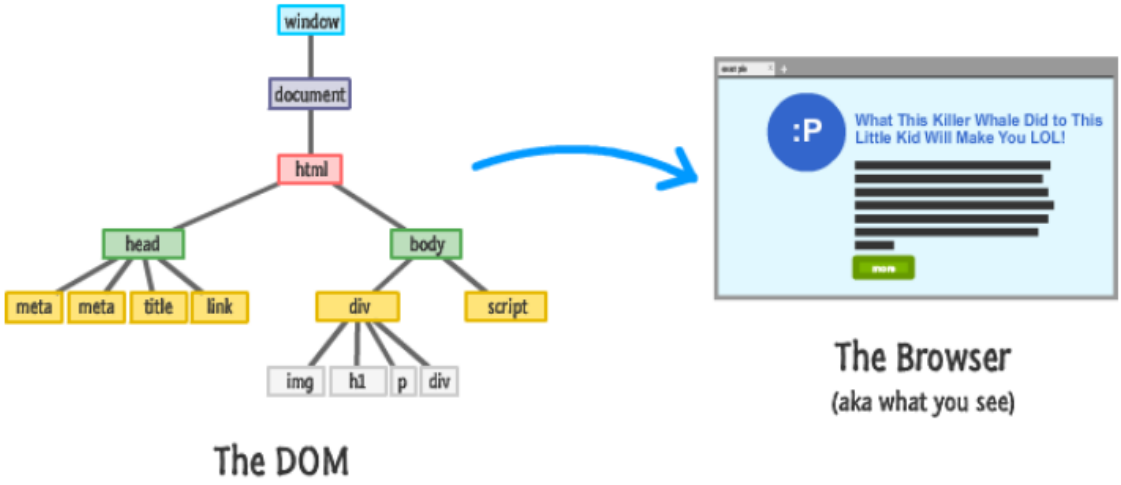
Temat 9

Temat: Model DOM 3- Przechodzenie przez DOM, tworzenie i usuwanie elementów

Przechodzenie przez DOM

Wiesz już, że DOM wygląda jak drzewo. Elementy w DOM są ułożone w hierarchii, która definiuje to, co ostatecznie widzisz w przeglądarce:



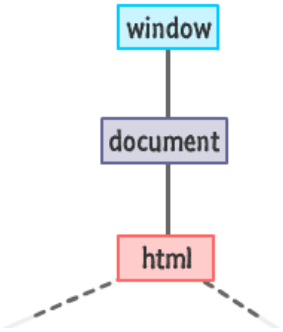
Aby operować na takich obiektach, musimy dobrze opanować sztukę "spacerowania" po nich.

Każdy element na stronie tworzy tak zwany **node** czyli pojedynczy węzeł drzewa. Takimi nodami są nie tylko elementy, ale także tekst w nich zawarty. Nas głównie będą interesować nody, które są elementami - np. buttony, divy itp.

Znalezienie właściwej drogi

Zanim znajdziesz elementy i wykonasz na nich operacje, musisz najpierw dotrzeć do miejsca, w którym znajdują się te elementy. Najprostszym sposobem rozwiązania tego problemu jest po prostu rozpoczynanie od góry i przesuwanie w dół.

Widok z góry DOM składa się z elementów okna , dokumentu i html :



Ze względu na to, jak ważne są te trzy rzeczy, DOM zapewnia łatwy dostęp do nich poprzez **window**, **document** i **document.documentElement**:

```
var windowObject = window;
var documentObject = document;
var htmlElement = document.documentElement;
```

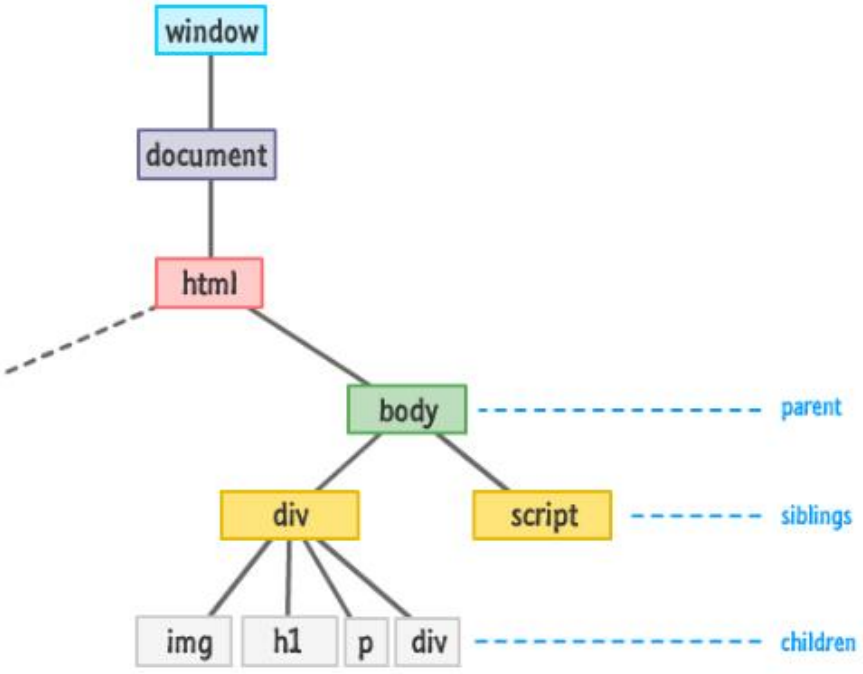
Należy zauważyć, że zarówno **window**, jak i **document** są właściwościami globalnymi. Nie musisz ich jawnie deklarować.

Najwyższe węzły drzewa są dostępne bezpośrednio jako documentwłaściwości:

- **<html> = document.documentElement** – najwyższy węzeł dokumentu to document.documentElement. To jest węzeł DOM tagu <html>.
- **<body> = document.body** – szeroko stosowany węzeł DOM
- **<head> = document.head** –<head> Znacznik jest dostępny przez document.head.

Gdy przejdziesz poniżej poziomu elementu HTML, DOM zaczyna się rozgałęziać. W tym momencie istnieje kilka sposobów poruszania się. Jednym ze sposobów jest użycie znanych Ci już **querySelector i querySelectorAll** pozwalające precyzyjnie uzyskać pożądane elementy. Gdy jednak **nie wiesz, gdzie chcesz się udać**, metody querySelector i querySelectorAll nie będą wystarczające.

Warto pamiętać, że wszystkie elementy w DOM mają co najmniej jedną kombinację **rodziców** , **rodzeństwa** i **dzieci:**



Niemal każdy element, w zależności od punktu startowego, może odgrywać wiele rodzinnych ról.

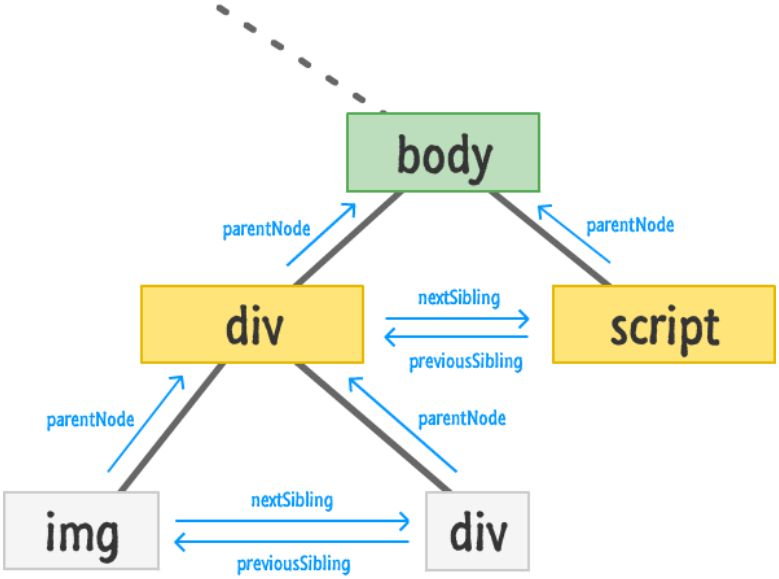
Do przechodzenia między nimi można użyć kilku właściwości, np.: **firstChild, lastChild, parentNode, children, previousSibling, nextSibling**

Rodzeństwo i rodzic

Z tych właściwości najłatwiej jest poradzić sobie z rodzicami i rodzeństwem. Odpowiednie właściwości to **`parentNode , previousSibling i nextSib`**ling .
Rodzeństwo to węzeł, który jest dzieckiem tego samego rodzica. Na przykład <head> i <body> są rodzeństwem:

- <body> - mówimy, że jest "następnym" lub "prawym" rodzeństwem <head>,
- <head> -mówimy, że jest "poprzednim" lub "pozostawionym" rodzeństwem <body>.

Poniższy schemat przedstawia sposób działania tych trzech właściwości:



Właściwość **`parentNod`**e wskazuje na element nadrzędny elementu. Właściwości **`previousSibling i nextSibling`** umożliwiają elementowi znalezienie poprzedniego lub następnego rodzeństwa.

Dzieci: `childNodes, firstChild, lastChild, children`

Są dwa terminy, z których będziemy od teraz korzystać:

- **Węzły potomne (lub dzieci)** - elementy będące bezpośrednimi dziećmi. Innymi słowy, są one zagnieżdżone dokładnie w danym elemencie. Na przykład, <head> i <body> są dziećmi elementu <html>.
- **Potomkowie** - wszystkie elementy, które są zagnieżdżone w danym, w tym dzieci, ich dzieci i tak dalej.

Na przykład w poniższym kodzie:

```
<html>
<body>
  <div>Begin</div>

  <ul>
    <li>
      <b>Information</b>
    </li>
  </ul>
</body>
</html>
```

<body> ma dzieci i <ul> (i kilka pustych węzłów tekstowych).

Z kolei wszyscy potomkowie <body>, to bezpośrednie dzieci: , <ul>, ale także więcej elementów zagnieżdżonych, takich jak <li> (jest dzieckiem <ul>) i <b> (jest dzieckiem <li>).

`childNodes` - Kolekcja zapewnia dostęp do wszystkich węzłów potomnych, w tym węzły tekstowe.

Przykład 1. Przygotuj stronę html wykorzystującą poniższy kod. Zapisz przykład jako *T9p1.html*

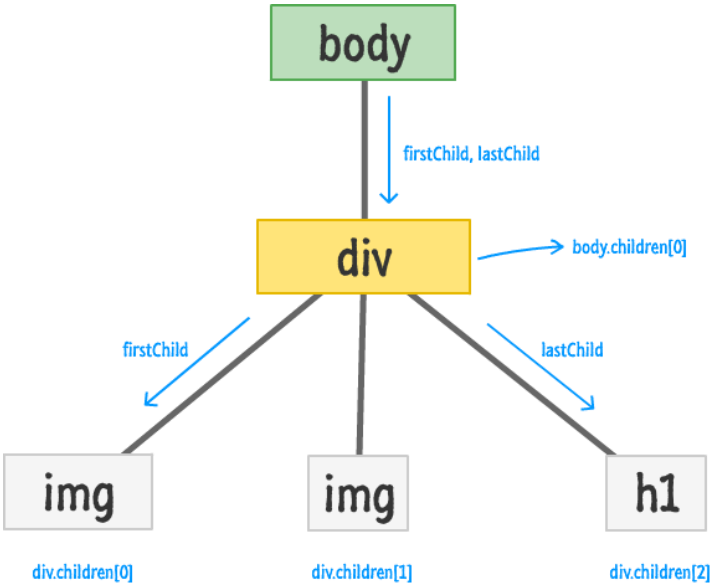
```
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="utf-8">
    <title>DOM, przechodzenie po drzewie - T9p1</title>
    <style>
    body{
    background-color: #002255;
    color: #fff;
    font-size: 18px;
    }
    </style>
</head>
<body>
    <div>Początek</div>
    <ul>
        <li>Informacje</li>
    </ul>
    <div>Koniec</div>

    <script>
    for (var i = 0; i < document.body.childNodes.length; i++) {
      alert( document.body.childNodes[i] );
    }
    </script>
    ...więcej rzeczy...
</body>
</html>
```

Zwróć uwagę na interesujący szczegół. Jeśli uruchomimy powyższy przykład, ostatni pokazany element to <script>. W rzeczywistości dokument ma jeszcze element poniżej, ale w momencie wykonania skryptu przeglądarka go jeszcze nie przeczytała, więc skrypt go nie widzi.

`childNodes` wygląda jak tablica, to jednak raczej *kolekcja* - specjalny obiekt podobny do tablicy, który możemy przeglądać tak jak tablicę. Kolekcja ta ma również długość, czyli właściwość, która podaje liczbę dzieci.

`firstChild , lastChild i children`



Właściwości **`firstChild i lastChild`** odnoszą się do pierwszego i ostatniego elementu potomnego rodzica. Jeśli rodzic ma tylko jedno dziecko, tak jak w przypadku elementu body w przykładzie na powyższym diagramie, to zarówno firstChild, jak i lastChild wskazują na to samo. Jeśli element nie ma elementów podrzędnych, wówczas te właściwości zwracają wartość **`null`** .

Sprawdzanie, czy dziecko istnieje

Aby sprawdzić, czy element ma dziecko, możesz wykonać:

Przykład 2. Przygotuj stronę html wykorzystującą poniższy kod. Zapisz przykład jako *T9p2.html*

```
<body>
    <div>Początek</div>
    <ul>
        <li>Informacje</li>
    </ul>
    <div id="div2"></div>
    <script>
    //Sprawdzanie, czy istnieje dziecko w body i wyświetlenie kilku dzieci
    var bodyElement = document.body;
    if (bodyElement.firstChild) {
        alert("są dzieci");
        alert("Pierwsze dziecko:"+bodyElement.firstChild);
        alert("Drugie dziecko:"+bodyElement.firstChild.nextSibling);
        alert("Ostatnie dziecko:"+bodyElement.lastChild);
    }
    else {
        alert("dzieci brak");
    }
    //Sprawdzanie, czy istnieje dziecko  w drugim div
    var divElement = document.getElementById('div2');
    if (divElement.firstChild) {
        alert("W drugim divie są dzieci");
    }
    else {
        alert("W drugim divie dzieci brak");
    }
    </script>
</body>
```

Przykład 3. Przygotuj stronę html wykorzystującą poniższy kod. Przeanalizuj dokładnie kod oraz wynik działania skryptu w konsoli. Zapisz przykład jako *T9p3.html*

```
<body>
    <div class="text-cnt">
    <p id="text">
        Mała
        <strong style="color:red">Ala</strong>
        miała
        <span style="color:blue">kota</span>
    </p>
    </div>
    <script>
    var text = document.querySelector('#text');

        console.log(text.parentElement) //wskazuje na nadrzędny nod będący elementem - div.text-cnt
        console.log(text.parentNode) //wskazuje na nadrzędny nod - div.text-cnt

        console.log(text.firstChild) //"Mała "
        console.log(text.lastChild) //"" - html jest sformatowany, więc ostatnim nodem jest znak nowej linii

        console.log(text.firstElementChild) //pierwszy element - <strong style="color:red">Ala</strong>
        console.log(text.lastElementChild) //ostatni element - <span style="color:blue">kota</span>

        console.log(text.children); //[strong, span] - kolekcja elementów
        console.log(text.children[0]) //wskazuje na 1 element - <strong style="color:red">Ala</strong>

        console.log(text.childNodes) //[text, strong, text] - kolekcja wszystkich dzieci - nodow
        console.log(text.childNodes[0]) //"Mała"

        console.log(text.firstElementChild.nextElementSibling) //kolejny brat-element pierwszego elementu - <span style="color:blue">kota</span>
        console.log(text.firstElementChild.nextSibling) //kolejny brat-node pierwszego elementu - "miała"

        console.log(text.firstElementChild.previousElementSibling) //poprzedni brat-element pierwszego elementu - null, bo przed pierwszym strong nie ma elementow
        console.log(text.firstElementChild.previousSibling) //poprzedni brat-node pierwszego elementu - "Mała"

        //powyższe możemy łączyć
        console.log(text.children[0].firstChild) //pierwszy element i w nim pierwszy nod : "Ala"
        console.log(text.children[0].firstElementChild) //null - w pierwszym strong nie mamy już elementów
        console.log(text.firstChild.firstElementChild) //null - nie ma elementu w pierwszym tekście
        console.log(text.firstElementChild.firstElementChild) //null - nie ma elementy w strong
        console.log(text.firstElementChild.firstChild) //"Ala"
    </script>
```

Tworzenie i usuwanie elementów

Bardzo często aplikacje i aplikacje interaktywne dynamicznie tworzą elementy HTML i wstawiają je do DOM.

Tworzenie obiektu za pomocą `createElement`

Aby utworzyć nowy element na stronie możemy skorzystać z metody

```
document.createElement(typ)
```

Sposób działania createElement jest dość prosty. Wywołujesz go za pośrednictwem obiektu dokumentu i podajesz nazwę tagu elementu, który chcesz utworzyć. W poniższym fragmencie tworzysz element akapitu reprezentowany przez literę p :

```
document.createElement("p");
```

Po utworzeniu nowego elementu warto też ustawić jego właściwości:

```
var el = document.createElement("div");

el.id = "myDiv";
el.innerText = "Tekst w divie";
el.setAttribute("title", "To jest tekst w dymku");
el.classList.add("module");
el.style.setProperty("background-color", "#FF6633");
```

Jeśli uruchomisz tę linię kodu jako część aplikacji, zostanie ona wykonana i zostanie utworzony element. Tworzenie elementu jest prostą częścią. To jednak nie wystarczy. Utworzony w powyższy sposób element jest dostępny dla skryptu, ale nie ma go jeszcze w drzewie dokumentu. Musimy go tam wstawić metodą

`parentElement.appendChild(nowyElement)`

Przykład 4. Przygotuj stronę html wykorzystującą poniższy kod. Zapisz przykład jako *T9p4.html*

```
    <style>
    body{
    background-color: #002255;
    color: #fff;
    font-size: 18px;
    }
    .test-first{
    background-color: #333;
    height:400px;
    width:400px;
    }
    .module{
    background-color: #999;
    height:200px;
    width:200px;
    }
    </style>
</head>
<body>
        <div class="test-first">
        </div>
        <script>
            // tworzymy element
            var el = document.createElement("div");
            el.id = "myDiv";
            el.innerText = "Tekst w divie";
            el.setAttribute("title", "To jest tekst w dymku");
            el.classList.add("module");
            var div = document.querySelector(".test-first"); //pobieramy miejsce docelowe
            div.appendChild(el); //wstawiamy element do drzewa dokumentu
    </script>
</body>
```

Aby utworzyć węzły DOM, istnieją dwie metody:

- `document.createElement(tag)` – Tworzy nowy element z podanym znacznikiem:
  `var div = document.createElement('div');`
- `document.createTextNode(text)` –Tworzy nowy *węzeł tekstowy* z podanym tekstem:
  `var textNode = document.createTextNode('Jestem tu');`

Do wstawiania elementów na stronę używaliśmy dotychczas innerHTML. Za pomocą innerHTML wstawiamy kod html w dany element. Ta właściwość nie daje jednak referencji do elementów we wstawianym html. Bardzo często będziemy chcieli za chwilę wykonać jakąś akcję na wstawianych elementach - np podpiąć im kliknięcie, zmienić tekst itp. Jeżeli będziemy korzystać z innerHTML, będziemy musieli te elementy po wstawieniu dodatkowo pobrać.

Przykład 5. Przygotuj stronę html wykorzystującą poniższy kod. Zapisz przykład jako *T9p5.html*

```
    <style>
    body{
    background-color: #002255;
    color: #fff;
    font-size: 18px;
    }
    .summer{
    background-color: #999;
    height:100px;
    color: #fff;
    font-size: 30px;
    }
    </style>
</head>
<body>
        <h1 id="theTitle" class=" summer">Co tu się dzieje?</h1>
        <button onclick=insert()>Dodaj tekst</button>
    <script>
    function insert() {
        var newElement = document.createElement("p");
        var tekst= prompt("Podaj tekst do wstawienia")
        newElement.textContent = tekst;

        document.body.appendChild(newElement);
        }
    </script>
</body>
```

Zwróć uwagę, że **`parentElem.appendChild(node)`** dołącza element **`node`** jako ostatnie dziecko **`parentElem`**.

Jeśli chcesz wstawić nowy element w innym miejscu, możesz to zrobić, wywołując funkcję **`insertBefore`**. Funkcja **`insertBefore`** przyjmuje dwa argumenty. Pierwszym argumentem jest element, który chcesz wstawić. Drugi argument jest odniesieniem do rodzeństwa (znanego również jako dziecko rodzica), które chcesz poprzedzić.

Przykład 6. Przygotuj stronę html wykorzystującą poniższy kod. Zapisz przykład jako *T9p6.html*

```
<body>
    <h1 id="theTitle">Lista zakupów</h1>
    <button onclick=insert1()>Dodaj pozycję na drugim miejscu</button>
    <button onclick=insert2()>Dodaj pozycję na przedostatnim miejscu</button>
    <ol id="list">
      <li>pieczywo</li>
      <li>mleko</li>
      <li>ser</li>
    </ol>

    <script>
    function insert1() {
        var newLi = document.createElement('li');
        var tekst= prompt("Podaj co jeszcze kupić")
        newLi.innerText = tekst;

        var list= document.querySelector("#list");
        list.insertBefore(newLi, list.children[1]);
        }
    function insert2() {
        var newLi = document.createElement('li');
        var tekst= prompt("Podaj co jeszcze kupić")
        newLi.innerText = tekst;

        list.insertBefore(newLi, list.lastChild.previousSibling);
        }
    </script>
</body>
```

Usuwanie elementów

Aby usunąć element, możemy:

1. wywołać funkcję **`removeChild`** na rodzicu elementu, który chcemy usunąć.

Jeżeli nie mamy bezpośredniego dostępu do elementu nadrzędnego elementu i nie chcemy tracić czasu na znajdowanie go, można usunąć ten element za pomocą właściwości **`parentNode`.**

```
Element.parentNode.removeChild(Element);
```

Przykład7. Przygotuj stronę html wykorzystującą poniższy kod. Zapisz przykład jako *T9p7.html.*

```
    <style>
    body{
    background-color: #002255;
    color: #fff;
    font-size: 18px;
    }
    .module{
    background-color: #009;
    height:100px;
    width:100px;
    border: 1px solid #fff;
    margin-bottom:5px;
    }
    </style>
</head>
<body>
        <div class="test">
        </div>
        <button onclick=add()>Dodaj element</button>
        <button onclick=del()>Usuń element</button>
        <script>
        function add() {
            var el = document.createElement("div");
            el.id = "myDiv";
            el.innerText = "Tekst w divie";
            el.classList.add("module");
            var div = document.querySelector(".test");
            div.appendChild(el);
            }
        function del() {
            var div= document.querySelector(".test");
            var el=div.lastChild;//ostatnie dziecko
            el.parentNode.removeChild(el);//przechodzimy do rodzica, aby móc usunąć element
            }
    </script>
```

Usuwamy element , wywołując removeChild na jego obiekcie nadrzędnym, podając element.parentNode . Wygląda to okrężnie, ale działa dobrze.

2. wywołać funkcję **`node.remove()`**, która usuwa node z jej miejsca. Np.:
   `var p = document.querySelector("#paragraf");`
   `p.remove();`

W przeciwieństwie do **`removeChild`** metoda **`remove()`** nie jest wspierana przez przeglądarki IE. Jeżeli musisz je wspierać, wtedy powinieneś użyć removeChild.

Podczas realizacji ćwiczeń wykorzystaj narzędzia poznane podczas tej lekcji.

Ćwiczenie 1 Przygotuj stronę html. Umieść w niej div oraz przycisk uruchamiający funkcję wstawiającą dynamicznie do przygotowanego diva: tekst nagłówkowy „Polecane strony", dwa linki do wybranych stron oraz obrazek.

Zapisz skrypt pod nazwą t9cw1.html

Ćwiczenie 2 Przygotuj stronę html. Umieść w niej:

- nagłówek z tekstem: „Pudełka niespodzianek",
- pustego diva z ustawioną klasą, w której określisz kolor jego tła, minimalną wysokość oraz właściwość pozwalającą ustawiać elementy blokowe obok siebie (użyj flexboxa)
- przycisk uruchamiający funkcję wstawiającą dynamicznie do przygotowanego diva:
    - dodatkowy div - ustaw mu dynamicznie atrybut klasy i w stylach sformatuj klasę, podając wymiary kwadratowego pudełka (np. 200px), kolor tła oraz obramowanie
- przycisk uruchamiający funkcję wstawiającą dynamicznie do **ostatniego utworzonego diva** jego zawartość:
    - akapit, zawierający tekst podawany przez użytkownika
    - obrazek. Dodaj potrzebne atrybuty dodawanego obrazka (src oraz klasę). W klasie obrazka ustaw jego wymiary tak, aby nie przekraczały wymiarów rodzica(ostatniego utworzonego diva). Możesz wykorzystać obrazki z poprzedniej lekcji (lub własne) oraz dodać możliwość podania przez użytkownika wartości atrybutu src obrazków.
- przycisk uruchamiający funkcję usuwającą dynamicznie utworzone pudełka **zaczynając od pierwszego**

Przetestuj działanie skryptu uruchamiając go przyciskami w różnej kolejności.

Zapisz skrypt pod nazwą T9cw2.html

Ćwiczenie 3 Przygotuj skrypt tworzący tabliczkę mnożenia liczb od a*a do b*b., w którym liczby a i b podaje użytkownik w formularzu. Okoduj odpowiednie komunikaty związane z błędami przy podawaniu danych. Ustaw odpowiednie style strony i tabeli.

Zapisz skrypt pod nazwą T9cw3.html

Spróbuj osiągnąć zbliżony efekt:

Tabliczka mnożenia

Podaj liczbę początkową:

Podaj liczbę końcową:

Utwórz

a = 4, b = 7

	4	**5**	**6**	**7**
4	16	20	24	28
5	20	25	30	35
6	24	30	36	42
7	28	35	42	49

Tabliczka mnożenia

Podaj liczbę początkową:

Podaj liczbę końcową:

Utwórz

Początek nie może byc większy od końca!!