**Własne obiekty**

Pewnie słyszeliście już nazwę "programowanie obiektowe"? Jeżeli jeszcze nie programowaliście obiektowo, to nie ma się co martwić - JavaScript właściwie cała opiera się na programowaniu obiektowym ;).  
W waszych skryptach nieraz i nie dwa odwołujecie się do obiektów. Na przykład **window**, **document**, **array**, **string** itp. to typowe obiekty, które posiadają swoje metody i właściwości :).

Oczywiście jak i inne języki, tak i Javascript umożliwia tworzenie własnych obiektów.

**Tworzenie pojedynczego obiektu**

Aby utworzyć nowy pojedynczy obiekt możemy skorzystamy z poniższej konstrukcji:

var myObject = {

name: "Marcin",

height: 184,

print : function() {

console.log(this.name)

}

}

Nasz obiekt już na starcie posiada dwie ustalone właściwości - name i height, oraz jedną metodę, która wypisuje jego imię.

myObject.print(); //wypisze w konsoli "Marcin"

console.log(myObject.height); //wypisze 184

Aby odwołać się do danego obiektu z jego wnętrza stosujemy instrukcję **this**. Dzięki temu możemy w łatwy sposób wywoływać z wnętrza inne metody danego obiektu lub korzystać z jego właściwości:

var myObject = {

number: 100,

square : function() {

return this.number \* this.number

}

}

obiekt2.number = 200;

console.log(obiekt2.square())

Gdy nasz obiekt już istnieje, możemy do niego dodawać nowe metody lub właściwości, tak samo jak to robimy normalnie dla obiektu window:

var x = 10;

//jest równoznaczne z

window.x = 10;

var pisz = function() {...}

//jest równoznaczne z

window.pisz = function() {...}

Tak naprawdę deklarując zwykłe zmienne globalne i funkcje definiujemy po prostu właściwości i metody dla obiektu window. JS nie wymaga stosowania odwołania do window, więc praktycznie zawsze pomijamy to odwołanie.

Podobnie właściwości i metody możemy deklarować dla nowych obiektów:

var myObject = {

name: "Marcin",

height: 184,

print : function() {

alert(this.name)

}

}

myObject.weight = 73; //dodaliśmy nową właściwość

myObject.printDetail = function() {

return {

height : this.height,

name :

}

}

myObject.printDetail()

**Tworzenie obiektu za pomocą konstruktora**

Powiedzmy, że zamiast pojedynczego obiektu chcemy utworzyć ich kilka. Każdy obiekt powinien posiadać jakieś właściwości i metody np. width, height i wypisz().

Aby utworzyć kilka podobnych obiektów posłużymy się klasą obiektu. Czym jest klasa? To rodzaj foremki, która opisuje nam jak będą wyglądać tworzone na jej podstawie nowe obiekty. W JS w przeciwieństwie do innych języków nie mamy mechanizmu konstruktora, ale samą klasę możemy stworzyć za pomocą zwykłej funkcji:

function SuperObjectClass(\_width,\_height) {

this.width = \_width;

this.height = \_height;

this.print = function() {

console.log(this.width + 'x' + this.height)

}

}

var myObject1 = new SuperObjectClass(200, 100);

var myObject2 = new SuperObjectClass(300, 200);

myObject1.print(); //wypisze 200x100

myObject2.width = 600;

myObject2.print() //wypisze 600x200

**Usuwanie właściwości**

Aby usunąć właściwość obiektu, skorzystamy z operatora **delete**:

var myObject = {

color: 'zielony',

size: 'duzy',

price: 'wielka'

}

console.log(price);

delete myObject.price; //Usuwamy właściwość cena

console.log(price); //wypisze błąd

**instrukcja Prototype**

Przypuśćmy, że mamy już stworzone jakieś obiekty. Po jakimś czasie chcielibyśmy dodać do nich nową metodę. Dodajemy więc nową metodę do naszej foremki, ale okazuje się, że dostaną ją dopiero nowe obiekty stworzone na podstawie tej formy. Aby zmienić prototyp (a co za tym idzie i wszystkie obiekty stworzone na jego podstawie) posłużymy się instrukcją **prototype**.

function SuperObjectClass() {

this.name = 'Marcin';

this.height = 183

}

//dodaję właściwość i metodę do prototypu

SuperObjectClass.prototype.weight = 73;

SuperObjectClass.prototype.showInfo = function () {

alert(this.name + ' ma ' + this.height + 'cm wzrostu.')

}

var myObject = new SuperObjectClass();

myObject.showInfo();

console.log(myObject.weight); //wypisze sie 73

Instrukcja prototype przydaje się, gdy do danego obiektu np String czy Array chcemy dodać dodatkową funkcjonalność np:

String.prototype.firstCapital = function() {

return this.charAt(0).toUpperCase() + this.substr(1);

}

function mixLetterSize() {

var tekst = '';

for (var x=0; x<this.length; x++) {

tekst += (x%2==0)?this.charAt(x).toUpperCase() : this.charAt(x).toLowerCase();

}

return tekst;

}

String.prototype.mixLetterSize = mixLetterSize;

var text1 = "marcinek";

console.log(text1.firstCapital()) //wypisze Marcinek

var text2 = "marcinek";

console.log(text2.mixLetterSize()) //wypisze mArCiNeK

**Obiekty nadrzędne, kontekst obiektu**

Przypuśćmy, że mamy obiekt w obiekcie:

var SuperObjectClass = function() {

this.name = 'Marcin';

this.height = 183

this.button = null;

this.init = function() {

this.button = document.createElement('input');

this.button.addEventListener('click', function() {

this.value = this.height ???????

});

document.querySelector('body').appendChild(this.button);

}

this.init();

}

var someObject = new SuperObjectClass();

Po wywołaniu metody init tworzymy nowy guzik. Po jego kliknięciu powinien on ustawić swoje value na height SuperObjectClass. Jak jednak to zrobić, skoro instrukcja this wewnątrz obiektu guzika wskazuje na niego samego?  
Są na to dwa sposoby: posłużenie się dodatkową zmienną that:

var SuperObjectClass = function() {

this.name = 'Marcin';

this.height = 183

this.button = null;

this.init = function() {

var that = this;

this.button = document.createElement('input');

this.button.value = 'Kliknij';

this.button.type = 'button';

this.button.addEventListener('click', function() {

alert('To jest ' + this.nodeName); //przycisk

alert('Wzrost Marcina: ' + that.height); //obiekt SuperObiectClass

});

document.querySelector('body').appendChild(this.button);

}

this.init();

}

var someObject = new SuperObjectClass();

lub skorzystanie z instrukcji **bind()**, za pomocą której możemy przekazać kontekst do danej funkcji:

var SuperObjectClass = function() {

this.name = 'Marcin';

this.height = 183

this.button = null;

this.init = function() {

this.button = document.createElement('input');

this.button.value = 'Kliknij';

this.button.type = 'button';

this.button.addEventListener('click', function(e) {

//tutaj już this wskazuje na SuperObjectClass

//dlatego dany przycisk musimy pobrac za pomocą e.target

alert(e.target.value = this.height);

}.bind(this));

document.querySelector('body').appendChild(this.button);

}

this.init();

}

var someObject = new SuperObjectClass();

Bardzo dużo osób neguje stosowanie dodatkowej zmiennej do przekazywania kontekstu obiektu. Pamiętać należy jednak, że stosując instrukcję **bind()** zatracamy dostęp do this danego pod obiektu.

**Watch Unwatch**

Javascript udostępnia nam metodę **watch("nazwaWlasciwosci", funkcja(id, staraWartosc, nowaWartosc)**, która służy do podglądu właściwości obiektów. Jej działanie jest takie samo jak w innych językach. Gdy podglądana wartość się zmieni, wówczas watch wywoła funkcję podaną w drugim atrybucie. Przekazujemy jej 2 parametry - pierwszy określa nazwę obserwowanej właściwości, drugi to funkcja z 3 atrybutami: id, oldValue, newValue. Parametry te są wypełniane automatycznie. Gdy obserwowana wartość się zmieni, funkcja podana w 2 atrybucie zostanie wywołana:

var obiekt = {p:1}

//śledzenie właściwości "p" obiektu "obiekt"

obiekt.watch("p", function (id, oldValue, newValue) {

console.log("o." + id + " zmieniło sie z " + oldValue + " na " + newValue)

return nowaWart;

});

//teraz przy każdej zmianie wartości obiektu wywołany zostanie callback

//który śledzi zmiany w obiekcie

obiekt.p = 2

obiekt.p = 3

Aby zakończyć obserwowanie, korzystamy z metody **unwatch("nazwa\_obserwowanej\_wlasciwości")**.

obiekt.unwatch('p')'

[**Spis treści**](http://kursjs.pl/index.php)

[**Słowem wstępu...**](http://kursjs.pl/kurs/obiekty.php)

[**Super podstawy**](http://kursjs.pl/kurs/obiekty.php)

[**DOM**](http://kursjs.pl/kurs/obiekty.php)

[**Średnio zaawansowane**](http://kursjs.pl/kurs/obiekty.php)

* [**Własne obiekty**](http://kursjs.pl/kurs/obiekty.php)
* [Moduły](http://kursjs.pl/kurs/moduly.php)
* [Zdarzenia](http://kursjs.pl/kurs/events.php)
* [Myszka](http://kursjs.pl/kurs/mouse.php)
* [Grafika na stronie](http://kursjs.pl/kurs/obrazki.php)
* [Okienka dialogowe](http://kursjs.pl/kurs/okienka_dialogowe.php)
* [Windows](http://kursjs.pl/kurs/okna.php)
* [Obiekt Date](http://kursjs.pl/kurs/date.php)
* [Obiekt Math](http://kursjs.pl/kurs/math.php)
* [Random](http://kursjs.pl/kurs/random.php)
* [Obiekt Cookie](http://kursjs.pl/kurs/cookie.php)
* [Storage](http://kursjs.pl/kurs/storage.php)
* [AJAX](http://kursjs.pl/kurs/ajax/ajax.php)
* [Wyrażenia regularne](http://kursjs.pl/kurs/regular.php)
* [Formularze](http://kursjs.pl/kurs/formularze/formularze.php)
* [Formularze - weryfikacja danych](http://kursjs.pl/kurs/formularze/formularze_walidacja.php)

[**jQuery**](http://kursjs.pl/kurs/obiekty.php)

[**Canvas**](http://kursjs.pl/kurs/obiekty.php)

[**Gry**](http://kursjs.pl/kurs/obiekty.php)

[**Zastosowania praktyczne**](http://kursjs.pl/kurs/obiekty.php)

[**Dodatki**](http://kursjs.pl/kurs/obiekty.php)

**Inne strony**