# Technologie Internetu. Podstawy JavaScript

Aleksander Denisiuk (denisjuk@pja.edu.pl)
Polsko-Japońska Akademia Technik Komputerowych
Wydział Informatyki w Gdańsku
ul. Brzegi 55, 80-045 Gdańsk

22 kwietnia 2020

# Podstawy JavaScript



Najnowsza wersja tego dokumentu dostępna jest pod adresem
http://users.pja.edu.pl/~denisjuk/

# Wprowadznie

# Wprowadzenie do JavaScript



- ✔ Język skryptowy (głównie) do tworzenia interaktywnych stron internetowych.
- ✔ nie jest Javą, inna nazwa ECMAScript
- ✔ Ma C-podobną składnię. Różni się od innych języków programowania
- ✔ Nie jest kompilowany, jest dołączany do HTML i interpretowany w przeglądarce
  - ✘ `V8` — Chrome i Opera.
  - ✘ `SpiderMonkey` — Firefox.
  - ✘ `Trident`, `Chakra` —IE, `ChakraCore` — Microsoft Edge
  - ✘ `Nitro`, `SquirrelFish` — Safari
- ✔ serwerowy (Node.js)
- ✔ desktopowy (Unity, Qt, programy biurowe, etc)

# Typowe zastosowania w przeglądarce



- ✔ Zmienić stronę: napisać na niej tekst, dodać lub usunąć element, zmienić styl elementu.
- ✔ Reagować na zdarzenia: kliknięcie myszką, załadowanie strony (elementu), etc. W odpowiedzi na zdarzenie wykonuję się funkcja (`callback`).
- ✔ Wykonywać zapytania do serwera i załadowywać nowe dane bez przeładowania całej strony (AJAX, COMET).
- ✔ Czytać i ustawiać cookies, walidować dane, wyświetlać komunikaty, zapytania do użytkownika
- ✔ Zapamiętywać dane po stronie klienta (`sessionStorage`, `localStorage`)
- ✔ etc

# Ograniczenia w przeglądarce



- ✔ Nie ma bezpośredniego dostępu do systemu plików i systemu operacyjnego
- ✔ Nie ma dostępu do danych w innych oknach i kartach (Same Origin Policy)
  - ✘ chyba że skrypt sam otworzył to okno/kartę
  - ✘ specjalny kod w obu oknach
- ✔ Dostęp do lokalizacji, kamery, mikrofonu, etc — tylko za pozwoleniem użytkownika
- ✔ Nie można wysłać zapytania na inny serwer (domena, port, protokół)
  - ✘ są pewne ograniczone możliwości

# Zalety JavaScript



- ✔ Zintegrowany z przeglądarką
- ✔ Łatwo się robi proste rzeczy
- ✔ Jest wspierany prawie wszędzie
- ✔ Jest aktywnie rozwijaną technologią
- ✔ Współczesne przeglądarki są bardo zbliżone do standardu:
  - ✘ Nowy standard ECMAScript 2015 (ES6)
  - ✘ Najnowszy standard: ES6+

# Alternatywy JavaScript



- ✔ Brak

# Rozszerzenia JavaScript



- ✔ CoffeeScript — „uproszczenie” składni
- ✔ TypeScript — typizacja danych, duże systemy, Microsoft
- ✔ Flow — typizacja danych, Facebook
- ✔ Dart — ma własne środowisko uruchomieniowe, Google

# Konsola przeglądarki



- ✔ Zobacz
  - ✘ `F12`, `Ctrl+Shift+J` (Firefox, Chrome)

# Włączenie JavaScript w HTML

# JavaScript wewnątrz HTML



```
<html>
<body>
    <h1>Silnia</h1>
   <script>
            S=1;
        for(var i=1; i<=3; i++) {
            S*=i;
        }
    alert("3!="+S)
    </script>
</body>
</html>
```

✔ W dowolnym miejscu dokumentu

# JavaScript w head



```
<html>
<head>
    <script>
    function silnia(x) {
        s=1;
        for(var i=1; i<=x; i++) s*=i;
        alert(x+"!="+s)
  }
  </script>
</head>
<body>
  <input type="button" onclick="silnia(3)"
                value="Silnia" />
</body>
</html>
```

# Archaiczna składnia



✔

```
<script type="text/javascript"
```

✔

```
<script language="javascript"
```

✔

```
<script type="text/javascript"
      language="javascript"><!--
    ...
//--></script>
```

# Wykonanie skryptów



- ✔ Przeglądarka
  - ✘ renderuje i wyświetla dokument przed `<script>`
  - ✘ po znaczniku `<script>` przełącza się w tryb JavaScript i wykonuje skryptów
  - ✘ po ukończeniu skryptu wraca w tryb HTML i kontynuuje renderowanie dokumentu

# Zewnętrzny JavaScript



```
<html>
<head>
  <script src="URL"></script>
</head>
<body>
</body>
</html>
```

- ✔ Można podłączyć kilka skryptów
- ✔ Jeżeli jest atrybut `src`, zawartość znacznika `<script>` zostanie zignorowana

# Problem



- ✔ Wolne załadowanie i wykonanie skryptu może naruszyć funkcjonalność strony:

```
<p>ważna informacja czeka na skrypt...</p>
<script src="http://edu.pl/script.js"></script>
<p>... ważna informacja!</p>
```

- ✔ Zobacz

# Rozwiązanie



1. Skrypt na dole strony
   - ✔ nie zawsze jest dobrym rozwiązaniem
2. Atrybut `async`
   - ✔ skrypt nie blokuje przeglądarki, wykonywany po załadowaniu
3. Atrybut `defer`
   - ✔ skrypty są wykonywane w tej kolejności, w jakiej są umieszczone w dokumencie

```
<p>ważna informacja...</p>
<script async src="http://edu.pl/script.js"></script>
<p>... już nie czeka na skrypt!</p>
```

✔ Zobacz

# Uwaga



- ✔ `async` i `defer` dotyczą tylko skryptów zewnętrznych

# Podstawy JavaScript

# Średnik



```
a = 5
a = 5;

alert(3 +
1
+ 2);

alert("Tu będzie błąd")
[1, 2].forEach(alert)
```

✔ zawsze stawić

# Komentarze



```
/*
  wieloliniowy
*/
//jednoliniowy
}
```

# Tryb „strict”



- ✔ nowy standard (ES5)
  - ✘ nie ma 100% kompatybilności wstecz
  - ✘ w szczególności, nie można nie deklarować zmiennych
- ✔ `"use strict"` (`'use strict'`)
- ✔ na początku skryptu
- ✔ na początku funkcji
- ✔ nie ma sposobu wyłączyć ten tryb
- ✔ w konsoli przeglądarki jest wyłączony
- ✔ zawsze korzystać (IE10+)

# let



```
let konus;
let message = 'Hello!';
let user = 'John', age = 25;
```

- ✔ Nie są typowane
- ✔ Użycie tylko po deklaracji
- ✔ Zasięg: blok {...}
- ✔ Wielkość liter ma znaczenie
  - ✘ $, znaki narodowe

# Stałe



- ✔ Duże małe litery

```
const apple = 5;
apple = 6; // Błąd
```

- ✔ W danych złożonych można zmieniać składowe:

```
const user = {
  name: "Olek"
};

user.name = "Olga"; // może być
user = 5; // Błąd
```

- ✔ Duże vs małe litery

```
const COLOR_ORANGE = "#FF7F00";
const pageLoadTime = ...;
```

# var



- ✔ Nie ma blokowego zasięgu

```
var i = 0
{
  var i=5
  alert(i) // 5
}
alert(i) // znowu 5
```

- ✔ Lokalne wewnątrz funkcji

```
a = 1;
var b = 2;
function go() {
  a = 6;
  var b=7;
}
```

# var



✔ Przy wejściu do funkcji tworzy się zmienne `var`

```
function cmp(a,b) {
  if (a>b) {
    res = 1
  } else if (a<b) {
    res = -1
  } else {
    var res = 0
  }
  return res
}
```

✔ Używać `let`

✘ można spotkać `var`

# Liczby



- ✔ `float64` — błędy zaokrąglenia

```
console.log( 0.1 + 0.2 )
   // 0.30000000000000004
```

- ✔ specjalne wartości:
  - ✘ `1/0 = Infinity`
  - ✘ `-1/0 = -Infinity`
  - ✘ `"something"/2 = NaN`
    - ✓ `NaN + 1 = NaN`
    - ✓ `NaN == NaN` // false
    - ✓ `isNaN(NaN)` // true
    - ✓ `isFinite(Infinity)` // false
- ✔ nie ma komunikatów o błędach
- ✔ skrypt się nie zatrzymuje

# Funkcje matematyczne



- ✔ `Math.floor()`, `Math.round()`, `Math.ceil()` — zaokrąglenie
- ✔ `Math.abs()`
- ✔ `Math.sin()`
- ✔ etc.

# BigInt



✔ Firefox i Chrome

```
const bigInt
    = 12345678901234567890123456789012345678901234567890n;
```

# String



- ✔ Unicode
- ✔ W apostrofie/cudzysłowie

```
let str = "Hello";
let str2 = 'To samo';
let phrase = `Może zawierać wyrażenia: ${str}`;

let name = "World"
alert( `Hello,, ${name}!` );
alert( `wynik: ${1 + 2}`
```

- ✔ Nie ma typu dla pojedyńczej litery
- ✔ stringi wielowierszowe

```
`...it is impossible
to achieve the aim without suffering...`
```

# Boolean



- ✔ Dwie wartości

```
let nameFieldChecked = true;
let ageFieldChecked = false;

let isGreater = 4 > 1;
```

# null



- ✔ Specjalna wartość
  - ✘ nic, puste, nieznane

```
let age = null;
```

# undefined



- ✔ Specjalna wartość
  - ✘ nie inicjowane

```
let x;
alert(x); // undefined
```

# `object` oraz `symbol`



- ✔ `object` — złożony typ danych, obiekt
- ✔ dane typu `symbol` wykorzystane są jako unikatowe klucze

# Operator `typeof`



- ✔ Dwie formy:
  - ✘ `typeof x`
  - ✘ `typeof(x)`

```
typeof undefined // "undefined"
typeof 0 // "number"
typeof NaN // "number"
typeof 10n // "bigint"
typeof true // "boolean"
typeof "foo" // "string"
typeof Symbol("id") // "symbol"
typeof Math // "object"  (1)
typeof null // "object"  (2)
typeof alert // "function"  (3)
```

# Na tekst



- ✔ W sposób naturalny
- ✔ Automatycznie w niektórych funkcjach:

```
let value = true;
alert(typeof value); // boolean
alert(value);        // "true"
```

- ✔ W sposób jawny:

```
value = String(value); // value == "true"
alert(typeof value); // string
```

# Na liczby



✔ Automatycznie w wyrażeniach matematycznych:

```
alert( "6" / "2" ); // 3
```

✘ operator + łączy tekst

```
alert( '1' + '2' ); // '12'
alert( 1 + '2' ); // '12'
alert( '1' + 2 ); // '12'
```

✔ W sposób jawny:

```
let str = "123";
alert(typeof str); // string
let num = Number(str); // liczba 123
alert(typeof num); // number
```

# Na liczby, wartości specjalne



- ✔ `undefined` ↦ `NaN`
- ✔ `null` ↦ 0
- ✔ `true` ↦ 1
- ✔ `false` ↦ 0
- ✔ `string`: spacje na początku i na końcu są ignorowane, w razie pomyłek wynikiem będzie `NaN`
  - ✘ pusty tekst konwertuje się na zero

```
alert( Number("   ") ); // 0
alert( Number("   123   ") ); // 123
alert( Number("123z") );      // NaN
alert( Number(true) );        // 1
alert( Number(false) );       // 0
```

# Na liczby, parseInt oraz parseFloat



```
alert( parseInt('100px') ); // 100
alert( parseFloat('12.5em') ); // 12.5
alert( parseInt('12.3') ); // 12
alert( parseFloat('12.3.4') ); //12.3
alert( parseInt('a123') ); // NaN
alert( parseInt('0xff') ); // 255
alert( parseInt('0b11111111') ); // 255
alert( parseInt('0o377') ); // 255
alert( parseInt('0377') ); // 255

alert( parseInt('0xff', 16) ); // 255
alert( parseInt('ff', 16) ); // 255
alert( parseInt('2n9c', 36) ); // 123456
```

# Na typ logiczny



- ✔ Do `false` konwertuje się:
  - ✘ `false`
  - ✘ `null`
  - ✘ `undefined`
  - ✘ `""`
  - ✘ 0
  - ✘ `NaN`
- ✔ Do `true` — reszta:
  - ✘ `"0"`
  - ✘ `"false"`
  - ✘ etc

# Porównywanie



- ✔ Operatory <, <=, >, >=, ==, !=
- ✔ Tekst: porządek leksykograficzny
  - ✘ `console.log("ą">"a")` ??
  - ✘ `console.log("ą">"b")` ??
  - ✓ `str.localeCompare(str2)`
    - ✗ liczba ujemna, jeżeli `str < str2`
    - ✗ liczba dodatnia, jeżeli `str > str2`
    - ✗ 0 jeżeli `str == str2`
  - ✓ `"ą".localeCompare("b")`

# Porównywanie różnych typów



- ✔ Konwersja na liczby

```
alert( '2' > 1 ); // ??
alert( '01' == 1 ); // ??
alert( true == 1 ); // ??
alert( false == 0 ); // ??
```

- ✔ `null` oraz `undefined` przy porównywaniu `==` z czymkolwiek (oprócz `null` i `undefined`) zawsze dają `false`
- ✔ `NaN` przy wszystkich porównywaniach daje `false`
- ✔ przy konwersji na `Number` `null` daje 0, a `undefined` — `NaN`

# Curiosum



```
alert(null >  0); //false
alert(null == 0); //false
alert(null >= 0); //true

alert(undefined >  0); //false
alert(undefined == 0); //false
alert(undefined <  0); //false
```

# Porównywanie a identyczność



- ✔ Powównywanie:
  - ✘ `val1 == val2`
    - ✓ `true`, jeżeli watrości są równe (po ewentualnych konwersjach typów)
  - ✘ `val1 != val2`
- ✔ Identyczność:
  - ✘ `val1 === val2`
    - ✓ bez konwersji typów
    - ✓ `true`, jeżeli watrości i typy są równe
      - ✗ `NaN === NaN` daje `false`
      - ✗ `isNaN(NaN)`
  - ✘ `val1 !== val2`

# Interakcja z użytkownikiem

# alert



- ✔ Wyświetla *modalne* okno z komunikatem

```
alert("Hello, World!");
```

- ✔ Zobacz

# prompt



```
result = prompt(title, default);
```

- ✔ Wyświetla modalne okno z nagłówkiem `title` i polem tekstowym, wypełnionym przez `dafault`
- ✔ Zwraca zawartość pola albo `null`
- ✔ Zobacz
- ✔ Można nie podawać drugiego argumentu

```
result = prompt(title);
```

- ✘ IE wstawi w polu `undefined`
- ✘ lepiej

```
result = prompt(title, '');
```

# confirm



```
result = confirm(title);
```

- ✔ Wyświetla modalne okno z zapytaniem `title`
- ✔ Zwraca `true` albo `false`
- ✔ Zobacz

# Instrukcje sterujące

# Wybór



```
if (WARUNEK) {...}

if (WARUNEK) {...} else {...}

if (WARUNEK) {
    ...
} else if (INNY_WARUNEK){
    ...
}

var x = WARUNEK ? wyrażenie_1 : wyrażenie_2;
```

# switch

```
switch(x) {
 case 'value1':
    ...
 [break;]

 case 'value1':
    ...
 [break;]

 default:
    ...
 [break;]
}
```

✔ Używany jest operator ===

# Pytanie



- ✔ Czy zadziała `alert`?

```
if ("0") {
  alert( 'Hello, World!' );
}
```

- ✔ Zobacz

# Logiczne operatory



- ✔ Koniunkcja: `x && y ...`
  - ✘ zwraca pierwszą wartość, która się konwertuje do `false` (albo ostatnią wartość)
- ✔ Logiczna suma: `x || y ...`
  - ✘ zwraca pierwszą wartość, która się konwertuje do `true` (albo ostatnią wartość)
- ✔ Negacja: `! x`
  - ✘ wynik jest typu `Boolean`
  - ✘ Co wyświetlą `alerty`?

```
alert( null || 2 || undefined );
alert( null && 2 && undefined );
alert( null || 2 && 3 || 4 );
```

  - ✘ Zobacz 1, Zobacz 2, Zobacz 3

# Pętle



```
while (WARUNEK) {
  // kod
}

do {
  // kod
} while (WARUNEK);

for (INICJALIZACJA; WARUNEK; ZMIANA) {
  // kod
}
```

# Przerywanie pętli



- ✔ `break` — wyjście z pętli
- ✔ `continue` — przejście do nowej iteracji

# Funkcje

# Deklaracja funkcji



```
function nazwa(parametry) {
  ciało funkcji
}
```

✔ Przykładowo:

```
function showMessage() {
  alert( 'Hello, World!' );
}
```

# Zmienne lokalne, zewnętrzne i globalne



✔ Jak w innych językach programowania

```
let userName = 'Olek';
let timeOfDay = 'wieczór';

function greetings() {
  let userName = "Iwona";
  let message = 'Dobry ' + timeOfDay + ', ' + userName;
  alert(message);
}

greetings();
```

# Parametry



✔ Jak w przykładzie

```
function showMessage(from, text) {
  alert(from + ': ' + text);
}
showMessage('Ania', 'Witajcie wszyscy!');
```

✔ Parametry domyślne

```
function showMessage(from, text = 'Witajcie!') {
  alert( from + ": " + text );
}
showMessage('Ania');
```

✘ domyślnie: `undefined`

✘ może być wyrażenie:

```
... text = someFunction() ){
```

# Domyślne parametry w starym JavaScripcie



```
function showMessage(from, text) {
  if (text === undefined) {
    text = 'Witajcie';
  }
  ...
}
```

```
function showMessage(from, text) {
  text = text || 'Witajcie';
  ...
}
```

# Zwracana wartość



```
function sum(a, b) {
  return a + b;
}
```

- ✔ Bez `return` wartość `undefined`
- ✔ Pusty `return ;` — też wartość `undefined`
- ✔ Uwaga na średnik

```
return
(some + long + expression);
```

- ✘ poprawka:

```
return (
 some + long + expression
 );
```

# Funkcje lokalne



✔ Funkcje zadeklarowane w bloku, nie są dostępne poza blokiem

```
if (true) {
  sayHi(); // działa
  function sayHi() {
    alert("Cześć!");
  }
}

sayHi(); // błąd!
```

# Wyrażenia funkcyjne

# Deklaracja funkcji



- ✔ Funkcja jest obiektem
- ✔ Deklaracja tworzy funkcję i zapisuje na nią referencję

```
function sayHello(name) {
    alert("Hello "+name)
}
alert(sayHello)
```

- ✔ Zobacz

# Kopiowanie funkcji



- ✔ przy deklacji funkcji tworzy się zmienna, vartością której jest funkcja
- ✔ `func` nie jest nazwą funkcji, tylko nazwą zmiennej

```
function func() { alert(1); }
var g = func; // skopiowano
func = null;  // zmiana wartości
g(); // działa
func(); // nie działa (null() O_o?)
```

# Przed wykonaniem



- ✔ funkcja jest tworzona przed wykonaniem skryptu
- ✔ można wywołać przed deklaracją

```
sayHello("Bolek");

function sayHello(name) {
  alert("Hello, " + name);
}
```

# Warunkowa deklaracja funkcji



- ✔ nie jest możliwa

```
'use strict'
let age = prompt("Ole masz lat?", 18);

if (age >= 18) {
  function sayHi() {alert('Zapraszam!');  }
} else {
  function sayHi() {alert('Tylko od 18 roku'); }
}

sayHi();
```

- ✔ Zobacz

# Wyrażenie funkcyjne



```
let f = function(parametry) {
    // kod funkcji
};
```

✔ przykładowo:

```
let sayHi = function(person) {
    alert("Hello, " + person);
};

sayHi('Olek');
```

# W trakcie wykonania



- ✔ funkcja jest tworzona w trakcie wykonania skryptu
- ✔ nie można wywołać przed deklaracją
  - ✘ nie działa:

```
sayHello("Bolek");
let sayHello = function(name){
  alert("Hello, " + name);
}
```

  - ✘ tylko tak:

```
let sayHello = function(name){
  alert("Hello, " + name);
}
sayHello("Bolek");
```

# Warunkowa deklaracja funkcji



- ✔ Jest możliwa

```
'use strict'
let age = prompt("Ole masz lat?", 18);
let sayHi;

if (age >= 18) {
  sayHi = function(){alert('Zapraszam!');}
} else {
  sayHi = function(){alert('Tylko od 18 roku');}
}

sayHi();
```

- ✔ Zobacz

# Mianowane wyrażenie funkcyjne



- ✔ identyfikator, przywiązany do funkcji
- ✔ Named Function Expression

```
var f = function SayHi(a,b) { ... };
```

- ✔ `SayHi` jest widoczne tylko wewnątrz funcji
- ✔ pozwala na odwoływania się do samej funkcji

# Przykład. Silnia



- ✔ pierwsze podejście

```
function f(n) {
  return n ? n*f(n-1) : 1;
};

alert( f(5) ); // 120
```

- ✔ następujący kod nie działa:

```
function f(n) {
  return n ? n*f(n-1) : 1;
};
var g=f; f=null;

alert( g(5) );
```

# Silnia. Poprawka



```
var f = function factorial(n) {
  return n ? n*factorial(n-1) : 1;
};

var g = f;
f = null;

alert( g(5) ); // 120
```

# Running at place



- ✔ w przypadku *wyrażenia funkcyjnego* można wywołać funkcję natychmiast po definicji
- ✔ przykład: imlpementacja przestrzeni nazw

```
(function() {

  // zmienne lokalne
  var a = 1 , b = 2;
  // kod skryptu

})();
```

- ✔ po co nawias?
- ✔ funkcja anonimowa, nie można ponownie wywołać

# Running at place. Prykład



```
var res = function(a,b){ return a+b }(2,2);
alert(res); // 4
```

✔ nawias jest nie konieczny, ale zalecony (dobry styl)

```
var res = (function(a,b){return a+b })(2,2);
alert(res); // 4
```

# Funkcje *callback*i



- ✔ Funkcja może być przekazywana jako argument do innej funkcji

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}
function showOk() {
  alert( "Cieszę się" );
}
function showCancel() {
  alert( "Szkoda" );
}
ask("Are you sure?", showOk, showCancel);
```

# Z wykorzystaniem funkcji anonimowych



```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}

ask(
  "Are you sure?",
  function() { alert("Cieszę się"); },
  function() { alert("Szkoda"); }
);
```

# Funkcje strzałki

# Funkcje-strzałki



- ✔ ES6

```
let inc = x => x+1; // function(x){return x+1;}
alert( inc(1) ); // 2

let sum = (a,b) => a + b;

let getTime =
    () => new Date().getHours() + ':'
            + new Date().getMinutes()
```

# Nawias klamrowy



- ✔ Jeżeli ciało strzałki umieszczono w nawisie klamrowym, to koniecznie musi być `return`

```
let getTime = () => {
  let date = new Date();
  let hours = date.getHours();
  let minutes = date.getMinutes();
  return hours + ':' + minutes;
};

alert( getTime() ); // bieżący czas
```

# Wygodna notacja dla callbacków



```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}

ask(
  "Are you sure?",
  () => alert("Cieszę się"),
  () => alert("Szkoda")
);
```