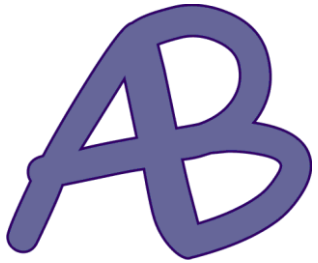
Aplikacje frontendowe



Wykład

DOM
Document Object Model

Czym jest DOM

DOM (Document Object Model)

- DOM to interfejs programistyczny dla dokumentów HTML i XML.
- Reprezentuje stronę jako strukturę drzewa, gdzie każdy węzeł jest częścią dokumentu (np. elementy, atrybuty, teksty).

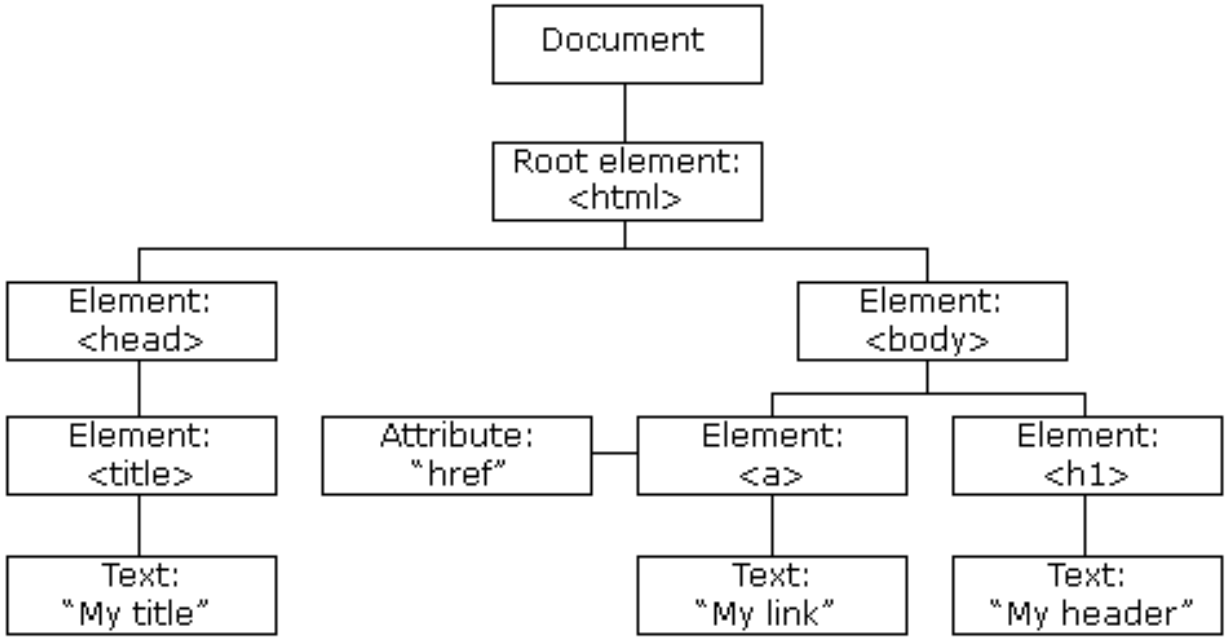
Przykład: Hello World staje się w DOM strukturą z węzłem div jako rodzicem i węzłem tekstowym jako dzieckiem.

Znaczenie DOM w web development:

- Pozwala na dynamiczne modyfikowanie struktury, stylu i zawartości strony webowej.
- Integracja z JavaScript umożliwia interaktywność strony.

Czym jest DOM

Na podstawie kodu HTML zapisanego w pliku budowany jest obiekt DOM.
Dopiero na podstawie tego obiektu wyświetlana jest strona.



DOM nie jest zależny od języka programowania

Czym jest DOM

Czy DOM to to samo co HTML?

```
<body>
    <h1>Tytuł strony</h1>
    <p>Lorem ipsum dolor sit amet,
consectetur adipiscing elit. in
laoreet ex odio scelerisque mi.
<span>Sed sit amet <em>egestas
</span> Proin fermentum sit amet nibh
eget facilisis. </em>Suspendisse
ultrices et lacus at placerat..</p>
    <ul>
        <li>Punkt pierwszy</li>
        <li>Punkt drugi
        <li>Punkt trzeci</li>
        <li>Punkt czwarty</li>
    </ul>
</body>
```

```
▼ <body>
    <h1>Tytuł strony</h1>
  ▼ <p>
      "Lorem ipsum dolor sit amet, consectetur adipiscing elit. in
      laoreet ex odio scelerisque mi. "
    ▶ <span> ⋯ </span>
      <em> Proin fermentum sit amet nibh eget facilisis. </em>
      "Suspendisse ultrices et lacus at placerat.."
    </p>
  ▼ <ul>
    ▼ <li>
        ::marker
        "Punkt pierwszy"
      </li>
    ▼ <li>
        ::marker
        "Punkt drugi "
      </li>
    ▶ <li> ⋯ </li>
    ▶ <li> ⋯ </li>
    </ul>
    <!-- Code injected by live-server -->
  ▶ <script> ⋯ </script>
</body>
```

Po lewej stronie widzimy kod z zaznaczonymi kolorem błędami. Po prawej elementy wyświetlone przez przeglądarkę. Jak widać błędy zostały naprawione. Dodane zostały też dodatkowe węzły np. pseudo-elementy ::marker.

Czym jest DOM

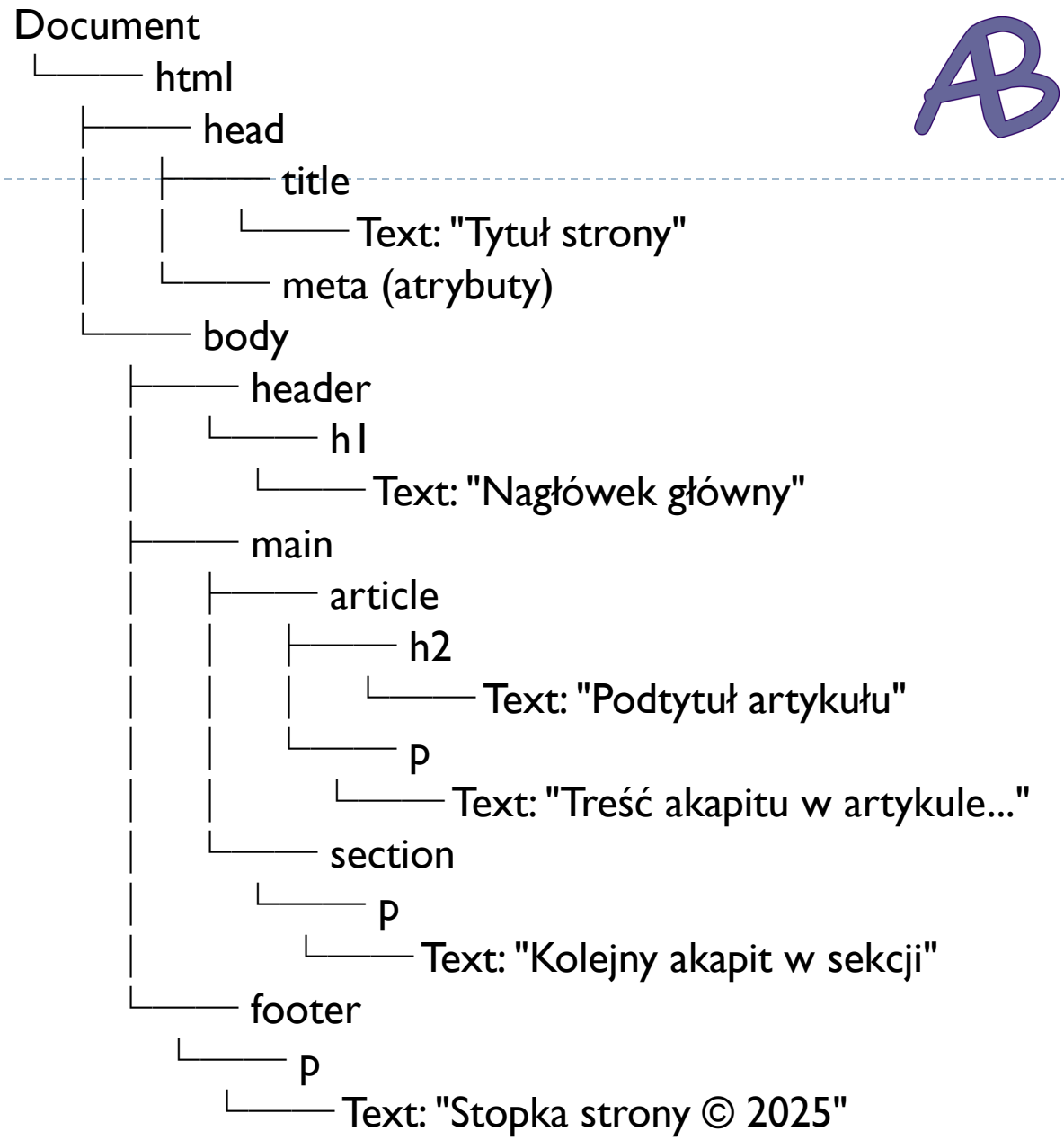
Rodzaje węzłów

Typ węzła	Stała w JS	Przykład w dokumencie	Możliwe metody
Element	Node.ELEMENT_NODE	, , <p>	.getAttribute(), .appendChild()
Text	Node.TEXT_NODE	Hello World (tekst)	.textContent, .data
Comment	Node.COMMENT_NODE	<!-- komentarz -->	(rzadko używane programowo)
Document	Node.DOCUMENT_NODE	cały dokument HTML/XML	.documentElement, .createElement()

Czym jest DOM

Drzewo dokumentu

- ✓ Każdy węzeł typu Element (np. <h1>) może mieć węzeł potomny typu Text, zawierający rzeczywistą treść.
- ✓ Węzły Text mają stałą nodeType === 3 i właściwość .textContent lub .data, którą można modyfikować lub odczytać.



Czym jest DOM

- ✓ Węzły dokumentu

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>DOM</title>
</head>
<body>
    <h1>Tytuł strony</h1>
    <p>Lorem ipsum dolor sit amet</p>
</body>
</html>
```

```
> document.body.childNodes
NodeList(5) [text, h1, text, p, text] i
  0: text
  1: h1
  2: text
  3: p
  4: text
  length: 5
  [[Prototype]]: NodeList
```

```
> document.childNodes
NodeList(2) [<!DOCTYPE html>, html] i
  0: <!DOCTYPE html>
  1: html
  length: 2
  [[Prototype]]: NodeList

childNodes: NodeList(3)
  0: head
  1: text
  2: body
  length: 3
  [[Prototype]]: NodeList
```

Węzły Vs elementy

Znajdowanie elementów HTML

childNodes vs children

Cecha	childNodes	children
Zwracany typ	**NodeList** (wszystkie typy węzłów: elementy, tekst, komentarze, itp.)	**HTMLCollection** (tylko węzły typu Element)
Zawartość	Węzły: Element, Text, Comment, DocumentType	wyłącznie węzły Element
Aktualność	snapshot (statyczny NodeList od momentu pobrania) *	live (HTMLCollection automatycznie się aktualizuje) *
Główne zastosowanie	pełny dostęp do struktury dokumentu, w tym białych znaków i komentarzy	operacje na elementach (np. manipulacje atrybutami, stylem)

* Do tego jeszcze wrócimy pod koniec prezentacji

Znajdowanie elementów HTML

childNodes vs children

```
<ul id="lista">
  <li>Item 1</li>
  <!-- komentarz -->
  <li>Item 2</li>
</ul>
<script>
  const lista = document.getElementById("lista");
  console.log(lista.childNodes);
  // NodeList: [Text("\n  "), li, Text("\n  "), Comment(" komentarz "),
Text("\n  "), li, Text("\n")]

  console.log(lista.children);
  // HTMLCollection: [li, li]
</script>
```

Znajdowanie elementów HTML

nextSibling vs nextElementSibling

Cecha	nextSibling	nextElementSibling
Zwracany typ	najbliższy sąsiedni **węzeł** (Node)	najbliższy sąsiedni **element** (Element)
Uwzględniane węzły	wszystkie: tekstowe, komentarze, itp.	tylko węzły elementów (ignoruje tekst i komentarze)
Główne zastosowanie	niskopoziomowa nawigacja po każdym węźle DOM	bezpieczne przeskakiwanie między elementami

Znajdowanie elementów HTML

nextSibling vs nextElementSibling

```
<div id="kontener">
  <span id="first">A</span>
  <!-- komentarz -->
  <span id="second">B</span>
</div>
<script>
  const first = document.getElementById("first");
  console.log(first.nextSibling);
  // Comment(" komentarz ")

  console.log(first.nextElementSibling);
  // <span id="second">B</span>
</script>
```

Poruszanie się po DOM-ie
Znajdowanie elementów HTML

Znajdowanie elementów HTML

Znajdowanie elementów HTML

Aby móc manipulować elementami HTML należy najpierw je odnaleźć. Można to zrobić na kilka sposobów:

1. Znajdowanie elementów HTML według identyfikatora
2. Znajdowanie elementów HTML według nazwy tagu
3. Znajdowanie elementów HTML według nazwy klasy
4. Znajdowanie elementów HTML za pomocą selektorów CSS

Znajdowanie elementów HTML

Najprostszym sposobem na znalezienie elementu HTML w modelu DOM jest użycie identyfikatora elementu.

```
const element = document.getElementById(„id_elementu");
```

- ✓ Jeśli element zostanie znaleziony, metoda zwróci element jako obiekt.
- ✓ Jeśli element nie zostanie znaleziony, element będzie zawierał .null
- ✓ W tym wypadku zakładamy, że w dokumencie istnieje tylko jeden element o takim id

Znajdowanie elementów HTML

Możliwe jest też wyszukanie wszystkich elementów należących do określonej klasy.

```
const collection =
document.getElementsByClassName("nazwa_klasy")
```

- ✓ Metoda zwraca obiekt typu HTMLCollection listę elementów HTML przypominającą tablicę.
- ✓ Dostęp do elementów w kolekcji można uzyskać za pomocą indeksu (zaczyna się od 0).
- ✓ Właściwość length zwraca liczbę elementów w kolekcji.

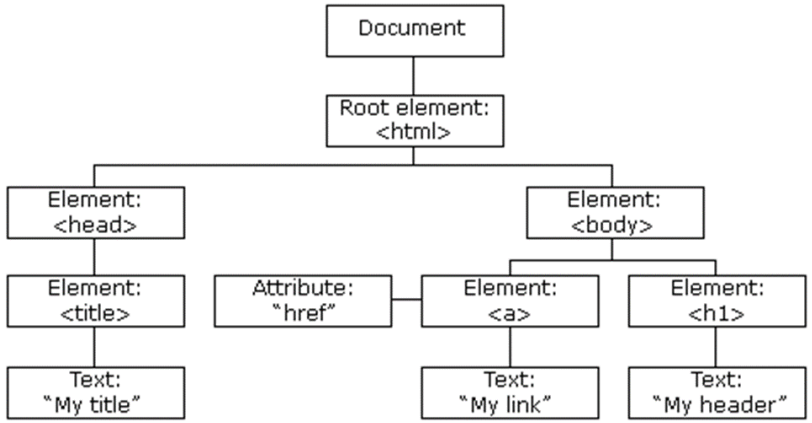
Znajdowanie elementów HTML

Przy wyszukiwaniu elementów pamiętać należy że przeszukujemy tylko wskazaną gałąź drzewa DOM.

Możemy więc przeszukiwać cały dokument, lub też rozpocząć przeszukiwanie od konkretnego miejsca.

```
const start = document.getElementsByTagName(„article");
const element = start[0].getElementsByTagName(„p");
```

Przykładowo wyszukujemy wszystkie akapity w pierwszym artykule na stronie



Znajdowanie elementów HTML

Możliwe jest wyszukanie wszystkich elementów o określonym tangu, np. wszystkich akapitów.

```
const collection = document.getElementsByTagName(„p");
```

- ✓ Metoda zwraca obiekt typu HTMLCollection listę elementów HTML przypominającą tablicę.
- ✓ Dostęp do elementów w kolekcji można uzyskać za pomocą indeksu (zaczyna się od 0).
- ✓ Właściwość length zwraca liczbę elementów w kolekcji.

Znajdowanie elementów HTML

```
const collection = document.getElementsByTagName(„p");
```

- ✓ Przykład: zmiana koloru fontu we wszystkich elementach listy na stronie.

```
const collection = document.getElementsByTagName(„li");
for (let i = 0; i < collection.length; i++) {
  collection[i].style.color = "red";
}
```

- ✓ Przykład: Zmiana koloru wszystkich elementów listy o konkretnym ID

```
const kontener = document.getElementById("id_listy");
const elementy = kontener.getElementsByTagName("li");
for (let i = 0; i < elementy.length; i++) {
  elementy[i].style.color = "red";
}
```

Znajdowanie elementów HTML

Najczęściej wykorzystywanym obecnie sposobem poruszanie się po obiekcie DOM jest wyszukiwanie elementów za pomocą identyfikatorów CSS

```
const myNode = document.querySelector("p");
```

- ✓ Metoda zwraca pierwszy napotkany element o pasującym selektorze
- ✓ Metoda zawsze zwraca pojedynczy element

Znajdowanie elementów HTML

Możemy także odnaleźć wszystkie elementy pasujące do wybranego selektora CSS

```
const myNodeList = document.querySelectorAll("p");
```

- ✓ Metoda zwraca obiekt typu NodeList (kolekcja węzłów).
- ✓ Dostęp do elementów w kolekcji można uzyskać za pomocą indeksu (zaczyna się od 0).
- ✓ Właściwość length zwraca liczbę elementów w kolekcji.
- ✓ NodeList jest prawie taki sam jak HTMLCollection.

Znajdowanie elementów HTML

`document.querySelectorAll();`

- ✓ Przykład: Pobieranie elementów na podstawie bardziej złożonych selektorów

```
// Selektor pobierający elementy 'p' wewnątrz elementów z klasą 'container'
var paragraphs = document.querySelectorAll(".container p");
```

- ✓ Przykład: Selekcja kombinacji selektorów:

```
// Pobieranie wszystkich elementów z klasą 'highlight' lub 'selected'
var highlightsAndSelected = document.querySelectorAll(
  ".highlight, .selected"
);
```

- ✓ Przykład: Selekcja na podstawie atrybutów::

```
// Pobieranie wszystkich elementów (pól dialogowych) z atrybutem type='number'
let dataTypeElements = document.querySelectorAll('[type="number"]');
//wypełnienie pierwszego z nich wartością "0"
dataTypeElements[0].value = "0";
```

Znajdowanie elementów HTML

W querySelectorAll możemy stosować pełnię składni CSS:

```
<nav class="nav">
  <ul>
    <li>
      <a href="/">Home</a>
    </li>
    <li>
      <a href="/about">About</a>
    </li>
  </ul>
</nav>;
```

- ✓ Przykład:

```
// Pobierz wszystkie linki wewnątrz <nav class="nav">
const links = document.querySelectorAll("nav.nav > ul > li > a");
links.forEach((link) => {
  console.log(`Link: ${link.textContent}, href: ${link.href}`);
});
```

Znajdowanie elementów HTML

Wyszukiwanie i obsługa błędów

Metody getElementById lub querySelector mogą zwrócić null:

- ✓ Przykład:

```
const submitBtn = document.getElementById("submit");
if (!submitBtn) {
  console.error("Brak elementu #submit w DOM – sprawdź czy ID się zgadza!");
} else {
  submitBtn.addEventListener("click", handleSubmit);
}
```

Wyszukanie w dokumencie przycisków i przypisanie do nich słuchacza reagującego na kliknięcie

„Live” vs „Static”

Live kolekcje

Kolekcja live to obiekt, który automatycznie odzwierciedla bieżący stan drzewa DOM. Jeżeli w drzewie pojawi się lub zniknie węzeł pasujący do kryteriów kolekcji, jej zawartość natychmiast się zmieni.

Metody zwracające „live” kolekcje:

- `document.getElementsByTagName(tagName) → HTMLCollection`
- `document.getElementsByClassName(className) → HTMLCollection`
- `document.getElementsByName(name) → NodeList (również live)`
- `parentNode.childNodes → NodeList (live, zawiera wszystkie węzły-dzieci)`

„Live” vs „Static”

Live kolekcje

Zalety:

- Automatyczna synchronizacja: nie musisz ponownie pobierać kolekcji po modyfikacji DOM.
- Przydatne, gdy chcesz stale monitorować zmiany (np. plugin, który dynamicznie dodaje/usuwa elementy).

Wady:

- Koszt utrzymania: każda operacja na DOM (dodanie/usunięcie węzła) wymaga przebudowy kolekcji może spowolnić aplikację przy dużej liczbie elementów.
- Nieoczekiwane zmiany podczas iteracji: jeśli w pętli modyfikujesz DOM, kolekcja zmieni się „w locie”, co może prowadzić do przeskoczeń lub dublowania elementów.

„Live” vs „Static”

„Static” kolekcje (NodeList)

Kolekcja static jest fotografię (snapshot) drzewa DOM w momencie wywołania. Po jakichkolwiek dalszych zmianach w DOM jej zawartość pozostaje niezmieniona.

Metody zwracające „static” kolekcje:

- `document.querySelectorAll(selector) → static NodeList`
- `element.querySelectorAll(...) → static NodeList`

Zalety:

- Przewidywalność: iteracja zawsze przebiega po tej samej, niezmiennej liście.
- Lepsza wydajność: brak ciągłego śledzenia zmian DOM.

Wady:

- Po modyfikacji drzewa musisz ponownie wywołać metodę, by otrzymać zaktualizowaną listę.

„Live” vs „Static”

„Live” vs „Static”

```
const liveColl   = document.getElementsByTagName('p');      // stale aktualizowane
const staticList = document.querySelectorAll('p');          // snapshot

// dodajemy nowy <p>
document.body.appendChild(document.createElement('p'));

console.log(liveColl.length);  // np. 6 (jeśli było 5)
console.log(staticList.length); // 5 (bez zmian)
```

Kiedy co wybrać?

- Gdy potrzebujesz dynamicznego śledzenia zmian: HTMLCollection.
- Gdy zamierzasz wykonać operację jednorazową: NodeList

Jak przekonwertować HTMLCollection i NodeList na tablice

Jak przekonwertować HTMLCollection i NodeList na tablice

```
// HTMLCollection (live) oraz NodeList (static)
const htmlColl  = document.getElementsByClassName('item');
const nodeList  = document.querySelectorAll('.item');


// Konwersja
const arrFromHTML  = Array.from(htmlColl);
const arrFromNode  = Array.from(nodeList);


// Teraz możesz używać map, filter, reduce itp.
const ids = arrFromNode.map(el => el.id);
```

Metoda wbudowana stworzy nową tablicę z iterowalnego obiektu lub obiektu posiadającego właściwość length

Jak przekonwertować HTMLCollection i NodeList na tablice

Jak przekonwertować HTMLCollection i NodeList na tablice

Operator rozproszenia (...)

```
const nodeList = document.querySelectorAll("p");
const arrSpread = [...nodeList];

arrSpread.forEach((p) => console.log(p.textContent));
```

Krótszy zapis, działa na iterowalnych strukturach (NodeList zawsze, HTMLCollection)

Literatura:

- Negrino Tom, Smith Dori, ***Po prostu JavaScriopt i Ajax, wydanie VI***, Helion, Gliwice 2007.
- Lis Marcin, JavaScript, Ćwiczenia praktyczne, wydanie II, Helion, Gliwice 2007.
- http://www.w3schools.com/JS/js_popup.asp
- Beata Pańczyk, wykłady opublikowane na stronie http://www.wykladowcy.wspa.pl/wykladowca/pliki/beatap