**Lekcja 14**

## Temat: Model DOM1- wyszukiwanie elementów na stronie, modyfikacja stylów

Każda strona HTML składa się z elementów. Na samej górze jest środowisko - czyli okno przeglądarki – **window**, które zawiera w sobie wszystkie obiekty, funkcje i właściwości. W tym środowisku znajduje się **document** (czyli nasza strona). Na stronie znajduje się duża ilość różnych obiektów i elementów.
Do odzwierciedlenia ułożenia elementów JS korzysta z DOM czyli Document Object Model. Jest to model, interfejs, który za pomocą metod i właściwości umożliwia działanie na dokumencie (czyli elementach strony).

**Model obiektowy dokumentu** (ang. Document Object Model, DOM) stanowi API dla dokumentów HTML i XML. Odpowiada za dwie rzeczy: zapewnia reprezentację struktury dokumentu oraz określa, w jaki sposób odnosić się do tej struktury z poziomu skryptu. DOM przedstawia stronę jako ustrukturyzowaną grupę węzłów.

Podczas ładowania strony internetowej przeglądarka tworzy obiektowy model dokumentu w postaci hierarchii (drzewa) w którym każdy znacznik HTML, atrybut znacznika, tekst znajdujący się pomiędzy znacznikami jest reprezentowany jako obiekt w hierarchii DOM. Obiekty nazywane są **węzłami** (ang. **nodes**) drzewa dokumentu.

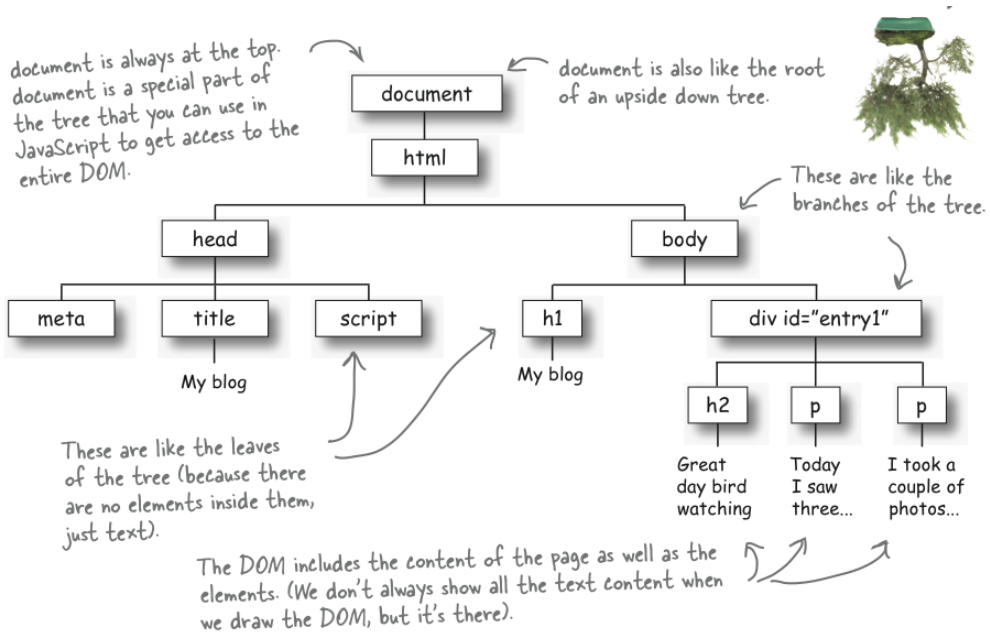
Dzięki obiektowemu modelowi dokumentu HTML za pomocą JavaScript możemy tworzyć dynamiczne strony internetowe, a w szczególności za pomocą JavaScript możemy:

- zmieniać znaczniki HTML i ich atrybuty
- zmieniać style CSS
- tworzyć oraz usuwać znaczniki HTML i ich atrybuty
- reagować na wszystkie zdarzenia

**Przykład 1 hierarchii DOM HTML:**

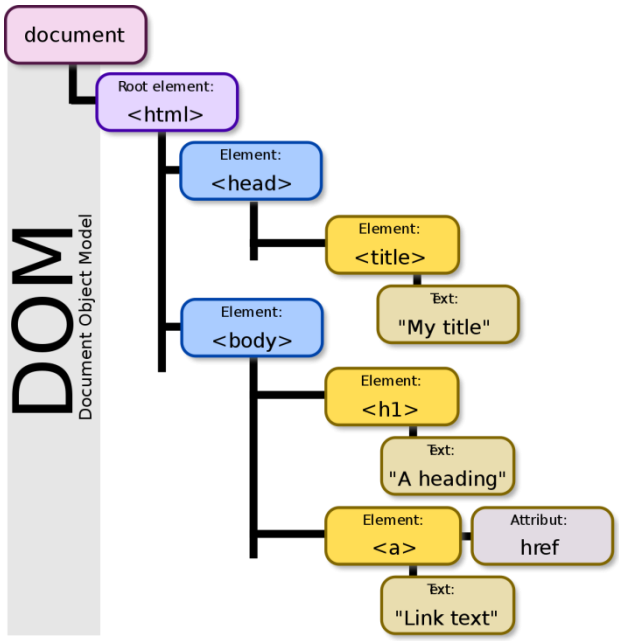
Poniżej znajduje się przykład tego, jak prosty plik HTML jest reprezentowany przez jego DOM.

```
<!doctype html>
<html lang="en">
<head>
  <title>My blog</title>
  <meta charset="utf-8">
  <script src="blog.js"></script>
</head>
<body>
  <h1>My blog</h1>
  <div id="entry1">
    <h2>Great day bird watching</h2>
    <p>
      Today I saw three ducks!
      I named them
      Huey, Louie, and Dewey.
    </p>
    <p>
      I took a couple of photos ...
    </p>
  </div>
</body>
</html>
```




Źródło: http://cs.wellesley.edu/~cs110/reading/DOM-JQ.html

**Przykład 2 hierarchii DOM HTML:**




Źródło: https://en.wikipedia.org/wiki/Document_Object_Model

**Zależności między węzłami:**

- Wszystkie węzły są ze sobą powiązane
- Każdy węzeł (oprócz document) ma rodzica
- Większość węzłów ma co najmniej jedno dziecko
- Węzły są rodzeństwem, jeśli mają tego samego rodzica
- Potomkami węzła są jego dzieci, dzieci tych dzieci, itd.
- Poprzednikami węzła są jego rodzice, rodzice tych rodziców, itd.

W celu modyfikacji obiektu znajdującego się na stronie internetowej musimy go najpierw wyszukać (zwykle go zapamiętujemy w zmiennej w celu późniejszego wykorzystania).

## Metody wyszukiwania elementów strony internetowej:

1. Starsze rozwiązania**:**
   - **document.getElementById("identyfikator")** – wyszukanie elementu o podanym w nawiasach identyfikatorze (id=”identyfikator”); Uwaga: nie może być na stronie dwóch elementów o tym samym identyfikatorze.
   - **document.getElementsByTagName("znacznik")** – wyszukanie wszystkich obiektów odpowiadających znacznikowi podanemu w nawiasach; ponieważ może być ich wiele zwracana jest tablica obiektów
   - **document.getElementsByClassName("klasa")** – wyszukanie wszystkich obiektów posiadających atrybut class podany w nawiasach (class=”klasa”) w postaci tablicy obiektów
2. Nowsze rozwiązania:
   - **querySelector(selector) -** przyjmuje argument, którym jest selektor CSS elementu, jaki chcemy znaleźć. Zwraca **pierwszy znaleziony element** (nawet jeśli istnieją inne elementy), który mógł zostać namierzony przez selektor
   - **querySelectorAll(selector) -** wszystkie dopasowania

**Ad1.**

Metoda **getElementById** zwraca pojedynczy element.

Metody **getElementsByClassName i getElementsByTagName** zwracają kolekcję HTMLCollecion, która zachowuje się jak tablica. Ta kolekcja jest "żywa", co oznacza, że jest automatycznie aktualizowana, jeśli dodatkowe elementy z tym znacznikiem lub klasą zostaną dodane do dokumentu.

Aby uzyskać dostęp do elementów otrzymanych tablic posługujemy się pętlą lub odwołujemy do konkretnego elementu otrzymanej tablicy.

**Ad2.**

Dają potężne możliwości dotarcia bezpośrednio do potrzebnego elementu. Mają jednak także swoje ograniczenia, co powoduje, że wszystkie wcześniejsze sposoby dotarcia do elementów strony nadal są wykorzystywane. Mówiliśmy, że funkcje getElementsByTagName i getElementsByClassName tworzą żywe kolekcje, co oznacza, że np. dodanie kolejnego elementu z klasą wykorzystaną przez funkcję powoduje aktualizację wyników jej działania. **querySelectorAll** nie tworzy takiej żywej kolekcji, więc każde dodanie nowego elementu z klasą zastosowaną w tej funkcji wymaga pewnych korekt.

**Pobieranie elementu za pomocą querySelector() i querySelectorAll()**

Funkcja **querySelector** działa w następujący sposób:

```
const element = document.querySelector("selektor_CSS");
```

Funkcja querySelector przyjmuje argument, którym jest selektor CSS elementu, jaki chcemy znaleźć. Zwraca ona **pierwszy znaleziony element** (nawet jeśli istnieją inne elementy), który mógł zostać namierzony przez selektor. Argumentami mogą być bardzo złożone selektory: np.

- **#id** - identyfikator
- **p.class1** – akapit z klasą
- **img.class2** – obrazek z klasą
- **div > p** – akapit- dziecko, którego rodzicem jest div
- **img[src='zdjecie1.png']** - obraz, którego atrybut src jest ustawiony na zdjecie1.png
- **ul li:first-of-type .btn** - pierwszy element klasy .btn w pierwszym li listy ul

Ponieważ metoda querySelector() zwraca tylko pierwszy element pasujący do podanych selektorów, aby otrzymać wszystkie dopasowania, należy użyć metody **querySelectorAll ()** .

Funkcje querySelector i querySelectorAll są niezwykle przydatne w złożonych dokumentach, w których kierowanie na dany element często nie jest proste. Opierając się na dobrze przyjętej składni selektora CSS, możemy rzucać dowolnie małą lub szeroką sieć na elementy, które chcemy odnaleźć. Jeśli chcemy otrzymać dostęp do wszystkich obrazków, możemy po prostu powiedzieć **querySelectorAll("img")** . Jeśli chcemy otrzymać dostęp tylko do elementu img zawartego w div , możemy powiedzieć **querySelector ("div + img").**

## Zmiana stylu elementu za pomocą JavaScript

Mamy dwa sposoby na zmianę stylu elementu za pomocą JavaScript. Jednym ze sposobów jest **ustawienie właściwości CSS bezpośrednio na elemencie**. Innym sposobem jest **dodanie lub usunięcie wartości klasy z elementu**, co może spowodować, że pewne reguły stylu zostaną zastosowane lub zignorowane.

### Bezpośrednie ustawianie stylu

Przy pracy ze stylami CSS korzystamy m. in. z obiektu `style` zawartego w każdym węźle drzewa DOM odzwierciedlającym znacznik HTML. Obiekt ten umożliwia określenie właściwości CSS i ustawienie jej wartości.

Jeżeli wskazanie do danego elementu witryny znajduje się w zmiennej `obj`, dostęp do obiektu style otrzymamy, pisząc:

```
obj.style;
```

Obiekt ten jest kolekcją atrybutów CSS przypisanych danemu elementowi strony.

W sposobie tym odwołujemy się więc do właściwości CSS. Jeżeli właściwość składa się z jednego słowa, to zapisujemy ją tak jak w css. Jeżeli właściwość składa się z kilku słów oddzielonych myślnikiem, wtedy dla takiej właściwości musimy zastosować zapis **camelCase**.

A zatem atrybut w formacie CSS, definiowany w kodzie HTML jako:

```
nazwa-atrybutu
```

zmieni się w następującą postać:

```
nazwaAtrybutu
```

**Przykładowo**, atrybut:

```
font-weight
```

jako właściwość obiektu `style` będzie miał postać:

```
fontWeight
```

```
list1.style.bacgroundColor=”#49b433”;
```

Możemy tutaj zastosować też zapis:

```
list1.style['background-color']=”#49b433”;
```

**UWAGA:** niektóre słowa w JavaScript są zastrzeżone i nie można ich używać bezpośrednio. Przykładem właściwości CSS, która należy do tej kategorii specjalnej, jest float . W CSS jest to właściwość layoutu. W języku JavaScript oznacza coś innego. Aby użyć właściwości, której nazwa jest całkowicie zarezerwowana, należy poprzedzić właściwość znakami: **css,** w której zmienna **`float`** zmienia się na **`cssFloat`** .

### Dodawanie i usuwanie klas za pomocą JavaScript

Drugie podejście polega na dodawaniu i usuwaniu wartości klas, które z kolei zmieniają stosowane reguły stylu.

To podejście bazuje na tym, że zasadniczo ustawianie wyglądu powinno się odbywać za pomocą CSS - a dokładnie klas w CSS, a JS powinien tylko odpowiednio manipulować tymi klasami - dodawać je, odejmować itp. Dzięki temu jeżeli w przyszłości zajdzie konieczność np. zmiany danego przycisku, zmienisz tylko odpowiednie klasy w CSS, a JS zostawisz w spokoju.

Aby zarządzać klasami CSS danego elementu używamy właściwości **`classList`**, która udostępnia kilka metod:

- **`add("nazwa-klasy")`** - dodawanie klasy. Jeśli te klasy już istnieją w atrybucie elementu, to są one ignorowane.
- **`remove("nazwa-klasy")`** - usuwanie klasy. **Uwaga:** Usunięcie klasy, która nie istnieje, NIE powoduje błędu.
- **`toggle("nazwa-klasy")`** - przełączanie (jak nie ma to dodaje, jak jest to usuwa) klasy
- **`contain("nazwa-klasy")`** - sprawdza czy element ma taką klasę

**Przykład 1**. Ten sam efekt uzyskany różnymi narzędziami:

a)

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="utf-8">
    <title>DOM, dynamiczna zmiana html - T14p1a</title>
    <style>
    body{
    background-color: #002255;
    color: #fff;
    font-size: 20px;
    }
    .square{
    background-color: #ccc;
    width:100px;
    height:100px;
    margin:5px;
    }
    </style>
</head>

<body>
    <div id="container">
        <div class="square">
            kwadrat1
        </div>
        <div class="square">
            kwadrat2
        </div>
        <div class="square">
            kwadrat3
        </div>
        <div class="square">
            kwadrat4
        </div>
    </div>
    <button onclick="changeLayout()">Zmień układ</button>
    <button onclick="changeSquare()">Zmień kwadraty</button>

    <script>
    function changeLayout(){
        let layout = document.getElementById('container');
        layout.style.backgroundColor='#1F6553';
        layout.style.display='flex';
    }
    function changeSquare(){
        let square1 = document.getElementsByClassName('square');
        square1[0].style.backgroundColor='red';
        for (let i=1;i<square1.length;i++){
        let squares1=square1[i].style;
        squares1.backgroundColor='#48D4B0';
        squares1.width="200px";
        squares1.height="200px";
        }
    }
    </script>
</body>
</html>
```

b)

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="utf-8">
    <title>DOM, dynamiczna zmiana html - T14p1b</title>
    <style>
    body{
    background-color: #002255;
    color: #fff;
    font-size: 20px;
    }
    .square{
    background-color: #ccc;
    width:100px;
    height:100px;
    margin:5px;
    }
    .flex1{
    display: flex;
    background-color:#1F6553;
    }
    .square2{
    background-color: #f00;
    }
    .square3{
    background-color: #48D4B0;
    width: 200px;
    height: 200px;
    }
    </style>
</head>
```

```
<body>
    <div class="container">
        <div class="square">
            kwadrat1
        </div>
        <div class="square">
            kwadrat2
        </div>
        <div class="square">
            kwadrat3
        </div>
        <div class="square">
            kwadrat4
        </div>
    </div>
    <button onclick="changeLayout()">Zmień układ</button>
    <button onclick="changeSquare()">Zmień kwadraty</button>
```

```
    <script>
    function changeLayout(){
        let uklad = document.querySelector('.container');
        uklad.classList.add('flex1');
    }
    function changeSquare(){
        let kw = document.querySelectorAll('.square');
        kw[0].classList.add('square2');
        for (let i=1;i<kw.length;i++){
        let kwa=kw[i].classList.add('square3');
        }
    }
    </script>
</body>
</html>
```

**Przykład 2**. Wykorzystanie zaawansowanych selektorów podczas wyszukiwania elementów drzewa DOM:

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="utf-8">
    <title>DOM, dynamiczna zmiana html - T14p2</title>
    <style>
    body{
    background-color: #002255;
    color: #fff;
    font-size: 18px;
    }
    </style>
    <script src="skryptT14p2.js"></script>
</head>
```

```
<body>
    <h2 class="class1">Nagłówek z klasą ="class1".</h2>
    <h3 class="class2">Nagłówek z klasą ="class2".</h3>
    <p class="class1">Akapit z klasą ="class1".</p>
    <p class="class1">Drugi akapit z klasą ="class1".</p>
    <p id="id1">Akapit bez klasy z identyfikatorem</p>
    <ul>
        <li>punkt pierwszy</li>
        <li>punkt drugi</li>
        <li class="class2">punkt trzeci</li>
        <li class="class2">punkt czwarty</li>
    </ul>
    <ol>
        <li>punkt pierwszy</li>
        <li>punkt drugi</li>
        <li class="class2">punkt trzeci</li>
        <li class="class2">punkt czwarty</li>
    </ol>
    <p><button onclick="myFunction()">Zmień kolor tła pierwszego akapitu  z klasą="class1"</button></p>
    <p><button onclick="myFunction2()">Zmień kolor tła i tekst akapitu z identyfikatorem</button></p>
    <p><button onclick="myFunction3()">Zmień formatowanie pierwszego punktu listy wypunktowanej</button></p>
    <p><button onclick="myFunction4()">Zmień formatowanie  punktów listy numerowanej w klasie class2</button></p>
    <p><button onclick="myFunction5()">Zmień formatowanie  punktów  w klasie class2</button></p>
</body>
</html>
```

skryptT14p2.js

```
function myFunction() {
    document.querySelector("p.class1").style.backgroundColor =
    "red";
}
function myFunction2() {
    document.querySelector("#id1").style.backgroundColor = "green";
    document.querySelector("#id1").innerHTML="<b>Świetnie!!!</b>";
}
function myFunction3() {
    document.querySelector("ul li:first-of-type").style.
    backgroundColor = "#2d7589";
}
function myFunction4() {
    let list1=document.querySelectorAll("ol li.class2 ")
    for(let i = 0; i< list1.length; ++i){
        list1[i].style.backgroundColor = "#49b433";
    }
}
function myFunction5() {
    let list2=document.querySelectorAll("li.class2 ")
    for(let i = 0; i< list2.length; ++i){
        list2[i].style.backgroundColor = "#932ABD";
    }
}
```

**Przykład 3**. Wykorzystanie przełączania klas

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="utf-8">
    <title>DOM, dynamiczna zmiana html - T14p3</title>
    <style>
        .box{
        width: 50px;
        height: 50px;
        border: 1px #000 solid;
        background: #fff;
        display: inline-block;
        }
        .box--marked{
            background: #f00;
        }
    </style>
</head>
<body>
    <div class="box"></div>
    <div class="box"></div>
    <div class="box"></div>
    <div class="box"></div>
    <div class="box"></div>
```

```
    <script>
        const boxes = document.querySelectorAll('.box');
        //iterujemy po wszystkich boksach
        for(let i = 0; i< boxes.length; ++i)
        {
            boxes[i].onclick = function(e){
                this.classList.toggle('box--marked');
            };
        }
        /*Każdy element ma klasę .box i po kliku dodajemy
        albo usuwamy klasę .box--marked*/
    </script>
</body>
</html>
```