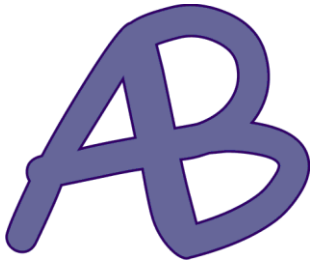
Aplikacje frontendowe



Wykład

Manipulowanie DOM-em
Tworzenie elementów HTML

innerHTML vs textContent vs innerText

Porównanie właściwości tekstowych (innerHTML, textContent, innerText)

- innerHTML – parsuje ciąg HTML i nadpisuje zawartość węzła
- textContent – ustawia/odczytuje czysty tekst, bez interpretacji HTML
- innerText – podobne do textContent, ale uwzględnia style CSS (np. display: none)

```
<div id="demo"></div>
<script>
  const el = document.getElementById("demo");
  el.innerHTML = "<strong>Bold</strong> tekst";
  console.log(el.innerHTML); // "<strong>Bold</strong> tekst"
  console.log(el.textContent); // "Bold tekst"
  console.log(el.innerText); // "Bold tekst"
</script>
```

innerHTML vs textContent vs innerText

innerHTML może prowadzić do XSS

XSS (Cross-Site Scripting) to rodzaj podatności bezpieczeństwa w aplikacjach webowych, która polega na umożliwieniu atakującemu wstrzyknięcia złośliwego kodu JavaScript do strony internetowej. Jeśli aplikacja bez odpowiedniego filtrowania wyświetla dane pochodzące od użytkownika przy użyciu np. innerHTML, atakujący może podstawić własny <script> i wykonać kod w przeglądarce innych użytkowników.

```
el.innerHTML = userInput;

// jeśli userInput = "<script>alert('XSS')</script>"
```

Z tego powodu zawsze należy **walidować i filtrować dane wejściowe**

Tworzenie i usuwanie elementów HTML

Dodawanie elementu

Kroki:

1. Tworzenie elementu - document.createElement(tagName)
2. Konfiguracja - (tekst, atrybuty, style)
3. Wstawienie do DOM - (appendChild / append / insertAdjacentElement)

Tworzenie i usuwanie elementów HTML

Dodawanie elementu

Metoda appendChild służy do dodawania nowego elementu jako ostatniego dziecka danego węzła (elementu) w drzewie DOM (Document Object Model).

```
parentNode.appendChild(newChild);
```

Przed dodaniem element należy utworzyć za pomocą metody createElement()

```
// Tworzymy nowy element <div>
let newDiv = document.createElement("div");
newDiv.textContent = "To jest nowy div";

// Znajdujemy element, do którego dodamy nowy element
let parentElement = document.getElementById("container");

// Dodajemy nowy element jako ostatnie dziecko elementu 'container'
parentElement.appendChild(newDiv);
```

Tworzenie i usuwanie elementów HTML

Dodawanie kilku elementów

Tworzy trzy nowe elementy listy <li> i dodaje je do elementu o id „myList”.

```
<ul id="myList"></ul>
<script>
  let list = document.getElementById("myList");
  let items = ["Element 1", "Element 2", "Element 3"];
  items.forEach((text) => {
    let listItem = document.createElement("li");
    listItem.textContent = text;
    list.appendChild(listItem);
  });
</script>
```

Tworzenie i usuwanie elementów HTML

Przenoszenie istniejącego elementu

Można także przenosić istniejące element w inne miejsca strony

```
let item = document.getElementById("item");
let newParent = document.getElementById("newParent");

// Przenosimy element 'item' do nowego rodzica 'newParent'
newParent.appendChild(item);
```

- ✓ Jeśli newChild już istnieje w DOM, zostanie przeniesiony z aktualnej lokalizacji do nowej.
- ✓ Element newChild może mieć tylko jednego rodzica w danym czasie. Przeniesienie go spowoduje jego usunięcie z aktualnego rodzica.
- ✓ Metoda appendChild zwraca dodany element newChild.

Tworzenie i usuwanie elementów HTML

DocumentFragment

- ✓ DocumentFragment to węzeł dokumentu, który może służyć jako kontener do grupowania innych węzłów.
- ✓ Typowym zastosowaniem jest utworzenie go, złożenie w nim poddrzewa DOM, a następnie dołączenie lub wstawienie fragmentu do modelu DOM
- ✓ Jest to węzeł typu dokumentu, który nie jest częścią aktywnego dokumentu DOM, co oznacza, że operacje na nim są wydajniejsze, ponieważ nie powodują wielokrotnego renderowania strony.

Korzyści związane z wydajnością są często zawyżane. W rzeczywistości w niektórych silnikach użycie a jest wolniejsze niż dołączanie do dokumentu w pętli.

Tworzenie i usuwanie elementów HTML

Przykład:

Utworzenie fragmentu zawierającego podpunkty listy na podstawie tablicy i dodanie go w całości do listy

```
<ul id="lista"></ul>
<script>
  const lista = document.getElementById("lista");
  const items = ["A", "B", "C"];
  // Wersja z DocumentFragment:
  const frag = document.createDocumentFragment();
  items.forEach((text) => {
    const li = document.createElement("li");
    li.textContent = text;
    frag.appendChild(li);
  });
  lista.appendChild(frag);
</script>
```

W tym przykładzie fragment tworzony jest w pamięci, elementy dodajemy do fragmentu, a potem dołączany do DOM przy użyciu metody append.

Tworzenie i usuwanie elementów HTML

Przykład – wersja 2

Podobnie jak w poprzednim przykładzie – tylko dla odmiany używamy innej pętli

```
const ul = document.querySelector("ul");
const fruits = ["Apple", "Orange", "Banana", "Melon"];

const fragment = new DocumentFragment();

for (const fruit of fruits) {
  const li = document.createElement("li");
  li.textContent = fruit;
  fragment.append(li);
}

ul.append(fragment);
```

Tworzenie i usuwanie elementów HTML

Przykład: - Tworzenie fragmentu jako niepowiązanej gałęzi drzewa DOM

```
const ul = document.querySelector("ul");
const fruits = ["Apple", "Orange", "Banana", "Melon"];

const fragment = document.createDocumentFragment();

for (const fruit of fruits) {
  const li = document.createElement("li");
  li.textContent = fruit;
  fragment.appendChild(li);
}

ul.appendChild(fragment);
```

Można też utworzyć fragment jako niepowiązaną gałąź dokumentu, a następnie przenieść go do właściwego modelu DOM

Tworzenie i usuwanie elementów HTML

Metoda Element.remove() usuwa element z modelu DOM.

```
const element = document.getElementById(„id_elementu");
element.remove(); // Usuwa element
```

Tworzenie i usuwanie elementów HTML

Metoda removeChild() interfejsu usuwa węzeł podrzędny z modelu DOM i zwraca usunięty węzeł.

```
    <div id="parent">
      <div id="child"></div>
    </div>
//-----------------------------------------------------
      const parent = document.getElementById("parent");
      const child = document.getElementById("child");
      const throwawayNode = parent.removeChild(child);
```

Tworzenie i usuwanie elementów HTML

Aby usunąć określony element bez konieczności określania jego węzła nadrzędnego:

```
const node = document.getElementById("child");
if (node.parentNode) {
  node.parentNode.removeChild(node);
}
```

Sprawdzamy, czy odnaleziony element ma rodzica. Jeżeli tak usuwamy potomka tego rodzica – czyli wskazany element

Aby usunąć wszystkie elementy podrzędne z elementu:

```
const element = document.getElementById("idOfParent");
while (element.firstChild) {
  element.removeChild(element.firstChild);
}
```

Dopóki istnieje pierwszy potomek – usuwamy pierwszego potomka

Zarządzanie klasami

Zarządzanie klasami (classList)

classList to interfejs umożliwiający łatwe zarządzanie klasami CSS elementu DOM. Oferuje metody manipulujące klasami:

- add(className) Dodaje klasę, jeśli jej jeszcze nie ma
- remove(className) Usuwa klasę, jeśli istnieje
- toggle(className) Przełącza klasę – dodaje jeśli nie ma, usuwa jeśli jest
- toggle(className, bool) Dodaje lub usuwa klasę w zależności od wartości logicznej bool
- contains(className) Zwraca true, jeśli klasa istnieje na elemencie
- replace(old, new) Zastępuje klasę old nową klasą new
- length Liczba klas przypisanych do elementu
- item(index) Zwraca nazwę klasy pod danym indeksem lub null

Zarządzanie klasami (classList)

```
<div id="box" class="box"></div>
<style>
  .highlight {
    background-color: yellow;
  }
  .hidden {
    display: none;
  }
  .box {
    width: 100px;
    height: 100px;
    border: 1px solid #333;
  }
</style>
```

```
<script>
  const box = document.getElementById("box");

  // Dodanie klasy
  box.classList.add("highlight");

  // Usunięcie klasy
  box.classList.remove("box");

  // Przełączenie klasy (toggle)
  box.classList.toggle("hidden");

  // Wymuszone dodanie klasy (toggle z bool)
  box.classList.toggle("highlight", true);

  // Sprawdzenie, czy klasa istnieje
  console.log(box.classList.contains("highlight")); // true

  // Zastąpienie jednej klasy inną
  box.classList.replace("highlight", "box");
</script>
```

Atrybuty

Atrybuty

Atrybuty – sprawdzanie warości:

Metoda getAttribute() interfejsu Element zwraca wartość określonego atrybutu

Jeśli dany atrybut nie istnieje, zwrócona wartość będzie równa .null

```
//w HTML
        <div id="div1">Cześć!</div>

//w konsoli
      const div1 = document.getElementById("div1");
      //=> <div id="div1">Cześć!</div>
      const exampleAttr = div1.getAttribute("id");
      //=> "div1"
      const align = div1.getAttribute("align");
      //=> null
```

Atrybuty

Atrybuty - modyfikacja:

- setAttribute(name, value) – ustawia atrybut niezależnie od typu
- removeAttribute(name) – usuwa atrybut
- hasAttribute(name) – sprawdza, czy atrybut istnieje

Atrybuty

Atrybuty:

Metoda setAttribute() interfejsu Element ustawia wartość atrybutu dla określonego elementu. Jeśli atrybut już istnieje, wartość jest aktualizowana; W przeciwnym razie dodawany jest nowy atrybut o określonej nazwie i wartości.

Aby uzyskać bieżącą wartość atrybutu, należy użyć getAttribute(); Aby usunąć atrybut, wywołaj metodę removeAttribute().

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

button.setAttribute("name", "helloButton");
button.setAttribute("disabled", "");
```

Atrybuty

Atrybuty – getAttribute vs. Właściwości

- ✓ Właściwości (np. element.href, element.id) są częścią obiektu DOM i mogą automatycznie zwracać pełne, przetworzone wartości.
- ✓ getAttribute() odczytuje dokładnie taką wartość, jaka została ustawiona w HTML.

Uwaga: Właściwości i atrybuty nie zawsze są zgodne – zwłaszcza w przypadku adresów URL, checked, value, itp.

Standardowe atrybuty to: id, href, value, src, type, name

Atrybuty

Atrybuty – getAttribute vs. Właściwości

```
<a id="link" href="https://example.com" data-info="sekret">Link</a>
<input id="input1" value="123" />
<script>
  const link = document.getElementById("link");
  const input = document.getElementById("input1");

  console.log(link.href);   // https://example.com/ (pełny URL)
  console.log(link.getAttribute("href")); // "https://example.com"
                                          //(dokładna wartość z HTML)

  console.log(input.value);   // "123" (aktualna wartość inputa)
  console.log(input.getAttribute("value")); // "123" (początkowa wartość z HTML)

  // Dane niestandardowe
  console.log(link.dataset.info); // "sekret"
  console.log(link.getAttribute("data-info")); // "sekret"
</script>
```

Atrybuty

Niestandardowe (custom): data-* (odczyt przez dataset lub getAttribute())

Atrybut data-* (czyt. „data dash”) to sposób na przechowywanie niestandardowych danych w znacznikach HTML w sposób zgodny ze specyfikacją.

```
<div id="produkt" data-id="123" data-kategoria="elektronika">
   Laptop
</div>
```

- ✓ Atrybuty muszą zaczynać się od data-, np. data-id, data-kategoria.
- ✓ W JavaScript są dostępne przez obiekt dataset

```
<script>
  const el = document.getElementById("produkt");
  console.log(el.dataset.id); // "123"
  console.log(el.dataset.kategoria); // "elektronika"
</script>
```

Style

Style

Style:

W JavaScript możemy modyfikować style elementów HTML na kilka różnych sposobów.

1. Używanie właściwości „style”. Pozwala ona bezpośrednio modyfikować styl elementu.
2. Dynamiczne modyfikowanie klas (zawartych w pliku .css)
3. Ustawianie wielu własności stylu jednocześnie za pomocą właściwości „cssText”
4. Dynamiczne tworzenie nowych reguł stylów i dodawanie ich do dokumentu

Style

1. Używanie właściwości „.style”. Pozwala ona bezpośrednio modyfikować styl elementu. Umożliwia dostęp do stylów inline (atrybut style w HTML)

Korzystamy z niej, aby dynamicznie ustawić poszczególne właściwości CSS.

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

element.style.color = "red";
element.style.backgroundColor = "yellow";
element.style.fontSize = "20px";
```

Uwaga: nazwy właściwości CSS w JS zapisujemy w camelCase, np. background-color → backgroundColor

Style

2. Używanie klasy CSS - Stylowanie elementów za pomocą klas CSS jest bardziej przejrzyste i skalowalne. Możemy dodawać lub usuwać klasy za pomocą metod „classList.add” i „classList.remove”.

```
//css
.tlo {
  color: white;
  background-color: blue;
  font-weight: bold;
}
```

```
//JavaScript
    let element = document.getElementById("myElement");
    element.classList.add(„tlo");
```

Patrz slajd:
Zarządzanie klasami (classList)

Style

2. Używanie klasy CSS

Zaawansowana technika: dynamiczne tworzenie arkuszy stylów i klas CSS

```
const style = document.createElement("style");
style.textContent = `
.dynamcznyStyl {
  color: white;
  background-color: green;
  padding: 10px;
}
`;
document.head.appendChild(style);
document.getElementById("myElement").classList.add(" dynamcznyStyl ");
```

Style

3. Możemy także ustawić wiele własności stylu jednocześnie za pomocą właściwości `cssText`.

```
let element = document.getElementById('myElement');
element.style.cssText =
    'color: purple; background-color: lightgray; border: 1px solid black;';
```

Style

4. Możemy dynamicznie tworzyć nowe reguły stylów i dodawać je do dokumentu

```
let style = document.createElement("style");
style.textContent = `
.myClass {
color: blue;
background-color: lightgreen;
padding: 10px;
}
`;
document.head.appendChild(style);
```

Powyższy kod tworzy nowy element `<style>`, definiuje styl dla klasy o nazwie`.myClass`, a następnie dodaje go do nagłówka dokumentu

Dzięki temu możemy oscylować wiele elementów jednocześnie

Przykład - część 1

- Skrypt który utworzy tabelę z „n” losowymi liczbami naturalnymi (od 1 do 10000), rozmieszczonymi w wierszach po 10 kolumn.
- Liczba „n” jest również losowana (od 50 do 200).

8193	2568	3089	8988	3668	7423	3145	746	8406	8125
7193	9012	3082	6148	6066	6328	6986	3807	4563	681
7818	1407	2384	5873	4992	4037	7728	824	6303	3930
2582	3665	3588	8950	6100	4877	751	7464	3262	7665
1575	2314	8817	4442	4282	9196	9856	364	8094	4600
9162	850	2439	426	9776	7308				

Przykład

```
<!DOCTYPE html>
<html lang="pl">
  <head>
    <meta charset="UTF-8" />
    <title>Tabela liczb losowych</title>
    <style>
      table {
        border-collapse: collapse;
        width: 80%;
      }
      td {
        border: 1px solid #333;
        padding: 4px;
        text-align: center;
      }
    </style>
  </head>
  <body>
    <div id="container"></div>
  </body>
</html>
```

W dokumencie umieszczamy kontener, w którym umieścimy tabelę

Przykład

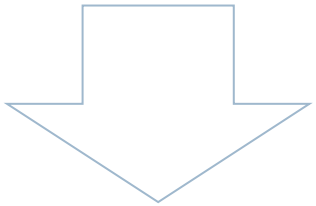
```
<script>
      const n = Math.floor(Math.random() * (200 - 50 + 1)) + 50;

      const container = document.getElementById("container");

      const table = document.createElement("table");
      const tbody = document.createElement("tbody");
```

Losowanie liczby n z przedziału od 50 do 200

Pobranie kontenera



Utworzenie elementów tabeli <table> i <tbody>

Przykład

Tworzenie wierszy tabeli

```
for (let i = 0; i < n; i += 10)
      {
        const tr = document.createElement("tr");
        for (let j = 0; j < 10 && i + j < n; j++)
        {
          const td = document.createElement("td");
          const value = Math.floor(Math.random() * 10000) + 1;
          td.textContent = value;
          tr.appendChild(td);
        }
        tbody.appendChild(tr);
      }
```

Nowy wiersz

Tworzenie do 10 komórek w wierszu (lub mniej jeśli to koniec danych)

Nowa komórka

Liczba 1–10000

Wpisanie liczby do komórki

dodanie komórki do wiersza

dodanie wiersza do ciała tabeli <tbody> utworzonego na poprzednim slajdzie.

Przykład

Dołączenie gotowych elementów do drzewa DOM

```
    table.appendChild(tbody);
    container.appendChild(table);
</script>
```

Dołączenie <tbody> do tabeli <table>

Dołaczenie gotowej tabeli do contenera zdefiniowanego w pliku html

Przykład - część 2

Skrypt który będzie przeglądał i modyfikował tabelę wygenerowaną przez poprzedni skrypt.

- Skrypt sprawdzi w których komórkach tabeli znajdują się liczby podzielne przez 11 i zmieni ich tło na zielone.

2756	2177	3915	5272	9488	7743	3457	7198	7141	4314
8255	3297	9238	9052	9666	1331	6481	5454	152	3320
7247	7261	7631	7434	6701	8813	2112	4048	1170	2643
4840	6473	1245	4174	187	9920	436	8462	8661	6977
1739	3065	2811	2730	1238	3493	7340	5541	2430	7293
9482	6602	8125	7946	7864	7552	2852	2332	2493	8829
41	7367	1575	772	7741	6333				

Przykład

Po wygenerowaniu tabeli przeszukujemy wszystkie komórki

```
<script>
  const cells = table.getElementsByTagName("td");
  for (const cell of cells)
   {
      const value = parseInt(cell.textContent);
      if (!isNaN(value) && value % 11 === 0)
      {
        cell.style.backgroundColor = "lightgreen";
      }
   }
</script>
```

getElementsByTagName zwróci strukturę HTMLCollection zawierającą wszystkie elementy o <td> czyli komórki tabeli

```
> table.getElementsByTagName("td");
< HTMLCollection(127) [td, td, td, td, td, td, td, td, td, td, td, td, td,
  td, td, td, td, td, td, td, td, td, td, td, td, td, td, td, td, td, td,
  td, td, td, td, td, td, td, td, td, td, td, td, td, td, td, td, td, td,
  td, td, td, td, td, td, td, td, td, td, td, td, td, td, td, td, td, td,
  td, td, td, td, td, td, td, td, td, td, td, td, td, td, td, td, td, td,
  td, td, td, td, td, td, td, td, td, td, td, td, td, td, …]
```

HTMLCollection „cells” można iterować jak tablicę – uzyskujemy w ten sposób dostęp do każdej komórki

Jeżeli zwrócona wartość nie jest pusta i jest podzielna przez 11, sięgamy do struktury style i zmieniamy jej własność odpowiadającą za kolor tła

Z przeglądany komórki pobierany zawartość i konwertujemy na liczbę

Literatura:

- Negrino Tom, Smith Dori, ***Po prostu JavaScriopt i Ajax, wydanie VI***, Helion, Gliwice 2007.
- Lis Marcin, JavaScript, Ćwiczenia praktyczne, wydanie II, Helion, Gliwice 2007.
- http://www.w3schools.com/JS/js_popup.asp
- Beata Pańczyk, wykłady opublikowane na stronie http://www.wykladowcy.wspa.pl/wykladowca/pliki/beatap