Technologie Internetu. JavaScript

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

18 listopada 2018

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
  - ✘ 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).
- ✔ Czytać i ustawiać cookies, walidować dane, wyświetlać komunikaty
- ✔ i inne

Ograniczenia w przeglądarce

- ✔ Nie ma bezpośredniego dostępu do systemu plików i systemu operacyjnego
  - ✘ HTML5 application cache
- ✔ Nie ma dostępu do danych w innych oknach i kartach
  - ✘ chyba że skrypt sam otworzył to okno/kartę
  - ✘ specjalny kod w obu oknach
- ✔ Nie można wysłać zapytania na inny serwer (domena, port, protokół)

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
- ✔ Dart — ma własne środowisko uruchomieniowe, Google
- ✔ BabelJS — translacja nowych możliwości na ES5

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"
        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;

var a = "long
  string"
return
  result

var a = "long \
  string"
return result
```

✔ zawsze stawić

Komentarze i zmienne


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

- ✔ Zmienne są typowane dynamicznie
- ✔ W starym JavaScriptcie deklaracja zmiennych nie jest konieczna
  - ✘ tak nie robić

```
a = 1;
var b = 'String';
```

Bloki

✔ Bloki nie tworzą zmiannych lokalnych

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

Lokalne zmienne

✔ `var` robi zmienną lokalną wewnątzr funkcji

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

Zasięg zmiennych

✔ przy wejściu do funkcji tworzy się kopie lokalne wszystkich `var`

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

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
- ✔ zawsze korzystać (IE10+)

Zmienne `let`

- ✔ ES6
- ✔ Zmienna lokalna:
  1. Użycie tylko po deklaracji
  2. Zasięg: blok `{...}`
     - ✘ w szczególności, na każdej iteracji pętli jest osobna lokalna zmienna:

```
for(let i = 0; i<10; i++) { /* ... */ }
```

Stałe

- ✔ ES6

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

- ✔ W danych złożonych można zmieniać składowe:

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

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

Typy danych JavaScript

- ✔ Wszystkie dane są obiektami
- ✔ Funkcje są obiektami
- ✔ Proste stałe interpretuje się jako obiekty

```
'Olek'.length
```

- ✔ Jest różnica między protsymi stałymi a obiektami

```
alert(typeof "test") // string
alert(typeof new String("test")) // object
```

- ✔

```
console.log(typeof "test") // string
```

Number

- ✔ `float64` — błędy zaokrąglenia

```
alert(0.1+0.2)
    // 0.30000000000000004
```

- ✔ specjalne wartości:
  - ✘ `1/0=Number.POSITIVE_INFINITY`
  - ✘ `-1/0=Number.NEGATIVE_INFINITY`
  - ✘ `Number("something")=NaN`
    - ✓ `NaN + 1 = NaN`
    - ✓ `NaN == NaN // false`
    - ✓ `isNaN(NaN) // true`

Konwersja na Number


```
var str = "002"
var a = Number(str) // 2
+"0.1"  // => 0.1
+"0.1z" // => Number.NaN

parseFloat("0.1zf") = 0.1
parseInt("08f.4", 10) = 8
parseFloat("smth") = Number.NaN

parseInt("0x10") = 16
parseInt("010") = 8
parseInt("010",10) = 10
```

Funkcje matematyczne

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

String

- ✔ unicode
- ✔ cudzysłów ∼ apostrof
- ✔ wyrażenia regularne
- ✔ `match`, `replace`

```
"Jude hej".replace(/(.*?)\s(.*)/,"$2, $1!")
      // => hej, Jude!|
```

String w ES2015

- ✔ *Lewy* apostrof
  - ✘ stringi wielowierszowe

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

  - ✘ wyrażenia regularne

```
'use strict';
let apples = 2;
let oranges = 3;

alert(`${apples} + ${oranges}
   = ${apples + oranges}`);
// 2 + 3 = 5
```

Boolean

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

null

✔ Specjalny typ, jedna wartość

```
var age = null;
```

✘ age jest nieznany

undefined

- ✔ Specjalna typ, jedna wartość
  - ✘ warzość zmienej nie jest nadana

```
var x;
alert( x ); // "undefined"
```

  - ✘ można nadać w sposób jawny

```
var x = 123;
x = undefined;
```

Konwersja a porównywanie. Wartości specjalne

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

Konwersja a porównywanie. 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
  - ✘ `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

Uwagi

- ✔ Niektóre wersje Safari zwracają zamiast `null` pusty text (`''`)
- ✔ 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;]
}
```

Pytanie

- ✔ Czy zadziała `alert`?

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

- ✔ Zobacz

Logiczne operatory

- ✔ Koniunkcja: `x && y`
- ✔ Logiczna suma: `x || y`
- ✔ Negacja: `! x`

Pętle


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

do {
  // kod
} while (WARUNEK);

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

Przerywanie pętli

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

Tablice

Tablice sekwencyjne


```
arr = new Array("my", "array")
alert(arr.length)  // 2

arr = [ "my", "array" ]
alert(arr[0])

arr = ["one", "two"]
arr.push("three")

for(var i=0; i<arr.length; i++) {
  alert(arr[i])
}
```

Stos i Kolejka


```
var arr = [3,5,7]
arr.push(9)
var last = arr.pop()     //= 9
var last = arr.pop()    // = 7
alert(arr.length)    // = 2

var arr = [4,6,8]
arr.unshift(2) // arr = [2,4,6,8]
arr.unshift(0) // arr = [0,2,4,6,8]
var last = arr.shift()
      // last = 0, arr = [2,4,6,8]
arr.shift()  // arr = [4,6,8]
```

Tablice assocjacyjne


```
obj = {
  n: 1,
  str: "Olek"
}

alert(obj.n) // kropka
alert(obj["n"])
var key = "str"
alert(obj[key])


a = { }  // pusty obiekt
alert(a.something)  // undefined
alert(a.blabla === undefined)
```

Object.keys(obj)

```
var user = {
  name: "Piotr",
  age: 30
}

var keys = Object.keys(user);

alert(keys); // name, age
```

✔ IE9+

Metody tablicsplit


```
var names = 'Olek, Bolek, Lolek';

var arr = names.split(', ');

for (var i=0; i<arr.length; i++) {
  doSomething(arr[i]);
}
```

- ✔ drugi argument — wielkość tablicy

```
alert( "a,b,c,d".split(',', 2) ); // a,b
```

- ✔ rozbić na litery:

```
var str = "test";
alert( str.split('') ); // t,e,s,t
```

join

```
var arr = ['Olek', 'Lolek', 'Bole'];

var str = arr.join(';');

alert(str); // Olek;Lolek;Bolek
```

Usuwanie elementu — `delete`


```
var arr = ['Olek', 'Lolek', 'Bole'];

delete arr[1];

alert(arr[1]); // undefined
```

- ✔ usuwa się para `klucz => wartość`
- ✔ inne metody:
  - ✘ `pop`, `shift`
  - ✘ `splice`

Usuwanie elementu — splice


```
arr.splice(index[, deleteCount, el1, ..., elN])
```

- ✔ usunąć `deleteCount` elementów
- ✔ poczynając od `index`
- ✔ wstawić na ich miejsce `el1, ..., elN`

splice. Przykłady


```
arr.splice(index[, deleteCount, el1, ..., elN])
```

- ✔ usunąć jeden element, poczynając od miejsca 1
  `arr.splice(1, 1);`
- ✔ usunąć przedostatni element `arr.splice(-2, 1);`
- ✔ usunąć wszyskie elementy, poczynając od miejsca 2
  `arr.splice(1);`
- ✔ usunąć trzy elementy, na ich miejsce wstawić dwa inne
  `arr.splice(0,3,"raz","dwa");`
- ✔ wstawić dwa elementy `arr.splice(3,0,"raz","dwa");`
- ✔ elementy usunięte `removed=arr.splice(-3,2);`

Kopiowanie elementów — `slice`


```
arr.slice(start, end);
```

- ✔ pierwotna tablica się nie zmienia
- ✔ `end` się nie zalicza
- ✔ `start` może być ujemnym
- ✔ bez `end` kopiowanie do końca tablicy
- ✔ bez `start` kopiowanie całej tablicy

Sortowanie — sort


```
var arr = [ 1, 2, 15 ];

arr.sort();

alert( arr );  // 1, 15, 2
```

Własna funkcja porównująca


```
function compareNumeric(a, b) {
  if (a > b) return 1;
  if (a < b) return -1;
}
var arr = [ 1, 2, 15 ];

arr.sort(compareNumeric);
alert(arr);  // 1, 2, 15
```

- ✔ funkcja porównywania $f(a, b)$ zwraca wartość
  - ✘ dodatnią, jeżeli $a > b$
  - ✘ ujemną, jeżeli $a < b$
  - ✘ inną, jeżeli $a = b$

Obrócenie tablicy — reverse


```
var arr = [1,2,3];
arr.reverse();

alert(arr); // 3,2,1
```

Połączenie tablic — concat


```
var arr = [1,2];
var newArr = arr.concat(3,4);

alert(newArr); // 1,2,3,4

var arr = [1,2];
var newArr = arr.concat( [3,4], 5);

alert(newArr); // 1,2,3,4,5
```

Wyszukiwanie elementów — `indexOf/lastIndexOf`


```
arr.indexOf(searchElement[, fromIndex])
```

- ✔ IE9+
- ✔ Zwraca numer elementu `searchElement` w tablicy, albo -1, jeżeli takiego elementu nie ma
- ✔ Przy wyszukiwaniu używany jest operator identyczności (`===`)
- ✔ Jeżeli dany jest `fromIndex`, to wyszukiwanie zaczyna się od tego miejsca

```
arr.lastIndexOf(searchElement[, fromIndex])
```

- ✔ Szuka od końca tablicy

Iteratory tablic

forEach

- ✔ `arr.forEach(callback[, thisArg])`
- ✔ odpala na dażdym elemencie `callback(item, i, arr)`
  - ✘ `item` jest bieżącym elementem
  - ✘ `i` jest indeksem
  - ✘ `arr` jest całą tablicą
- ✔ `thisArg` pozwala odpalić iterator w innym kontekście
- ✔ wydrukować tablicę:

```
function logArrayElements(element, index, array) {
  console.log('a[' + index + '] = ' + element);
}

[2, 5, , 9].forEach(logArrayElements);
```

filter

- ✔ `arr.filter(callback[, thisArg])`
- ✔ Tworzy nową tablicę z elementów, na których `callback(item, i, arr)` zwróci `true`
  - ✘ `item` jest bieżącym elementem
  - ✘ `i` jest indeksem
  - ✘ `arr` jest całą tablicą
- ✔ `thisArg` pozwala odpalić iterator w innym kontekście
- ✔ tablica z liczb dodatnich:

```
var arr = [1, -1, 2, -2, 3];
var positiveArr = arr.filter(function(number) {
  return number > 0;
});
alert( positiveArr ); // 1,2,3
```

map

- ✔ `arr.map(callback[, thisArg])`
- ✔ Tworzy nową tablicę — wyniki odpalania `callback(item, i, arr)` na każdym elemencie
  - ✘ `item` jest bieżącym elementem
  - ✘ `i` jest indeksem
  - ✘ `arr` jest całą tablicą
- ✔ `thisArg` pozwala odpalić iterator w innym kontekście
- ✔ Długości łańcuchów tekstowych:

```
var names = ['HTML', 'CSS', 'JavaScript'];
var nameLengths = names.map(function(name) {
  return name.length;
});
alert( nameLengths ); // 4,3,10
```

every/some

- ✔ `arr.every(callback[, thisArg])` zwraca `true`, jeżeli `callback` zwróci `true` dla każdego elementu
- ✔ `arr.some(callback[, thisArg])`zwraca `true`, jeżeli `callback` zwróci `true` dla co najmniej jednego elementu
- ✔ Pzykład:

```
var arr = [1, -1, 2, -2, 3];

function isPositive(number) {
  return number > 0;
}

alert( arr.every(isPositive) ); // false
alert( arr.some(isPositive) ); // true
```

reduce/reduceRight

- ✔ `arr.reduce(callback[, initialValue])` odpala `callback(previousValue, currentItem, i, arr)` na każdym elemencie
  - ✘ `previousValue` wynikiem obliczenia na poprzedniej iteracji
  - ✘ `item` jest bieżącym elementem
  - ✘ `i` jest indeksem
  - ✘ `arr` jest całą tablicą
- ✔ Jeżeli nie ma `initialValue`, to iteracje się zaczynają od drugiego elementu, a pierwszy jest wykorzystany jako `previousValue`
- ✔ `arr.reduceRigh(callback[, initialValue])` — analogicznie, tylko z prawej strony

Przykład z `reduce`

- ✔ Suma elementów:

```
var arr = [1, 2, 3, 4, 5]
var result = arr.reduce(function(sum, current) {
  return sum + current;
}, 0);
alert( result ); // 15
```

- ✔ Co będzie bez `initialValue`?

```
var arr = [1, 2, 3, 4, 5]
var result = arr.reduce(function(sum, current) {
  return sum + current;
});
alert( result ); // ?
```

Obiekty

Model obiektowy JavaScript

- ✔ Obiekt == Tablica assocjacyjna == Hash
- ✔ Przechowuje dowolne pary `klucz => vartość`
- ✔ Metoda obiekta — jest *funkcją*, dodaną do tablicy

Podstawy obiektów

- ✔ Utworzenie obiektu

```
var o = new Object()
var o = {}
```

- ✔ Dodanie właściwości

```
o.test = 5
o["test"] = 5

var name = 'test'
o[name] = 5
```

Właściwości obiektów

- ✔ Dostęp do właściwości

```
alert(o.test)
alert(o['test'])
var o = {}
alert(o.nosuchkey)    // => undefined
```

- ✔ Sprawdzanie właściwości

```
if (typeof x != 'undefined')
if (window.x) { ... }
if (window.x !== undefined)
```

- ✔ Usuwanie właściwości

```
o.test = 5
delete o.test
```

Metody obiektów

✔ Dodanie metody

```
var rabbit = {}
rabbit.run = function(n) {
    alert("Ran "+n+" miles!")
}

rabbit.run(10)
```

Wyliczenie właściwości


```
for(var key in object) {
  // key - nazwa pola
  // object[key] - wartosc
  ...
}
```

✔ Z filtracją właściwości prototypu

```
for(prop in object)
      if (object.hasOwnProperty(prop)) {
    //...
}
```

this

- ✔ W konstruktorze: tworzony obiekt

```
function Animal(name) {
    this.name = name
}
Cat = new Animal('Tygrys');
```

- ✔ W metodzie obiektu: obiekt

```
kot= { name : 'Tygrys'}
kotka= { name : 'Latka'}
say= function{alert('Jestem '+this.name)}
kot.sayHi=say;
kotka.sayHi=say;
kot.sayHi(); // Tygrys
kotka.sayHi(); // Latka
kotka['sayHi']()
```

this w bezpośrednim wywolaniu

- ✔ w starym standardzie

```
function func() {
  alert(this); // [object Window]
               // lub [object global]
}

func();
```

- ✔ współczesny standard (IE⩾10)

```
function func() {
  "use strict";
  alert(this); // undefined
}

func();
```

this w call i apply


```
var user = {
  firstName: "Aleksander",
  lastName: "Kowalski"
};

function showName() {
  alert( this.firstName + ' ' + this.lastName );
}

showName.call(user)  // "Aleksander Kowalski"

func.apply(context, [arg1, arg2 ... ]);
```

Zapożyczenie metody


```
function sayHi() {
  arguments.join = [].join;
  var argStr = arguments.join(':');
  alert(argStr);  // 1:2:3
}

sayHi(1, 2, 3);
```

✔ Bez kopiowania do `arguments`

```
function sayHi() {
  var join = [].join;
  var argStr = join.call(arguments, ':');
  alert(argStr);  // 1:2:3
}
```

1 2 3

Konstruktor

✔ Każda funkcja może stworzyć obiekt

```
function Animal(name) {
    this.name = name
    this.canWalk = true
}

var animal = new Animal("Tygrys")
alert(animal instanceof Animal)
        // true
```

Dziedziczenie prototypowe

- ✔ Obiekty dziedziczą bezpośredno od obiektów (omijając klasy)
- ✔ Ukryta referencja `[[prototype]]` wskazuje, jaki obiekt jest prototypem
- ✔ Dostęp do prototypu przez właściwość `prototype`
  - ✘ Referencja na prototyp jest tworzona operatorem `new` podczas tworzenia obiektu.
  - ✘ Wartością referencji będzie wartość właściwości `prototype` funkcji-konstruktora.
  - ✘ Wartość `prototype` określa, od jakiego obiektu będą dziedziczyć nowe obiekty
  - ✘ Jeżeli obiekt nie ma pewnej właściwości, jej się szuka w prototypie

Przykład dziedziczenia


```
function Animal(name) {
    this.name = name
    this.canWalk = true
}
var animal=new Animal('Tygrys')

function Rabbit(name) {
    this.name = name
}

Rabbit.prototype = animal;
```

Przykład dziedziczenia, cd


```
big = new Rabbit('Chuk')
small = new Rabbit('Bunks')

alert(big.name)  // Chuk
alert(small.name) // Bunks

alert(big.canWalk) // true

animal.canWalk = false

alert(big.canWalk)  // false
alert(small.canWalk)  // false
```

Przykrycie


```
animal.canWalk = false

small.canWalk = true

alert(big.canWalk)   // false
alert(small.canWalk)   // true
```

Łańcuch dziedziczenia

- ✔ W podstawie zawsze jest obiekt `Object`
- ✔ Domyślnym prototypem jest obiekt `Object`

Dynamiczne dodanie metody


```
Animal.prototype.move = function(n) {
    this.distance = n
    alert(this.distance)
}


function Animal(n) {
    .....
    this.move = function(n) {
        this.distance = n
        alert(this.distance)
    }
}
```

Przykład statycznych danych


```
function Hamster() {  }
Hamster.prototype = {
    food: [],
    found: function(something) {
        this.food.push(something)
    }
}
speedy = new Hamster()
lazy = new Hamster()

speedy.found("apple")
speedy.found("orange")

alert(speedy.food.length) // 2
alert(lazy.food.length) // 2 (!??)
```

Poprawka przykładu


```
function Hamster() {
    this.food = []
}
Hamster.prototype = {
    food: [],
    found: function(something) {
        this.food.push(something)
    }
}
```

Funkcje

Deklaracja funkcji

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

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

Funkcja jako obiekt

✔ jak do każdego obiektu, do funkcji można dodać właściwość

```
sayHello.Counter=5;
alert(sayHello.Counter);
```

Kopiowanie funkcji


```
sayHi = sayHello;
alert(sayHi);
sayHi('Lolek');
sayHello('Bolek');
alert(sayHello()); // ?
```

Function Declaration


```
function func(){alert(1);}
```

- ✔ 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

```
var age = 20;

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

sayHi();
```

Function Expression


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

✔ przykładowo:

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

  - ✘ tylko tak:

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

Warunkowa deklaracja funkcji

✔ jest możliwa (nawet zalecona)

```
var age = 20;
var sayHi;

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

sayHi();
```

Running at place

- ✔ w przypadku *Function Expression* 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
```

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
- ✔ przestarzała opcja `arguments.callee` jest wyłączona ze standardu

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
```

Lista argumentów


```
function func(a,b) {
  alert(arguments[0])
  alert(arguments[1])
  alert(arguments[2])
}
func(1,2,3)

function sum() {
  var s = 0
  for(var i=0; i<arguments.length; i++) {
      s += arguments[i]
  }
  return s
}
```

Argumenty domyślne

- ✔ ES6

```
function showMenu(title = "Brak",
                  width = 100, height = 200) {
  alert(title + ' ' + width + ' ' + height);
}

showMenu("Menu"); // Menu 100 200
```

- ✔ Domyślne argumenty dla `undefined`

```
showMenu(undefined, null); // Brak null 200
```

Właściwość `name`

- ✔ ES6
- ✔ Nazwa funkcji

```
function f() {} // f.name == "f"
let g = function g() {}; // g.name == "g"
```

- ✔ Nawet dla funkcji anonimowych

```
let g = function () {}; // g.name == "g"

let user = {
  // user.sayHi.name == "sayHi"
  sayHi: function() {}
};
```

Funkcje lokalne

- ✔ ES6
- ✔ Funkcje zadeklarowane w bloku, nie są dostępne poza blokiem

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

sayHi(); // błąd!
```

Funkcje-strzałki

✔ ES6

```
let inc = x => 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


```
let arr = [5, 8, 3];

let sorted = arr.sort( (a,b) => a - b );

alert(sorted); // 3, 5, 8
```

✔ strzałki wykorzystują `arguments` i `this` funkcji zewnętrznej

```
function f() {
  let showArg = () => alert(arguments[0]);
  showArg();
}

f(1); // 1
```

Domknęcia

Funkcje


```
function outer() {
    var outerVar;

    var func = function() {
        var innerVar
        ...
        x = innerVar + outerVar
    }
    return func
}
```

Przykład


```
function addHideHandler(sourceId, targetId){
 var sourceNode =
        document.getElementById(sourceId)
  var handler = function() {
   var targetNode =
        document.getElementById(targetId)
        targetNode.style.display = 'none'
    }
    sourceNode.onclick = handler
}

addHideHandler("clickToHide", "info")
```

Mechanizm domknięć —I

- ✔ Każde wywołanie funkcji przechowuje wszystkie zmiennne w obiekcie `[[scope]]`
- ✔ Obiekt `[[scope]]` nie jest dostępny dla programisty
- ✔ Każde wywołanie `var` tworzy nową właściwość tego obiektu
- ✔ Każda użyta zmienna jest wyszukiwana w obiekcie `[[scope]]`
- ✔ Każda zmiana zmiennej lokalnej odpowiada zmianie właściwości obiektu `[[scope]]`
- ✔ Po zakończeniu funkcji obiekt `[[scope]]` jest niszczony

Mechanizm domknięć —II


```
function sum(x,y) {
  // utworzony obiekt [[scope]]
  ...
  // do [[scope]] dodane z
  var z
  // znalezione, [[scope]].z = x+y
  z = x+y
  // znalezione, return [[scope]].z
  return z

  // koniec funkcji
  // [[scope]] jest niszczony razem z z
}
```

Mechanizm domknięć —III

- ✔ Funkcja włożona ma referencję na obiekt `[[scope]]` funkcji zewnętrznej
- ✔ Każda użyta zmienna jest wyszukiwana w swoim obiekcie `[[scope]]`, potem w obiekcie `[[scope]]` funkcji nadrzędnej, etc
- ✔ Ponieważ wewnętrzna funkcja możet zostać wywołana później, referencja oraz sam obiekt `[[scope]]` funkcji nadrzędnej zostaje
- ✔ Funkcja wewnętrzna w każdym momenci ma do niego dostęp

Mechanizm domknięć —IV


```
function addHideHandler(sourceId, targetId){
    // nowy [[scope]], sourceId, targetId
    // do [[scope]] dodane sourceNode
    var sourceNode =
      document.getElementById(sourceId)
    // do [[scope]] dodane handler
    var handler = function() {
        var targetNode =
          document.getElementById(targetId)
        targetNode.style.display = 'none'
    }
    sourceNode.onclick = handler
    // zostaje referencja!!!
}
```

Przykład 2


```
function makeShout() {
    var phrase = "Hi!"

    var shout = function() {
        alert(phrase)
    }

    phrase = "There!"

    return shout
}

shout = makeShout()
// jaki komunikat?
shout()
```

Przykład 3


```
function sum(a) {
  return function(b) {
    return a+b
  }
}
 sum(1)(3) //4
```

Przykład 4 (błędny)


```
function addEvents(divs) {
    for(var i=0; i<divs.length; i++) {
        divs[i].innerHTML = i
        divs[i].onclick = function() {
                alert(i) }
    }
}
```

- ✔ wszystkie div'y wyświelają ten sam komunikat
- ✔ jaki?
- ✔ czemu?

Przykład 4 (poprawiony)


```
function getHandler(x){
    return function() { alert(x) }
}

function addEvents2(divs) {
    for(var i=0; i<divs.length; i++) {
        divs[i].innerHTML = i
        divs[i].onclick = getHandler(i)
    }
}
```

Przykład 4 (inaczej)


```
function addEvents2(divs) {
    for(var i=0; i<divs.length; i++) {
        divs[i].innerHTML = i
        divs[i].onclick = (function(x) {
            return function() { alert(x) }
        } ) (i)
    }
}
```

Wyjątki

Wyjątki


```
try {
  ...
  throw {message : "server timeout"}
  ..
} catch (e) {
  alert("File not found")
}
finally{
  zawsze();
}
```

- ✔ blok `finally` jest opcjonalny
- ✔ w `throw` może być dowolny obiekt

Obiekt Error


```
try {
  ...
  throw new Error('connection refused')
  ..
} catch (err) {
  alert(err.message)
}
```

✔ W FF `err.stack`

Obiekt Error


```
ConnError.prototype=Error;
try {
  ...
  throw new ConnError('connection refused')
  ..
} catch (err) {
  if(err instanceof ConnError){
    reconnect();
  }
  else{
    alert(err.message)
  }
}
```

JSON

Format JSON

- ✔ RFC 4627
- ✔ format dla danych w oparciu o JavaScript
- ✔ zpisuje się jako obiekt $\{\cdots\}$ lub tablica `[···]`, które zawierają
  - ✘ tekst w cudzysłowie (nie w apostrofie)
  - ✘ liczby
  - ✘ stałe `true`/`false`
  - ✘ `null`

Przykład JSON


```
{
  "name": "Aleksander",
  "surname": "Kowalski",
  "age": 35,
  "isAdmin": false
}
```

Metody `stringify` oraz `parse`

- ✔ *serializacja* —

  `JSON.stringify(value, replacer, space)`

  zamienia wartość w tekst (IE8+)
- ✔

  `JSON.parse(value, reviver)`

  zamienia tekst w obiekt JavaScript

stringify oraz parse. Przykład


```
var event = {
  title: "Conference",
  date: "Today"
};

var str = JSON.stringify(event);
alert( str ); // {"title":"Conference",
              // "date":"Today"}

event = JSON.parse(str);
```

Serializacja `toJSON`

- ✔ Przy serializacji obiektu jest wywołana metoda `toJSON`.
- ✔ Jeżeli nie ma takiej metody, to wymienia się wszystkie właściwości, oprócz funkcji

```
function Room() {
    this.number = 103;
    this.occupy = function() {};
}
event = {
  title: "Confertence",
  date: new Date(2014, 3, 6),
  room: new Room()
}; alert( JSON.stringify(event) );
```

- ✔ Zobacz

toJSON

- ✔ Iny przykład

```
function Room() {
    this.number = 103;

 this.toJSON = function() {
     return this.number;
  };

}

alert( JSON.stringify( new Room() ) );
```

- ✔ Zobacz

JSON a elementy DOM

- ✔ Elementy DOM — to są skomplikowane obiekty, które mają cykliczne referencje

```
var user = {
  name: "Aleksander",
  age: 25,
  elem: document.body
}

alert( JSON.stringify(user) ); //
```

- ✔ Zobacz

Wyłączenie właściwości w `stringify`

- ✔ W drugim argumencie (`replacer`) funkcji `stringify` można wskazać tablicę właściwości, które należy serializować

```
var user = {
  name: "Aleksander",
  age: 25,
  elem: document.body
}

alert( JSON.stringify(user,["name", "age"]));
```

- ✔ Zobacz

Funkcja jako `replacer`

- ✔ Jako `replacer` można użyć funkcji

  `function(key, value)`,

  która dla klucza `key` zwraca:

  - ✘ serializowane `value`
  - ✘ `undefined`, jeżeli nie trzeba tej właściwości serializować
- ✔ funkcja `replacer` działa rekurencyjnie

Funkcja replacer — przykład


```
var user = {
  name: "Aleksander",
  age: 25,
  elem: document.body
}
var str = JSON.stringify(user, function(key, value)
  if (key == 'elem') return undefined;
  return value;
} );

alert(str);
```

✔ Zobacz

Formatowanie serializacji

- ✔ Trzeci argument funcji `stringify` określa formatowanie
  - ✘ liczba: poziomy włożoności formatuje się podaną ilością spacji
  - ✘ tekst: na każdym poziomie wstawia się ten tekst

```
var user = {
  name: "Aleksander",
  age: 25,
  roles: {isAdmin: false, isEditor: true}
};
var str = JSON.stringify(user, "", 4);
alert(str);
```

- ✔ Zobacz

Pasrowanie

- ✔ Przykład

```
var str = '{ \
  "title":"Conference", \
  "date":"2014-03-06T00:00:00.000Z" \
  }  ';

var event = JSON.parse(str);

alert( event.date.getDate() )
```

- ✔ jaki wynik?
- ✔ Zobacz

Parsowanie intelektualne

- ✔ Drugi argument `reviver` funcji `parse` jest funkcją

  `function (key, value)`,

  która zwraca:

  - ✘ przekształcony parametr `value`
  - ✘ `undefined`, jeżeli `value` nie trzeba zmieniać

```
var event = JSON.parse(str,
 function(key, value) {
  if (key == 'date') return new Date(value);
  return value;
})
```

- ✔ Zobacz

Parsowanie za pomocą `eval`

- ✔ Jest starą metodą (IE7-)

```
var str= '{ \
    "name": "Aleksander", \
    "age": 25 \
}';
var user = eval('(' + str + ')');
alert(user.name)
```

- ✔ niebezpieczeństwo — można przekazać dowolny skrypt
- ✔ lepiej unikać