azt határozza meg, hogy az elem látható-e vagy sem. Ha az értéke `true`, az elem el van rejtve, ha `false`, akkor látható.

Szintaxis:

```
let isHidden = element.hidden;
element.hidden = true;
```

Példa:

```
<!DOCTYPE html>
<html lang="hu">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>hidden Példa</title>
    </head>
    <body>
        <div id="myElement">Ez az elem el lesz rejtve</div>
        <script>
            let element = document.getElementById("myElement");
            element.hidden = true;
            console.log(element.hidden); // Output: true
        </script>
    </body>
</html>
```

Összegzés

Property	Leírás
`nodeType`	Visszaadja a csomópont típusát numerikus értékként.
`nodeName`	Visszaadja a csomópont nevét.
`tagName`	Visszaadja az elem címkéjének nevét.
`innerHTML`	Lekérdezi vagy beállítja az elem belső HTML tartalmát.
`outerHTML`	Lekérdezi vagy beállítja az elem teljes HTML kódját.
`textContent`	Lekérdezi vagy beállítja az elem szöveges tartalmát.
`hidden`	Boolean érték, amely meghatározza, hogy az elem látható-e.

Ezek a propertyk alapvető eszközök a DOM manipulációban, amelyek lehetővé teszik az elemek tartalmának és láthatóságának dinamikus kezelését a JavaScript segítségével.

DOM Manipulációs Metódusok Magyarázata

createElement

A `createElement` metódus egy új HTML elemet hoz létre. A létrehozott elem még nincs a DOM-ban, így azt hozzá kell adni egy meglévő elemhez.

Szintaxis:

```
let newElement = document.createElement(tagName);
```

- **tagName**: A létrehozandó HTML elem típusa (pl. `div`, `p`, `span`).

Példa:

```
let newDiv = document.createElement("div");
newDiv.textContent = "Ez egy új div elem";
document.body.appendChild(newDiv);
```

createTextNode

A `createTextNode` metódus egy új szövegcsomópontot hoz létre. A szövegcsomópontot hozzá kell adni egy meglévő elemhez, hogy megjelenjen a DOM-ban.

Szintaxis:

```
let newText = document.createTextNode(data);
```

- **data**: A létrehozandó szöveg.

Példa:

```
let newText = document.createTextNode("Ez egy új szövegcsomópont");
let newParagraph = document.createElement("p");
newParagraph.appendChild(newText);
document.body.appendChild(newParagraph);
```

append

Az `append` metódus új csomópontokat vagy szövegeket ad hozzá egy elem belsejének végéhez. Több csomópontot vagy szöveget is hozzáadhat egyszerre.

Szintaxis:

```
parentElement.append(nodesOrDOMStrings);
```

- **nodesOrDOMStrings**: Az új csomópontok vagy szövegek, amelyeket hozzá szeretnénk adni.

Példa:

```
let parentDiv = document.getElementById("parentDiv");
let newDiv = document.createElement("div");
newDiv.textContent = "Új elem";
parentDiv.append(newDiv, " és szöveg");
```

prepend

A `prepend` metódus új csomópontokat vagy szövegeket ad hozzá egy elem belsejének elejéhez. Több csomópontot vagy szöveget is hozzáadhat egyszerre.

Szintaxis:

```
parentElement.prepend(nodesOrDOMStrings);
```

- **nodesOrDOMStrings**: Az új csomópontok vagy szövegek, amelyeket hozzá szeretnénk adni.

Példa:

```
let parentDiv = document.getElementById("parentDiv");
let newDiv = document.createElement("div");
newDiv.textContent = "Új elem";
parentDiv.prepend(newDiv, " és szöveg");
```

before

A `before` metódus új csomópontokat vagy szövegeket ad hozzá egy elem előtt. Több csomópontot vagy szöveget is hozzáadhat egyszerre.

Szintaxis:

```
referenceNode.before(nodesOrDOMStrings);
```

- **nodesOrDOMStrings**: Az új csomópontok vagy szövegek, amelyeket hozzá szeretnénk adni.

Példa:

```
let referenceDiv = document.getElementById("referenceDiv");
let newDiv = document.createElement("div");
newDiv.textContent = "Új elem";
referenceDiv.before(newDiv, " és szöveg");
```

after

Az `after` metódus új csomópontokat vagy szövegeket ad hozzá egy elem után. Több csomópontot vagy szöveget is hozzáadhat egyszerre.

Szintaxis:

```
referenceNode.after(nodesOrDOMStrings);
```

- **nodesOrDOMStrings**: Az új csomópontok vagy szövegek, amelyeket hozzá szeretnénk adni.

Példa:

```
let referenceDiv = document.getElementById("referenceDiv");
let newDiv = document.createElement("div");
newDiv.textContent = "Új elem";
referenceDiv.after(newDiv, " és szöveg");
```

replaceWith

A `replaceWith` metódus kicseréli az elemet egy vagy több új csomópontra vagy szövegre.

Szintaxis:

```
oldNode.replaceWith(nodesOrDOMStrings);
```

- **nodesOrDOMStrings**: Az új csomópontok vagy szövegek, amelyekkel az elemet kicseréljük.

Példa:

```
let oldDiv = document.getElementById("oldDiv");
let newDiv = document.createElement("div");
newDiv.textContent = "Új elem";
oldDiv.replaceWith(newDiv, " és szöveg");
```

insertAdjacentHTML

Az `insertAdjacentHTML` metódus HTML szöveget illeszt be az elem egy megadott pozíciójába. Az illesztett szöveg HTML-ként értelmeződik, és a megfelelő elemek létrejönnek.

Szintaxis:

```
element.insertAdjacentHTML(position, text);
```

- **position**: Az illesztés helye (`'beforebegin'`, `'afterbegin'`, `'beforeend'`, `'afterend'`).

- **text**: A beszúrandó HTML szöveg.

Példa:

```
let referenceDiv = document.getElementById("referenceDiv");
referenceDiv.insertAdjacentHTML("beforebegin", "<div>Új elem</div>");
referenceDiv.insertAdjacentHTML("afterend", "<div>Új elem</div>");
```

insertAdjacentText

Az `insertAdjacentText` metódus szöveget illeszt be az elem egy megadott pozíciójába. Az illesztett szöveg nem HTML-ként, hanem egyszerű szövegként kerül beillesztésre.

Szintaxis:

```
element.insertAdjacentText(position, text);
```

- **position**: Az illesztés helye (`'beforebegin'`, `'afterbegin'`, `'beforeend'`, `'afterend'`).
- **text**: A beszúrandó szöveg.

Példa:

```
let referenceDiv = document.getElementById("referenceDiv");
referenceDiv.insertAdjacentText("beforebegin", "Új szöveg");
referenceDiv.insertAdjacentText("afterend", "Új szöveg");
```

insertAdjacentElement

Az `insertAdjacentElement` metódus egy meglévő DOM elemet illeszt be az elem egy megadott pozíciójába.

Szintaxis:

```
element.insertAdjacentElement(position, newElement);
```

- **position**: Az illesztés helye (`'beforebegin'`, `'afterbegin'`, `'beforeend'`, `'afterend'`).
- **newElement**: A beszúrandó DOM elem.

Példa:

```
let referenceDiv = document.getElementById("referenceDiv");
let newDiv = document.createElement("div");
newDiv.textContent = "Új elem";
referenceDiv.insertAdjacentElement("beforebegin", newDiv);
```

remove

A `remove` metódus eltávolítja az elemet a DOM-ból.

Szintaxis:

```
element.remove();
```

Példa:

```
let element = document.getElementById("elementToRemove");
element.remove();
```

Összegzés

Metódus	Leírás
`createElement`	Új HTML elemet hoz létre.
`createTextNode`	Új szövegcsomópontot hoz létre.
`append`	Új csomópontokat vagy szövegeket ad az elem végéhez.
`prepend`	Új csomópontokat vagy szövegeket ad az elem elejéhez.
`before`	Új csomópontokat vagy szövegeket ad az elem elé.
`after`	Új csomópontokat vagy szövegeket ad az elem után.
`replaceWith`	Kicseréli az elemet egy vagy több új csomópontra vagy szövegre.
`insertAdjacentHTML`	HTML szöveget illeszt be az elem egy megadott pozíciójába.
`insertAdjacentText`	Szöveget illeszt be az elem egy megadott pozíciójába.
`insertAdjacentElement`	Egy meglévő DOM elemet illeszt be az elem egy megadott pozíciójába.
`remove`	Eltávolítja az elemet a DOM-ból.

Ezek a metódusok alapvető eszközök a DOM manipulációban, amelyek lehetővé teszik az elemek dinamikus létrehozását, módosítását, beillesztését és eltávolítását.