ferrante

Praktycznie zaawansowany JavaScript

Jak pisaliśmy JavaScript – dobre i złe strony

Ćwiczenie

Co możesz powiedzieć o swoim kodzie JavaScript? Co lubisz, co Cię denerwuje, a co ogranicza? Czego nie jesteś pewien?

typeof null == "object"

```
1. var foobar = null;
2. typeof foobar; // object
```

Tablice

- zapomnij o *new Array*

```
1. var foo = [];
2. var len = 5;
3.
4. foo[len-1] = 0;
5. foo.length; // 5
6. typeof foo[2] // undefined
```

Object literals

- podstawy

```
1. var obj = {
2.      property: "foobar",
3.      method: function() {}
4. };
5. typeof obj; // object
6. obj.property; // "foobar"
7. obj.method();
```

- pusty obiekt

```
1. var obj = {};
```

- tablica jako obiekt

```
1. var arr = [];
2. typeof arr; // object
```

.prototype w konstruktorach

- uaktualnianie właściwości obiektu *.prototype*

```
1. var fn = function() { };
2. fn.prototype.foo = "bar";
3.
4. var obj = new fn;
5.
6. fn.prototype.foo = "foo";
7. obj.foo; // "foo" !
```

Badanie obiektów

- dziwolągi

```
1. var obj = {
2.      5: "moim kluczem jest cyfra pięć"
3. };
4. obj.5; // błąd
```

```
1. var obj = {
2.      5piec: "moim kluczem jest cyfra pięć" // błąd
3. };
```

```
1. var obj = {
2.      5: "moim kluczem jest cyfra pięć"
3. };
4. obj['5']; // "moim kluczem jest cyfra pięć"
5. obj[5]; // "moim kluczem jest cyfra pięć"
```

instanceof

- *instanceof Array* niekoniecznie dobry

```
1. var arr = [];
2. arr instanceof Array // true
```

```
1. var Array = function() {};
2. var arr = [];
3. arr instanceof Array // false
```

Obiekty natywne

- *Object.prototype*

```
1. Object.prototype.say = function() { alert("hello world!"); };
2.
3. var obj = {};
4. obj.say(); // "hello world!"
5.
6. var arr = [];
7. arr.say(); // "hello world!"
```

- *Array.prototype*

```
1. Array.prototype.polishLength = function() {
2.      return "Długość tej tablicy to " + this.length;
3. };
4. var arr = [1, 2, 3];
5. arr.polishLength(); // Długość tej tablicy to 3"
```

Scope

- ważne "ciekawostki"

```
1. var i = 10;
2.
3. var fn = function() {
4.      alert(i); // undefined !
5.      var i = 100;
6.      alert(i); // 100
7. }
8.
9. fn();
```

Closures

```
1. var arr = [1, 2, 3];
2.
3. var i = 0;
4. for (var l = arr.length; i < l; i++) {
5.      arr[i] = (function(i) {
6.              return function() {
7.                      alert(i);
8.              }
9.      })(arr[i]);
10.}
11.
12.arr[0]();  // 1 !
```

this

- this, czyli "definicja jest trudna"
- this jako kontekst

```
1. var fn = function() {
2.      this.foobar = "foo";
3. };
4.
5. var obj = new fn;
6. obj.foobar; // foo
```

```
1. var fn = function() {
2.      this.foobar = "foo";
3. };
4.
5. var bar = fn();
6. bar.foobar; // błąd !
```

window

- window jako kontekst

```
1. var fn = function() {
2.      this.foobar = "foo";
3. };
4.
5. fn();
6. foobar; // foo !
```

```
1. var fn = function() {
2.      this === window; // true
3. };
4.
5. fn();
```

window

- ciekawostki

```
1. <script type="text/javascript">
2.      var x = "foobar" in window; // true
3.      var foobar = 2;
4. </script>
```

```
1. <script type="text/javascript">
2.      alert(window['foobar']); // undefined
3.      var foobar = 2;
4.      alert(window['foobar']); // 2
5. </script>
```

Funkcje jako obiekty

- podstawy

```
1. var fn = function() {};
2. fn.foobar = "2";
3. fn.foobar; // 2
```

```
1. var fn = function() {};
2. fn.method = function() { return this === fn; };
3. fn.method(); // true
```

Rekurencja anonimowa

- *arguments.callee*

```
1. var factorial = function(x) {
2.      if (x > 0) {
3.              return x * arguments.callee(x-1);
4.      }
5.      return 1;
6. };
7.
8. factorial(4); // 24
```