tart

yeni medya mutfağı

JavaScript'te Nesne Tabanlı Programlama

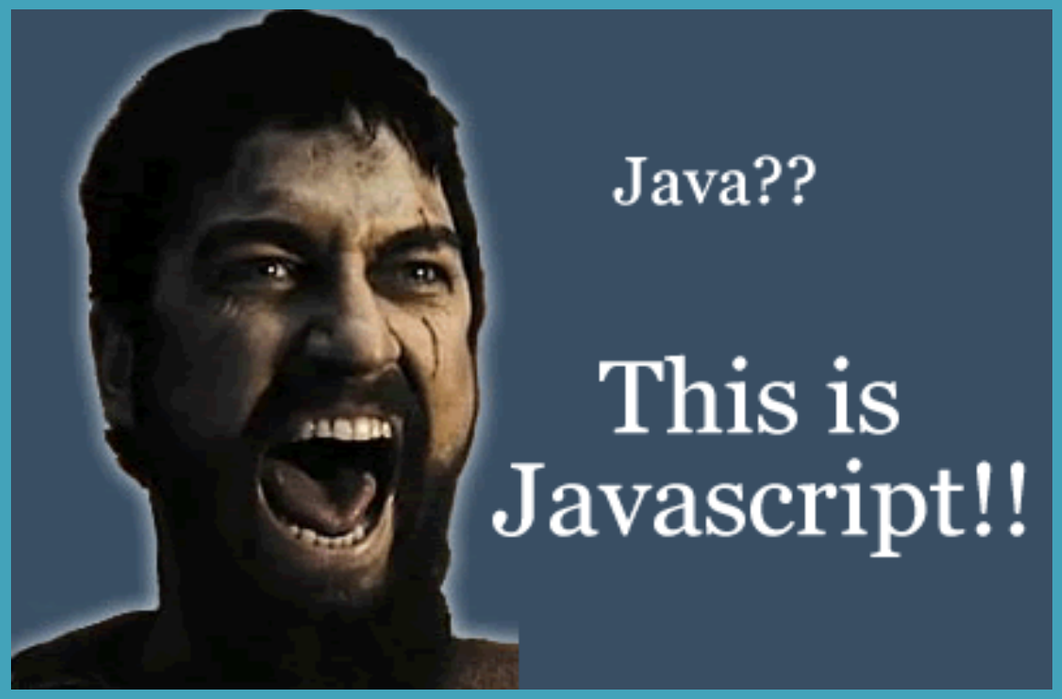
Sezgi Şensöz | FE201

JavaScript

- Dünyanın en çok kullanılan programlama dili
  - Sadece janjanlı animasyonlar için kullanılmıyor
  - Dinamik content

Yanlış anlaşılan dil: JavaScript

- Dünyanın en çok yanlış anlaşılan dili
  - –Java == JavaScript ??
  - –Ama syntax benziyor? (C, Java)
  - –Fonksiyonel diller (Scheme, Lisp)
  - –Toy language?



JavaScript: The World's Most Misunderstood Programming Language :
http://www.crockford.com/javascript/javascript.html

Veri Tipleri

- Primitive veri tipleri (string, boolean, number)
- undefined
- null
- Geri kalan “her şey” Object sınıfından türeyen (inherit) sınıflara ait objeler
- Primitive type lar başka type'a cast edilebilirler
  - –parseInt("12")
  - –var num = 12;
  - –num.toString()

Veri Tipleri

- Object → {}
- Array → []
- Function → var fn = function (){};
- RegExp → /[expression]/
- Number
- String
- Boolean
- Date
- Error
- Math

JSON (JavaScript Object Notation)

```
1  var objectLiteral = {
2          'a' : 2,
3          'b' : 's',
4          c : 'naber',
5          'd' : [8, 10],
6          'e' : { o : true }
7  }
8
9  objectLiteral
10
11 var arrayLiteral = [
12         'tart',
13         2,
14         true,
15         { a: 9}
16 ]
17
18 console.log(arrayLiteral)
```

- Öğeler virgül ile ayrılır
- Son öğeden sonra virgül konulmaz (bkz: Explorer)
- Value'lar başka bir obje veya array olabilir

- Object Literal'de süslü parantez ve
  - Key : value eşleştirmesi bulunur.

- Array Literal'de köşeli parantez içine tanımlanırlar. Otomatik arttırılan rakam key'lerden oluşan objelerdir.

https://gist.github.com/3132436

Scope / Context

- JavaScript, block scope'lu değil, function scope'lu bir dildir
- Her scope, dışındaki scope'daki verilere erişir ama içerideki verilere erişemez
- “var” ile belirtilmeyen değişkenler, “global” scope'a aittir ve global scope'un altındaki her değer için erişilebilir olur

1. Scope

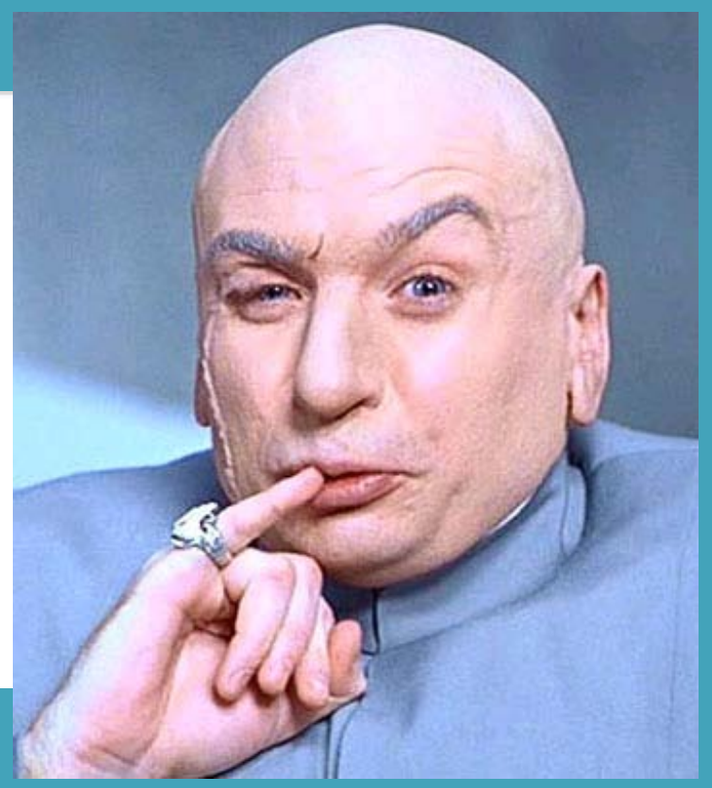
Scope / Context

```
(function () {
    var outerMember = "outer function's scope";

    (function () {
        var innerMember = "inner function's scope";
        console.log("test for accessing outer function");
        console.log(outerMember);
    })();

    console.log("test for accessing inner function");
    console.log(innerMember);
})();

/* output will be :
 *
 * test for accessing outer function
 * outer function's scope
 * test for accessing inner function
 * ReferenceError: innerMember is not defined
 */
```

1. Scope

Scope / Context

```
1  for (var i=0, link; i<2; i++) {
2      link = document.createElement("a");
3      link.innerHTML = "Link " + i;
4      link.onclick = function () {
5          alert(i);
6      };
7      document.body.appendChild(link);
8  };
```



1. Scope

Scope / Context

- “this” çalışan fonksiyonun ait olduğu “context”i belirtir
- context, methodun bağlı olduğu “scope”a eşittir ama değiştirilebilir

```
1  var foo = function () {
2      console.log(this);
3  };
4
5  var someObject = {};
6  someObject.someMethod = function () {
7      console.log(this);
8  };
9
10 var anotherObject = {"hell" : "yeah"};
11
12 foo(); // global
13 someObject.someMethod(); //someObject
14 someObject.someMethod.call(anotherObject); //anotherObject
15
```

https://gist.github.com/3132453

1. Context

Sınıflar

- JavaScript'de sınıflar da birer fonksiyondur
- “new” ile çağırılan fonksiyonlar, kendi “context”lerini döner
- “new” ile çağırılan ve obje dönen bu fonksiyonlara “constructor” denir
- İsimlendirme standartları gereği, genelde büyük harfle başlar

```
1  var Car = function(maxSpeed) {
2      this.maxSpeed = maxSpeed
3      console.log("Car class initilized and speed is " + this.maxSpeed)
4  }
5
6  var renault = new Car(200)
7
8  var ferrari = new Car(350)
9
10 /**
11  output will be:
12     Car class initilized and speed is 200
13     Car class initilized and speed is 350
14 */
```

Sınıflar

- Construct edilen her objenin .constructor property'si, o objenin hangi sınıftan construct ediliğini belirtir

```
1  var Car = function(maxSpeed) {
2      this.maxSpeed = maxSpeed
3      console.log("Car class initilized and speed is " + this.maxSpeed)
4  }
5
6  var renault = new Car(200)
7
8  renault.constructor === Car
```

.prototype

- Function sınıfından türeyen her obje construct edildiğinde “.prototype” isimli bir attribute'u bulunur
- .prototype, sınıftan türeyen tüm objelere **“referans”** olarak aktarılır
- Alt sınıfların objeleri de, üst sınıfın prototype zincirindeki öğelere erişebilir
- Bu sayede, **inheritance** sağlanmış olur

```
1  var Car = function(maxSpeed) {
2      this.maxSpeed = maxSpeed
3  };
4
5  Car.prototype.setMaxSpeed = function(newSpeed) {
6      this.maxSpeed = newSpeed;
7  };
8
9  Car.prototype.getSpeed = function() {
10     console.log("Car class initilized and speed is " + this.maxSpeed)
11 };
12
13 var renault = new Car(200);
14 renault.setMaxSpeed(150);
15 renault.getSpeed();
```

https://gist.github.com/3132362

Sınıflar

- Public, private, “privileged”
- protected ???

```
var SomeClass = function () {
    var privateMember = "this is private";

    this.publicMethod = function () {
        console.log("you can call this directly from instance");
    };

    this.privilegedMethod = function () {
        console.log("this can access private member: " + privateMember);
    };
};

var someObj = new SomeClass();
someObj.publicMethod(); // you can call this directly from instance
someObj.privilegedMethod();  //this can access private member: this is private
console.log(someObj.privateMember); //undefined
```

1. Information Hiding

Inheritance

- Bildiğiniz “klasik” inheritance JavaScript'te yoktur
- Ama, genel anlamda “kodu tekrar kullanılabilir” yapan yöntemler vardır
- Bunlara örnek olarak

–Functional pattern

–Pseudoclassical pattern

Inheritance

```
var Phone = function(data) {
        var that = {};

        that.number = data.number;
        that.getNumber = function() {
                return "lala " + that.number;
        }

        that.getOwner = function() {
                return "Phone's owner is " + data.owner;
        }

        return that;
}


var SmartPhone = function(data) {
        var that = Phone(data);

        that.size = data.size;

        that.sendMail = function(to, message) {
                console.log('mail will send to ' + to + ' with this message: ' + message);
        }

        that.getNumber = function() {
                return "Number is " + that.number;
        }

        return that;
}

var armaganPhone = Phone({ number : '0511', owner : 'Armagan'});
armaganPhone.getOwner(); // "Phone's owner is Armagan"
armaganPhone.getNumber(); // "lala 0511"

var myPhone = SmartPhone({ number : '000', owner : 'Sezgi', size: 4});
myPhone.sendMail('Armagan', 'naber'); // mail will send to Armagan with this message: naber
myPhone.getNumber(); // "Number is 000"
```

1. Inheritance

https://gist.github.com/3132308

Inheritance

```
1  var User = function(name){
2     this.name = name;
3  }
4
5  User.prototype.getName = function() {
6     return "user's name is " + this.name;
7  }
8
9  var FacebookUser = function(name, surname){
10    User.call(this, name);
11    this.surname = surname;
12 };
13
14 function tempCtor() {};
15 tempCtor.prototype = User.prototype;
16 FacebookUser.prototype = new tempCtor();
17 FacebookUser.prototype.constructor = FacebookUser;
18
19 FacebookUser.prototype.getName = function() {
20    return "Hello " + User.prototype.getName.call(this);
21 }
22
23 var ahmet = new FacebookUser("ahmet", "mahmut");
24 ahmet.getName();
```

https://gist.github.com/3132299

1. Inheritance

Inheritance

- Functional'da her instance için tüm methodlar kopyalanırken (hafız kullanımı), pseudoclassical'da prototype chain'i ortak kullanılır
- Functional inhertinace'da “instanceof” ile test edilemez
–ahmet instanceof FacebookUser //false
- Functional pattern'da, protoype chain'i kullanılamaz, dolayısıyla runtime'da, sınıftan türeyen tüm instance'lara etki eden bir değişiklik yapılamaz

Inheritance

- Functional'da her instance için tüm methodlar kopyalanırken (hafıza kullanımı), pseudoclassical'da prototype chain'i ortak kullanılır
- Functional inhertinace'da “instanceof” ile test edilemez
–facebookUser instanceof FacebookUser //false
- Functional pattern'da, protoype chain'i kullanılamaz, dolayısıyla runtime'da, sınıftan türeyen tüm instance'lara etki eden bir değişiklik yapılamaz
- Functional pattern'da, ana sınıfın methodunu çağırmak için sınıf içinde tekrar kopyalamak gerekir.

Kaynakça & Tavsiye edilen linkler/kitaplar:

- http://javascript.crockford.com/
- http://azer.googlecode.com/files/ileri_seviye_javascript.pdf
- http://bolinfest.com/javascript/inheritance.php
- http://killdream.github.com/blog/2011/10/understanding-javascript-oop/
- http://www.amazon.com/JavaScript-Good-Parts-Douglas-Crockford/dp/0596517742
- http://www.amazon.com/Pro-JavaScript-Techniques-John-Resig/dp/1590597273
- http://www.amazon.com/Professional-JavaScript-Developers-Wrox-Guides/dp/0764579088

Sorular ?

- 10 yılda programlama öğrenin http://norvig.com/21-days.html

Teşekkürler

www.tart.com.tr / sezgi.sensoz@tart.com.tr

GitHub
github.com/mefallit

Twitter
Twitter.com/mefallit