Bogate Interfejsy Użytkownika JavaScript

Łukasz Rybka · Gdańsk 2015

O mnie

Z WYKSZTAŁCENIA

Z WYKSZTAŁCENIA

fizyk

O mnie

Z WYKSZTAŁCENIA

fizyk

Z ZAWODU

O mnie

Z WYKSZTAŁCENIA

fizyk

Z ZAWODU

projektant-programista / kontrybutor Open Source / freelancer

O mnie

Z WYKSZTAŁCENIA

fizyk

Z ZAWODU

projektant-programista / kontrybutor Open Source / freelancer

Z ZAMIŁOWANIA

O mnie

Z WYKSZTAŁCENIA

fizyk

Z ZAWODU

projektant-programista / kontrybutor Open Source / freelancer

Z ZAMIŁOWANIA

wykładowca / prelegent / szkoleniowiec

Wykład - materiały

Literatura:

http://smoczysko.github.io/#/courses/rui

Wykład - materiały

Literatura:

David Flanagan "JavaScript: The Definitive Guide"

http://smoczysko.github.io/#/courses/rui

Wykład - materiały

Literatura:

David Flanagan "JavaScript: The Definitive Guide"

Douglas Crockford "JavaScript: The Good Parts"

http://smoczysko.github.io/#/courses/rui

Wykład - materiały

Literatura:

David Flanagan "JavaScript: The Definitive Guide"

Douglas Crockford "JavaScript: The Good Parts"

LinkedIn JavaScript Bootcamp

http://smoczysko.github.io/#/courses/rui

Wykład - materiały

Literatura:

David Flanagan "JavaScript: The Definitive Guide"

Douglas Crockford "JavaScript: The Good Parts"

LinkedIn JavaScript Bootcamp

Mozilla Developer Network

http://smoczysko.github.io/#/courses/rui

Ustalenia

Ustalenia

Pytania mile widziane!

Pytania mile widziane!

Konsultacje po zajęciach - informacja odpowiednio przed (np. mailowo)

Pytania mile widziane!

Konsultacje po zajęciach - informacja odpowiednio przed (np. mailowo)

Slajdy to tak naprawdę ściągawka dla wykładowcy ;)

Ustalenia

Pytania mile widziane!

Konsultacje po zajęciach - informacja odpowiednio przed (np. mailowo)

Slajdy to tak naprawdę ściągawka dla wykładowcy ;)

Wykład nie jest obowiązkowy . . .

Organizacja zajęć

Organizacja zajęć

4 wykłady po 2 godziny (bez przerwy)

Organizacja zajęć

4 wykłady po 2 godziny (bez przerwy)

4 laboratoria po 3 godziny (bez przerwy)

Tematyka zajęć

Tematyka zajęć

JavaScript library tips & tricks

Tematyka zajęć

JavaScript library tips & tricks

Tworzenie pluginu biblioteki jQuery

Tematyka zajęć

JavaScript library tips & tricks

Tworzenie pluginu biblioteki jQuery

Underscore.js

Tematyka zajęć

JavaScript library tips & tricks

Tworzenie pluginu biblioteki jQuery

Underscore.js

Modularność w JavaScript: system.js

Scope

> (…) an association of a name to an entity (…) is the part of a computer program where the binding is valid: where the name can be used to refer to the entity

Scope

Wyróżniamy dwa rodzaje zasięgu w JavaScript:

Scope

Wyróżniamy dwa rodzaje zasięgu w JavaScript:

globalny

Scope

Wyróżniamy dwa rodzaje zasięgu w JavaScript:

globalny

lokalny

Zasięg globalny

```
<script type="text/javascript">
    var monkey = "Gorilla";

    function greetVisitor () {
        return alert("Hello dear blog reader!");
    }

    console.log(monkey);
    console.log(greetVisitor());
</script>
```

Zasięg lokalny

```
<script type="text/javascript">
    function greetVisitor(name) {
        var message = "Hello " + name + "!";
        console.log(message);
    }

    greetVisitor("John");1

    console.log(name);2
    console.log(message);3
</script>
```

1. Wypisze "Hello John!"
2. Wypisze undefined - dostęp do argumentów posiada wyłącznie funkcja
3. Wypisze undefined - message jest zmienną lokalną wewnętrznej funkcji

Zasięg funkcji wewnętrznej

```
<script type="text/javascript">
    var name = "John";

    function greetVisitor() {
        var message = "Hello " + name + "!";
        console.log(message);
    }

    greetVisitor();1

    console.log(name);2
</script>
```

1. Wypisze "Hello John!"
2. Wypisze "John"

Zasięg funkcji wewnątrz funkcji

```
<script type="text/javascript">
    function square(x) {
        function logSquare() {
            var value = x * x;
            console.log('Square value of ' + x + ' is ' + value);
        }

        console.log(value);
        logSquare();
    }

    console.log(square(2));
    console.log(square(3));
</script>
```

Closure

```
<script type="text/javascript">
    function makeAdder(x) {
        return function(y) {
            return x + y;
        };
    }

    var add5 = makeAdder(5);
    var add10 = makeAdder(10);

    console.log(add5(2)); // ??
    console.log(add10(2)); // ??
</script>
```

Zastosowania domknięć w JavaScript

Zamiennik obiektów z jedną metodą (zamiast np. bind'owania)

Zastosowania domknięć w JavaScript

Zamiennik obiektów z jedną metodą (zamiast np. bind'owania)

Przekazywanie stałych wartości do event handlerów

Zastosowania domknięć w JavaScript

- Zamiennik obiektów z jedną metodą (zamiast np. bind'owania)
- Przekazywanie stałych wartości do event handlerów
- Prostsze rozszerzanie prototypu

Zastosowania domknięć w JavaScript

- Zamiennik obiektów z jedną metodą (zamiast np. bind'owania)
- Przekazywanie stałych wartości do event handlerów
- Prostsze rozszerzanie prototypu
- Ukrywanie roboczych zmiennych i funkcji skryptu/biblioteki

The Infamous Loop Problem

```
<script type="text/javascript">
    function addLinks () {
        for (var i=0, link; i<5; i++) {
                link = document.createElement("a");
                link.innerHTML = "Link " + i;
                link.onclick = function () {
                        alert(i);
                };
                document.body.appendChild(link);
        }

    window.onload = addLinks;
</script>
```

The Infamous Loop Problem - rozwiązanie

```
<script type="text/javascript">
    function addLinks () {
        for (var i=0, link; i<5; i++) {
                link = document.createElement("a");
            link.innerHTML = "Link " + i;

            link.onclick = function (num) {1
                return function () {2
                    alert(num);3
                };
            }(i);1

            document.body.appendChild(link);
        }
    }

    window.onload = addLinks;
</script>
```

1. Definicja funkcji, która tworzy domknięcie i przekazanie parametru
2. Faktyczna funkcja wywoływana na onclick
3. Wyświetlenie liczby "zamrożonej" w kontekście domknięcia

Yahoo JavaScript Module Pattern

```
<script type="text/javascript">
    var person = function () {
        var name = "Robert";
        return {
                getName : function () {
                        return name;
                },
                setName : function (newName) {
                        name = newName;
                }
        };
    }();

    console.log(person.name);
    console.log(person.getName());
    person.setName("Robert Nyman");
    console.log(person.getName());
</script>
```

Asynchronous Loop Problem

```
<script type="text/javascript">
    for (var i = 0; i < 4; i++){
        setTimeout(function(){
            console.log(i);
        }, 1000);
    }
</script>
```

Immediately-Invoked Function Expression

Immediately-Invoked Function Expression

Inaczej nazywane "IIFE" lub "Immediate Execution Function"

Immediately-Invoked Function Expression

Inaczej nazywane "IIFE" lub "Immediate Execution Function"

Każde wywołanie funkcji tworzy domknięcie - także IIFE

Immediately-Invoked Function Expression

Inaczej nazywane "IIFE" lub "Immediate Execution Function"

Każde wywołanie funkcji tworzy domknięcie - także IIFE

Wyrażenie funkcyjne zamiast tradycyjnej deklaracji

Immediately-Invoked Function Expression

Inaczej nazywane "IIFE" lub "Immediate Execution Function"

Każde wywołanie funkcji tworzy domknięcie - także IIFE

Wyrażenie funkcyjne zamiast tradycyjnej deklaracji

Tworzy własny, funkcyjny scope

IIFE - sposób zapisu

```
<script type="text/javascript">
    // Notacja według Crockford'a
    (function(){
        /* code */
    }());

    // Poniższa notacja jest równoznaczna tej wyżej - kwestia "gustu"
    (function(){
        /* code */
    })();
</script>
```

Asynchronous Loop Problem - rozwiązanie

```
<script type="text/javascript">
    for (var i = 0; i < 4; i++){
        (function (num) {
            setTimeout(function(){
                console.log(num);
            }, 1000);
        }(i));
    }
</script>
```

Hoisting

Hoisting

hoist - "raise (something) by means of ropes and pulleys."

Hoisting

hoist - "raise (something) by means of ropes and pulleys."

Logiczna interpretacja kodu tłumacząca działanie skryptu

Hoisting

hoist - "raise (something) by means of ropes and pulleys."
Logiczna interpretacja kodu tłumacząca działanie skryptu

Interpreter nie modyfikuje naszego kodu !

> **var statements** and **function** declarations are (conceptually) moved to the top of their enclosing scope.
>
> — LinkedIn JavaScript Bootcamp

Hoisting - deklaracji zmiennych

Hoisting - deklaracji zmiennych

Deklaracja zmiennej podlega hoistingowi,
przypisanie wartości nie

Hoisting - deklaracji zmiennych

Deklaracja zmiennej podlega hoistingowi, przypisanie wartości nie

Zmienne otrzymują domyślną wartość undefined

Hoisting - deklaracji zmiennych

Deklaracja zmiennej podlega hoistingowi, przypisanie wartości nie

Zmienne otrzymują domyślną wartość undefined

Zmienna jest dostępna w całym scope, w którym została zadeklarowana

Hoisting - pozostałe zasady

Hoisting - pozostałe zasady

Przypisanie wartości zmiennej pozostaje bez zmian

Hoisting - pozostałe zasady

Przypisanie wartości zmiennej pozostaje bez zmian

W pierwszej kolejności hoistingowi podlegają deklaracje zmiennych, w drugiej deklaracje funkcji

Hoisting - przykład #1

```
(function () {
    var foo = 1;

    console.log(foo);

    function inner() {
        console.log(foo);

        foo = 3;

        console.log(foo);
    }

    inner();

    console.log(foo);
})();
```

Hoisting - przykład #1

```
(function () {
    var foo = undefined;

    function inner() {
        console.log(foo);

        foo = 3;

        console.log(foo);
    }

    foo = 1;

    console.log(foo);

    inner();

    console.log(foo);
})();
```

Hoisting - przykład #2

```
(function () {
    var foo = 1;

    console.log(foo);

    function inner() {
        console.log(foo);
        var foo = 3;
        console.log(foo);
    }

    inner();

    console.log(foo);
})();
```

Hoisting - przykład #2

```
(function () {
    var foo = undefined;

    function inner() {
        var foo = undefined;
        console.log(foo);
        foo = 3;
        console.log(foo);
    }

    foo = 1;

    console.log(foo);

    inner();

    console.log(foo);
})();
```

Hoisting - przykład #3

```
(function () {
    var foo = 1;

    function inner() {
        if (!foo) {
            var foo = 10;
        }

        console.log(foo);
    }

    inner();

    console.log(foo);
})();
```

Hoisting - przykład #3

```
(function () {
    var foo = undefined;

    function inner() {
        var foo = undefined;

        if (!foo) {
            foo = 10;
        }

        console.log(foo);
    }

    foo = 1;

    inner();

    console.log(foo);
})();
```

Hoisting - przykład #4

```
(function () {
    foo();

    return;

    function foo() {
        console.log("Hello ;)");
    }
})();
```

Hoisting - przykład #4

```
(function () {
    function foo() {
        console.log("Hello ;)");
    }

    foo();

    return;
})();
```

Prototype - czyli dlaczego to wszystko ma znaczenie?

Prototype - czyli dlaczego to wszystko ma znaczenie?

Biblioteka stworzona w 2005 roku

Prototype - czyli dlaczego to wszystko ma znaczenie?

Biblioteka stworzona w 2005 roku

„Zakazana” w użyciu produkcyjnym (mimo wszystko ma ~2% wykorzystania rynkowego, głównie legacy systems)

Prototype - czyli dlaczego to wszystko ma znaczenie?

Biblioteka stworzona w 2005 roku

„Zakazana” w użyciu produkcyjnym (mimo wszystko ma ~2% wykorzystania rynkowego, głównie legacy systems)

Bardzo dobra baza przydatnego kodu!

Prototype - antywzorce

Prototype - antywzorce

Future-proofing

Prototype - antywzorce

Future-proofing

Shadowing

Prototype - antywzorce

Future-proofing
Shadowing
Nadpisywanie host i native objects

Host and Native objects

Host and Native objects

Native object: obiekt implementujący specyfikację ECMAScript, np. Object, Date, Math

Host and Native objects

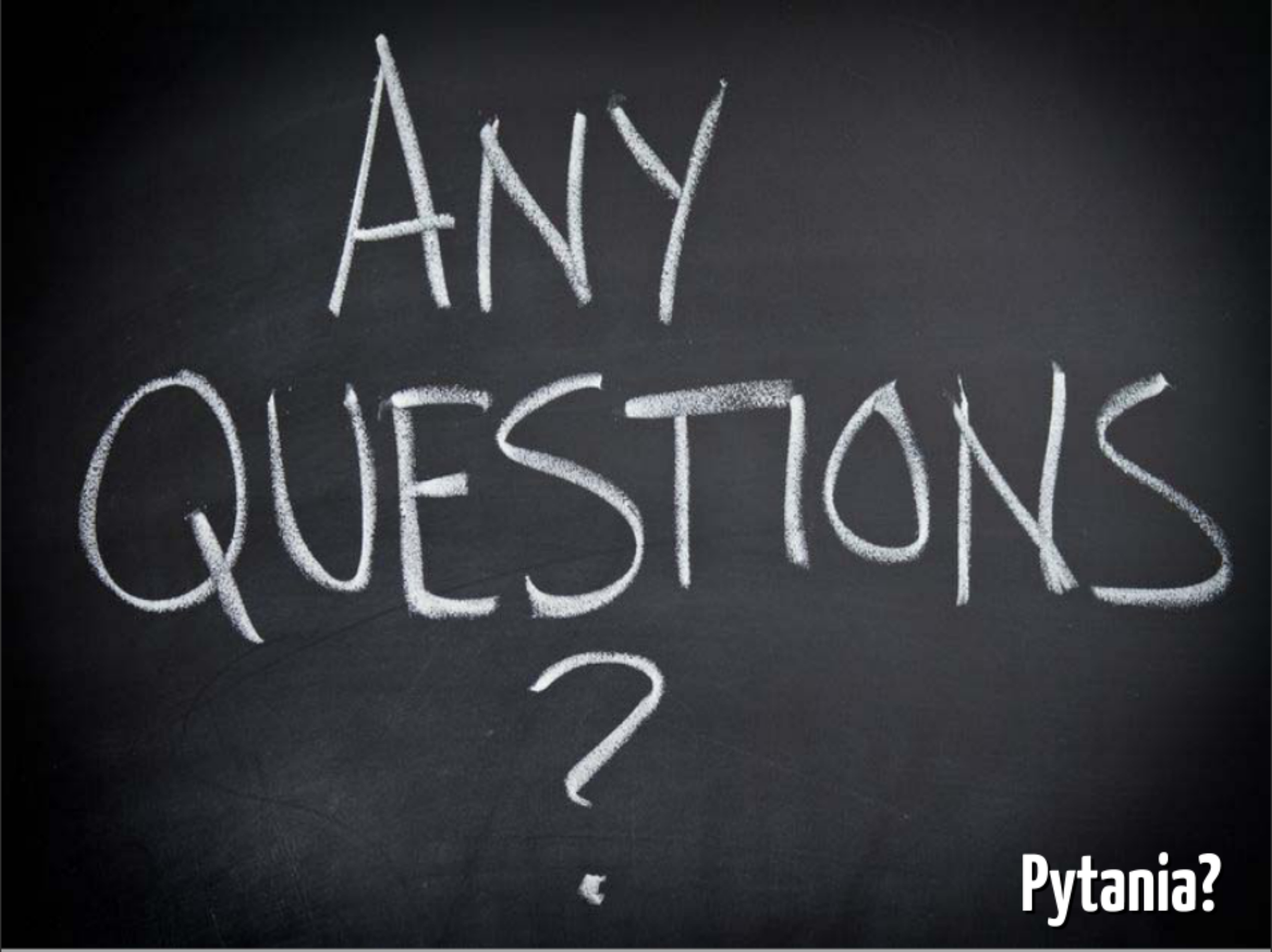
Native object: obiekt implementujący specyfikację ECMAScript, np. Object, Date, Math

Host object: obiekt dostarczany i specyficzny dla środowiska uruchomieniowego (przeglądarki/silnika JavaScript), np. window, document, XMLHttpRequest

Verbotten

Sandboxing

```
var sb, iframe = document.createElement('iframe'),
    document.body.appendChild(iframe);

sb = window.frames[1];

sb.Array.prototype.remove = function (member) {
    // ...
};

var arr = new sb.Array('carrot', 'potato', ...);

arr.remove('potato');

console.log(Array.prototype.remove); // undefined!
```



Pytania?