



JavaScript - Warsztaty

Który framework wybrać? Ile z nich znać?



Image source: <https://jelvix.com/blog/top-10-best-javascript-frameworks-list-in-2017>

Wszystkie frameworki zostały napisane przez osoby znające czysty JavaScript



**Vanilla JS is most power
and lightweight
cross-platform framework**

Lexical Scope.

Execution Context (globalny, lokalny).

Call stack.

Thread.

V8.

Compilation.

Function Scope VS Global Scope

Strict Mode

```
1 // "use strict"
2
3 var foo = "bar";
4
5 function bar() {
6     var foo = "baz";
7
8     function baz(foo) {
9         foo = "bam";
10        bam = "yay";
11    }
12    baz();
13 }
14
15 bar();
```

undefined
VS
not declared

Reference
error

```
1  // "use strict"
2
3  var foo = "bar";
4
5  function bar() {
6      var foo = "baz";
7
8      function baz(foo) {
9          console.log(foo);
10         foo = "bam";
11         // console.log(bam);
12         bam = "yay";
13     }
14     baz();
15 }
16
17 bar();
18 console.log(bam);
19 console.log(foo);
```

Higher order functions vs callbacks

Higher order function to funkcja, która zwraca lub do której przekazujemy inne funkcje.

Callback to funkcja zwracana lub przekazywana do funkcji.

JavaScript pozwala na zwracanie, przekazywanie funkcji, co jest jedną z najważniejszych jego cech.

```
1  function copyAndManipulateOnArray(array, fn) {
2      let result = [];
3      for (let i = 0; i < array.length; i++) {
4          result.push(fn(array[i]));
5      }
6      return result;
7  }
8  function multiplyBy2(v) {
9      return v * 2;
10 }
11
12 let manipulatedArray = copyAndManipulateOnArray([1, 2, 3], multiplyBy2);
```

Closure – funkcja ma dostęp do zmiennych z leksykalnego kontekstu, nawet jeżeli została wywołana w dowolnym innym miejscu.

W JavaScript każde wywołanie funkcji tworzy nowy, lokalny execution context.

```
1  function makeCounter() {
2      var i = 0;
3
4      return function () {
5          console.log(++i);
6      };
7  }
8
9  var counter = makeCounter();
10 counter(); // logs: 1
11 counter(); // logs: 2
12
13 var counter2 = makeCounter();
14 counter2(); // logs: 1
15 counter2(); // logs: 2
16
17 i; // ReferenceError
```


IIFE – Immediately-Invoked Function Expressions

W poniższym przykładzie moglibyśmy użyć również `for(let i=0; ...`

```
1  var elems = document.getElementsByTagName('a');
2
3  for (var i = 0; i < elems.length; i++) {
4
5      elems[i].addEventListener('click', function (e) {
6          e.preventDefault();
7          alert('I am link #' + i);
8      }, 'false');
9
10 }
11
12 // VS
13
14 var elems = document.getElementsByTagName('a');
15
16 for (var i = 0; i < elems.length; i++) {
17
18     elems[i].addEventListener('click', (function (lockedInIndex) {
19         return function (e) {
20             e.preventDefault();
21             alert('I am link #' + lockedInIndex);
22         };
23     })(i), 'false');
24
25 }
```

Moduły

Służą w celu ukrywania wewnętrznych implementacji, natomiast eksponowaniu tylko publicznego API. Klasyczny moduł:

```
1  var foo = (function () {  
2  
3      var obj = { bar: "bar" };  
4  
5      return {  
6          bar: function () {  
7              console.log(obj.bar);  
8          }  
9      };  
10  
11  })();  
12  
13  foo.bar();
```

ES6+ wprowadził system modułów (słowa kluczowe `import`, `export`), ale moduły muszą być umieszczone w oddzielnych plikach

Obiekty

Kilka sposobów na definiowanie obiektów:

```
1  let player1 = {  
2      name: "John",  
3      surname: "Smith",  
4      score: 0,  
5      getFullName: function () {  
6          return [name, surname].join(" ");  
7      },  
8      incrementScore: function () {  
9          player1.score++;  
10     }  
11 };
```

```
16  
17  let player2 = {}  
18  player2.name = "Jack";  
19  player2.surname = "Black";  
20  // ...  
21
```

Obiekty

Kilka sposobów na definiowanie obiektów:

```
23  // better
24
25  function playerCreator(name, surname, score) {
26      let newPlayer = {};
27      newPlayer.name = name;
28      newPlayer.surname = surname;
29      newPlayer.score = score;
30      newPlayer.incrementScore = function () {
31          newPlayer.score++;
32      };
33      return newPlayer;
34  };
35
36  let player3 = playerCreator("Will", "Kennedy", 2);
37  let player4 = playerCreator("Tim", "Kang", 1);
38  player3.incrementScore();
39  player4.incrementScore();
```


Obiekty

Kilka sposobów na definiowanie obiektów - **Prototype**

```
41 // even more better, most common professional approach
42
43 function Player(name, surname, score) {
44     this.name = name;
45     this.surname = surname;
46     this.score = score;
47 }
48 Player.prototype.incrementScore = function () {
49     this.score++;
50 };
51
52 let player5 = new Player("Mats", "Koenig", 0)
53 player5.incrementScore();
54
55 // MODULE
```

Nazwane funkcje, zamiast funkcji anonimowych

- **Lepsze debugowanie**
- **Self-reference, co jest przydatne np. przy rekurencji**
- **Dokumentowanie kodu poprzez odpowiednie nazwy**