Resurse suplimentare pe care recomand să le parcurgi

Modificarea conținutului elementelor

JavaScript oferă mai multe metode pentru a modifica conținutul elementelor HTML:

1. Proprietatea innerHTML

Proprietatea `innerHTML` permite citirea sau modificarea conținutului HTML al unui element:

```
// Citirea conținutului HTML
const container = document.querySelector('#container')
console.log(container.innerHTML) // Afișează tot conținutul HTML din interior

// Modificarea conținutului HTML
container.innerHTML = '<h2>Titlu nou</h2><p>Paragraf nou</p>'

// Adăugarea de conținut HTML (păstrând conținutul existent)
container.innerHTML += '<p>Încă un paragraf</p>'
```

Avantaje:

- Permite adăugarea rapidă de conținut HTML complex
- Operație simplă și directă

Dezavantaje:

- Poate fi vulnerabilă la atacuri de tip XSS (Cross-Site Scripting) dacă folosiți conținut nevalidat
- Elimină toți event listeners atașați elementelor existente
- Performanță scăzută pentru actualizări frecvente

2. Proprietatea textContent

Proprietatea `textContent` manipulează doar conținutul text al unui element, ignorând marcajul HTML:

```
// Citirea conținutului text
const title = document.querySelector('h1')
console.log(title.textContent) // Afișează doar textul, fără HTML

// Modificarea conținutului text
title.textContent = 'Titlu modificat'
```

Avantaje:

- Mai sigur decât innerHTML (evită injectarea de HTML)
- Mai eficient pentru manipularea textului simplu
- Returnează tot textul, inclusiv cel din elementele ascunse

3. Proprietatea innerText

Similar cu `textContent`, dar respectă stilurile CSS:

```
// Citirea textului vizibil
const paragraph = document.querySelector('p')
console.log(paragraph.innerText) // Afișează doar textul vizibil

// Modificarea textului
paragraph.innerText = 'Text nou în paragraf'
```

Diferențe față de textContent:

- `innerText` este afectat de CSS (nu returnează text din elemente ascunse)
- `innerText` poate declanșa recalcularea stilurilor (mai lent)
- `textContent` returnează conținutul exact, inclusiv spațiile și sfârșiturile de linie

4. Metoda createElement și appendChild

Pentru o abordare mai structurată și mai sigură, putem crea elemente noi și le putem adăuga în DOM:

```
// Crearea unui nou element
const newParagraph = document.createElement('p')

// Adăugarea de conținut în noul element
newParagraph.textContent = 'Acesta este un paragraf creat prin JavaScript'

// Adăugarea elementului în DOM
const container = document.querySelector('#container')
container.appendChild(newParagraph)
```

Manipularea atributelor elementelor

1. Metode standard pentru atribute

DOM oferă metode pentru a lucra cu atributele elementelor:

```
const link = document.querySelector('a')

// Citirea unui atribut
const hrefValue = link.getAttribute('href')
console.log(hrefValue) // Afișează valoarea atributului href

// Verificarea existenței unui atribut
const areTarget = link.hasAttribute('target')
console.log(areTarget) // true sau false

// Setarea unui atribut
link.setAttribute('target', '_blank')
link.setAttribute('title', 'Link extern')

// Eliminarea unui atribut
link.removeAttribute('target')
```

2. Proprietăți pentru atribute comune

Pentru atributele frecvent folosite, elementele HTML au proprietăți JavaScript directe:

```
const image = document.querySelector('img')

// Citirea atributelor prin proprietăți
console.log(image.src) // Adresa completă (absolută)
console.log(image.alt)
console.log(image.width)

// Modificarea atributelor prin proprietăți
image.src = 'images/new-image.jpg'
image.alt = 'Descriere nouă'
image.width = 300
```

Diferența dintre getAttribute și proprietăți directe:

- `getAttribute` returnează exact valoarea din HTML (de ex. imagine.getAttribute("src") poate returna o cale relativă)
- Proprietățile directe pot returna valori procesate (de ex. imagine.src va returna URL-ul complet, absolut)

3. Clasa classList

Pentru manipularea claselor CSS, obiectul `classList` oferă metode utile:

```
const element = document.querySelector('#myElement')

// Adăugarea unei clase
element.classList.add('highlight')

// Eliminarea unei clase
element.classList.remove('old-class')

// Verificarea existenței unei clase
if (element.classList.contains('active')) {
  console.log('Elementul este activ')
}

// Comutarea unei clase (adaugă dacă nu există, elimină dacă există)
element.classList.toggle('selected')

// Înlocuirea unei clase
element.classList.replace('old-class', 'new-class')

// Adăugarea mai multor clase
element.classList.add('bold', 'italic', 'underline')
```

Alternativ, putem manipula toate clasele deodată folosind proprietatea `className`:

```
// Citirea tuturor claselor ca string
console.log(element.className) // "class1 class2 class3"

// Înlocuirea tuturor claselor
element.className = 'new-class1 new-class2'
```

4. Atributul style

Pentru manipularea stilurilor inline, putem folosi proprietatea `style`:

```javascript
const box = document.querySelector('.box')

// Setarea proprietăților CSS individuale
box.style.color = 'red'
box.style.backgroundColor = '#f0f0f0'
box.style.padding = '10px'
box.style.borderRadius = '5px'

// Notă: Proprietățile CSS cu cratimă devin camelCase în JavaScript
// De exemplu: background-color devine backgroundColor
```

Alternativ, putem seta mai multe stiluri deodată folosind `cssText`:

```javascript
box.style.cssText = 'color: blue; font-size: 16px; text-align: center;'
```

5. Proprietatea dataset pentru atribute data-*

HTML5 permite definirea atributelor personalizate cu prefixul `data-`. JavaScript oferă acces la aceste atribute prin proprietatea `dataset`:

```html
<div id="user" data-id="123" data-name="Ion Popescu" data-role="admin"></div>
```

```javascript
const user = document.querySelector('#user')

// Citirea atributelor data-*
console.log(user.dataset.id) // "123"
console.log(user.dataset.name) // "Ion Popescu"
console.log(user.dataset.role) // "admin"

// Modificarea atributelor data-*
user.dataset.status = 'active'
user.dataset.lastLogin = '2023-05-15'

// Atributele cu mai multe cuvinte folosesc camelCase
// HTML: data-last-login se accesează ca dataset.lastLogin
```

Manipularea stilurilor prin JavaScript

1. Manipularea claselor vs. stiluri inline

Există două abordări principale pentru modificarea stilului elementelor:

Manipularea claselor (recomandată):

```
// CSS definit separat
/*
.error {
  color: red;
  border: 1px solid red;
  background-color: #ffebee;
}
*/

// JavaScript doar aplică sau elimină clasa
element.classList.add('error')
```

Stiluri inline (pentru modificări dinamice):

```
// Setare directă a stilurilor
element.style.color = 'red'
element.style.border = '1px solid red'
element.style.backgroundColor = '#ffebee'
```

Recomandări:

- Folosiți clasele CSS pentru stiluri consistente și mențină separarea preocupărilor
- Folosiți stiluri inline pentru modificări dinamice (de ex. animații, poziționare)

2. Obținerea stilurilor computate

Pentru a obține stilurile efective (după aplicarea CSS-ului), folosim `getComputedStyle()`:

```javascript
const element = document.querySelector('#myElement')
const styles = window.getComputedStyle(element)

console.log(styles.color) // rgb(255, 0, 0)
console.log(styles.fontSize) // 16px
console.log(styles.marginLeft) // 10px

// Pentru pseudo-elemente
const pseudoStyles = window.getComputedStyle(element, '::before')
console.log(pseudoStyles.content)
```

Manipularea formularelor

Formularele HTML au proprietăți și metode speciale:

```javascript
// Accesarea unui formular
const form = document.querySelector('#myForm')
// sau
const form = document.forms['myForm'] // accesare după atributul name

// Accesarea elementelor din formular
const username = form.elements['username'] // după atributul name
const password = form.querySelector("input[type='password']") // folosind querySelector

// Citirea și modificarea valorilor
console.log(username.value) // valoarea curentă
username.value = 'utilizator_nou' // modificarea valorii

// Verificarea stării checkbox-urilor și radio button-urilor
const acceptTerms = form.elements['terms']
console.log(acceptTerms.checked) // true sau false
acceptTerms.checked = true // bifează checkbox-ul

// Selectarea opțiunilor din select
const country = form.elements['country']
console.log(country.value) // valoarea opțiunii selectate
country.value = 'ro' // selectează opțiunea cu value="ro"

// Sau pentru select multiplu
const skills = form.elements['skills']
for (const option of skills.options) {
  if (option.value === 'javascript') {
    option.selected = true
  }
}

// Dezactivarea elementelor de formular
username.disabled = true
form.elements['submit-btn'].disabled = true
```

Metode elementare pentru inserare și eliminare

Iată câteva operații de bază pentru modificarea DOM-ului:

```
// Adaugă un element la sfârșitul unui container
container.appendChild(newElement)

// Adaugă la o poziție specifică
container.insertBefore(newElement, referenceElement)

// Metode moderne
container.append(newElement) // Similar cu appendChild, dar acceptă și text
container.prepend(newElement) // Adaugă la început
referenceElement.before(newElement) // Inserează înainte
referenceElement.after(newElement) // Inserează după

// Elimină un element
element.remove()

// Înlocuiește un element
oldElement.replaceWith(newElement)
```