

Kurs Front-End **Developer**
Model DOM

DOCUMENT OBJECT MODEL (DOM) (I-***)

DOM jest to sposób reprezentacji złożonych dokumentów XML i HTML w postaci **modelu obiektowego**.

Dzięki niemu skrypty mają **dynamiczny dostęp** do dokumentu oraz mogą go aktualizować tzn. zmieniać style, treść i jego strukturę.

DOM jest **modelem hierarchicznym** i udostępnia zestaw obiektów odzwierciedlających dokument.

DOM jest to **standardowy interfejs** umożliwiający dostęp i manipulację obiektami.

DOCUMENT OBJECT MODEL (DOM) (I-***)

Standard W3C DOM dzieli się na trzy części:

- CORE DOM – model dla wszystkich typów dokumentów;
- XML DOM – model dla dokumentów XML;
- **HTML DOM** – model dla dokumentów HTML.

HTML DOM definiuje:

- elementy HTML jako **obiekty**;
- **właściwości** wszystkich elementów HTML;
- **metody** dostępu do wszystkich elementów HTML;
- **zdarzenia** dla wszystkich elementów HTML.

HTML DOM definiuje jak uzyskać, zmieniać, dodawać lub usuwać elementy HTML.

DOCUMENT OBJECT MODEL (DOM) (I-****)

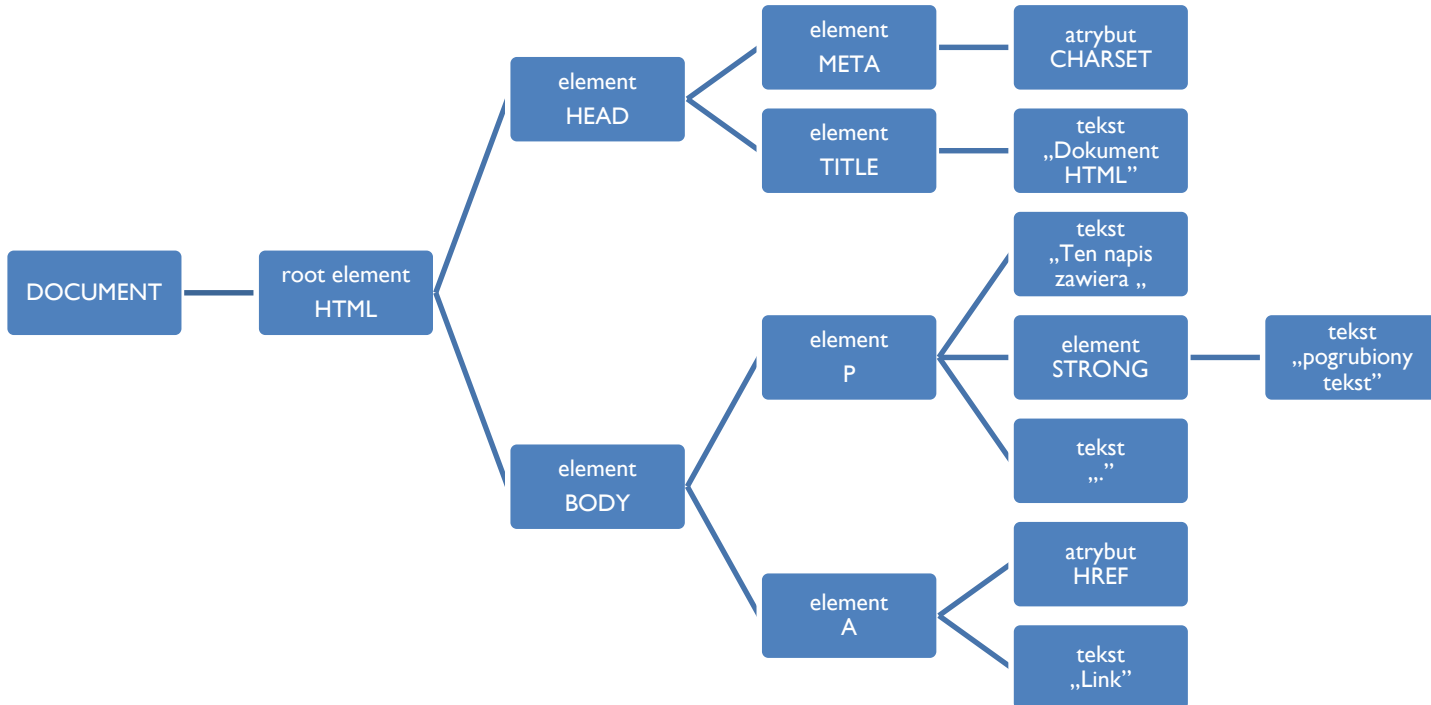
Gdy strona jest ładowana, przeglądarka tworzy **model DOM**.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Dokument HTML</title>
  </head>
  <body>
    <p>
      Ten napis zawiera
      <strong>pogrubiony tekst</strong>

    </p>
    <a href="#">Link</a>
  </body>
</html>
```

DOCUMENT OBJECT MODEL (DOM) (I-****)

Graficzna reprezentacja modelu DOM



Jest to struktura drzewiasta.

OBIEKTY DOCUMENT (I-***)

Głównym obiektem w modelu DOM jest **obiekt** *document*.

Obiekt *document* reprezentuje całą stronę internetową.

Aby uzyskać dostęp do dowolnego elementu na stronie HTML, zawsze należy zacząć od dostępu do obiektu *document*.

Obiekt *document* zawiera właściwości i metody, które pozwalają uzyskać dostęp do wszystkich obiektów, z poziomu JavaScriptu.

document.documentElement to główny element dokumentu (korzeń), czyli *<html>*.

document.body to obiekt reprezentujący *<body>* dokumentu, natomiast

document.head to obiekt reprezentujący *<head>* dokumentu.

Obiekt *document* jest częścią obiektu *window*, a więc można uzyskać do niego dostęp poprzez ***window.document***.

ZNAJDOWANIE ELEMENTÓW (2-***)

Elementy HTML możemy znaleźć na kilka sposobów:

1. za pomocą **identyfikatora** elementu - metoda *getElementById*;
2. za pomocą **nazwy klasy** elementu - metoda *getElementsByClassName*;
3. za pomocą **nazwy tagu** elementu - metoda *getElementsByTagName*;
4. za pomocą **selektora CSS** - metody *querySelectorAll* i *querySelector*.

ZNAJDOWANIE ELEMENTÓW (2-***)

```
<body>
  <section id="sectionFirst">
    <div id="parFirst">
      <p>Tekst w tym akapicie.</p>
      <a class="link" href="#">Akapit Link 1</a>
      <a class="link" href="#">Akapit Link 2</a>
    </div>
    <div id="parSecond" >
      <p>Tekst w tym akapicie.</p>
    </div>
    <a class="link" href="#">Link 1</a>
    <a class="link" href="#">Link 2</a>
    <a class="link" href="#">Link 3</a>
    <a class="link" href="#">Link 4</a>
  </section>
</body>
```


ZNAJDOWANIE ELEMENTÓW (2-***)

// pobierze element o id="parFirst"

```
let elementFirst = document.getElementById( "parFirst" );
```

// pobierze wszystkie element o class="link"

```
let elementsTable = document.getElementsByClassName( "link" );
```

// pobierze wszystkie elementy p

```
let allP = document.getElementsByTagName( "p" );
```

// pobierze wszystkie elementy o class="link"

```
let allLink = document.querySelectorAll( ".link" );
```

// pobierze pierwszy element o class="link"

```
let firstLink = document.querySelector( ".link" );
```

RELACJE MIĘDZY WĘZŁAMI (3-****)

Elementy na stronie tworzą **hierarchiczne drzewo – drzewo węzłów**.

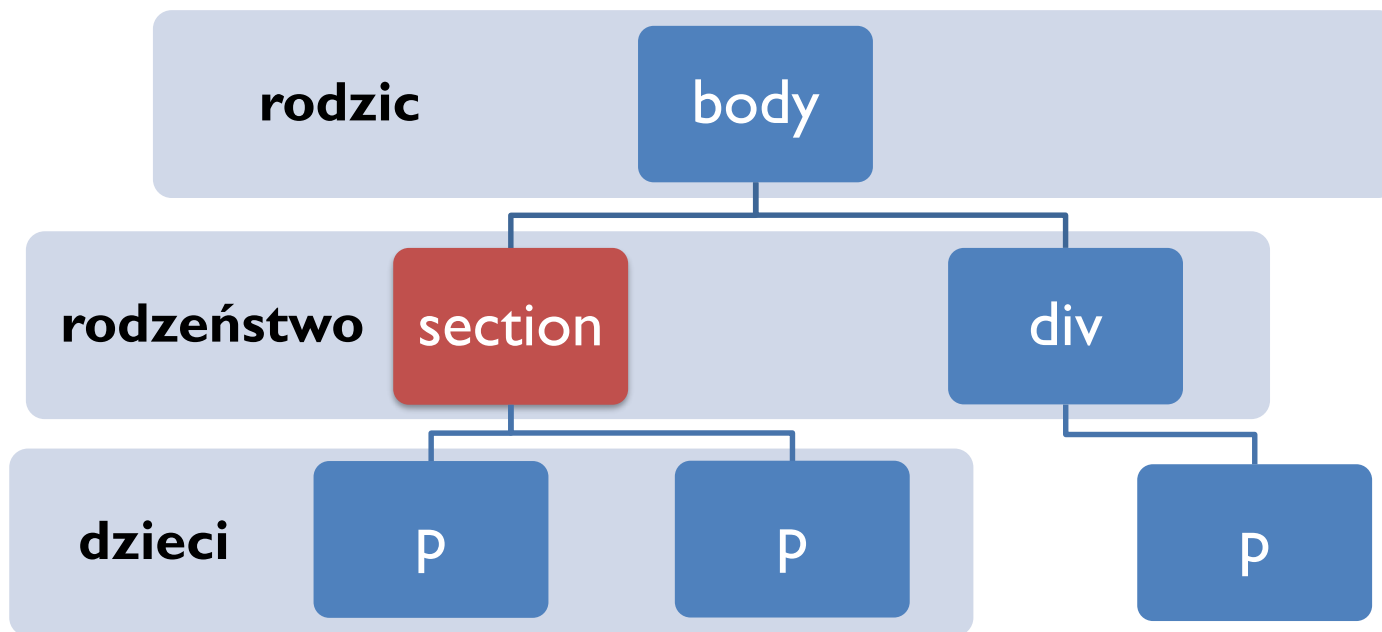
Każdy element na stronie jest tak zwanym węzłem (ang. **node**) czyli pojedynczy węzeł. Należą do nich: cały dokument, elementy, atrybuty, tekst oraz komentarze.

W relacjach między nodami możemy mówić o:

- byciu **rodzicem (parent)** w stosunku do innego węzła;
- byciu **rodzeństwem (sibling)** dla innego węzła;
- byciu **dzieckiem (child)** w stosunku do innego węzła.

RELACJE MIĘDZY WĘZŁAMI (3-****)

Relacja w stosunku do `<section>`



RELACJE MIĘDZY WĘZŁAMI (3-****)

W drzewie modelu DOM, górny węzeł nazywa się **root** lub **root node** (korzeń główny, węzeł główny).

Każdy węzeł ma dokładnie jednego **rodzica (parent node)**, z wyjątkiem głównego węzła (nie ma rodzica).

Węzły mogą posiadać **dzieci (child nodes)**.

Rodzeństwem (sibling node) są węzły na tym samym poziomie drzewa modelu DOM.

RELACJE MIĘDZY WĘZŁAMI (3-****)

Dzięki modelowi **DOM**, możemy uzyskać dostęp do każdego węzła w drzewie za pomocą JavaScript.

Właściwości te pomagają poruszać się po drzewie węzłów.

RELACJE MIĘDZY WĘZŁAMI (3-****)

parentElement lub *parentNode* – zwraca rodzica danego węzła

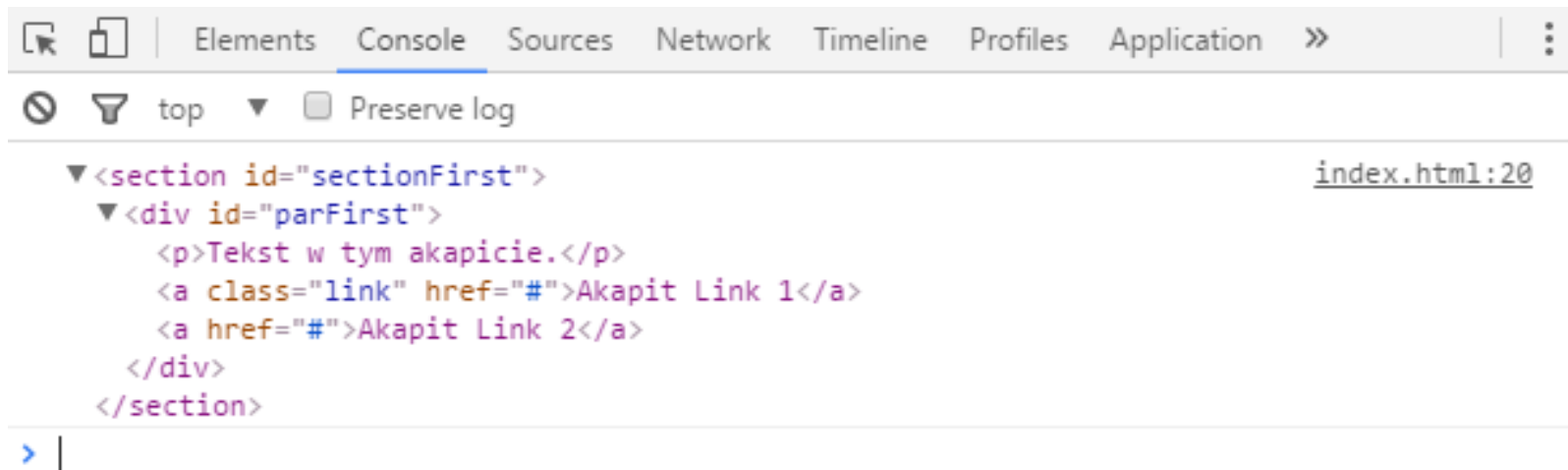
```
<body>
  <section id="sectionFirst">
    <div id="parFirst">
      <p>Tekst w tym akapicie.</p>
      <a class="link" href="#">Akapit Link 1</a>
      <a href="#">Akapit Link 2</a>
    </div>
  </section>
</body>
```

```
let elementParent = document.getElementById( "parFirst" ).parentNode;
console.log( elementParent );
```

RELACJE MIĘDZY WĘZŁAMI (3-****)

Consola Javascript i logi:

```
let elementParent = document.getElementById( "parFirst" ).parentNode;  
console.log( elementParent );
```



RELACJE MIĘDZY WĘZŁAMI (3-****)

childNodes – lista dzieci danego węzła - wszystkich

childNodes[nodenumbr] – dane dziecko danego węzła

children – lista dzieci danego węzła – tylko jeśli są elementami HTML

firstElementChild – pierwsze dziecko danego węzła będące elementem HTML

lastElementChild – ostatnie dziecko danego węzła będące elementem HTML

RELACJE MIĘDZY WĘZŁAMI (3-****)

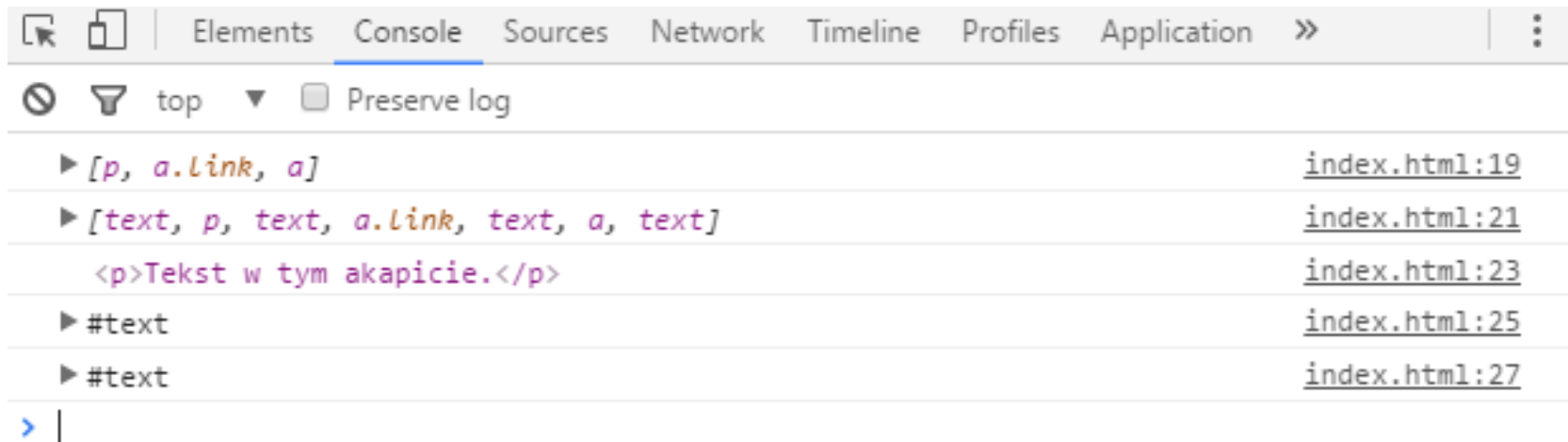
```
<body>
  <section id="sectionFirst">
    <div id="parFirst">
      <p>Tekst w tym akapicie.</p>
      <a class="link" href="#">Akapit Link 1</a>
      <a href="#">Akapit Link 2</a>
    </div>
  </section>
</body>
```

```
console.log( document.getElementById( "parFirst" ).children );
console.log( document.getElementById( "parFirst" ).childNodes );
console.log( document.getElementById( "parFirst" ).childNodes[1] );
console.log( document.getElementById( "parFirst" ).firstElementChild );
console.log( document.getElementById( "parFirst" ).lastElementChild );
```

RELACJE MIĘDZY WĘZŁAMI (3-***)

Console Javascript i logi:

```
console.log( document.getElementById( "parFirst" ).children );  
console.log( document.getElementById( "parFirst" ).childNodes );  
console.log( document.getElementById( "parFirst" ).childNodes[1] );  
console.log( document.getElementById( "parFirst" ).firstElementChild );  
console.log( document.getElementById( "parFirst" ).lastElementChild );
```



RELACJE MIĘDZY WĘZŁAMI (3-****)

previousElementSibling – poprzedni element danego węzła, który ma tego samego rodzica

nextElementSibling – następny element danego węzła, który ma tego samego rodzica

```
<body>
  <section id="sectionFirst">
    <div id="parFirst">
      <p>Tekst w tym akapicie.</p>
      <a id="link" href="#">Akapit Link 1</a>
      <a href="#">Akapit Link 2</a>
    </div>
  </section>
</body>
```

```
console.log( document.getElementById( "link" ).nextElementSibling );
console.log( document.getElementById( "link" ).previousElementSibling );
```

RELACJE MIĘDZY WĘZŁAMI (3-****)

Wymienione powyżej właściwości np. *childNodes*, *previousSibling*, *nextSibling* pobierają wszystkie węzły łącznie z komentarzami oraz tekstem. Aby zobaczyć jakiego typu jest „node” można użyć **właściwości** *nodeType*.

Własność ta zwraca typ danego węzła w postaci numeru:

- 1 – element HTML
- 2 – atrybut elementu
- 3 – tekst
- 8 – komentarz
- 9 – dokument
- 10 – dokument definicji

TWORZENIE WĘZŁÓW (4-****)

Do tworzenia nowych węzłów służą metody:

- `document.createElement(typ)` – tworzy element HTML
- `document.createTextNode(tekst)` – tworzy tekst
- `document.createAttribute(nazwa)` – tworzy atrybut

```
let btn = document.createElement( "button" );
```

```
let textBtn = document.createTextNode( "Click me" );
```

```
let classBtn = document.createAttribute( "class" );
```

DODAWANIE WĘZŁÓW (4-***)

Po stworzeniu węzłów musimy dodać je do DOM.

Możemy to zrobić za pomocą metod:

- *appendChild*(nowyWezel) – dodaje węzeł jako ostatnie dziecko danego węzła
- *insertBefore*(nowyWezel, istniejacyWezel) – dodaje węzeł przed innym dzieckiem danego węzła
- *replaceChild*(nowyWezel, istniejacyWezel) – zamień dziecko danego węzła na nowy węzeł
- *setAttributeNode*(nowyObiektAtrybut) – dodaje obiekt atrybutu do danego węzła
- *setAttribute*(nowyAtrybut, wartoscAtrybutu) – dodaje atrybut wraz z wartością tego atrybutu do danego węzła lub edytuje istniejący atrybut

DODAWANIE WĘZŁÓW (4-***)

```
let btn = document.createElement( "button" ); // tworzy element <button>  
let textBtn = document.createTextNode( "Click me" ); // tworzy tekst  
let classAtr = document.createAttribute( "class" ); // tworzy atrybut class
```

```
classAtr.value= "btn"; // ustawia wartość dla atrybutu class
```

```
document.body.appendChild( btn ); // dodaj element do elementu <body>  
btn.appendChild( textBtn ); // dodaj tekst do elementu <button>  
btn.setAttributeNode( classAtr ); // dodaj atrybut class do elementu <button>
```

```
btn.setAttribute( "href" , "#" ); // dodaj nowy atrybut wraz z wartością do elementu <button>
```

USUWANIE WĘZŁÓW (4-***)

Do usuwania elementów służą metody:

- *removeChild*(wezel) – usuwa dziecko danego węzła
- *removeAttribute*(nazwaAtrybutu) – usuwa atrybut danego węzła
- *removeAttributeNode*(atrybut) – usuwa atrybut danego węzła

```
btn.removeAttribute( "class" );  
document.body.removeChild( btn );
```


MANIPULACJA DOM - HTML I CSS (5-***)

innerHTML – służy do pobierania i ustawiania kodu HTML w danym elemencie

```
document.getElementById( "header" ).innerHTML = "Tekst do elementu";
```

innerText – pobiera i ustawia tekst znajdujący się w element HTML (bez zagnieżdżonych elementów HTML)

```
document.getElementById( "link" ).innerText = "Tekst do tagu";
```

outerHTML – pobiera i ustawia kod HTML wraz z elementem HTML

```
document.getElementById( "link" ).outerHTML;
```

MANIPULACJA DOM - HTML I CSS (5-***)

Możemy również pobrać i zmieniać/ustawić wartości atrybutów za pomocą następującej składni:

```
document.getElementById( id ).attribute = nowaWartosc;
```

np.

```
document.getElementById( "link" ).href = "https://akademia108.pl";
```

```
let idLinku = document.getElementById( "link" ).id;
```

MANIPULACJA DOM - HTML I CSS (5-***)

className – zwraca i ustawia/zmienia listę klas elementu

```
document.getElementById( "link" ).classList;  
document.getElementById( "link" ).className = "btn btn-default";
```

Możemy także dodawać oraz ustawiać/zmieniać style CSS dla danego elementu za pomocą następującej składni:

```
document.getElementById( id ).style.wlasnoscCSS = nowaWartosc;
```

np.

```
document.getElementById( "link" ).style.color = "red";
```

MANIPULACJA DOM - HTML I CSS (5-***)

Więcej właściwość CSS:

http://www.w3schools.com/jsref/dom_obj_style.asp

EVENTY (6-***)

BEGIN NAVIGATION

Eventy to czynności, które użytkownik wykonuje podczas odwiedzania strony.

Większość zdarzeń wywoływana jest przez użytkownika. Istnieją też zdarzenia, które nie są bezpośrednio spowodowane przez użytkownika.

Za pomocą JavaScriptu można kontrolować **eventy** i na nie reagować.

W języku JavaScript istnieje gotowy zestaw predefiniowanych zdarzeń.

Korzystając z eventów musimy mieć pewność, że wszystkie elementy na stronie są załadowane.

DOMContentLoaded (6-***)

Event **DOMContentLoaded**, gwarantuje, że skrypt zacznie swoje działanie wtedy, gdy całe drzewo DOM zostanie już wczytane, a element, na którym pracujemy już istnieje.

W praktyce wszystkie skrypty operujące na elementach DOM powinny korzystać z tego eventu.

Ogólna konstrukcja użycia tego eventu ma postać:

```
document.addEventListener( "DOMContentLoaded", function() {  
    // tutaj trafia skrypt operujący na elementach ze strony  
});
```

Callback (6-***)

callback - jest to tak zwana funkcja zwrotna. Jest ona uruchamiana po wystąpieniu jakiegoś zdarzenia (eventu).

Eventy są budowane na tej funkcji.

EVENTY MYSZY (6-****)

Obsługa eventów **myszki**:

click - zachodzi, gdy obiekt został kliknięty

dblclick - zachodzi, gdy podwójnie klikniemy na obiekt

mouseover - zachodzi, gdy kursor znalazł się na obiekcie

mouseout - zachodzi, gdy kursor opuścił obiekt

EVENTY KLAWIATURY (6-***)

Obsługa eventów **klawiatury**:

keydown - zachodzi, gdy klawisz na klawiaturze zostaje wciskany

keyup - zachodzi podczas zwalniania klawisza na klawiaturze

keypress - zachodzi, gdy klawisz klawiatury został wciśnięty

EVENTY DOTYKOWE (6-***)

Obsługa eventów **dotykowych**:

touchmove - zachodzi, gdy palec przemieszcza się po ekranie dotykowym

touchstart - zachodzi, gdy palec dotyka ekranu dotykowego

touchedend - zachodzi podczas zabrania palca z ekranu dotykowego

EVENTY FORMULARZY (6-***)

Obsługa eventów formularzy:

blur - zachodzi, gdy dane pole formularza przestaje być aktywne

change - zachodzi, gdy dane pole formularza ulega zmianie

focus - zachodzi podczas uaktywnienia danego pola formularza

submit - zachodzi podczas przesyłania formularza

INNE TYPY EVENTÓW (6-***)

Pełna lista typów eventów:

http://www.w3schools.com/jsref/dom_obj_event.asp

OBSŁUGA EVENTÓW (6-***)

Aby zdarzenie było dostępne dla danego obiektu, należy je dla niego zarejestrować.

Jest kilka sposobów obsługi zdarzeń:

- rejestrowanie zdarzenia bezpośrednio w kodzie HTML
- przypisywanie zdarzeń przy użyciu HTML DOM
- rejestrowanie zdarzeń za pomocą metody *addEventListener()*

OBSŁUGA EVENTÓW (6-***)

Rejestrowanie zdarzenia bezpośrednio w kodzie HTML

```
<a href="#" onclick="alert('click' )" > kliknij </a>
```

OBSŁUGA EVENTÓW (6-***)

Przypisywanie zdarzeń przy użyciu HTML DOM

```
function showText() {  
    console.log('click');  
}
```

```
document.getElementById( "link" ).onclick = showText;
```

OBSŁUGA EVENTÓW (6-***)

Rejestrowania zdarzeń opierającego się na metodzie
addEventListener()

```
function showText() {  
    console.log('click' );  
}
```

```
document.getElementById( "link" ).addEventListener( 'click', showText );
```


USUWANIE EVENTÓW Z ELEMENTU (6-***)

Można to zrobić za pomocą metody *removeEventListener()*

```
document.getElementById( "link" ).removeEventListener( 'click',  
showText );
```

Nie da się usunąć eventów, które zostały dodane za pomocą funkcji anonimowych! Gdyż drugi paramet metody wymaga podania nazwy funkcji.

ODCZYT WŁAŚCIWOŚCI EVENTU (6-***)

Javascript udostępnia nam **specjalne właściwości**, dzięki którym możemy bardziej dokładnie badać każde zarejestrowane zdarzenie.

Aby odczytać właściwości zdarzenia musimy posłużyć się **pseudo parametrem**, który będziemy przekazywać do deklarowanej funkcji (w naszych przykładach taki parametr nazwiemy **e**).

```
document.getElementById( "link" ).addEventListener( 'click', function(e){  
    console.log(e);  
} );
```

ODCZYT WŁAŚCIWOŚCI EVENTU (6-***)

Można więc sprawdzić typ elementu, który wywołał dane zdarzenie.

```
document.getElementById( "link" ).addEventListener( 'click', function(e){  
    console.log(e.type);  
} );
```

I wiele innych właściwości.

WSTRZYMANIE DOMYŚLNEJ AKCJI (6-***)

Większość elementów na stronie wykonuje domyślne akcje.

Linki przenoszą w jakieś miejsca, formularz się wysyła itp.

Po podpięciu zdarzeń pod obiekt będą one wywoływane na początku, jednak zaraz po nich wykonana zostanie domyślna czynność.

Aby zapobiec wykonaniu się domyślnej czynności można skorzystać z metody *preventDefault()*.

WSTRZYMANIE DOMYŚLNEJ AKCJI (6-***)

```
document.getElementById( "link" ).addEventListener( 'click', function(e){  
    e.preventDefault();  
    alert( 'Ten link nigdzie nie przeniesie!' );  
} );
```

ZATRZYMANIE NASŁUCHU INNYCH ZDARZEŃ (6-***)

Po odpaleniu zdarzenia, domyślnie przechodzi ono po obiektach od dołu hierarchii do góry - dążąc do dokumentu.

Aby przerwać tę wędrówkę oraz kolejne nasłuchy, można skorzystać z metody *stopPropagation()*.

ZATRZYMANIE NASŁUCHU INNYCH ZDARZEŃ (6-***)

```
<div id="exampleDiv">
  <a id="exampleLink" href="">Kliknij mnie</a>
</div>
```

```
document.getElementById( 'exampleDiv' ).addEventListener( 'click', function(e) {
    alert( 'Kliknięto div' );
} );
```

```
document.getElementById( 'exampleLink' ).addEventListener( 'click', function(e) {
```

```
    /*
```

Bez dwóch poniższych linii kliknięcie na link spowoduje wyświetlenie komunikatu "Kliknięto link", a następnie komunikatu "Kliknięto div". Dzięki dodaniu tych linii kodu wyświetli się tylko pierwszy komunikat.

```
    */
```

```
    e.preventDefault();
    e.stopPropagation();
    alert( 'Kliknięto link' );
```

```
});
```

WARSZTATY – USTAW TŁO

W HTML stwórz 2 paragrafy oraz przycisk z napisem “Ustaw tło paragrafów”.

Za pomocą `addEventListener` obsługuj zdarzenia kliknięcia w przycisk.

Po kliknięciu w przycisk “Ustaw tło paragrafów” zostanie wywołana ta funkcja. W funkcji `ustawTlo()` zmień style paragrafów, ustawiając kolor tła pierwszego paragrafu na czerwony i drugiego paragrafu na żółty.

WARSZTATY – POBIERZ IMIE

Stwórz formularz

```
<form id="formularz">
```

```
  Imię: <input type="text" name="fname" value="Krystian">
```

```
  <br>
```

```
  Nazwisko: <input type="text" name="lname" value="Dziopa">
```

```
  <br>
```

```
  <input type="submit" value="Submit">
```

```
</form>
```

Następnie podczas wysyłania formularza zablokuj jego domyślne działanie. Pobierz wartości obydwu pól formularza (czyli fname i lname) i wyświetl je w konsoli.

WARSZTATY – WALIDACJA FORMULARZA

Zwaliduj formularz znajdujący się pliku **index.html**.

Walidacja ma sprawdzać, czy pola **“imie i nazwisko”**, **“adres email”** nie są puste oraz sprawdza, czy pierwszy checkbox (zgoda-marketingowa-1) jest zaznaczony.

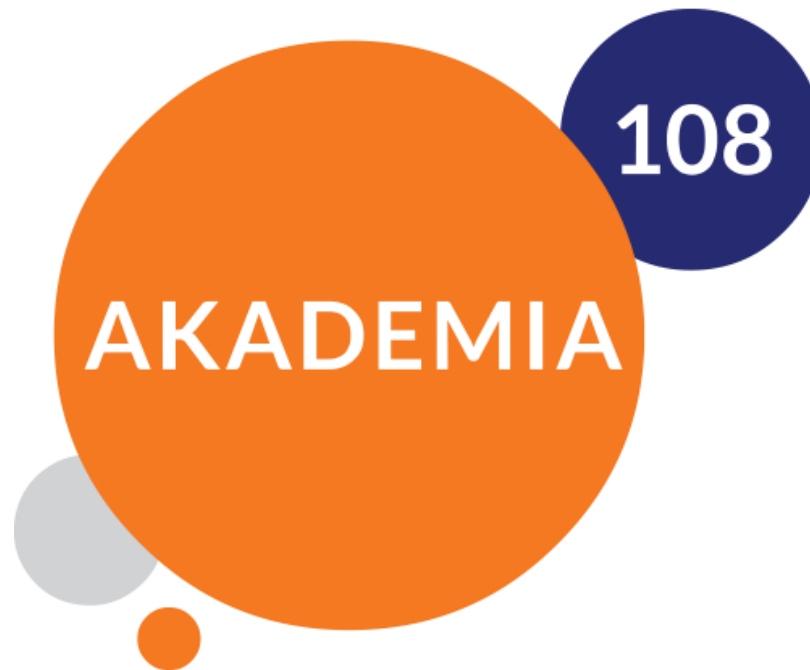
Jeżeli pola są puste lub checkbox jest niezaznaczony, to pod każdym niepoprawnie wypełnionym polem wyświetl komunikat informujący o błędzie.

Po zaznaczeniu checkboxa (wszystkie zgody), wszystkie pozostałe checkboxy mają zostać automatycznie zaznaczone.

WARSZTATY – KALKULATOR ODSETEK

Stworz kalkulator odsetek z lokaty, który liczy zyski z uwzględnieniem podatku belki.

Kalkulator liczy zyski wg. wzoru podanego w instrukcjach do tego zadania.



Akademia 108

<https://akademia108.pl>