Javascript jak każdy znany mi język programowania posiada zmienne. Posiada także zasięg zmiennych. Jednak dla programistów przyzwyczajonych do C (Javy, C++, C#) zasada działania zasięgu zmiennych może okazać się mylna...

Prosty kod

{codecitation class='brush: js'}

```
var x = 10;

function example()

{

    var x = &amp;quot;To w ogóle inny typ!&amp;quot;;

    alert(x);

}

alert(x);
```

example();{/codecitation}

Powyższy kod wyświetli:

{codecitation}10

To w ogóle inny typ!{/codecitation}

Póki co żadnych niespodzianek. Czego jednak spodziewasz się w takim wypadku:

{codecitation class='brush: js'}

```
function  example()

{

    for (var i = 0; i < 1; i++)

    {
```

```
        var x = &quot;To jest moja wartość&quot;;

         alert(x);

    }

     alert(x);

}

example();

alert(x);{/codecitation}
```

Wynik?

{codecitation}To jest moja wartość

To jest moja wartość

undefined{/codecitation}

O ile 3 alert nie powinien nikogo dziwić, to drugi jest niespodzianką. **W JavieScript klamry w pętlach nie tworzą zasięgu**. Raz stworzona zmienna jest widoczna do końca funkcji [jeśli jest stworzna poza funkcją, to jest zmienną globalną).

Dylemant

Co jednak, kiedy chcesz stworzyć zasięg? Czy na pewno nie ma jakiegoś wytrychu? Skoro zadaje to pytanie, to z pewnością odpowiedź brzmi "tak". W C starczyło stworzyć nowy blok kodu objęty klamrami... A w JS?

Tworzenie "bloku" kodu

Starczy stworzyć anonimową funkcję w funkcji. Funkcja tworzy nowy zasięg. Można ją od razu wywołać. Nie jestem pewien, czy nie zwalnia to skryptu (pewnie tak), jest jednak często sposobem na wredny zasięg zmiennych:)

{codecitation class='brush: js'}

function example()

```
{

    (function() {

        for (var i = 0; i < 1; i++)

        {

            var x = &amp;quot;To jest moja wartość&amp;quot;;

            alert(x);

        }

    })(); // tu uzywamy operatora wywołania funkcji, a więc automatycznie wywołujemy powyższy blok kodu

     alert(x);

}

alert(x);{/codecitation}
```

Działanie takiego kodu będzie mniejszą niespodzianką dla programistów z doświedczeniem w językach dziedziczących po C. Jednak powyższe zastosowanie uważam za niepotrzebne (w końcu po co 90% kodu funckji ma być inną funkcją? Nie lepiej dać po prostu inną nazwę zmiennej?)

Rozsądne zastosowanie

Często jest tak, że mamy kod HTML z fragmentem JS:

{codecitation class='brush: js'}<html>

<head>

<title>Przykład z youthcoders</title>

<script type="text/javascript">

var x = {'a' : 10, 'b': -1};

(function(obj) {

// mamy tu wlasny zasięg, i możemy bezpieczniej używać zmiennych

alert(obj.a);

})(x); // ... przekazując je jako parametry do funkcji anominowej

</script>

</head>{/codecitation}

Taki kod pozwala nam poczuć się choćby trochę bezpieczniej. Gdy załączasz wiele plików JS z różnych źródeł, w końcu nastąpi konflikt nazw. Warto zatem zabezpieczyć się choćby w ten sposób (lepiej jest użyć przestrzeni nazw, ale o tym kiedy indziej). Do naszego skryptu przekazujemy tylko te zmienne, które uważamy za niezbędne. Pozostałe też - jako zmienne globalne będą w anonimowej funkcji widoczne. Jeśli jednak każdą deklarację zmiennej poprzedzisz słowem kluczowym 'var' to twoje zmienne nie nadpiszą wartości globalnych.

To rozwiązanie często wykorzystywane jest w takim celu:

{codecitation class='brush: js'}(function($){

// tu mam dostęp do jQuery za pomocą $

})(jQuery);

{/codecitation}

Wiele skryptów tworzy obiekty wykorzystując właśnie nazwę $. Dlatego twórcy jQuery zdecydowali się na trzymanie referencji do frameworka w zmiennej o nazwie 'jQuery'. Wewnątrz naszej anonimowej funkcji pod zmienną o nazwie $ na pewno zawsze będziemy mieli jQuery. Warto, różne rzeczy w JS są możliwe :) Zarówno dobre, jak i złe :)