
PGS
SOFTWARE

PGS SOFTWARE

Akademia Developera – edycja FrontDev

Javascript - interakcja z przeglądarką

PAWEŁ LULA FRONT-END DEVELOPER

Praktyki

- Trwa rekrutacja na praktyki obowiązkowe (160h) oraz nadprogramowe (min 120h)
- Nie zwlekajcie z wyslaniem CV na praktyki obowiązkowe (termin na uczelni do 15 maja)
- **Pamiętajcie aby zamieszczać w CV klauzulę dotyczącą zgody na przetwarzanie danych osobowych**

Co było na poprzednim wykładzie?

- zmienne i funkcje
- wybrane właściwości języka
- zakres zmiennych, domknięcia, hoisting
- przepływ kontroli
- typy danych
- operatory
- this
- obiektowość i dziedziczenie prototypowe

O czym będzie dziś?

- Osadzanie skryptów na stronie
- Objekty window oraz document
- Dostęp oraz modyfikacja elementów
- Dodawanie oraz usuwanie elementów
- Zdarzenia
- Asynchoniczność

Osadzanie skryptów na stronie

Skrypty osadzone

```
<script>
    console.log('Only Javascript');
</script>
```

Skrypty zewnętrzne

```
<script type="text/javascript"
        src="myScript.js">
</script>
```

Gdzie lepiej umieszczać skrypty zewnętrzne?

- Na końcu znacznika 'body'
- Wewnątrz 'head', ale musimy użyć atrybutu defer lub async

```
<script type="text/javascript"
    src="myScript.js"
    defer>
</script>
```

Document Object Model (DOM)

Czym jest DOM?

DOM jest sposobem odczytu dokumentu, w którym cała struktura przetwarzanego pliku zapisywana jest w obiekcie (pamięci). Dzięki temu, możemy w szybki sposób uzyskać dostęp do dowolnego elementu dokumentu. Każdy element możemy odczytać, przetworzyć, usunąć, a nawet rozszerzyć o kolejne elementy podrzędne.

Dostęp do elementów DOM z poziomu JS

getElementById

```
<body>
    <div id="exampleId">id</div>
    <div class="exampleClass">div</div>
    <p class="exampleClass2">paragraph</p>
</body>

var id = document.getElementById('exampleId');
// <div id="exampleId">id</div>
```

getElementByClassName

```
<body>
    <div id="exampleId">id</div>
    <div class="exampleClass">div</div>
    <p class="exampleClass2">paragraph</p>
</body>

var elements = document.getElementsByClassName(
// [div.exampleClass]
```

getElementsByTagName

```
<body>
    <div id="exampleId">id</div>
    <div class="exampleClass">div</div>
    <p class="exampleClass2">paragraph</p>
</body>

var elements = document.getElementsByTagName('p
 // [p.exampleClass2]
```

querySelector

```
<body>
    <div id="exampleId">id</div>
    <div class="exampleClass">div</div>
    <p class="exampleClass2">paragraph</p>
</body>

var element = document.querySelector('.exampleC
 //  <div class="exampleClass">div</div>
```

querySelectorAll

```
<body>
    <div id="exampleId">id</div>
    <div class="exampleClass">div</div>
    <p class="exampleClass2">paragraph</p>
</body>

var elements = document.querySelectorAll('.exam
 // [div.exampleClass, p.exampleClass]
```

Które elementy zostaną wybrane?

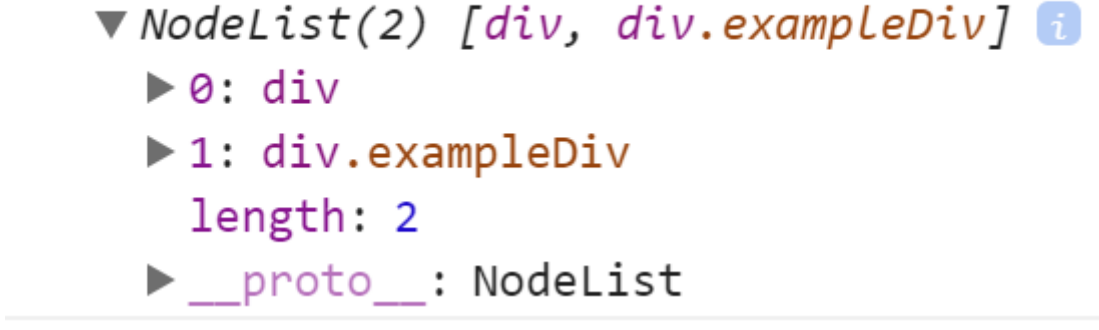
```
<body>
    <div>
      <div class="exampleDiv">div</div>
      <p id="exampleParagraph">paragraph1</p>
      <span class = "exampleSpan">span1</span>
      <span class = "exampleSpan">span2</span>
    </div>
</body>
```

```
var y = document.querySelector('div > span');
console.log(y);
```

```
<span class="exampleSpan">span1</span>
```

```
<body>
    <div>
      <div class="exampleDiv">div</div>
      <p id="exampleParagraph">paragraph1</p>
      <span class = "exampleSpan">span1</span>
      <span class = "exampleSpan">span2</span>
    </div>
 </body>
```

```
var x = document.querySelectorAll('div');
console.log(x);
```


▼ NodeList(2) [div, div.exampleDiv]
▶ 0: div
▶ 1: div.exampleDiv
length: 2
▶ __proto__: NodeList

Dodawanie i usuwanie elementów w DOM

Usuwanie elementów

```
var el = document.getElementById('exampleId');
el.remove();

// lub

el.parentNode.removeChild(element)
```

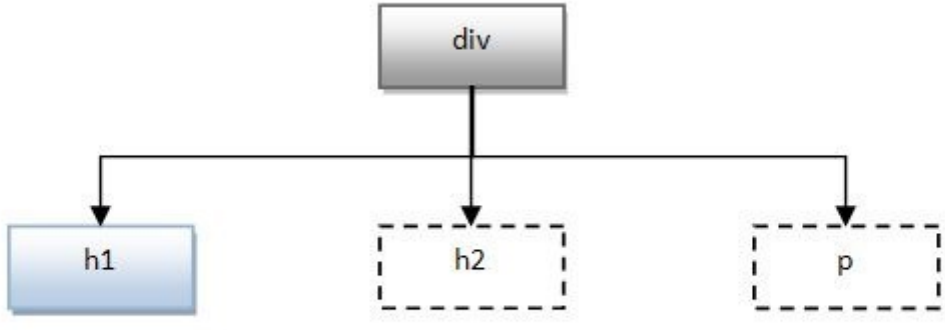
Dodawanie elementów

```
var btn = document.createElement("button");
var t = document.createTextNode("click me");
btn.appendChild(t);
document.body.appendChild(btn);
```

Poruszanie się po drzewie DOM

parentNode

Polecenie **paretNode** umożliwia wskoczenie do elementu znajdującego się powyżej danego.

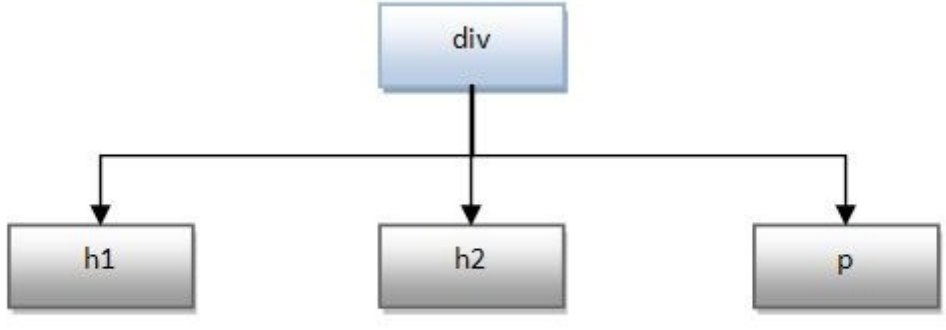


parentNode dla elementu h1

```
var h1 = document.querySelector('h1');
console.log(h1.parentNode); // div
```

childNodes

Jeśli chcemy uzyskać wszystkie elementy, które znajdują się w we wskazanym węźle możemy użyć polecenia **childNodes**. Warto sprawdzić czy wskazany węzeł w ogóle posiada jakieś inne węzły:



childNodes dla elementu div

```
var div = document.querySelector('div');
console.log(div.childNodes); // [h1, h2, p]
```

Inne

- firstChild
- lastChild
- previousSibling
- nextSibling

Window Object

Obiekt window znajduje się na szczycie hierarchii obiektów i reprezentuje okno przeglądarki. Jest też obiektem domyślnym,tzn. do jego metod i właściwości można się odwoływać bezpośrednio, z pominięciem jego nazwy.

Np.:

```
window.alert('text');
// or
alert('text');
```

Najważniejsze właściwości obiektu Window

window.navigator

Jest to pole, które posiada szereg informacji na temat przeglądarki i jej możliwości.

```
window.navigator
```

- ...
- geolocation: Geolocation
- language: "pl-PL"
- platform: "Win32"
- userAgent: "Mozilla/5.0 (Windows NT 10.0; Win64; x64) ...
- ...

window.location

Ten obiekt zawiera informacje dotyczące URL danej strony.

```
window.location
```

- ...
- host: "www.google.pl"
- hostname: "www.google.pl"
- href: "https://www.google.pl/_/chrome/newtab?ie=UTF-8"
- origin: "https://www.google.pl".
- pathname: "/_/chrome/newtab"
- ...

window.open(url) oraz window.close()

To metody, które pozwalają na otwarcie oraz zamknięcie okna.

```
window.open('pgs-soft.com')

window.close()
```

window.setTimeout() oraz window.setInterval()

Metody służą do wykonywania danej funkcji w pewnych odstępach czasu. Pierwsza z nich wykonuje podaną funkcję jeden raz, po upływie określonej liczby milisekund. Druga wykonuje funkcję co określoną liczbę milisekund, wiele razy.

```
function dot(){ console.log('.'); }
setTimeout(dot,2000);
//po 2 sekundach na konsoli pojawi się kropka

var id=setInterval(dot, 2000);
//po upływie każdych 2 s, wykona się funkcja

clearInterval(id);
//od tego momentu funkcja przestanie się wykony
```

window.document

Odnosi się do aktualnie załadowanego dokumentu czyli strony WWW.

```
window.document
```

- ...
- #document
- <!DOCTYPE html>
- <html lang="en-PL">
- <head>...</head>
- ...

Bonus: Console API

Obiekt console dysponuje kilkoma metodami, które wyświetlą podane im na wejściu informacje w konsoli.

Najważniejsze metody

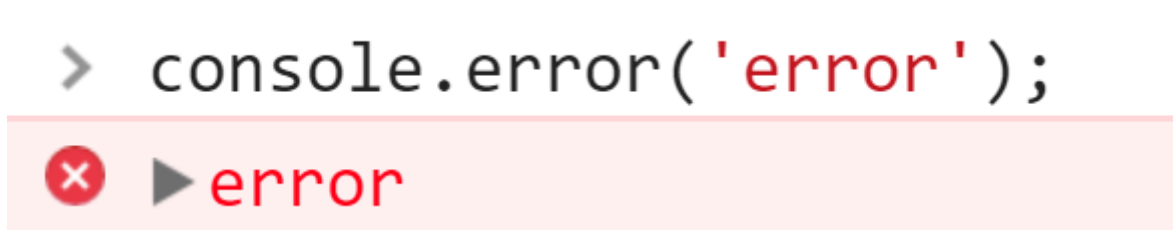
log

Zwykły wpis

```
> console.log('log');
log
```

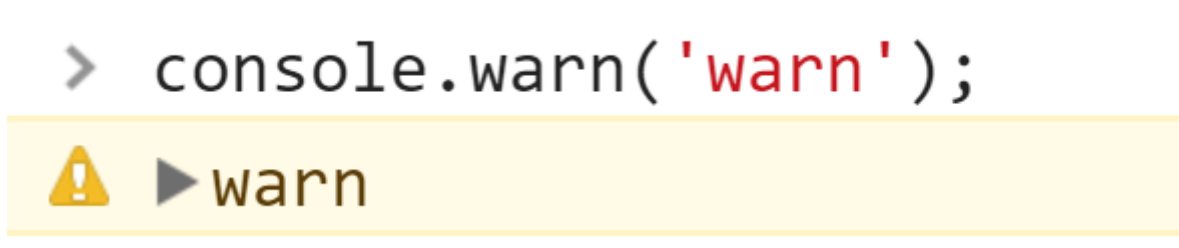
error

Komunikat błędu



warn

Komunikat ostrzeżenia



info

Informacja

```
> console.info('info');
  info
```

table

Wyświetla dane tabelaryczne jako tabelę

```
> console.table([1,2,3]);
```

VM235:1

(index)	Value
0	1
1	2
2	3

▶ Array(3)

time oraz timeEnd

time - uruchamia timer timerEnd - zatrzymuje timer

```
console.time('mójIdentyfikator');
// kod którego wydajność chcemy sprawdzić
console.timeEnd('mójIdentyfikator');
```

```
> console.time('time')
< undefined
> console.timeEnd('time')
  time: 1776.171875ms
```

Zdarzenia

Elementy DOM dostarczają szereg zdarzeń, które wywoływane są kiedy zachodzi interakcja użytkownika ze stroną.

```
var myBtn = document.getElementById("myBtn");
var listener = function() {
    console.log('wciśnięto przycisk');
};

myBtn.addEventListener("click", listener);
```

Przykłady wbudowanych zdarzeń:

- **change** - wywoływany jest w momencie gdy obiekt zmieni swoją zawartość
- **click** - zdarzenie kliknięcia elementu
- **error** - wywoływany jest kiedy w skrypcie wystąpi błąd
- **focus** - wywoływany jest kiedy element staje sie aktywny (przeciwieństwo blur)
- **keydown** - wywoływany jest w momencie naciśnięcia klawisza klawiatury
- **mouseover** - występuje w momencie najechania na element kursorem myszki (przeciwieństwo mouseout)
- **submit** - występuje w momencie zatwierdzenia formularza

onload vs DOMContentLoaded?

- **onload** - wszystkie zasoby zostały pobrane (obrazki, skrypty, style)
- **DOMContentLoaded** - HTML jest gotowy do interakcji (bez czekania na style, skrypty, obrazki)

Unobtrusive JavaScript

Unobtrusive

```
var btn = document.getElementById('btn');
btn.addEventListener('click', myFunction);
```

Obtrusive

```
<button id="btn" onclick="myFunction()">
    Click me
</button>
```

Bąbelkowanie (ang. bubbling)

```
<section onclick="alert('section')">
    SECTION
    <div onclick="alert('div')">
        DIV
        <p onclick="alert('p')">P</p>
    </div>
</section>
```

- Propagowanie zdarzenie w górę aż do napotkania objektu 'document'
- Nie wszystkie zdarzenia są propagowane, np. submit, focus, blur

Przerwanie bąbelkowania

```
<div onclick="alert('alert się nie wykona')">
    <button onclick="event.stopPropagation()">
        Click me
    </button>
</div>
```

Przerwanie bąbelkowania - c.d.

- Nie przerywaj bąbelkowania bez potrzeby!

Przerwanie domyślnych akcji

```
<a href="pgs-soft.com"
   onclick="return false">
    PGS Software
</a>

// or

<a href="pgs-soft.com"
   onclick="event.preventDefault()">
    PGS Software
</a>
```

Przechowywanie danych w przeglądarce

Cookies (ciasteczka)

Session storage

Local storage

Cookies (ciasteczka)

```
// Zapisywanie
document.cookie = "test1=Hello";
document.cookie = "test2=World;max-age=31536000

// Odczytywanie
var cookie = document.cookie;
console.log(cookie); // 'test1=Hello; test2=Wor

// Usuwanie
document.cookie = "test1=; expires=Thu, 01 Jan
console.log(document.cookie); // 'test2=World'
```

- Są przesyłane w każdym zapytaniu do backendu
- Backend może też modyfikować ciasteczka
- Do obsługi ciasteczek używamy dodatkowych bibliotek z lepszym API
- Są ograniczenia co do rozmiaru ciasteczek, najczęściej jest to 4096 bajtów

Session storage

```
// Zapisywanie
sessionStorage.setItem('test1', 'Hello');

// Odczytywanie
var value = sessionStorage.getItem('test1');
console.log(value) // 'Hello'

// Usuwanie
sessionStorage.removeItem('test1');
```

Local storage

```
// Zapisywanie
localStorage.setItem('test1', 'Hello');

// Odczytywanie
var value = localStorage.getItem('test1');
console.log(value) // 'Hello'

// Usuwanie
localStorage.removeItem('test1');
```

Różnica pomiędzy sessionStorage a localStorage?

- sessionStorage jest kasowany po zakończeniu sesji (np. zamknięcie karty)

Komunikacja z backendem

XMLHttpRequest

Służy do pobierania danych przy użyciu protokołu HTTP. Wbrew nazwie może być stosowany do obsługi dokumentów w wielu formatach, nie tylko XML, ale także JSON.

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (req.readyState === 4) {
        if(req.status === 200) {
            console.log(req.responseText);
        } else {
            console.error("Błąd podczas ładowa
        }
    }
};
xhttp.open("GET", "http://example.com/movies.js
```

Promise (obietnice)

Obietnice to obiekty reprezentujące wartości, które będzie można wykorzystać w przyszłości. Przy ich użyciu można przechwycić wynik operacji asynchronicznej, np. zdarzenia, i obsłużyć ją w jednolity sposób.

```
var myPromise = new Promise(function(resolve, r
  setTimeout(function() {
    resolve("Success!");
  }, 1000);
});

myPromise
    .then(function(message) {
        console.log('Hura! ' + message);
    });
```

Promise (obietnice) - cd.

```
var myPromise = new Promise(function(resolve, r
  setTimeout(function() {
    reject('Something is not yes');
  }, 1000);
});

myPromise
    .then(function() {
        console.log('Zostanę pominięty');
    })
    .catch(function(error) {
```

Fetch

Fetch API jest narzędziem do komunikowania się z różnymi źródłami danych podobnie jak XMLHttpRequest. Różnica między nimi sprowadza się do tego, że Fetch API korzysta z Promises.

```
fetch('http://example.com/movies.json')
    .then(function(response) {
        return response.json();
    })
    .then(function(myJson) {
        console.log(myJson);
     });
```

Dziękuję za uwagę !

Odwiedź:

www.facebook.com/AkademiaDeveloperaRzeszow

PAWEŁ LULA • **FRONT-END DEVELOPER**