მომხმარებლის ობიექტების შექმნა

კიდევ ერთხელ დავუბრუნდეთ ობიექტების ცნებას JavaScript-ში, და შევისწავლოთ მათი შექმნის მეთოდები მომხმარებლის მიერ. მანამდე კი შევნიშნოთ, რომ როგორც ასეთი ობიექტი, არის ზოგადი წარმოდგენა, რაღაც უნივერსალური, რასაც კონკრეტულ შემთხვევაში შეიძლება მიეცეს, კონკრეტული რეალიზაცია, მაგალითად ზოოლოგიაში არის ფრინველთა კლასი - განზოგადოებულად წარმოდგენილი, ხერხემლიან ცხოველთა ნაწილი და ამ კლასში არის კონკრეტული წარმომადგენელი - ვთქვათ ფლამენგო. JavaScript-ის ობიექტებშიც არსებობს ასეთი ანალოგია. ობიექტი რომელსაც ქმნის მომხმარებელი ზოგადი სახით წარმოადგენს ობიექტის პროტოტიპს (object prototype) ან კლასს (class). ობიექტის პროტოტიპის კონკრეტულ რეალიზაციას ეწოდება ეგზემპლარი (ფლამინგოს ანალოგი)

თვისებები

ობიექტი, საჭირო თვისებების და მეთოდების ჩვენებით, მომხმარებელსაც შეუძლია შექმნას. მიზნად დავისახოთ ისეთი **Card-ობიექტის** შექმნა, რომელსაც ექნება **name, address, workphone** და **homephone** თვისებები.

პირველი, რასაც ვაკეთებთ, ობიექტების კონსტრუქტორად წოდებული ფუნქციის განსაზღვრა გახლავთ. სწორედ, მისი მეშვეობით შევქმნით შემდგომ ახალ-ახალ **Card-ობიექტს.** ცხადია, ფუნქციის სახელადაც **Card** -ს ვირჩევთ.
ასევე, ლოგიკურია ერთნაირი ან მსგავსი სახელები ჰქონდეს ობიექტის თვისებებსა და კონსტრუქტორის შესაბამის პარამეტრებს. მიღებულია მათი შეთანადებისას გამოყენებულ იქნეს **this** საკვანძო სიტყვა.
საბოლოოდ, აი, როგორ გამოიყურება **Card -ობიექტების** კონსტრუქტორი:

```
function Card (name,  address, work, home) {
this.name = name;
this.address = address;
this.workphone = work;
this.homephone = home;
}
```

ამ კოდში, მაგალითად, **this.workphone = work** ოპერატორი გვამცნობს, რომ **Card -ობიექტს** ექნება **workphone** თვისება, რომლის მნიშვნელობა განისაზღვრება **Card-ფუნქციის work** პარამეტრის მნიშვნელობით.

აქვე აღვნიშნოთ, რომ **Card** გახლავთ ობიექტის ზოგადი სახელი - ანუ ეს არის ობიექტის პროტოტიპი, კლასი. რაც შეეხება ახალ ობიექტებს (ანუ ობიექტის ეგზემპლარებს), თითოეულს ექნება ნებისმიერი ინდივიდუალური სახელი. ობიექტის ეგზემპლარი შეიძლება შევქმნათ **new** ოპერატორის საშუალებით:

holmes = new Card (“შერლოკ ჰოლმსი”, “221B ბეიკერ-სტრიტი ”, “555-1234”, “555-1111”);

დასაშვებია ობიექტის თვისებების მნიშვნელობების მოგვიანებით განსაზღვრაც:

```
holmes = new Card();
holmes. name = ”შერლოკ ჰოლმსი”;
holmes. address = ” 221B ბეიკერ-სტრიტი”;
holmes.workphone = ”555-1234”
holmes.homephone = ”555-1111”;
```

ლისტინგი 1 ობიექტის პროტოტიპის შექმნა კონსტრუქტორით

```
<html>
 <head>
 <title> ობიექტის პროტოტიპის შექმნა კონსტრუქტორით </title>
 <script language="javascript">
 <!-- ვინიღბებით!
 function Card(name,address,work,home) {
 this.name = name;
 this.address = address;
 this.workphone = work;
 this.homephone = home;
 }
 holmes=new Card("შერლოკ ჰოლმსი", " 221B ბეიკერ-სტრიტი","555-1234","555-1111");
 document.write("<b>"+holmes.name+"</b>"+"<br>"+holmes.address+"<br>");
 document.write(holmes.workphone+"<br>"+holmes.homephone);
 // ვიხსნით ნიღაბს. -->
 </script>
 </head>
 <body>
 </body>
 </html>
```

მომხმარებელს შეუძლია შექმნას ობიექტი სხვა გზითაც, ანუ ე.წ. ლიტერალის საშუალებით. ამ შემთხვევაში ცარიელ ობიექტს შევქმნით შემდეგნაირად :

```
var empty={ };    // ცარიელი ობიექტი შეიქმნა  ობიექტის ლიტერალით.
```

ობიექტის ლიტერალი - ეს არის ფიგურულ ფრჩხილებში მოთავსებული თვისებების სია, გამოყოფილი მძიმით. აღსანიშნავია, რომ თვისების სახელწოდება და მისი მნიშვნელობა ორწერტილით გამოიყოფა. შესაძლებელია ობიექტს დაემატოს ასევე ახალი თვისება, რომელიც მინიჭების ოპერატორის შეიძენს მნიშვნელობას. იხილე ლისტინგი 2. ობიექტის შექმნის ოპერაციებში დამხურავი ფიგურული ფრჩხილის შემდეგ იწერება წერტილმძიმე.

ლისტინგი 2. ობიექტის შექმნა ლიტერალით და ახალი თვისების მინიჭება

```
<html>
<head><title> ობიექტი ლიტერალით</title>
<body>
<script>
  var homer = {
    name: "ლუარსაბ თათქარიძე",
    age: 34,
    married: true
  };
  document.write("<b>"+homer.name+"</b>"+ "<br>")
  document.write(homer.age + " წლის"+"<br>");// იბეჭდება age თვისების პირველი
მნიშვნელობა
  homer.age = 35;  //age თვისებას ენიჭება ახალი მნიშვნელობა
  document.write("age თვისების მნიშვნელობა შეიცვალა და გახდა "+homer.age +
"<br>");  //იბეჭდება age თვისების ახალი მნიშვნელობა

  homer.male = "მამაკაცი";  //ვუმატებთ ახალ თვისებას თავისი მნიშვნელობით
  document.write("ახალი male თვისების მნიშვნელობაა - "+"<b>"+homer.male+"</b>");
//გამოგვყავს ახალი თვისების მნიშვნელობა
</script>
</head>
</body>
</html>
```

ობიექტის თვისების მნიშვნელობა შეიძლება იყოს ასევე ობიექტი, მაგალითად ლისტინგ 3-ში.
ლისტინგი 3. **თვისების მნიშვნელობა ობიექტია**

```
<html><head><title> თვისების მნიშვნელობა ობიექტია</title>
</head>
<body>
<script>
  var obj = {
    river: "მისისიპი",
    colors: {
      first: "ყვითელი",
      second: "ცისფერი"
    }
  };
    //თვისების მნიშცნელობის წცდომისთვის გამოიყენებს სტანდარტული სინტაქსისი
  document.write(obj.colors.first+" მდინარე "+obj.river);
</script>
</body>
</html>
```

შესაძლებელია თვისებების ანულირება, ამისთვის იყენებენ ოპერატორ **delete**-ს და თუ კი მოხდება შემდეგ მისი გამოტანის მცდელობა, იქნება შეტყობინება **undefined** - ლისტინგი 4.

ლისტინგი 4. თვისების მნიშვნელობა ობიექტია

```
<html><head><title> თვისების მნიშვნელობა ობიექტია</title>
</head>
<body>
<script>
  var obj = {
    river: "მისისიპი",
    colors: {
      first: "ყვითელი",
      second: "ცისფერი"
    }
  };
  delete obj.colors.first;
  //თვისების მნიშცნელობის წცდომისთვის გამოიყენებს სტანდარტული სინტაქსისი
  document.write(obj.colors.first+" მდინარე "+obj.river);
</script>
</body>
</html>
```

შეგვიძლია გამოვიყენოთ ციკლის ოპერატორი **for .. in** რათა მოვახდინოთ კონკრეტული ობიექტის თვისებების გადარჩევა და მასზე რაღაც ოპერაციების ჩატარება. ამ ოპერატოტს აქვს შემდეგი სახე:

```
for (var თვისების_სახელი in ობიექტის_სახელი) {
  ...   //ოპერატორები
}
```

ოპერატორი მუშაობას შემნეგნაირად, თვისების_სახელად აირჩევა ნებისმიერრი თვისების სახელი და შემდეგ იგი ფორმალურად ინარჩუნებს ამ თვისების მაჩვენებლკობას, ციკლის ტრიალისას იგი რეალურად მიიღებს ამ ობიექტის ყველა თვისების სახეს, ანუ იგი ფორმალური პარამეტრის თვისებას იძენს. ვნახოთ ის მოქმედებაში, ლისტინგ 5-ში.

ლისტინგ 5-ი. ციკლის ოპერატორი ობიექტში

```
<html><head><title> ციკლის ოპერატორი ობიექტში</title>
</head>
<body>
<script>
  var homer = {
```

```
  name: "ჰომერ სიმპსონი",
  age: 34,
  married: true
 };

 document.write("სანამ წავშლით: <br>");
 for (var name in homer) {
  document.write(name + " = " + homer[name] + "<br>");  //თვისების სახელის გამოყვანა
 }
  delete homer.age;
  document.write("<br> თვისების წაშლის შემდეგ: <br>")
 for (var name in homer) {
  document.write(name + " = " + homer[name] + "<br>");  //თვისების სახელებისა და
მათი მნიშვნელობების გამოტანა
 }
</script>
</script>

</body>
</html>
```

რეალიზაციის შედეგი

```
სანამ წავშლით:
name = ჰომერ სიმპსონი
age = 34
married = true

თვისების წაშლის შემდეგ:
name = ჰომერ სიმპსონი
married = true
```

იმის დასადგენათ, ამა თუ იმ ობიექტს, აქვს თუ არა, ესე თუ ის თვისება გამოიყენება ოპერატორი **if .. in** . **in** –ის მარცხენა მხარეს ჩაიწერება საძებნი თვისების სახელი, მარჯვნივ - ობიექტის სახელი. მაგალითად ლისტინგი 6.

ლისტინგი 6. პირობითი ოპერატორი ობიექტში

```
 <html><head><title>პირობითი ოპერატორი ობიექტში</title>
</head>
<body>
<script>
  var obj = {};
 if ("a" in obj) {
  obj.a = 1;
```

```
 }
 else {
  document.write("ასეთი თვისება არ არსებობს");
        }
</script>
</body>
</html>
```

რეალიზაციის შედეგი:

ასეთი თვისება არ არსებობს

როგორც ცნობილია, ობიექტის თვისებაზე და მეთოდზე წვდომა ხორციელდება ოპერატორ წერტილის ”.” საშუალებით. ობიექტის თვისებაზე წვდომა ასევე შესაძლებელია კვადრატული ფრჩხილების ”[ ]”ოპერატორის საშუალებით, როგორც გვახსოვს ”[ ]” გამოიყენება მასივებთან მუშაობისას. ამრიგად, შემდეგი ორი გამოსახულება ექვივალენტურია:

1. **obj.property**=10;
2. **obj[‘property’]**=10;

განვიხილოთ ლისტინგ 7-ს მაგალითზე

ლისტინგი 7. **თვისებაზე წვდომა [ ] ოპერატორით**

```
<html><head><title>თვისებაზე წვდომა [ ] ოპერატორით</title>
</head>
<body>
<script LANGUAGE="JavaScript">
 var obj = { name: "ჰომეროსი" };

 var str = "name";  //სტრიქონულ ცვლადს ვანიჭებთ თვისების სახელს

 //ჩანაწერი obj[str] ექვივალენტურია obj["name"]
 document.write(str + ": " + obj[str]+'  - obj[str]' +"<br>"+str + ": " + obj['name']+"  -
obj['name']" +"<br>"+str + ": " + obj.name+'  - obj.name');
</script>
</body>
</html>
```

მეთოდი

ახლა კი ვაჩვენოთ, როგორ ხდება ობიექტებში მეთოდის დამატება. (შევნიშნოთ, რომ ობიექტებში აუცილებელია თუნდაც ერთი მეთოდის არსებობა). მეთოდი გახლავთ ფუნქცია, რომელიც გარკვეული წესით დაამუშავებს ობიექტის თვისებებს. მიზნად

დავისახოთ ისეთი ფუნქციის შექმნა, რომელიც ეკრანზე გამოგვიყვანს **Card**-ობიექტის თვისებებს, სახელებს და მნიშვნელობებს.
ვუწოდოთ ამ ფუნქციას **PrintCard()**:

```
function PrintCard() {
Line1=”სახელი:  ” + this.name + ”<BR>\n”;
Line2=”მისამართი:   ” + this.address + ”<BR>\n”;
Line3=”ტელ.(სამსახ.):  ”+ this.workphone + ”<BR>\n”;
Line4=”ტელ.(სახლ.):  ” + this.homephone + ”<BR>\n”;
document.write (line1, line2, line3,  line4);
}
```

საინტერესოა, რომ **PrintCard** ფუნქცია არ საჭიროებს პარამეტრების ჩვენებას. იგი, როგორც ქვემოთ ვნახავთ, **Card**-ობიექტის კონსტრუქტორის მიერ გამოიძახება, როგორც მეთოდი, და სწორედ ამ ობიექტის თვისებებს იყენებს პარამეტრებად.
ფუნქციის შექმნის შემდეგ აუცილებელია ინფორმაციის მოთავსება **Card** -ობიექტის განსაზღვრებაში (ანუ **Card**-ფუნქციაში):

```
function  Card (name, address,  work,  home) {
this.name = name;
this.address = address;
this.workphone = work;
this.homephone = home;
this.PrintCard=PrintCard;
}
```

ვხედავთ, რომ მეთოდიც ისევე გამოცხადდა, როგორც თვისება. ოღონდ იგი ეყრდნობა შესაბამის ფუნქციას (მოცემულ შემთხვევაში **PrintCard**-ს).
ავამოქმედოთ შექმნილი მეთოდი **holmes** -ობიექტის ეგზემპლარისათვის.
ოპერატორს ექნება სახე:

```
holmes.PrintCard( );
```

ობიექტის კონსტრუქტორის განსაზღვრის შემდეგ შესაძლებელია მომხმარებლის ობიექტებისთვის საჭირო სიგრძის მასივის ფორმირებაც. ციკლის მეშვეობით ვქმნით ახალ ობიექტებს და შევუთანადებთ მათ მასივის ელემენტებს. მაგალითად:

```
i = 7;
cardarray[i] = newCard;
```

ლისტინგი 8 ობიექტის მეთოდის და თვისებების შექმნა

```
<HTML><HEAD>
<TITLE> ობიექტის შექმნა</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function PrintCard() {
line1 = "<B>სახელი: </B>" + this.name + "<BR>";
line2 = "<B>მისამართი: </B>" + this.address + "<BR>";
line3 = "<B>ტელ.(სამსხ): </B>" + this.workphone + "<BR>";
line4 = "<B>ტელ.(სახლ): </B>" + this.homephone + "<BR>";
```

```
 document.write(line1,line2,line3,line4);
}
function Card(name,address,work,home) {
  this.name = name;
  this.address = address;
  this.workphone = work;
  this.homephone = home;
  this.PrintCard = PrintCard;
}
</SCRIPT>
</HEAD>
<BODY>
<H1>პირადი ბარათები</H1>
სცენარი აქ იწყება.<HR>
<SCRIPT LANGUAGE="JavaScript">
//შექმენით ობიექტები
holmes = new Card("შერლოკ ჰოლმსი", "ლონდონი, 221B ბეიკერ-სტრიტი",
"555-1234", "555-111");
bond = new Card("ჯეიმს ბონდი", "ლონდონი, 232A პიკადილია", "125-1693", "412-
8422");
shtirlits = new Card("შტირლიცი", "ბერლინი, 2431 ალექსანდრპლაცი","432-4656",
"332-5234");
//გამოსახეთ ისინი
holmes.PrintCard();
bond.PrintCard();
shtirlits.PrintCard();
</SCRIPT>
<HR>სცენარის დასასრული
</BODY>
</HTML>
```

ჩაშენებული ობიექტების გაწყობა

JavaScript -ში შესაძლებელია ჩაშენებული ობიექტების შესაძლებლობების გაფართოებაც ახალი თვისებების და მეთოდების დამატების გზით.

დავუშვათ, გვსურს **String** ჩაშენებულ ობიექტში ჩავამატოთ **heading** მეთოდი, რომელიც ამა თუ იმ სტრიქონს ეკრანზე სასურველი დონის სათაურის რანგში გამოგვიყვანს.
მაშასადამე, ჩვენი მიზანი გახლავთ, შეგვეძლოს კოდში შემდეგი ბრძანების გამოყენება:
document. write (“ ეს ტესტია**”.heading(1));**
რიცხვი ”1” აქ **heading** მეთოდისთვის (ფუნქციისთვის) პარამეტრია. ცხადია, შეიძლებოდა მის მაგივრად აგვერჩია ”2”, ”3” და ა.შ. სიდიდეები.

პირველი, რაც უნდა განვახორციელოთ დასახული მიზნის მისაღწევად, გახლავთ **addhead** ფუნქციის განსაზღვრა, რომელსაც ექნება ერთი რიცხვითი პარამეტრი **level**:

```
function addhead (level) {
text = this.toString ( );
return (“<H”+level+“>”+text+“/h”+level+˜>”);
}
```

ადვილი შესამჩნევია, რომ **addhead** ფუნქციაში **HTML** -ოპერატორი იქმნება. წინა შემთხვევისაგან განსხვავებით, **String** ობიექტში **heading** მეთოდის ჩამატებას ჩვენ არაპირდაპირი გზით ვახორციელებთ ჩაშენებული ობიექტების მოდიფიცირება ხდება შემდეგი სპეციალური ოპერატორით:

```
String.prototype.heading = addhead;
```

საბოლოოდ, **String** ჩაშენებული ობიექტისთვის **heading (level)** მეთოდის შექმნის, ჩამატების და გამოყენების კოდს ექნება სახე:

```
<HTML>
<HEAD> <TITLE> მეთოდის ჩამატება </title>
<STYLE>
P  {font-family: LitNusx}
</style>
</head>
<BODY>
<SCRIPT LANGUAGE=“JavaScript”>
function addhead (level) {
text = this.toString ( );
return (“<H”+level+“>”+text+“</h”+level+“>”);
}
String.prototype.heading = addhead;
document.write (“<P>ეს ტესტია”.heading(1));
</body>
</html>
```

კიდევ ერთხელ გადავავლოთ თვალი შექმნილ კოდს. დასაწყისში ვქმნით **addhead( )** ფუნქციას, რომელსაც შემდეგ **prototype** საკვანძო სიტყვით გავამწესებთ **String** ობიექტის მეთოდად, **heading** სახელით. დასასრულ, ვახდენთ ამ მეთოდის შესაძლებლობების დემონსტრირებას “ეს ტესტია” სტრიქონის მაგალითზე. მოვიყვანოთ მწყობრში შესწავლილი მასალა - მიზნად დავისახოთ **Card** -ობიექტის რამდენიმე ეგზემპლარის შექმნა და ეკრანზე გამოყვანა:

ლისტინგი 9. **პირადი ბარათები**

```
< <HTML>
<HEAD> <TITLE> პირადი ბარათები </title>
<STYLE>
H2, P  {font-family: LitNusx}
</style>
<SCRIPT LANGUAGE="JavaScript">
function PrintCard()  {
line1="სახელი:   " + this.name + "<BR>";
line2="მისამართი:   " + this.address + "<BR>";
line3="ტელ.(სამსახ.):  "+ this.workphone + "<BR>";
line4="ტელ.(სახლ.):  " + this.homephone + "<BR>";
 document.write ("<P>" + line1 +  line2  +  line3  +  line4)
};
function Card (name,address,work,home) {
this.name = name;
this.address = address;
this.workphone = work;
this.homephone = home;
this.PrintCard=PrintCard;
};
</script>
</head>
<BODY>
<H2> პირადი ბარათები </h2>
<P> აქედან იწყება სცენარი.
<HR>
<SCRIPT LANGUAGE="JavaScript">
// ობიექტების შექმნა
gigi = new Card('გიგი გურული','კოსტავას 75','237-37-37','239-11-12');
lia = new Card ('ლია ბერიძე','რუსთაველის 25','293-23-34','236-45-55');
tea = new Card ('თეა გიგაური','წერეთელის 47','234-34-34','295-23-89');
// ობიექტების ასახვა
gigi.PrintCard();
lia.PrintCard();
tea.PrintCard();
</script>
<P> სცენარის დასასრული
</body>
</html>
```

ვხედავთ, რომ ეკრანზე ინფორმაციის უკეთ ასახვის მიზნით **PrintCard()** ფუნქცია რამდენადმე შეცვლილია.

აღსანიშნავია, რომ ზემოგანხილული წესით შეიძლება ობიექტისთვის შევქმნათ შვილობილი ობიექტები. ჯერ ვქმნით კონსტრუქტორის ფუნქციას ახალი

ობიექტისთვის და შემდეგ მშობლად გათვალისწინებულ ობიექტში ვამატებთ ახალ თვისებას. მაგალითად, თუ შევქმენით **Nicknames** ობიექტი თანამშრომელთა ფსევდონიმების დასაფიქსირებლად და გვსურს იგი **Card**-ობიექტთან მიმართებაში შვილობილად ვაქციოთ, ამ **Card**-ობიექტს კონსტრუქტორში უნდა დავუმატოთ შემდეგი ოპერატორი:

this.nick = new Nicknames ( );