Java Script

Cz.2

DOM

Hierarchia dokumentu

Każda strona HTML składa się z elementów - to już wiemy.

Na samej górze jest kontynent - czyli okno przeglądarki - window, które zawiera w sobie wszystkie obiekty, funkcje i właściwości. W tym oknie znajduje się nasze małe państewko - document (czyli nasza strona).

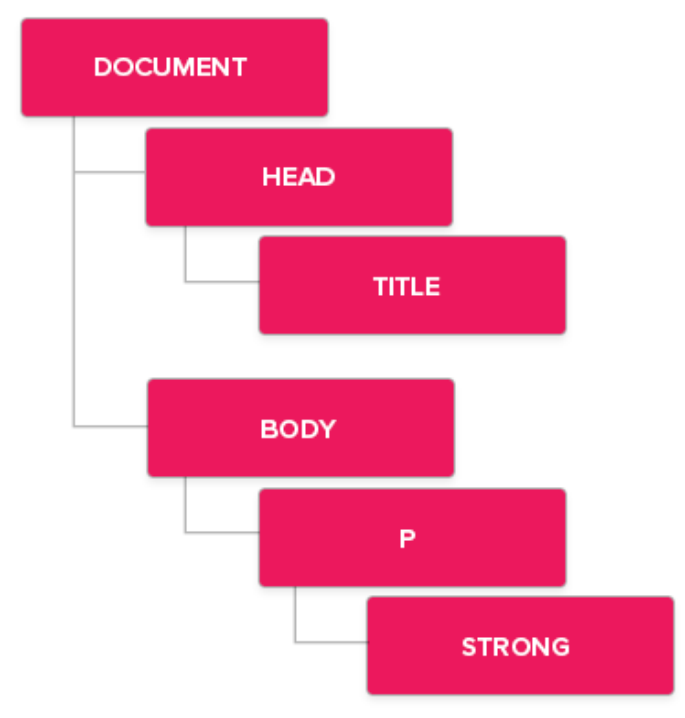
W tym państewku żyje od groma i tyć tyć różnych obiektów i ...elementów (zupełnie jak w życiu!).

Do odzwierciedlenia ułożenia elementów JS korzysta z DOM. DOM czyli Document Object Model.

Przykład:

Najlepiej uczyć się na przykładach, dlatego zacznijmy od bardzo prostej strony:

```
<!doctype html>
<html><head>
    <title>To jest tytuł strony</title>
</head>
<body>
    <p>
        Ten napis zawiera <strong>pogrubiony tekst</strong>
    </p>
</body>
</html>
```



Odwoływanie się do obiektów

Gdy DOM jeszcze nie istniał, odwoływanie się do obiektów wymagało korzystania ze specjalnych kolekcji. Większość obiektów na stronie pogrupowana jest w kolekcje które są swego rodzaju tablicami. I tak dla przykładu mamy kolekcję forms, która zawiera wszystkie formularze na naszej stronie, kolekcję images, która zawiera wszystkie znaczniki IMG na naszej stronie, kolekcję links zawierającą linki itp.

Korzystanie z takich kolekcji było całkiem miłe (np. żeby pobrać 1 formę na stronie korzystaliśmy z instrukcji forms[0]), jednak ograniczało się tylko to elementów, dla których stworzono kolekcje.

W dzisiejszych czasach takich ograniczeń nie mamy i możemy działać na każdym elemencie na stronie.

Zanim zaczniemy – słów kilka o Zdarzeniach…

Zdarzenia to czynności, które użytkownik wykonuje podczas odwiedzania naszej strony. Przykładowymi zdarzeniami mogą być np. przesunięcie kursora na obrazek (mouseover), kliknięcie jakiegoś linka (click), wysłanie formularza (submit), naciśnięcie klawisza (keypress) itp.

Javascript udostępnia kilkanaście typów **zdarzeń**:

Zdarzenie:	Opis:
blur	odpalane, gdy obiekt przestał być aktywny (np. input)
change	odpalane, gdy obiekt zmienił swoją zawartość (np. pole tekstowe)
click	odpalane, gdy obiekt został kliknięty (np. input)
dblClick	odpalane, gdy podwójnie klikniemy na obiekt (np. input)
focus	odpalane, gdy obiekt stał się wybrany (np. pole tekstowe)
keyDown	odpalane, gdy został naciśnięty klawisz na klawiaturze
input	podobne do powyższego, ale odpalane synchronicznie w czasie trzymania klawisza (np. przytrzymanie klawisza A w polu tekstowym)
keyUp	odpalane gdy puścimy klawisz na klawiaturze
load	odpalane, gdy obiekt został załadowany (np. cała strona)
mouseOver	odpalane, gdy kursor znalazł się na obiekcie

Javascript udostępnia kilkanaście typów zdarzeń (c.d.):

Zdarzenie:	Opis:
mouseOut	odpalane, gdy kursor opuścił obiekt
resize	odpalane, rozmiar okna przeglądarki jest zmieniany
select	odpalane, gdy zawartość obiektu została zaznaczona
submit	odpalane, gdy formularz został wysłany
unload	odpalane, gdy użytkownik opuszcza dana stronę

Powyższa lista zawiera tylko najczęściej używane zdarzenia. Takich zdarzeń jest o wiele, wiele więcej, chociaż prawdę mówiąc większości z nich i tak się nie używa.

Rejestrowanie zdarzenia bezpośrednio w kodzie HTML

Aby zdarzenie było dostępne dla danego obiektu, musimy je dla niego zarejestrować. Istnieje kilka metod na rejestrację zdarzenia dla obiektu.

Metoda inline (jako atrybut elementu) przypisywania zdarzeń polega na określeniu zdarzenia wewnątrz znacznika:

```
<a href="jakasStrona.html" onclick="alert('Kliknąłeś')"> kliknij </a>
```

UWAGA: Ten model rejestrowania zdarzeń jest zły, bo miesza JS z kodem HTML.

Zdarzenie jako właściwość obiektu

Pierwsza, starsza metoda przypisywania zdarzeń polega na ustawieniu zdarzenia jako właściwość danego obiektu:

```
function showText() {
    console.log('Kliknięto przycisk na stronie');
}

var element = document.getElementById('przycisk');
element.onclick = showText;
```

Zauważyłeś, że przy podpinaniu funkcji do zdarzeń pomijamy nawiasy? Robimy tak dlatego, ponieważ nie chcemy odpalać funkcji, a tylko ją podpiąć pod dane zdarzenie.

Aby usunąć wcześniej przypisane zdarzenie, wystarczy pod daną właściwość podstawić null:

```
obiekt.onclick = null;
```

Poza rejestrowaniem zdarzeń dla elementów, możemy też takie zdarzenia wywołać - np. onclick:

```
element.click()
```

Nowy model rejestracji zdarzeń

Problem z tradycyjnym modelem polega na tym, że do jednego elementu możemy podpiąć tylko jedną funkcję dla jednego rodzaju zdarzenia. Rejestrując nową funkcję do tego samego typu zdarzenia nadpisujemy starą.

Możemy to oczywiście obejść (wspólna funkcja odpalająca kilka funkcji), jednak jest to mało logiczne i przysparza kłopotów z późniejszym odrejestrowaniem takich zdarzeń.

Problemów takich nie mamy korzystając z "nowego" modelu rejestrowania zdarzeń opierającego się na metodzie addEventListener().

Więcej o tej wersji można poczytać w sieci…

Słowo kluczowe this

Javascript udostępnia słowo kluczowe this, które wskazuje na obiekt, który wywołał daną funkcję. Słowo to jest bardzo użyteczne w przypadku rejestrowania zdarzeń dla obiektów.

Przykładowo chcielibyśmy zmienić kolor czcionki dowolnego obiektu, do którego przypiszemy zdarzenie onmouseover:

```
function changeColor() {
this.style.color = '#CC0000';
}
```

DOM – wracamy do tego jak odwołać się do elementów strony HTML…

Pobieranie elementu za pomocą getElementById

Metoda getElementById(id) pobiera element o danym ID.

```
<input type="button" id="guzik" value="ok" />
```

```
var g = document.getElementById('guzik');
console.log(g.value); //korzystamy z okienka dialogowego by wypisać value pobranego guzika
```

getElementById Przykład:

```
<html>
    <head>
        <title>Przykład getElementById</title>
            <script type="text/javascript">
                function changeColor(newColor)
                    { var elem = document.getElementById("para1");
                        elem.style.color = newColor;   }
            </script>
    </head>
    <body>
    <p id="para1">Jakiś tekst</p>
    <button onclick="changeColor('blue');">niebieski</button>
    <button onclick="changeColor('red');">czerwony</button>
    </body>
</html>
```

getElementById **jest absolutną podstawą DOM. Jedną z najważniejszych zasad w programowaniu DOM jest unikalne identyfikowanie elementów, dzięki czemu można je przechwycić i manipulować nimi.**

Jeżeli nie ma elementu o podanym ID, funkcja zwraca NULL. Zauważ też, że implementacja DOM musi wiedzieć, które atrybuty mają typ ID. Atrybuty o nazwie "ID" nie mają typu ID, o ile nie zostanie to tak zdefiniowane. Implementacje, które nie wiedzą, czy atrybuty mają typ ID czy nie, powinny zwracać NULL.

KONIEC