Tworzenie Aplikacji Internetowych
Laboratorium 4

DOM (*ang.* Document Object Model)

DOM - obiektowy model dokumentu – sposób reprezentacji złożonych dokumentów XML i HTML w postaci modelu obiektowego. W modelu DOM HTML każdy element jest węzłem i może posiadać węzły potomne oraz rodzica. Struktura DOM HTML odpowiada strukturze elementów zagnieżdżonych w znacznikach HTML.

W JavaScript można odwołać się bezpośrednio do węzła:
- **window** – zawierającego informacje o oknie wyświetlającym daną stronę HTML
- **document** – zawierającego informacje o dokumencie HTML w danym oknie

Wybrane właściwości, metody i zdarzenia węzła **window**:
- **.history** – dostęp do historii związanej z przejściem do strony
- **.location** – zwraca bieżący adres strony internetowej
- **.innerWidth** – zwraca szerokość obszaru roboczego
- **.innerHeight** – zwraca wysokość obszaru roboczego
- **.pageXOffset / .scrollX** – liczba pikseli przesuniętych scrolem (poziom)
- **.pageYOffset / .scrollY** – liczba pikseli przesuniętych scrolem (pion)
- **.back**() – przekierowuje na poprzednią stronę
- .**close**() – zamyka bieżące okno
- .**print**() – włącza drukowanie strony
- .**open**(url, nazwa [, parametry]) – otwiera nowe okno
- **.onclose** – zdarzenie zamknięcie okna
- **.onunload** – zdarzenie wywołane przed zamknięciem okna
- .**onblur** – zdarzenie utracenia fokusu okna
- .**onfocus** – zdarzenie uzyskania fokusu w oknie
- .**onclick** – zdarzenie kliknięcia w dowolne miejsce na oknie
- .**onresize** – zdarzenie zmiany rozmiaru wyświetlanej przestrzeni w oknie

Wybrane właściwości, metody i zdarzenia węzła **document**:
- **.body** – bezpośredni dostęp węzła body
- **.head** – bezpośredni dostęp węzła head
- **.cookie** – zwraca przypisane ciasteczka
- **.activeElement** – zwaraca aktualnie aktywny element
- **.childElementCount** – zwraca liczbę potomków
- **.children** – zwraca potomków (kolekcja) // nie działa w IE
- **.childNodes** – zwraca potomków łącznie z węzłami prostymi (tekst, komentarze)
- **.forms** – zwraca wszystkie obiekty formularzy
- **.links** – zwraca wszystkie odnośniki ze strony
- **.scripts** – zwraca wszystkie skrypty ze strony
- **.styleSheets** – zwraca wszystkie style
- **.createElement("typ")** – tworzy nowy węzeł dom o typie podanym przez parameter
- **.removeElement(węzeł)** – usuwa dany węzeł **(także z innych węzłów)**
- **.appendChild(węzeł)** – dodaje nowy węzeł do dokumentu **(i innych węzłów)**
- **.prepend(węzeł)** – dodaje nowy węzeł na początku dokumentu **(j.w.)**
- **.adoptNode(węzeł)** – dodaje nowy węzeł z zewnętrznej strony
- **.getAnimations()** – zwraca aktywne animacje
- **.getElementById("id")** – zwraca element o danym id
- **.getElementByClassName("klasa")** – zwraca kolekcję elementów o przypisanej klasie
- **.getElementByTagName("tag")** – zwraca kolekcję znaczników danego typu
- **.querySelector("selektor")** – zwraca pierwszy element kolekcji zgodny z selektorem CSS
- **.querySelectorAll("selektor")** – zwraca elementy kolekcji zgodne z selektorem CSS

Poruszanie się po kolekcjach węzłów

Pobierając kolekcję węzłów można się po niej poruszać jak po standardowej tablicy (odwołując się do elementów za pomocą indeksów i do liczby elementów za pomocą .length) lub iteracyjnie za pomocą przekazywania kolejnych elementów do zmiennej:

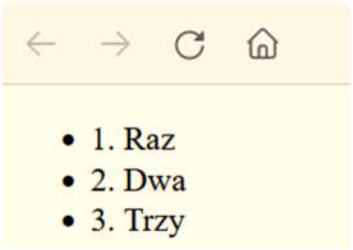
```
<script>
	// przykład 1 - pobranie węzłów wewnątrz body
	let wezly = document.body.childNodes; // pobranie węzłów
	for (let i = 0; i < wezly.length; i++) { // przejście pętlą
		console.log(i, wezly[i].tagName); // wypisanie znaczników
	}

	// przykład 2 - pobranie węzłów o określonej klasie
	let wklasy = document.getElementsByClassName("pola");
	for (let wezel of wklasy) { // przejście pętlą
		console.log(wezel.tagName); // wypisanie znaczników
	}
</script>
```

Wybrane właściwości, metody i zdarzenia węzłów odpowiadającym **znacznikom HTML:**

- **.innerHTML** – kod HTML wewnątrz węzła
- **.outerHTML** – kod HTML razem z kodem węzła
- **.innerText** – sam tekst wewnątrz znacznika
- **.textContent** – j.w. ale z uwzględnieniem elementów formatujących
- **.tagName** – nazwa znacznika
- **.setAttribute**("atrybut", "wartość") – ustawia atrybut
- **.removeAttribute**("atrybut") – usuwa atrybut
- **.getAttribute**("nazwa") – pobiera wartość atrybutu
- **.hasAttribute**("atrybut") – zwraca true jeżeli węzeł posiada atrybut
- **.parentElement** – rodzic elementu (null gdy brak)
- **.firstElementChild** – pierwszy potomek (null gdy brak)
- **.lastElementChild** – ostatni potomek (null gdy brak)
- **.nextElementSibling** – następny sąsiedni element (null gdy brak)
- **.previousElementSibling** – poprzedni sąsiedni element (null gdy brak)
- **.children** – kolekcja potomków
- **.classList.add("styl")** – dodanie stylu
- **.classList.remove("styl")** – usunięcie stylu
- **.classList.toggle("styl")** – dodanie stylu (gdy nie ma) inaczej usunięcie
- **.classList.contains("styl")** – sprawdzenie czy styl jest dodany
- **.style.właściwość** = "wartość" – nadanie wartości stylu
- **.clientHeight** – wysokość węzła (bez border i padding)
- **.clientWidth** – długość węzła (bez border i padding)
- **.attributes** – zwraca kolekcję atrybutów przypisanych do węzła
- **.id** – zwraca identyfikator węzła
- **.name** – zwraca nazwę węzła
- **.tabIndex** – pobiera lub określa pozycję przechodzenia tabulatorem

```
<ul class="lista"><li>Raz</li><li>Dwa</li><li>Trzy</li></ul>
<script>
	// przykład 3 - pobranie węzłów li wewnątrz listy o klasie lista
	let wezly = document.querySelectorAll(".lista li");
	for (let i = 0; i < wezly.length; i++) { // przejście pętlą
		wezly[i].prepend((i + 1) + ". "); // dodanie na początku węzła jego numeru
	}
</script>
```

Powyższy przykład wyświetli (przykład może nie zadziałać w starszych wersjach przeglądarki IE):



Dynamiczne tworzenie, modyfikowanie i usuwanie elementów.

Do każdego węzła HTML (pobranego dowolną metodą) można:

- Dodać węzeł: wezel.**appendChild**(nowy);
- Usunąć węzeł: wezel.**removeChild**(wybrany);

Nowe węzły można tworzyć metodą document.**createElement**("znacznik");
Poniższy przykład pokazuje utworzenie węzła i umieszczenie go w innym węźle:

```
<!DOCTYPE html>
<html>
    <head>
        <style>
            .wazny { font-weight: bold; }
        </style>
    </head>
    <body>
        <div id="tutaj"></div>
        <script>
            // utworzenie nowego węzła
            let nowywezel = document.createElement("span");
            nowywezel.innerHTML = "Tekst węzła"; // zawartość nowego węzła
            nowywezel.classList.add("wazny"); // przypisanie stylu CSS
            nowywezel.style.backgroundColor = "silver"; // nadanie koloru tła
            // pobranie węzła z body, w którym umieścimy węzeł
            let gdziedac = document.getElementById("tutaj");
            // umieszczenie nowego węzła wewnątrz pobranego
            gdziedac.appendChild(nowywezel);
        </script>
    </body>
</html>
```

Dynamiczne tworzenie elementów pozwala m.in. tworzyć elementy w pętlach, tworzyć elementy z zagnieżdżonymi innymi elementami wewnątrz (również utworzonymi dynamicznie), wyszukiwać i przenosić węzły, modyfikować wybrane węzły zgodnie z oczekiwaną funkcjonalnością itp.

```
<ul id="lista"></ul> <!-- przykład – dynamiczne dodanie elementów z tablicy do listy HTML -->
<script>
       let gdzied = document.getElementById("lista");
       let imiona = ["Azor", "Bingo", "Chip", "Dexter", "Elmo"];
       for (let i = 0; i < imiona.length; i++) {
              let nowapozycja = document.createElement("li");
              nowapozycja.innerHTML = imiona[i];
              gdzied.appendChild(nowapozycja);
       }
</script>
```

Zadanie 1

Napisać skrypt, który wyszuka wszystkie hiperłącza wewnątrz znacznika div o identyfikatorze id="tresc" i umieści je wewnątrz znacznika div o identyfikatorze id="hiperlacza". Hiperłącza pobrać stosując let linki = document.querySelectorAll("#tresc a"). Skrypt napisać dla poniższej strony:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <style>
        #hiperlacza { background: silver; border: 1px solid black; display: inline-block; }
        </style>
    </head>
    <body>
        <div id="tresc">Witaj na <a href="start.html">Mojej stronie</a>, polecam odwiedzić
takie serwisy jak: <ul><li><a href="http://google.com">Google</a></li><li><a
href="https://pl.wikipedia.org/wiki/Wiki">Wikipedia</a></li><li><a
href="https://stackoverflow.com/">StackOverflow</a></li></ul>Więcej informacji o mnie można
znaleźć <a href="omnie.html">tutaj</a>.</div>
        <div>Wszystkie linki na stronie:<br><div id="hiperlacza"></div></div>
        <script>
            // miejsce na skrypt

        </script>
    </body>
</html>
```

Sprawdzić czy po dodaniu hiperłączy do poszukiwanego znacznika zostały one usunięte z bazowych pozycji na stronie. Następnie rozwinąć skrypt tak, aby:

- dodał wewnątrz div o identyfikatorze id="hiperlacza" **kopie** znalezionych hiperłączy (węzłów) - tworzenie kopii węzła: let kopia = wezel.cloneNode(true); // wezel zastąpić znalezionymi węzłami
- dodał dynamicznie styl znalezionym znacznikom zmieniający ich kolor tła (w dowolny sposób)
- rozdzielił wstawiane hiperłącza dynamicznie utworzonym elementem br - po wstawieniu każdego hiperłącza należy utworzyć znacznik br: let br = document.createElement("br")
- zmienił treść wstawianych hiperłączy umieszczając w nich także adres hiperłączy (adres odnośników dostępny pod właściwością .href znalezionych węzłów).

Zdarzenia

Wszystkie zdarzenia można przypisywać do danych węzłów dynamicznie w następujący sposób:

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      function klikniecie(event) {
        console.log("Kliknięcie", event.clientX, event.clientY);
      }
      window.onclick = klikniecie; // sposób 1 - zdarzenie onclick dla window
      window.onkeyup = function(event) { // sposób 2 - zdarzenie onkeyup dla window
        console.log("Wciśnięcie", event.key, event.keyCode);
      }
    </script>
  </head>
<body></body>
</html>
```

Zdarzenia można również przypisywać dynamicznie utworzonym lub wyszukanym węzłom:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <style>
            body > div { min-height: 200px; background: rgb(150, 150, 150); }
            body > div div { float: left; width: 200px; height: 40px; }
            body > div div { background: rgb(200, 200, 200); border: 1px solid black; }
        </style>
    </head>
    <body>
        <div id="main">Kliknij mnie<br></div>
        <script>
            let pierwszyDiv = document.getElementById("main"); // odnalezienie div o id="main"
            pierwszyDiv.onclick = function() { // dodanie zdarzenia onclick - po kliknięciu
                let nowyDiv = document.createElement("div"); // tworzymy nowy div
                nowyDiv.innerHTML = "kliknij mnie"; // dodajemy zawartość nowego diva
                pierwszyDiv.append(nowyDiv); // umieszczamy go wewnątrz znalezionego
                nowyDiv.onclick = function() { // dla nowego również dodajemy zdarzenie
                    let r = Math.floor(Math.random() * 255); // losujemy składowe koloru
                    let g = Math.floor(Math.random() * 255);
                    let b = Math.floor(Math.random() * 255);
                    nowyDiv.style.backgroundColor = "rgb("+ r + "," + g + "," + b + ")";
                    pierwszyDiv.style.backgroundColor = "rgb(" + b + "," + r + "," + g + ")";
                }
            }
        </script>
    </body>
</html>
```

Zadanie 2

Celem tego zadania jest utworzenie podstawowej bazy gry w kości. W celu wykonania zadania należy:

- Dodać na stronie przycisk (lub hiperłącze) wywołujące funkcję JS tworzącą pięć nowych znaczników div. Nowe znaczniki powinny być umieszczone wewnątrz div o id="kosci". Przycisk (lub hiperłącze) może być umieszczony na stronie statycznie lub dynamicznie.
- Utworzone znaczniki powinny zawierać wewnątrz (.innerHTML) liczby losowe z zakresu od 1 do 6. Liczby te będą reprezentować wartości wyrzuconych kości. Do wygenerowania takich liczb wykorzystać metodę Math.random() – zwraca ona liczbę zmiennoprzecinkową w zakresie wartości od 0 do 1. Wylosowanie liczby całkowitej od 1 do 6: let losowa = 1 + Math.floor(Math.random() * 6);
- Za pomocą styli CSS (statycznie lub dynamicznie) nadać szerokość i wysokość tworzonym znacznikom (np. 100x100 px), margines 10px oraz nadać właściwość float: left (przypisywanie styli dynamicznieto np. znacznik.style.float = "left") – przy width, height i margin należy oprócz liczb podać "px".
- Po kliknięciu w przycisk tworzący pięć nowych znaczników, poprzednio utworzone znaczniki należy usunąć. Aby to zrobić można wymazać całą zawartość znacznika div o id="kosci" przypisując mu pusty atrybut .innerHTML (.innerHTML = "").
- Tworzonym dynamicznie znacznikom dodać obsługę zdarzenia onclick. Wywoływana funkcja powinna usuwać kliknięty znacznik ze strony HTML (do usunięcia znacznika wybrać metodę .removeChild(znacznik) i wywołać ją ze znacznika rodzica – tj. znacznika div o id="kosci"). Taki mechanizm pozwoli zasymulować wybranie kości do wymiany.
- Zapamiętać i wyświetlać (w dowolnym miejscu na stronie – np. wewnątrz znacznika span o id="info") ile użytkownik wybrał kości do wymiany. Dodać drugi przycisk lub hiperłącze generujący tylko tyle kości ile zostało wybranych do wymiany.
- **(opcjonalnie)** Samodzielnie rozszerzyć funkcjonalność tworzonej gry

Zadanie 3 (opcjonalne)

Dla poniższego skryptu (gra typu **memory** – należy szukać par podobnych pól odsłaniając maksymalnie dwa) dodać funkcjonalność:

- zliczającą liczbę wykonanych odsłonięć pól
- wyświetlający komunikat gratulacje po zakończeniu gry

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <style>
            @keyframes obrot {
                from { transform: rotate(0deg); }
                to { transform: rotate(360deg); }
            }
            .pole { position: fixed; width: 110px; height: 110px; margin: 10px; }
            .pole { text-align: center; font-size: 100px; padding: 8px; cursor: pointer; }
            .anim { animation: obrot 1s forwards; }
        </style>
    </head>
    <body>
        <div id="plansza"></div>
        <script>
            const liczbaKolorow = 8; // liczba kolorów na planszy
            const kolory = []; // zapamiętane kolory
            const polaMemory = []; // zapamiętane indeksy kolorów
            const bazowy = "rgb(50, 50, 50)"; // kolor bazowy
            let trybSprawdzenia = 0; // tryb sprawdzenia
            let zapamietanyIndeks = 0; // zapamiętany indeks koloru
            let zapamietanyA = null; // zapamiętany element a
            let zapamietanyB = null; // zapamiętany element b
            for (let i = 0; i < liczbaKolorow; i++) { // przypisanie pol
                polaMemory[i * 2] = i;
                polaMemory[i * 2 + 1] = i;
                let r = 70 + Math.floor(Math.random() * 185);
                let g = 70 + Math.floor(Math.random() * 185);
                let b = 70 + Math.floor(Math.random() * 185);
                kolory[i] = "rgb(" + r + "," + g + "," + b + ")";
            }
            for (let i = 0; i < 200; i++) { // losowe wymieszanie
                let indeks1 = Math.floor(Math.random() * polaMemory.length);
                let indeks2 = Math.floor(Math.random() * polaMemory.length);
                let tmp = polaMemory[indeks1];
                polaMemory[indeks1] = polaMemory[indeks2];
                polaMemory[indeks2] = tmp;
            }
            let px = 20, py = 20, pspace = 20, psize = 120, pgrid = 4;
            for (let i = 0; i < liczbaKolorow * 2; i++) { // utworzenie pól
                const pole = document.createElement("div"); // nowe pole
                pole.classList.add("pole");
                pole.style.backgroundColor = bazowy;
                pole.style.left = px + "px";
                pole.style.top = py + "px";
                px += psize + pspace;
                if ((i % pgrid) == pgrid - 1) { px = pspace; py += psize + pspace; }
                pole.onclick = function() { // obsługa kliknięcia w pole
                    if (polaMemory[i] == -1) return; // gdy element jest już odkryty
```

```
                    switch(trybSprawdzenia) {
                        case 0: { // odwracamy i zapamiętujemy pierwszy kolor
                            if (zapamietanyB != null) { // gdy się nie udało zakrywamy pola
                                zapamietanyA.innerHTML = zapamietanyB.innerHTML = "";
                                zapamietanyA.style.backgroundColor = bazowy;
                                zapamietanyB.style.backgroundColor = bazowy;
                            }
                            zapamietanyA = pole; // zapamiętujemy kliknięte pole
                            zapamietanyIndeks = i; // oraz jego indeks z tablicy
                            pole.style.backgroundColor = kolory[polaMemory[i]];
                            pole.innerHTML = polaMemory[i]; // wpisujemy indeks
                            trybSprawdzenia = 1; // będziemy sprawdzać drugie pole
                        } break;
                        case 1: { // sprawdzamy drugi kolor
                            if (zapamietanyA == pole) return;
                            pole.style.backgroundColor = kolory[polaMemory[i]];
                            pole.innerHTML = polaMemory[i]; // wpisujemy indeks
                            if (polaMemory[zapamietanyIndeks] == polaMemory[i]) {
                                polaMemory[i] = polaMemory[zapamietanyIndeks] = -1;
                                zapamietanyA.classList.add("anim");
                                pole.classList.add("anim");
                                zapamietanyA = zapamietanyB = null; // zostawiamy odkryte pola
                            } else { // gdy inny kolor
                                zapamietanyB = pole; // zapamiętujemy drugie pole
                            }
                            trybSprawdzenia = 0;
                        } break;
                    }
                }
                document.getElementById("plansza").appendChild(pole);
            }
        </script>
    </body>
</html>
```