Techniki Internetowe - wykład 4
Wprowadzenie do JavaScript
DOM i jego obsługa w JS

dr hab. inż. Jerzy Duda, 2022

Literatura (JavaScript)

Zasoby internetowe

- https://www.w3schools.com/js/default.asp
- http://developer.mozilla.org/en/docs/JavaScript
- **http://kursjs.pl/**

Książki

- Marijn Haverbeke: Zrozumieć JavaScript. Wprowadzenie do programowania, Helion 2015
  wersja angielska nieoficjalna wersja polska
- Marcin Lis: JavaScript. Ćwiczenia praktyczne, Wyd. III, Helion 2013
- Shelley Powers: JavaScript. Wprowadzenie, Helion 2007
- Michael Moncur: JavaScript dla każdego. Wydanie IV, Helion 2007
- Darmowe dostępne online: https://jsbooks.revolunet.com/

JavaScript – historia

Język **JavaScript** został stworzony w **1995 r.** przez ***Brendana Eicha*** w firmie Netscape

1996 r. pojawia się w przeglądarce *Netscape Navigator 2.0* (marzec) oraz *Microsoft Internet Explorer 3.0* (sierpień)

Początkowo projekt nazywał się ***Mocha***, później ***LiveScript***,
a dopiero na mocy porozumienia z *Sun* przemiano go na ***JavaScript***

Microsoft rozwija swoją wersję jako ***JScript***

1997 r. JavaScript został poddany standaryzacji przez **ECMA International**
i od tej pory ta organizacja zajmuje się rozwojem języka

Oficjalny standard JavaSrcipt nosi nazwę ***ECMA-262*** i został zatwierdzony w sierpniu **1997 r.** - w skrócie stosowana jest nazwa ***ECMA Script***

1998 r. standard **ECMA** został zaaprobowany także przez ISO jako ***ISO/IEC 16262***

ECMA Script – rozwój standardu

ECMA 1 – wersja zaimplementowana w 1997 r

ECMA 2 – zaimplementowana w 1998 r., drobne zmiany, standard ISO

ECMA 3 – wyrażenie regularne, bloki try/catch, zaimplementowana w 2000 r.

ECMA 5 – 'strict mode' zaimplementowany w 2010 r.

ECMA 6 (aka ECMA 2015) – określony w 2015 r.

- stałe (*const*), zmienne lokalne (*let*)
- klasy (*class*, *extends*), składowe statyczne
- moduły
- iteratory, generatory
- typ Symbol, kolekcje (Map, Set, WeakMap i WeakSet)

ECMA 2016 – określony w 2016 r., niewielkie zmiany – m.in. operator **

ECMA 2017 – funkcje *async* i *await,* typy SIMD, przeciążanie operatorów w OOP

ECMA 2018 – m.in. pętle asynchroniczne, spread operator (...)

ECMA 2019 – drobne zmiany m.in. prototypy obiekty Array

ECMA 2020 – ostatnia wersja z czerwca 2020, min. nullish operator (??)

Umieszczanie JavaScript na stronie

Skrypty JS umieszcza się w bloku <script>

```
<html>
 <body>
  <script type="text/javascript">
   document.write("Witaj świecie");
  </script>
 </body>
</html>
```

Można wykorzystywać skrypty zapisane w zewnętrznym pliku JS

```
<script src="xxx.js"></script>
```

Zmienne w JavaScript

Zmienne deklaruje się słowem kluczowym **var** bez podawania typu (typ ustawiany jest na **Undefined**)

Zmienne zadeklarowane w funkcji są widoczne tylko w jej obrębie (=**widoczność funkcyjna**)

Nazwy zmiennych muszą zaczynać się od litery lub podkreślnika

```
var x;
var nazwa;
var y = 10;
```

Zmienne niezadeklarowane słowem **var** są deklarowane automatycznie i mają zasięg globalny!

Ponowna deklaracja nie powoduje utraty pierwotnej wartości

```
z = 5;
var z;
document.write(z); // 5
```

W **ECMA6** wprowadzono zasięg lokalny – zmienne deklarowane słowem **let**

```
{ let a=10; }
console.log(a); //undefined
```

Typy w JavaScript

W JavaScript mamy pięć typów prostych

- **undefined**
- **null**
- **boolean**
- **numeric**
- **string**

oraz typ **object** (obiekt, JSON, tablica)

Typ ustawiany jest po pierwszym przypisaniu wartości do zmiennej

Typ zmiennej można odczytać operatorem **typeof**

```
var a = 'Tekst';
console.log(typeof(a));
// "string"
console.log(typeof(b));
// "undefined"
```

Tryb 'strict' w JavaScript

W **ECMAScript5** został wprowadzony tryb ścisły (*strict*)

Jego zastosowanie ma na celu zapobieganiu prostych błędów w kodzie JS

Włączany jest poleceniem **'use strict'**

Musi być to pierwsze polecenie w kodzie lub w funkcji (*strict* działa tylko dla tej funkcji)

W trybie *strict* wszystkie zmienne muszą być zadeklarowane przed ich użyciem

```
'use strict'
a = 5;
// błąd!
```

Dodatkowo sprawdzanie jest m.in. czy nie powtarzają się nazwy argumentów funkcji lub nazwy właściwości obiektów

Operatory w JavaScript (1)

Operatory arytmetyczne znane z C

+ - * / % ++ --

Operatory przypisania

= += -= *= /= %=

Operatory łączenia łańcuchów znakowych:

+ +=

```
x=5+5;
document.writeln(x);
x="5"+"5";
document.writeln(x);
x=5+"5";
document.writeln(x);
x="5"+5;
document.writeln(x);
```

Operatory w JavaScript (2)

Operatory porównywania

== != > < >= <= (od JS 1.3: === !==)

Operatory logiczne

&& || !

Operatory bitowe

& | ^ << >> >>> ~

Operatory przypisania z operacją bitową

<<= >>= >>>= &= |= ^=

Operator warunkowy

<warunek> ? <wartość_prawda> : <wartość_fałsz>

```
osoba = (plec=="K") ? "Pani" : "Pan ";
```

Operatory specjalne:

instanceof (1.4) **in** (1.4) **typeof** **new** **delete** **void**

Instrukcje warunkowe

Składnia **if** zgodna z C

```
if (warunek1)
{
 // instrukcje
}
else if (warunek2)
{
 // instrukcje
}
else
{
 // instrukcje
}
```

```
var d = new Date()
var time = d.getHours()
if (time<10)
{
 document.write("<b>Witaj rano!</b>");
}
else if (time>10 && time<18)
{
 document.write("<b>Witaj!</b>");
}
else
{
 document.write("<b>Witaj wieczorem!</b>");
}
```

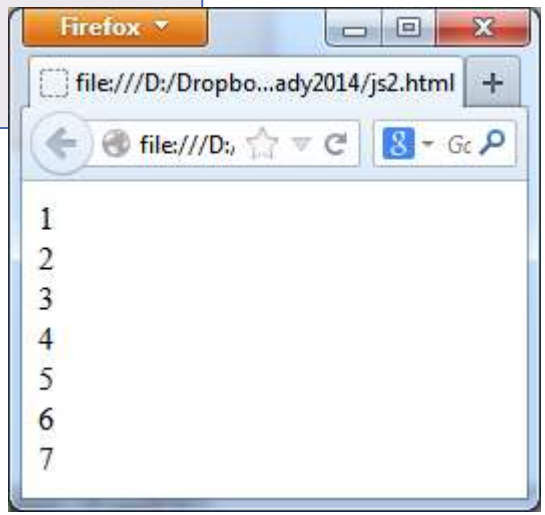
Składania **switch** również zgodna z C – pamiętajmy o break!

Pętle (1)

Pętla **for** – identycznie jak w C

```
for (wyr.początkowe; wyr.końcowe; wyr.modyfikujące)
{
    // instrukcje
}
```

```
for (let i=1;i<=7;i++)
{
 document.write(i+"<br />");
}
```



Podobnie pętle **while** oraz **do ... while**

```
while (wyrażenie warunkowe)
{
    instrukcje
}
```

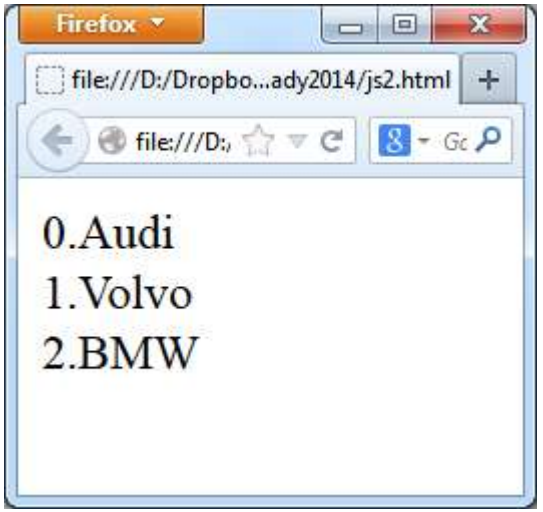
```
while (let i<=7)
{
 document.write(i+"<br />");
 i++;
}
```

Pętla foreach (2)

Do iteracji elementów w tablicy oraz obiektów może służyć instrukcja **for .. in**

```
for (zmienna in/of tablica/obiekt)
{
  // instrukcje
}
```

```
var auta = new Array();
auta[0] = "Audi";
auta[1] = "Volvo";
auta[2] = "BMW";
for (var x in auta)
{
 document.write(x+"."+auta[x]+"<br />");
}
```



Okna dialogowe

Alert box – wyświetla okienko z informacją dla użytkownika oraz przycisk **OK**

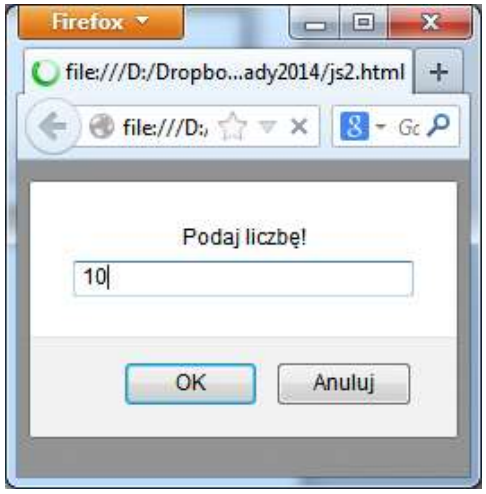
```
alert("Witaj świecie!");
```

Confirm box – wyświetla okienko z informacją oraz przyciski **OK** i **Anuluj**; jeżeli użytkownik naciśnie OK zwracane jest **true**

```
confirm("Witaj świecie!");
```

Prompt box – umożliwia pobranie wartości od użytkownika; jeżeli naciśnie **Anuluj** zwracana jest wartość **null**

```
x=prompt("Podaj liczbę!");
```



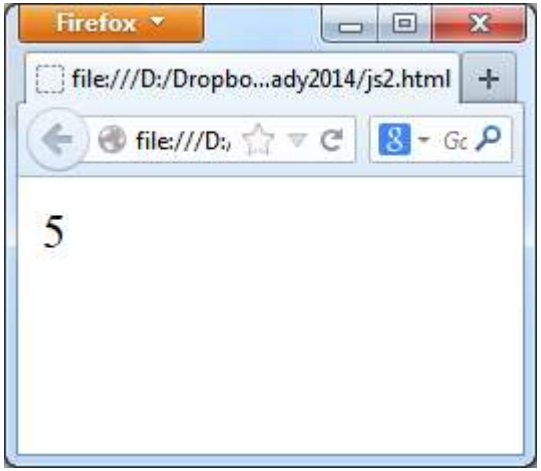
Funkcje

Funkcja tworzona jest w następujący sposób

```
function nawa_funkcji(arg1, arg2, ..., argN)
{
 // instrukcje
}
```

```
function suma(a,b)
{
 x = a + b;
 return x;
}

document.write(suma(2,3));
```



Obsługa zdarzeń w JavaScript (1)

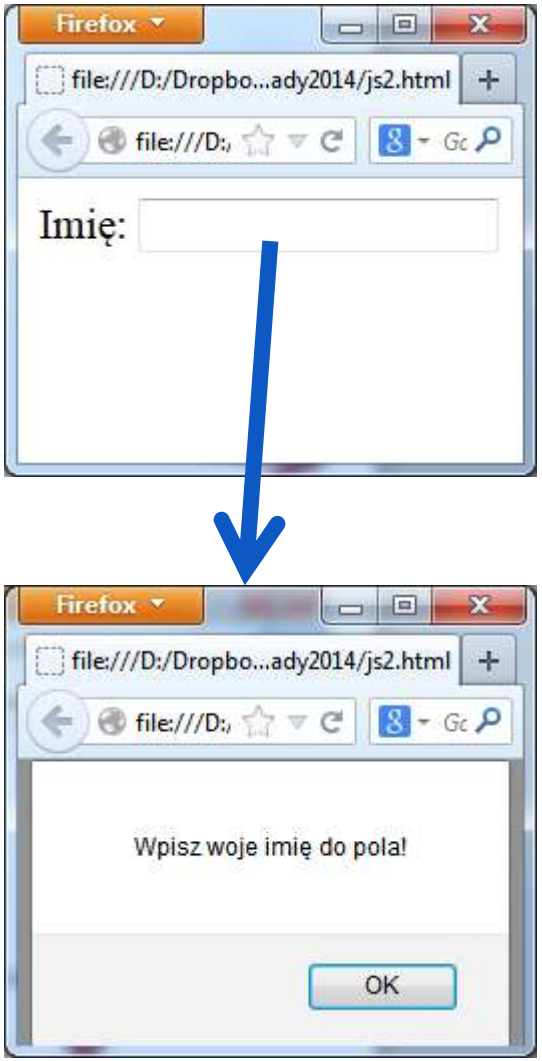
Większość elementów w HTML wyzwala zdarzenia, które mogą być obsłużone jako funkcje **JavaScript**

Podstawowe zdarzenia związane z wejściem od użytkownika:

- ***onfocus*** – element formularza uzyskał fokus
- ***onblur*** – utrata fokusu przez element np. w formularzu
- ***onchange*** – pole formularza zostało zmienione
- ***onreset*** – użytkownik zresetował obiekt; najczęściej formularz
- ***onsubmit*** – użytkownik wysłał obiekt; najczęściej formularz
- ***onkeydown*** – użytkownik nacisnął klawisz
- ***onkeyup*** – użytkownik zwolnił klawisz

Obsługa zdarzeń z formularza - przykład

```
<head>
 <script>
  function aktywuj()
   { alert("Wpisz woje imię do pola!"); }
 </script>
</head>
<body>
  Imię: <input type="text" onfocus="aktywuj()" />
</body>
```



Obsługa zdarzeń w JavaScript (2)

Podstawowe zdarzenia związane z obsługą myszki:

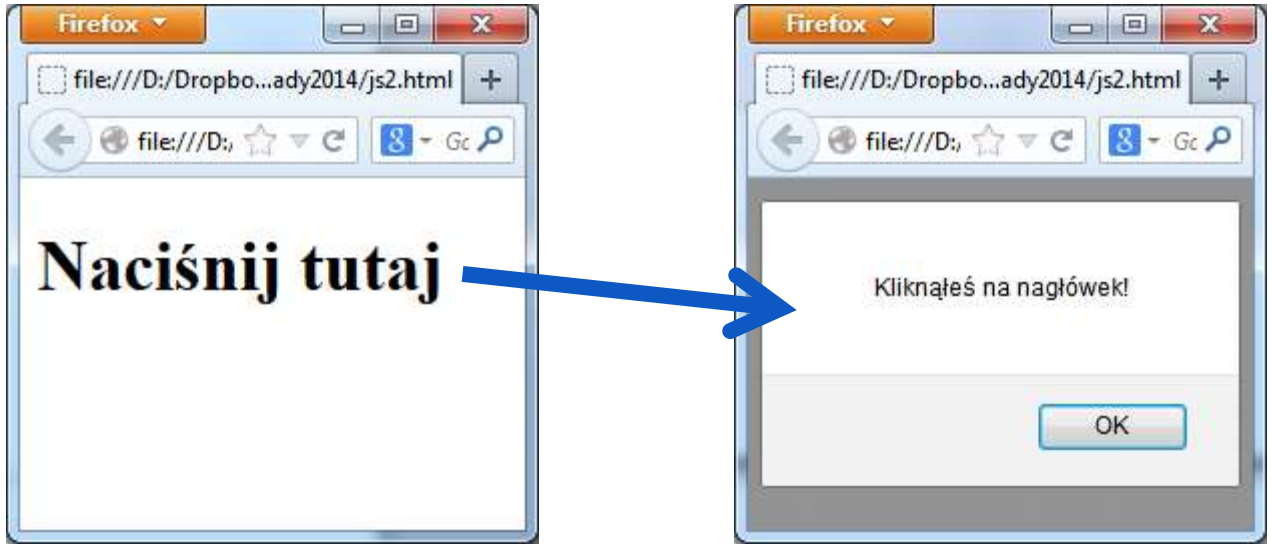
- ***onclick*** – użytkownik nacisnął przycisk na elemencie, np. odsyłaczu
- ***ondblclick*** – użytkownik wykonał podwójne kliknięcie
- ***onmouseover*** – wskaźnik myszki został przeniesiony na element
- ***onmouseout*** – wskaźnik myszki opuścił element, np. odsyłacz, grafikę

Inne ważne zdarzenia:

- ***onload*** – zakończono ładowanie grafiki lub strony HTML
- ***onunload*** – użytkownik opuścił stronę
- ***onerror*** – wystąpił podczas ładowania grafiki lub dokumentu (np. błąd w kodzie JS)

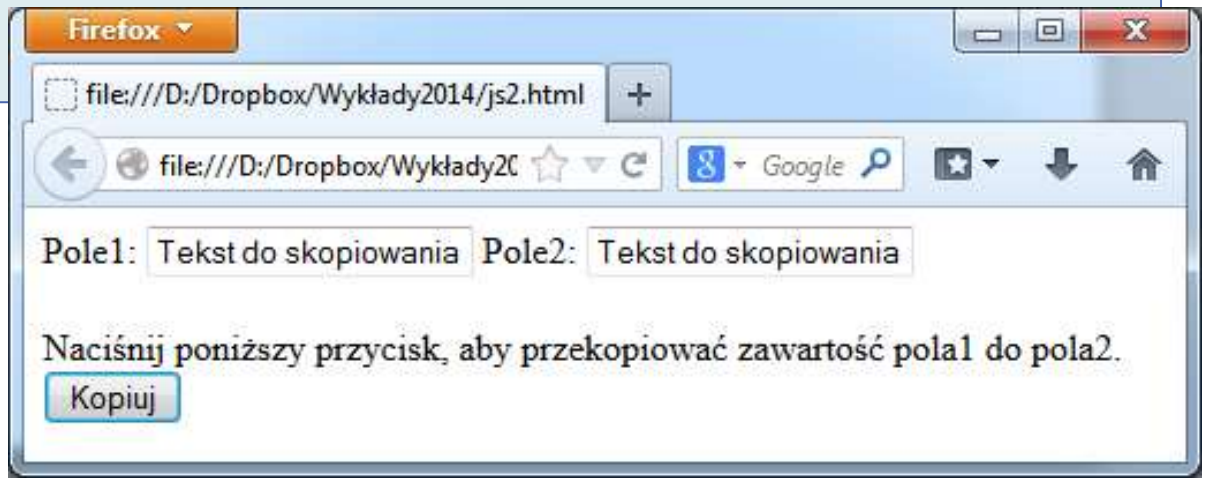
Obsługa zdarzenia kliknięcia – prosty przykład

```
<html>
 <body>
 <h1 onclick="alert('Kliknąłeś na nagłówek!');">
  Naciśnij tutaj</h1>
 </body>
</html>
```



Obsługa zdarzenia kliknięcia – przykład z wykorzystaniem DOM

```
<html>
 <body>
   Pole1: <input type="text" id="pole1" <br/>
   Pole2: <input type="text" id="pole2"> <br/><br/>
   Naciśnij poniższy przycisk, aby przekopiować zawartość pola1
   do pola2.<br/>
   <button onclick="document.getElementById('pole2').value=
    document.getElementById('pole1').value">Kopiuj</button>
 </body>
</html>
```



Obsługa błędów (1)

W JavaScript istnieją dwa sposoby obsługi błędów

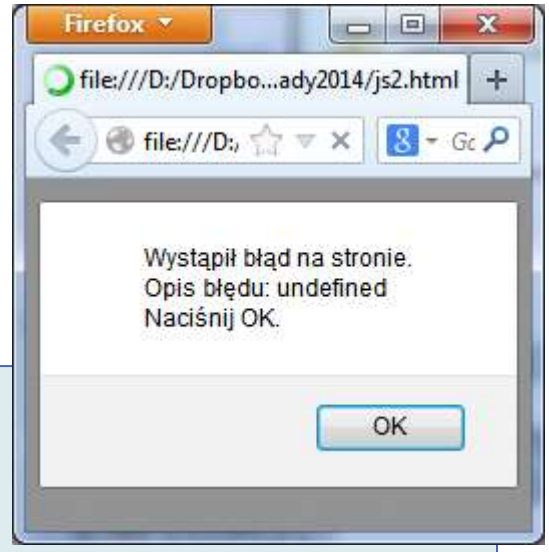
wykorzystanie bloku **try...catch** (od IE5)

wykorzystanie zdarzenia **onerror**

Blok **try...catch**

```
try
{
 //Kod do przetestowania
}
catch(err)
{
 //Obsługa błędów
}
```

```
try
{
  allert("Witaj!");
}
catch(err)
{
  txt="Wystąpił błąd na stronie.\n";
  txt+="Opis błędu:"
    +err.description+"\n";
  txt+="Naciśnij OK.\n";
  alert(txt);
 }
```



Obsługa błędów (2) – zdarzenie *onerror*

W przeciwieństwie do innych zdarzeń **onerror** nie jest przypisywany do żadnego elementu

Jest to funkcja, którą można ustawić dla obiektu **window**

```
<script
    type="text/javascript">
 window.onerror=function(){
 //obsługa błędu
 }
</script>
```

```
<head>
 <script type="text/javascript">
  window.onerror = function(){
   alert('Wystąpił błąd!')
 }
 </script>
</head>
<body>
 <script type="text/javascript">
  document.write('coś niedokończone'
 </script>
</body>
```

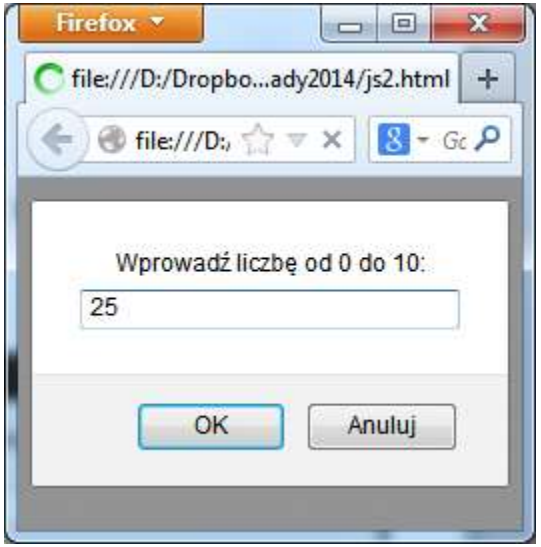
Wyrzucanie wyjątków

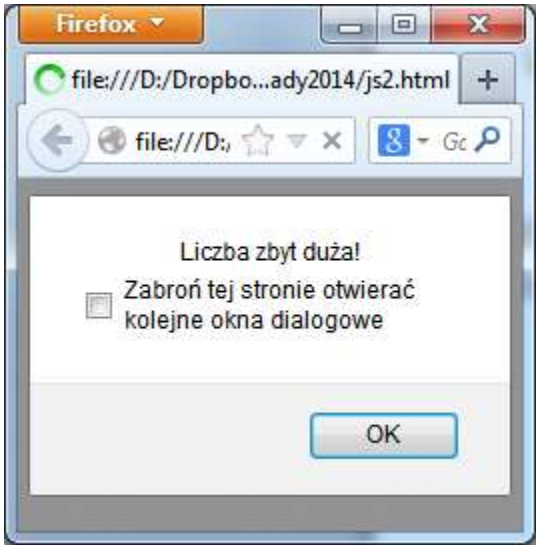
Wyjątki można wyrzucać instrukcją **throw**

```
throw(wyjątek)
```

Wyjątkiem może być łańcuch znakowy, liczba, wartość logiczna lub obiekt

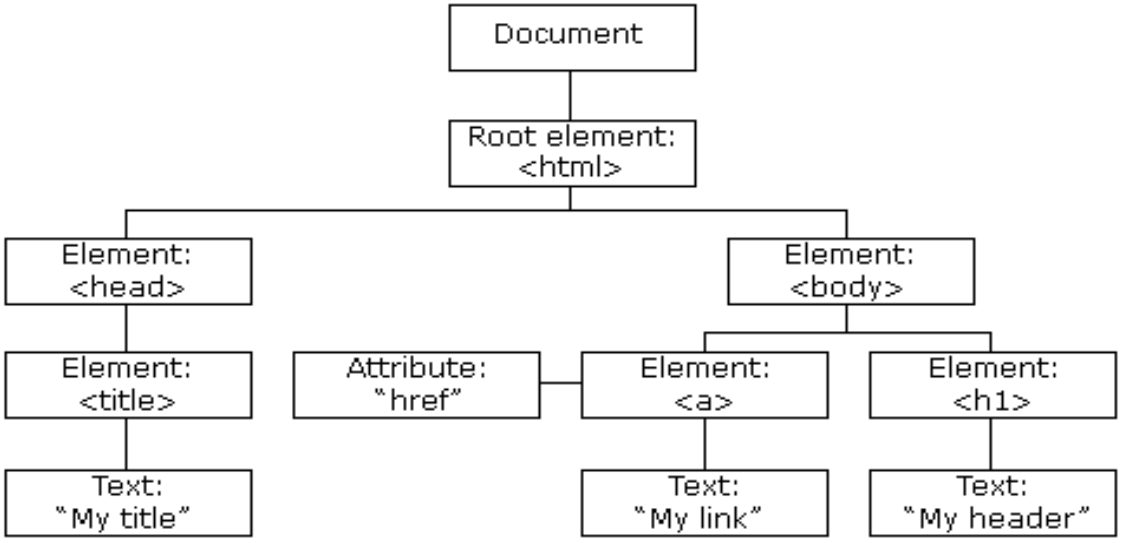
```
var x=prompt("Wprowadź liczbę od 0 do 10:","");
try
{
 if(x>10) throw "Blad1";
  else if(x<0) throw "Blad2";
}
catch(blad)
{
 if(blad == "Blad1")
   alert("Liczba zbyt duża!");
 if(blad == "Blad2")
   alert("Liczba zbyt mała!");
}
```





HTML DOM (Document Object Model)

Dokument (X)HTML jest węzłem głównym (początkowym)
Każdy znacznik w dokumencie HTML jest węzłem elementu
Teksty zawarte w elementach HTML są węzłami tekstowymi
Każdy atrybut HTML jest węzłem atrybutu
Komentarze są węzłami komentarza
Wszystkie węzły tworzą **drzewo dokumentu HTML (DOM)**




```
<html>
  <head>
    <title>My title</title>
  </head>
  <body>
     <a>My link</a>
    <h1>My header</h1>
</body>
</html>
```

Lista obiektów HTML DOM

Elementy HTML:

- **anchor** (<a>)
- **area** (na mapie obrazu)
- **base**
- **body**
- **button**
- **form**
- **frame**
- **frameset**
- **iframe**
- **image** (<img>)
- **link**
- **meta**
- **option**
- **select**
- **style**
- **table**
- **tableData** (<td>)
- **tableRow** (<tr>)
- **textarea**

document – cały dokument HTML

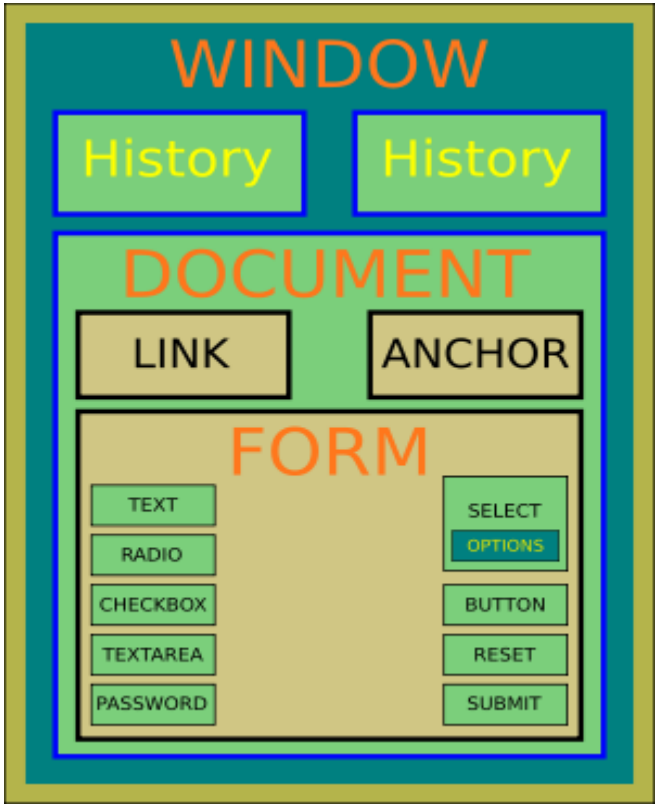
event – stan zdarzenia

Pola i przyciski w formularzu:

button
checkbox
file
hidden
password
radio
reset
submit
text

Dodatkowe obiekty JavaScript:

window
navigator
screen
history
location



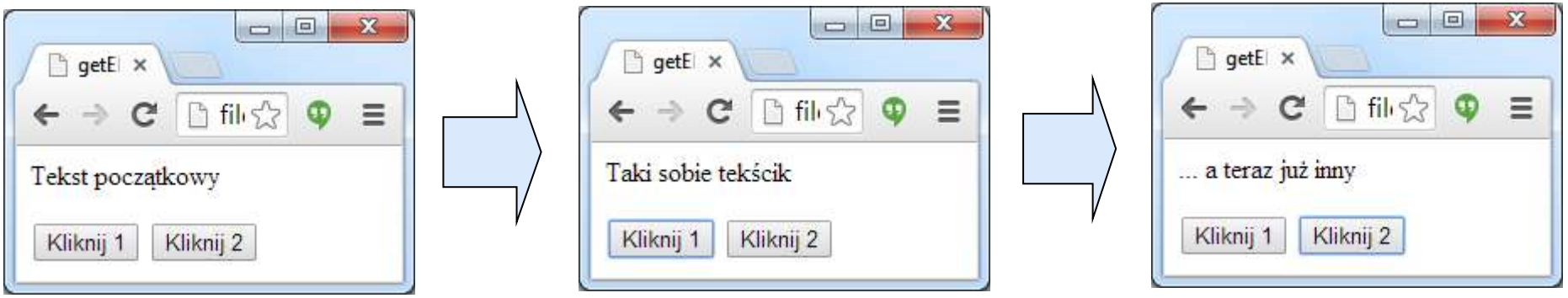
Dostęp do elementów DOM

Do węzłów DOM można dostać się stosując

- metody **getElementById()** i **getElementsByName()**
- właściwości **parentNode**, **firstChild** i **lastChild** węzła elementu

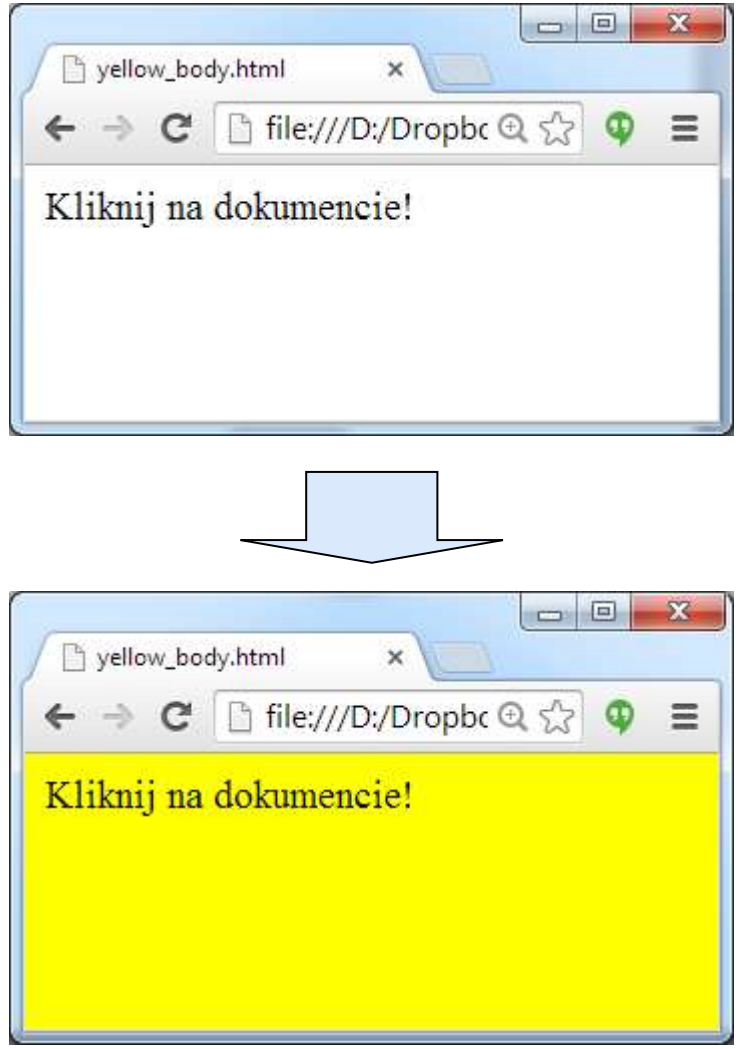
Metody ignorują strukturę drzewa i wyszukują elementy w całym dokumencie, właściwości uwzględniają strukturę

```
<p id="tekst">Tekst początkowy</p>
<button onclick="document.getElementById('tekst').innerHTML = 'Taki
    sobie tekścik';"> Kliknij 1 </button>
<button onclick="document.getElementById('tekst').innerHTML = '... a
    teraz już inny';"> Kliknij 2 </button>
```



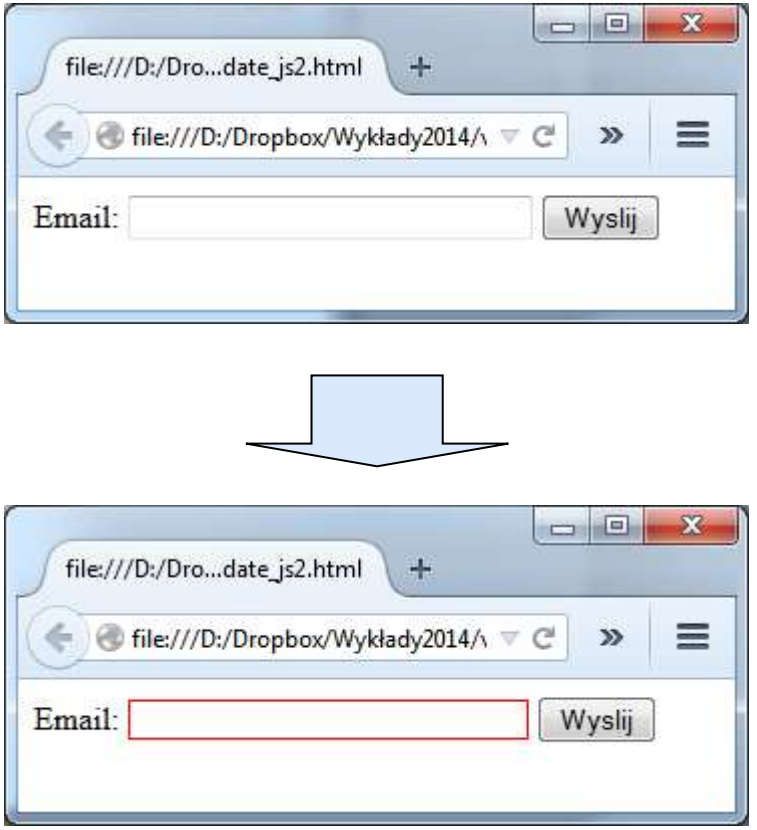
Przykład odwołania do elementu body

```
<html>
 <head>
  <script type="text/javascript">
   function ChangeColor()
   {
    document.body.style.background =
    "yellow";
   }
  </script>
 </head>
 <body onclick="ChangeColor()">
  Kliknij na dokumencie!
 </body>
</html>
```



Obsługa formularza raz jeszcze

```
<head>
 <script type="text/javascript">
  function wymagane(pole)
  {
   if (pole.value==null || pole.value=="")
    {pole.style.border="1px solid red"}
   else {return true}
  }
  function waliduj(form)
  {
   with(form)
   {
    if (!wymagane(email))
    {email.focus();return false;}
   }
  }
 </script>
</head>
```



Obiekt *window*

Obiekt ***window*** tworzony jest dla każdego okna i ramki w przeglądarce

Nie należy do standardu, ale wszystkie przeglądarki go implementują

Zawiera informacje o rozmiarze okna, ilości ramek czy obiekcie, który otwarł okno

Window zawiera także inne obiekty

- ***document*** – dokument HTML załadowany w przeglądarce
- ***history*** – obiekt z historią przeglądania
- ***location*** – obiekt zawierający informacje o bieżącym URL (ma m.in. metodę ***reload()***)
- ***navigation*** – obiekt zawierający informacje o przeglądarce
- ***screen*** – obiekt zawierający informacje o parametrach ekranu

```
document.write("Host: "+location.host+"<br/>");
document.write("Width: "+screen.width+"<br/>");
document.write("Height: "+screen.height+"<br/>");
document.write("Frames: "+window.length+"<br/>");
```

dokument załadowany z pliku, więc host=""

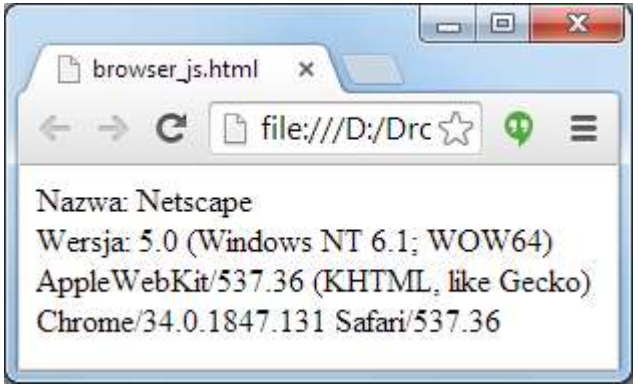
Obiekt *navigator*

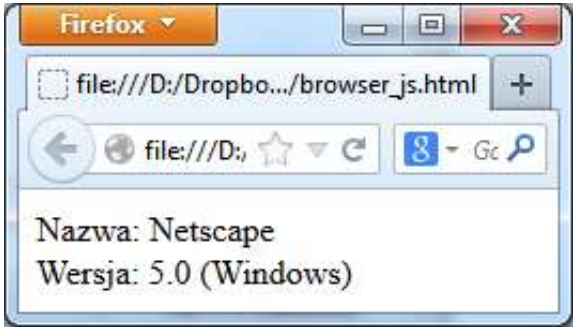
Obiekt **navigator** (właściwie ***window.navigator***) tworzony jest automatycznie przez silnik obsługujący JavaScript (**choć nie należy do standardu**!)

Navigator zawiera użyteczne informacje o przeglądarce

- ***appName*** – nazwa przeglądarki
- ***appVersion*** – wersja przeglądarki
- ***codeName*** – nazwa kodowa przeglądarki

```
<html>
 <body>
  <script type="text/javascript">
   var browser=navigator.appName;
   var version=navigator.appVersion;
   document.write("Nazwa: "+ browser);
   document.write("<br />");
   document.write("Wersja: "+ version);
  </script>
 </body>
</html>
```





Cookie w JS

Cookie - mały fragment tekstu, który serwis internetowy wysyła do przeglądarki i który przeglądarka wysyła z powrotem przy następnych wejściach na witrynę.

Aby sprawdzić, czy przeglądarka obsługuje ciasteczka można odczytać właściwość **navigator.cookieEnabled**

Obsługa ciasteczek w JavaScript sprowadza się do odczytu/zapisu właściwości **document.cookie**

Przykładowa funkcja ustawiająca cookie

escape() koduje tekst do postaci URI, np. "Żółw?!" -> "%u017B%F3%u0142w%21%3F"

```
function setCookie(c_name,value,expiredays)
{
 var ex_date.setDate(getDate()+expiredays);
 document.cookie = c_name+"="+escape(value)+
 ((expiredays==null) ? "" : ";expires="+exdate.toGMTString());
}
```

Jeżeli expires nie jest ustawiony cookie znika po zamknięciu przeglądarki

W nagłówku HTTP:
Set-Cookie: nazwa_własna=wartość; expires=DATA; path=ŚCIEŻKA; domain=DOMENA; secure

HTML5 API - WebStorage

W HTML5 wprowadzony został mechanizm **WebStorage**
W stosunku do *cookie* ma następujące zalety

- nie ma ograniczenia danych do 4kB, WS może używać do 5MB
- WS nie odsyła danych do serwera przy każdym zapytaniu
- poprawione zostało bezpieczeństwo

Można zapisywać do

- **Local strorage** (dane nie są usuwane automatycznie)
- **Session storage** (dane są usuwane po zamknięciu zakładki lub przeglądarki)

Dopuszczalny jest zapis jako
.setItem(klucz,wartość)
.klucz=wartość
["klucz"]=wartość
(analogicznie dla getItem)

```
localstorage.setItem("PI",3.14);
sessionstorage.dane = JSON.stringify({nazwa: 10}); //zapis JSON
var PI = localstorage.PI;
var dane = JSON.parse(sessionstorage["dane"]); //odczyt JSON
localstorage.removeItem("PI");
delete sessionstorage.dane;
```

HTML5 API – Geolocation API

HTML5 udostępnia też API do obsługi geolokalizacji **Web Geolocation API**
Geolokalizacja (pozycja urządzenia) działa najdokładniej dla urządzeń mobilnych (w oparciu o GPS)

Dla komputerów wyznaczana jest na podstawie adresu IP; da się ją nadpisać np. w FireFox

Preference Name	Status	Type	Value
geo.enabled	default	boolean	true
geo.provider.ms-windows-location	default	boolean	false
geo.wifi.uri	modified	string	data:application/json,{"location": {"lat": 40.7590, "lng": -73.9845}, "accuracy": 27000.0}
geo.wifi.xhr.timeout	default	integer	60000

API udostępnia przede wszystkim funkcję **getCurrentPosition()**

```
function getLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(showPosition);
  } else {
    x.innerHTML = "Geolocation is not supported by this browser.";
  }
}
function showPosition(pos) {
  x.innerHTML = "Lat:"+pos.coords.latitude+"<br>Long:"+pos.coords.longitude;
}
```

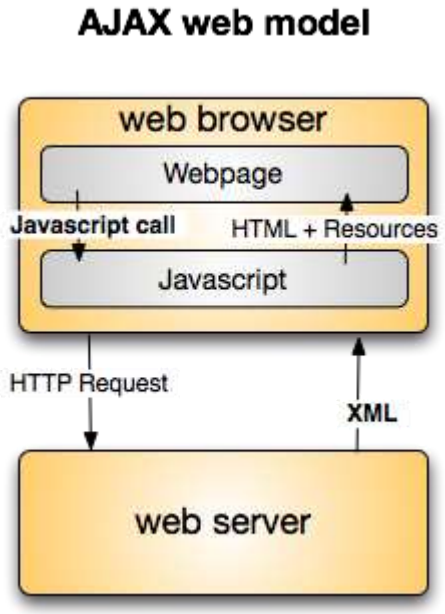
HTML5 API – Fetch API

Fetch API zostało wprowadzone do HTML w celu łatwej obsługi AJAX, bez konieczności korzystania z dodatkowych bibliotek typu *jQuery*

Jest to tzw. *Living Standard*

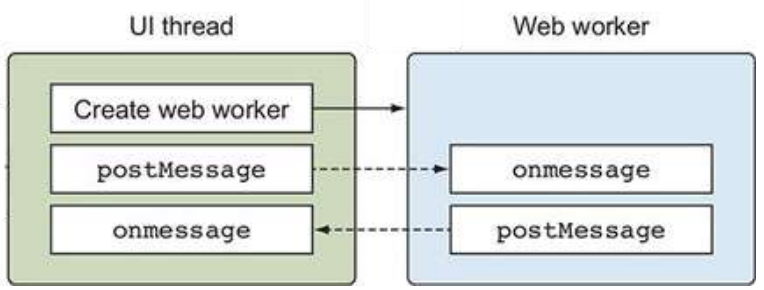
```
fetch(url)
  .then((response) => {
    return response.text();
  })
  .then((data) => {
    // do something with 'data'
  });
```

Skrócony zapis
(operator strzałkowy)



HTML5 API – Web Worker

WebWorker pozwala na uruchamianie skryptów w tle, które nie wpływają na wydajność strony; skrypty te są wykonywane w osobnych wątkach (na osobnych rdzeniach CPU)




```
function startWorker() {
  if (typeof(w) == "undefined") {
    w = new Worker("workers.js");
  }
  w.onmessage = function(event) {
    document.getElementById("result")
     .innerHTML = event.data;
  };
}

function stopWorker() {
  w.terminate();
  w = undefined;
}
```

```
var i = 0;

function timedCount() {
  i = i + 1;
  postMessage(i);
  setTimeout("timedCount()",500);
}

timedCount();
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_webworker

Dodatkowe kursy i materiały

Spis:

https://hackr.io/blog/best-javascript-courses

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide

https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/basic-javascript/

https://eloquentjavascript.net/

You Don't Know JS Yet

https://github.com/getify/You-Dont-Know-JS