ბ. ღვინეფაძე

WEB-დაპროგრამება

II ნაწილი

Javascript

“ტექნიკური უნივერსიტეტი”

საქართველოს ტექნიკური უნივერსიტეტი

გ. ღვინეფაძე

WEB-დაპროგრამება

II ნაწილი

Javascript

დამტკიცებულია სტუ-ს
სარედაქციო-საგამომცემლო
საბჭოს მიერ

თბილისი
2007

შაკ 681.3.06

წიგნში განხილულია WEB-დოკუმენტების შესაქმნელად განკუთვნილი სცენარების სპეციალიზებული ენა Javascript.

იგი განკუთვნილია ინფორმატიკის სპეციალობათა შემსწავლელი სტუდენტებისა და ამ საკითხით დაინტერესებული პირებისთვის.

რეცენზენტები: პროფ. ო. ნატროშვილი,
ასოც. პროფ. თ. სუხიაშვილი

ISBN 99940-14-80-3

JavaScript

შესავალი

WWW *(მსოფლიო აბლაბუდა)* **Web**-დოკუმენტების შესაქმნელად თავდაპირველად მხოლოდ **HTML** ენის შესაძლებლობებს იყენებდა. ამ მარტივი ენის დესკრიპტორებით *(ელემენტებით)* ხდება დოკუმენტის ფორმატირება, რაც ბროუზერს საშუალებას აძლევს **Web**-ფურცელი ავტომატურად ასახოს ეკრანზე. მაგრამ ინტერაქტიური *(დიალოგის)* ხასიათის ამოცანებისთვის საჭირო გახდა უფრო რთული, ე.წ. *სცენარების მომზადების ენების* გამოყენება. ისინი, ჩვენთვის ნაცნობი დაპროგრამების ენებისაგან განსხვავებით, არ საჭიროებენ პროგრამების კომპილაციას – მიმდევრობით სრულდება პროგრამის *(რომელსაც აქ სცენარი ეწოდება)* თითოეული სტრიქონი, ანუ ხდება მისი <u>ინტერპრეტირება.</u>

საკითხის ამგვარი გადაწყვეტა ძალზე აადვილებს სცენარების შექმნას და კორექტირებას – ყოველი ცვლილება ძალაში შედის ბროუზერის ფანჯარაში **Web**-ფურცლის ხელახლა გამოყვანისთანავე.

სცენარების მომზადების ენებს შორის დღეს მსოფლიოში ყველაზე პოპულარულია კომპანია **Netscape Communication Corporation**-ის მიერ შექმნილი **JavaScript** ენა. მასზე შექმნილი სცენარები გასაგებია ყველა პოპულარული ბროუზერისთვის *(მაგალითად, **VBScript** ენისაგან განსხვავებით, რომელიც მხოლოდ **Internet Explorer**-ზეა ორიენტირებული).*

JavaScript ენის კონსტრუქციებს გარეგნულად ბევრი აქვს საერთო **Java** ენის შესაბამის კონსტრუქციებთან, მაგრამ ეს უკანასკნელი უფრო მძლავრი, მაშასადამე, უფრო რთული ენაცაა. **Java**-ზეც შეიძლება **Web**-ფურცლებზე გამოსაყვანი ფრაგმენტებისათვის სპეციალური პროგრამების – ე.წ. *აპლეტების* დაწერა, მაგრამ ბროუზერის მიერ მათი უშუალოდ გამოყენება ვერ ხერხდება – აპლეტები ჯერ საკლასიფიკაციო ფაილებში უნდა იქნეს კომპილირებული. არის სხვა სირთულეებიც, რის გამოც, თუ განსაკუთრებით რთული ამოცანების გადაწყვეტა არ გვიწევს, **Web**-ფურცლების შექმნისას უპირატესობას ვანიჭებთ **JavaScript** ენას.

<u>ჩვენი პირველი სცენარები</u>

Notepad ტექსტურ რედაქტორში ავკრიფოთ პროგრამული კოდი, რომელშიც გამოყენებული იქნება მხოლოდ **HTML** ენის შესაძლებლობები:

```
<HTML>
<HEAD>
<TITLE>FIRST  PAGE</title>
<STYLE>
H3, P  {font-family: LitNusx}
</style>
</head>
```

```
<BODY>
<H3> კეთილი იყოს თქვენი მობრძანება ინტერნეტის სამყაროში! </h3>
<P> მაშ ასე, ორშაბათიდან ვიწყებთ ახალი საქართველოს შენებას!
</p>
</body>
</html>
```

JavaScript ენაზე დაწერილ სცენარს კი განვალაგებთ დესკრიპტორების შემდეგი წყვილის შიგნით: **<SCRIPT>** *სცენარი* </**script**> და მოვათავსებთ მას **HTML** ენაზე დაწერილ პროგრამაში:

```
<HTML>
<HEAD>
<TITLE>Next Page</title>
<STYLE>
H1, P {font-family: LitNusx}
</style>
</head>
<BODY>
<H1> კეთილი იყოს თქვენი მობრძანება ინტერნეტის სამყაროში! </h1>
<P> ვაგრძელებთ შრომითს საქმიანობას! ვისახავთ ახალ მიზნებს. თან
ოპერატიულად გატყობინებთ ჩვენი დოკუმენტის ბოლო ცვლილების თარიღს.
</p>
<SCRIPT LANGUAGE=”JavaScript”>
document.write(document.lastModified);
</script>
</body>
</html>
```

დავიმახსოვროთ ეს პროგრამები ჯერ **txt** და შემდეგ **html** გაფართოებით.

ვხედავთ, რომ მეორე მაგალითში სცენარი მოთავსებულია **Web-**დოკუმენტის პროგრამის სხეულში. მაგრამ, საერთოდ, შესაძლებელია, იგი სათაურის უბანშიც განვალაგოთ. ასეთ შემთხვევაში სცენარი ჩატვირთვისთანავე არ სრულდება – იგი სხვა სცენარების მიერ გამოიძახება, როგორც *ფუნქცია*.

სცენარი **HTML-**დესკრიპტორშიც შეიძლება განვალაგოთ ე.წ. *ხდომილობის დამმუშავებელი კონსტრუქციის* სახით. ხდომილობის დამმუშავებლისთვის <**SCRIPT**> </**script**> ფრჩხილების გამოყენება საჭირო აღარ გახლავთ.

დაბოლოს, სცენარი შეიძლება გაფორმდეს **js** გაფართოების მქონე ფაილის სახითაც. პროგრამაში მისი გამოძახება ხორციელდება ფაილის სახელის ჩვენებით <**SCRIPT**> წყვილის შიგნით.

აღვნიშნოთ, რომ სცენარში ხშირად მითითებულია **JavaScript** ენის ვერსიაც. ეს კეთდება იმ მიზნით, რომ ძველმა (*მაშასადამე, ნაკლები შესაძლებლობების მქონე)* ბროუზერებმა უნაყოფოდ არ სცადოს მათთვის გაუგებარი სცენარების შესრულება, რაც ზოგჯერ ბროუზერის “ჩამოკიდებასაც” კი იწვევს.

დავუშვათ, გვსურს **Web**-ფურცელზე ავსახოთ ჩვენი დაბადების დღიდან გასული წამების რაოდენობა.

გავითვალისწინოთ ის ფაქტი, რომ კომპიუტერში თარიღები მილიწამებში აითვლება და დავწეროთ შემდეგი კოდი:

```
<HTML>
<HEAD>
<TITLE>ჩემი ასაკი წამებში</title>
<STYLE>
H2, P {font-family: LitNusx}
</style>
</head>
<BODY>
<H2>წამია კაცის ცხოვრება, მხოლოდ საკითხავია რამდენი?</h2>
<HR>
<SCRIPT LANGUAGE="JavaScript">
now=new Date();
y2k=new Date("jan 01 1981 00:00:00");  // აქ უჩვენეთ დაბადების თარიღი
seconds=(now-y2k) / 1000;
document.write("<P>ჩემი დაბადებიდან გავიდა " + seconds  + " წამი");
</script>
</body>
</html>
```

დოკუმენტი ჩავტვირთოთ ბროუზერში.

მივაქციოთ ყურადღება შემდეგ გარემოებას: **JavaScript** ენა ცვლადებსა და სხვა კონსტრუქციებში ერთმანეთისგან განასხვავებს დიდსა და პატარა ასოებს.

ეკრანზე გამოყვანილ რიცხვს წერტილის შემდეგ მოსდევს ორი ათობითი ნიშანი *(რასაც **JavaScript** დუმილით ითვალისწინებს)*, შედეგის დამრგვალების-თვის შეიძლება გამოვიყენოთ **Math**.**round** *სპეციალური ფუნქცია*. იგი **document**. **write** ოპერატორის წინ უნდა ჩავსვათ:

seconds=**Math.round**(**seconds**);

კორექტირების შემდეგ ბროუზერის ფანჯარაში **Web**-ფურცლის ხელახლა გამოსაყვანად დავაწკაპუნოთ **Refresh** ღილაკზე.

ვნახოთ, თუ როგორ რეაგირებს ბროუზერი შეცდომაზე. ზემოაღწერილ ოპერატორს მივცეთ შემდეგი სახე:

seconds = **Math.round** (**second**;

ფურცლის ჩატვირთვისას, ცხადია, თავს იჩენს შეცდომა. **Netscape Navigator**-ბროუზერის ბოლო ვერსიებში მისამართის ველში *javascript:* ინფორმაციის შეტანის შემდეგ ეკრანზე აისახება **JavaScript Console** სამუშაო ფანჯარა. მასში გამოიყვანება შეცდომის შემცველი სტრიქონი შეცდომის მიზეზის მითითებით *(თუმცა ზოგჯერ ეს მიზეზი შეიძლება, არასწორად იყოს*

ახსნილი). ამავე ფანჯარაში გვეძლევა შეცდომის გამოსწორების შესაძლებლობაც.

ამჯერად, მიზნად დავისახოთ ამოცანაში გამოთვლილი დროის გამოყვანა წუთებშიც. **document.write** ოპერატორის შემდეგ კოდში ვამატებთ შემდეგ ფრაგმენტს:

minutes = seconds/60;
minutes = Math.round(**minutes**);
document.write ("<**P**>*ჩემი დაბადებიდან გავიდა* " + **minutes** + " *წუთი*");

ჩვენ გავეცანით **JavaScript**-ის მეშვეობით შექმნილ რამდენიმე მარტივ სცენარს. ენის უფრო რთულ კონსტრუქციებს მომდევნო თავებში შევისწავლით.

მივცეთ Web-ფურცელს უფრო მიმზიდველი სახე!

დავიწყოთ მდგომარეობის ამსახველი სტრიქონიდან (**Status Bar**). მიზნად დავისახოთ მასში მორბენალი სტრიქონის გამოყვანა. **Notepad**-ში ავკრიბოთ შემდეგი კოდი:

```
<HTML>
<HEAD>
<TITLE>შევქმნათ მორბენალი სტრიქონი!</title>
<STYLE>
H2,P  {font-family: LitNusx}
</style>
<SCRIPT LANGUAGE="JavaScript">
var msg= "Hello,  Baby!";
var spacer= "  ...                    ...  " ;
var pos=0;
function ScrollMessage() {
window.status=
msg.substring(pos, msg.length) + spacer + msg.substring(0,pos);
pos++;
if (pos > msg.length) pos=0;
window.setTimeout("ScrollMessage()", 200);
}
ScrollMessage();
</script>
</head>
<BODY>
<CENTER><H2>მორბენალი სტრიქონის მაგალითი</h2></center>
<P> შეხედეთ სტატუსის სტრიქონს! </p>
</body>
</html>
```

ამ კოდში გასარკვევად დაგვჭირდება **JavaScript**-ის რიგი შესაძლებლობების შესწავლა.

ფუნქციები და ობიექტები

პირველ ყოვლისა, გავეცნოთ **ფუნქციის** ცნებას. ენის აღნიშნული კონსტრუქცია სცენარის სტრუქტურის გამარტივებაში გვეხმარება.

ფუნქცია წარმოადგენს **JavaScript**-ის ოპერატორების ჯგუფს, რომლებიც ფუნქციის გამოძახების შემთხვევაში სრულდება, როგორც ერთი მთლიანობა.

მოვიყვანოთ ფუნქციის განსაზღვრის მაგალითი:

```
function Greet() {
alert ("აბა, ჰე! ")
}
```

ამ მაგალითში **function** საკვანძო სიტყვის გამოყენებით განვსაზღვრეთ **Greet**() ფუნქცია, რომლის გამოძახებისას *(იხ. ქვემოთ)* შესრულდება ფიგურულ ფრჩხილებში მოყვანილი ოპერატორების მიმდევრობა – მოცემულ შემთხვევაში ეს გახლავთ ერთადერთი **alert**() ოპერატორი, რომელიც წარმოადგენს ჩაშენებულ ფუნქციას. მისი დანიშნულებაა ეკრანზე რაიმე შეტყობინების გამოყვანა.

ფუნქციისათვის უფრო მეტი მოქნილობის მისანიჭებლად უმეტეს შემთხვევაში იყენებენ პარამეტრებს *(არგუმენტებს).*

პარამეტრები წარმოადგენს იმ ცვლადებს, რომელთა მნიშვნელობები ფუნქციას გადაეცემა მისი გამოძახების მომენტში.

მაგალითად, ზემომოყვანილი კოდის ფრაგმენტი შეიძლებოდა ასე გაგვერთულებინა:

```
function Greet (who) {
alert ( "აბა, ჰე, " + who);
}
```

ცხადია, ფუნქციის გამოძახების მომენტში **who** ცვლადს რაიმე მნიშვნელობა უნდა ჰქონდეს მინიჭებული.

ფუნქციას ტრადიციულად <**HEAD**> უბანში განსაზღვრავენ, ხოლო გამოძახებას ახდენენ კოდის სხეულში – მიუთითებენ ფუნქციის სახელს და, საჭიროების შემთხვევაში, განსაზღვრავენ პარამეტრების მნიშვნელობას. მაგალითად:

```
<HTML>
<HEAD>
<TITLE>ფუნქციების გამოყენება</title>
<STYLE>
H2, P  {font-family: LitNusx}
</style>
<SCRIPT LANGUAGE="JavaScript">
```

```
function Greet(who) {
alert("აბა,  ჰე, " + who  );
}
</script>
</head>
<BODY>
<H2>ვისწავლოთ ფუნქციებთან მუშაობა!</h2>
<SCRIPT LANGUAGE="JavaScript">
Greet("გიგლა!");
Greet("გაგა!");
</script>
</body>
</html>
```

Greet (“გიგლა!”) ოპერატორით ფუნქციის პირველი გამოძახებისას ეკრანზე გამოდის შესაბამისი შეტყობინების ფანჯარა:

OK-ზე დაწკაპუნების შემდეგ მას შეცვლის ასეთივე ფანჯარა, ოღონდ ამჟამად შეტყობინებაში გამოვა “აბა, ჰე, გაგა!”.

ფუნქციას არა მარტო შეტყობინების გამოყვანა შეუძლია – მისი მეშვეობით ხშირად გამოთვლიან რაიმე მნიშვნელობასაც. ამ მიზნით, ფუნქციას შესაძლოა რამდენიმე პარამეტრი გადაეცეს, მაგრამ სცენარისთვის უკან დასაბრუნებელი მნიშვნელობა კი მხოლოდ ერთადერთი შეიძლება იყოს და ეს მნიშვნელობა თვით ამ ფუნქციის სახელთან დაკავშირდება.

ქვემოთ მოყვანილი კოდის ფრაგმენტში **Average** ფუნქციას **return** ოპერატორით უბრუნდება **result** ცვლადის მნიშვნელობა, რომელიც წარმოადგენს ამ ფუნქციისათვის გადაცემული 4 პარამეტრის მნიშვნელობების საშუალო არითმეტიკულ სიდიდეს:

```
<SCRIPT LANGUAGE=”JavaScript”>
 function Average  (a,b,c,d)  {
 result=(a+b+c+d)/4;
 return  result;
}
</script>
```

Average ფუნქციის მნიშვნელობას კი შემდგომ სხვა ოპერატორებში გამოვიყენებთ. ცხადია, ფუნქციის გამოძახების მომენტში **a, b, c, d** ცვლადების

მნიშვნელობა განსაზღვრული უნდა იყოს. ეს შეიძლება პირდაპირი გზითაც მოხდეს, მაგალითად:

score = **Average**(3,4,5,6);

დასაშვებია ფუნქციის გამოძახება გამოსახულების შემადგენელი ნაწილის სახითაც:

alert (**Average**(1,2,3,4));

გავეცნოთ ენის სხვა, ასევე უმნიშვნელოვანეს კომპონენტს – **ობიექტებს**.

თუ ცვლადი მხოლოდ ერთ მნიშვნელობას შეიცავს *(რიცხვითს, ტექსტურს, თარიღის ტიპის და სხვ.),* ობიექტი შეიძლება წარმოვიდგინოთ, როგორც ერთ სახელთან დაკავშირებული *(ცხადია, ამავე დროს საკუთარი სახელის მქონეც)* ცვლადების კრებული. ამ ცვლადთაგან თითოეულს უწოდებენ **თვისებას**.

ობიექტს, მაგალითად, შეიძლება წარმოადგენდეს *მომხმარებელი*, ხოლო მისი თვისებები გახლდეთ: *სახელი, გვარი, მისამართი, ტელეფონის ნომერი* და სხვ. თითოეული ამ თვისებათაგანი რომელიმე კონკრეტული ობიექტისათვის, მაგალითად, **Bob**-ისთვის, ასე შეიძლება გამოვსახოთ:

Bob. address ან **Bob. phone**

თვისებების გარდა, ობიექტები ხშირად შეიცავენ **მეთოდებსაც**.

მეთოდი შეიძლება განვმარტოთ როგორც ფუნქცია, რომელიც გარკვეული წესით დაამუშავებს ობიექტის თვისებას *(ან თვისებებს).*

მაგალითად, **Bob.display()** მეთოდი საშუალებას იძლევა, ეკრანზე აისახოს **Bob**-ობიექტის ყველა თვისება.

JavaScript-ში საქმე გვაქვს 3 ტიპის ობიექტებთან:

- ჩაშენებული ობიექტები. ზემოთ ჩვენ უკვე გავეცანით ორ ასეთ ობიექტს: **Date()** და **Math().**
- ბროუზერის ობიექტები. ჩვენ მიერ განხილული **alert()** ფუნქცია, ფაქტობრივად, **windows**-ობიექტის ერთ-ერთ მეთოდს წარმოადგენს.
- მომხმარებლის მიერ შექმნილი ობიექტები.

სამივე ტიპის ობიექტებს უფრო დაწვრილებით შემდგომში გავეცნობით.

ხდომილობების დამუშავება

როგორც შესავალში აღვნიშნეთ, გარდა <**script**> უბანში მოთავსებისა, **HTML** კოდში შესაძლებელია სცენარი ხდომილობათა დამმუშავებლის როლშიც გამოვიყვანოთ.

ხდომილობათა მაგალითებია: თაგვის მაჩვენებლის მოთავსება გრაფიკული თუ ტექსტური სახის კავშირზე, თაგვის ღილაკზე ხელის დაჭერა ან დაწკაპუნება, **Web**-ფურცლის ეკრანზე გამოყვანის დამთავრება და სხვ. ამრიგად, ხდომილობის დამმუშავებელი სცენარის გამოძახების ინიციატორი შეიძლება იყოს როგორც მომხმარებელი, ასევე კომპიუტერიც.

გრაფიკულ კავშირზე თაგვის მაჩვენებლის მოთავსება აღინიშნება როგორც **MouseOver** ხდომილობა. მსგავსი სახე აქვს სხვა ხდომილობებსაც.

დავუშვათ, სწორედ ასეთ ხდომილობას აქვს ადგილი ეკრანზე გამოტანილი **button.gif** ნახატისათვის და მოითხოვება, ხდომილობის დამმუშავებლის როლში გამოძახებულ იქნეს სცენარი, ვთქვათ, **Highlight**() ფუნქციის სახით.

JavaScript-ში ამ მოთხოვნის რეალიზებას უზრუნველყოფს შემდეგი კონსტრუქცია:

<**IMG src**=”**button.gif**” **onMouseOver**=”**Highlight**()”>

ხაზს ვუსვამთ ზუსტად ასეთი სინტაქსისის გამოყენების აუცილებლობას.

შეგახსენებთ, რომ ფუნქცია, როგორც წესი, ოპერატორების კრებულს წარმოადგენს. თუ მისი გამოძახება ხშირად ხდება, კოდი ძალიან მარტივდება.

აქვე შევეხოთ ერთ საკითხსაც. ბროუზერების ძველი ვერსიებისთვის **JavaScript** ენა გაუგებარია. რომ არ მოხდეს მათი მუშაობის შეფერხება (ან ეკრანზე **JavaScript** ენის კოდის მექანიკურად გამოტანა), იყენებენ **HTML** ენის კომენტარებს:

<!-- კოდი -->

ახალი ბროუზერები დესკრიპტორების ამ წყვილს არავითარ ყურადღებას არ აქცევს და მასში მოთავსებულ კოდს ჩვეულებრივად ამუშავებს, განსხვავებით ძველი ბროუზერებისაგან, რომლებისთვისაც აღნიშნული ფრაგმენტი მხოლოდ კომენტარია და, ცხადია, მის იგნორირებას ახდენს.

ქვემოთ მოყვანილია მაგალითი, თუ როგორ შეიძლება “დავუმალოთ” კოდი ძველ ბროუზერს:

<**SCRIPT LANGUAGE**=”**JavaScript**”>

<**!--**

document.write(**“**თქვენს ბროუზერს შეუძლია **JavaScript**-თან მუშაობა”);

// -->

</**script**>

JavaScript-ის კოდში კომენტარების ტრივიალური დანიშნულებით გამოსაყენებლად მიმართავენ შემდეგ კონსტრუქციებს:

// ეს კომენტარია

a=a+1; *// ეს კომენტარია*

/ ეს კი გახლავთ*

სამ სტრიქონზე

*განთავსებული კომენტარი */*

ამრიგად, ჩვენ წარმოდგენა შეგვექმნა **JavaScript**-ის ფუნდამენტურ ცნებებსა და სცენარების შექმნის ძირითად საშუალებებზე. მომდევნო თავებში მათ უფრო დეტალურად განვიხილავთ.

ნაწილი II

JavaScript-ზე დაპროგრამების ძირითადი საშუალებები

ცვლადები

ცვლადებს **JavaScript**-ში მონაცემთა კონტეინერებსაც უწოდებენ. მოვიყვანოთ მათი სახელდების წესები:

- ცვლადის სახელის შემადგენლობაში შეიძლება შედიოდეს ლათინური ალფაბეტის დიდი და პატარა ასოები;
- სახელი არ შეიძლება ციფრით იწყებოდეს;
- ენა განასხვავებს დიდსა და პატარა ასოებს. მაგალითად: **Total** და **total** ორი სხვადასხვა ცვლადია.
- სახელის სიგრძე ოფიციალურად შეზღუდული არ გახლავთ, თუმცა იგი კოდის სტრიქონზე გრძელი არ უნდა იყოს.

სწორად დაწერილი სახელების მაგალითებია:

total_number_of_fish TotalNum Totalnum temp5 _var94

გლობალური და ლოკალური ცვლადები

გლობალური ცვლადებით შეიძლება ვისარგებლოთ ყველა იმ სცენარში, რომლებიც შედის მოცემული **HTML**-დოკუმენტის შემადგენლობაში.

ლოკალური ცვლადების მოქმედების არე კი იმ ფუნქციის ფარგლებს არ სცილდება, რომელშიც მოხდა მათი შექმნა.

მაშასადამე, გლობალური ცვლადები უნდა გამოვაცხადოთ მთავარ სცენარში. ამ მიზნით, შეიძლება გამოვიყენოთ **var** საკვანძო სიტყვა ან მხოლოდ მინიჭების ოპერატორი:

```
var students = 25;
    students = 25;
```

ეს ოპერატორები ერთმანეთის ტოლფასია, თუმცა, კოდის უკეთ აღქმის თვალსაზრისით, უპირატესობას პირველ ხერხს ვანიჭებთ.

რაც შეეხება ლოკალური ცვლადების გამოცხადებას, როგორც აღვნიშნეთ, ეს ხდება ფუნქციაში და, როგორც წესი, ვირჩევთ პირველ ხერხს. ლოკალური ცვლადის **var** საკვანძო სიტყვით გამოცხადება თავიდან გვაცილებს კონფლიქტური სიტუაციის წარმოქმნის საშიშროებას *(მხედველობაში გვაქვს ისეთი შემთხვევები, როცა მოცემული სახელის ცვლადი ადრე უკვე გამოცხადებული იყო, როგორც გლობალური).*

ქვემოთ მოყვანილ ლისტინგში **name1** და **name2** გლობალური ცვლადები სათაურშივე განისაზღვრება, **who** ლოკალური ცვლადი კი იქმნება **Greet**() ფუნქციაში:

```
<HTML>
<HEAD>
<TITLE>ფუნქციების გამოყენების სხვა მაგალითი</title>
<STYLE>
H2, P  {font-family: LitNusx}
</style>
<SCRIPT LANGUAGE="JavaScript">
var name1="გიგი";
var name2="გაგა";
function Greet(who) {
   alert("დღეს პროგრამაშია " + who  );
   var name2="გურამი";
}
</script>
</head>
<BODY>
<H2>ვისწავლოთ ფუნქციებთან მუშაობა!</h2>
<P>შეტყობინება გამოდის ორჯერად</p>
<SCRIPT LANGUAGE="JavaScript">
Greet(name1);
Greet(name2);
</script>
</body>
</html>
```

მივაქციოთ ყურადღება - **Greet()** ფუნქციაში ხელმეორედ იქმნება **name**2 ცვლადი *(ეს ხდება **var** ბრძანებით),* ახალი **name**2 ცვლადი ლოკალური ტიპისაა და მისთვის მნიშვნელობის მინიჭება არ ცვლის **name**2 გლობალური ცვლადის მნიშვნელობას.

შევნიშნოთ, რომ ფუნქციის პარამეტრებიც ლოკალური ცვლადებია. საერთოდ კი, ნებისმიერი ცვლადი, რომელიც ცხადდება ფუნქციაში ან პირველად მასში გამოიყენება, ლოკალურად მიიჩნევა.

გლობალური ცვლადების სათაურშივე გამოცხადება დაუწერელი წესია. თუმცა კი დასაშვებია, ასეთი ცვლადები პროგრამის ნებისმიერ ადგილას გამოცხადდეს.

აღვნიშნოთ, რომ თუ ცვლადს ვიყენებთ "ოფიციალურად" გამოცხადებამდე *(ან ღია სახით მნიშვნელობის მინიჭების გარეშე),* მას განესაზღვრება ნულოვანი მნიშვნელობა.

JavaScript-ში ცვლადებისთვის მნიშვნელობების მისანიჭებლად დასაშვებია შემდეგი ოპერატორების გამოყენებაც:

lines += 1; **lines** ++;

lines –= 1; **lines** – –;

ისინი ეკვივალენტურია **lines** = **lines**+1 და **lines**=**lines**–1 ოპერაციების.

++ და – – ოპერატორები შეიძლება ცვლადის სახელის წინაც დავსვათ. აღვნიშნოთ, რომ სიტუაციიდან გამომდინარე, მათ შეიძლება რამდენადმე განსხვავებული როლიც დაეკისროთ. დავუშვათ, **lines** ცვლადის მნიშვნელობაა 40. **alert** (**lines**++) და **alert** (++**lines**) გამოსახულების შესრულება სხვადასხვა შედეგებს მოგვცემს:

I შემთხვევაში ჯერ ეკრანზე აისახება მნიშვნელობა 40 და შემდეგ **line** ცვლადი შემდეგ მიიღებს 41-ის ტოლ მნიშვნელობას;

II შემთხვევაში კი – ჯერ ცვლადი გახდება 41-ის ტოლი, ხოლო მერე, სწორედ, ეს შედეგი აისახება ეკრანზე.

მონაცემთა ტიპები JavaScript-ში

- რიცხვითი ტიპი. როგორც წესი, განისაზღვრება მინიჭების ოპერაციისას: **total**=31 ან **total**=3.91
- სტრიქონული ტიპი. თუ იმავე ცვლადს მივანიჭებთ მნიშვნელობას **total="tree",** მაშინ ცვლადის ტიპი სტრიქონულად გარდაიქმნება.
- ბულის ანუ ლოგიკური. იღებს **true** და **false** მნიშვნელობებს.
- ნულოვანი. ასეთი ტიპი ენიჭება გამოუცხადებელ ცვლადს, რომლის მნიშვნელობა განისაზღვრება **null** სიტყვით. სწორედ, ამ მნიშვნელობას იღებს გამოუცხადებელი **fig** ცვლადი შემდეგ ოპერატორში: **document.write(fig)**;
  *(იგულისხმება, რომ **fig** ცვლადი ამ ოპერატორამდე გამოცხადებული არ იყო).*

ყველა დასაშვები შემთხვევისთვის **JavaScript** ავტომატურად ახორციელებს მონაცემთა ერთი ტიპის სხვაში გარდაქმნას. ზემოთ უკვე განვიხილეთ ერთი ასეთი შემთხვევა **total** ცვლადის მაგალითზე. მოვიყვანოთ სხვებიც:

ვთქვათ, **total** ცვლადის მნიშვნელობა გახლავთ 40.

document.write (*"ჯამის სიდიდეა "* + **total**) ოპერატორი ეკრანზე გამოიყვანს შეტყობინებას:

ჯამის სიდიდეა 40

ვხედავთ, რომ ეს ოპერატორი მუშაობს მაშინაც, როცა **total** ცვლადის ტიპი არასტრიქონულია *(მაგალითად, რიცხვითი ან ბულის ტიპის).* ამ შემთხვევაში ხდება მისი გარდაქმნა სტრიქონულ ტიპად.

თუმცა გარდაქმნა ყოველთვის ვერ ხერხდება. მაგალითად, თუ **total** ცვლადი სტრიქონული ტიპისაა, მაშინ **average = total / 3** ოპერატორი ვერ შესრულდება. ასეთ შემთხვევებში მიმართავენ შემდეგ გარდაქმნის ფუნქციებს:

parseInt() – ტექსტური ტიპი გადაჰყავს მთელრიცხოვან ტიპში;

parseFloat() – ტექსტური ტიპი გადაჰყავს მცურავწერტილიან რიცხვით ტიპში.

თუ ცვლადი, გარდა რიცხვითი მნიშვნელობისა, მარჯვნივ შეიცავს სხვა, ტექსტურ სიმბოლოებსაც, ხდება მათი იგნორირება. მაგალითად:

stringvar = ”**30** დათვი”;

numvar = parseInt (stringvar);

ოპერატორების შესრულების შედეგად **numvar** მიიღებს 30-ის ტოლ მნიშვნელობას.

ცვლადებისათვის მომხმარებლების მიერ მნიშვნელობების მინიჭება

აღნიშნული მიზნით გამოიყენება **prompt** ფუნქცია. ვაჩვენოთ ამ ფუნქციის გამოყენება შემდეგი კოდის მაგალითზე:

```
<HTML>
<HEAD>
<TITLE>მომხმარებელთან დიალოგის მაგალითი</title>
<STYLE>
H2, H3, P  {font-family: LitNusx}
</style>
</head>
<BODY>
<H2>ვისწავლოთ ფურცლის აწყობა!</h2>
<P>მიღებულია ინფორმაცია: </p>
<SCRIPT LANGUAGE="JavaScript">
saxeli=prompt("შეიტანეთ თქვენი სახელი");
gvari= prompt("შეიტანეთ თქვენი გვარი");
furclis_satauri= prompt("შეიტანეთ ფურცლის სათაური");
document.write(“<H2>” + furclis_satauri + “</h2>”);
document.write(“<H3>” + saxeli +  ” “ + gvari + “</h3>”);
</script>
<P>ფურცელი იმყოფება აწყობის სტადიაში.</p>
</body>
</html>
```

სტრიქონული მონაცემები (ტექსტი)

უფრო დაწვრილებით შევისწავლოთ სტრიქონული მონაცემები ანუ ტექსტი. როცა ცვლადს ვანიჭებთ ამა თუ იმ ტექსტურ მნიშვნელობას, **JavaScript** ქმნის ე.წ. **String** ობიექტს. აღვნიშნოთ, რომ ასეთი ობიექტის შექმნა უშუალოდაც შეიძლება. ვაჩვენოთ ორივე გზა ქვემოთ მოყვანილი ოპერატორების მაგალითზე:

test = ”ეს ტესტია”;

test = **new String** (“ეს ტესტია”);

შედეგად შეიქმნება ტოლფასი სტრიქონული ობიექტები. ობიექტში, გარდა მნიშვნელობისა, ინახება ინფორმაცია სტრიქონის სიგრძის შესახებაც **length** თვისების სახით. ვაჩვენოთ ამ თვისების გამოყენების მაგალითი:

test = ”ეს ტესტია.”;

document.write (**test.length**);

აღსანიშნავია, რომ **test.length** ცვლადი რიცხვითი ტიპისაა. იგი შეიძლება გამოვიყენოთ მათემატიკურ ოპერაციებში.

ჩვენთვის უკვე ცნობილია, რომ ობიექტი შეიძლება შეიცავდეს მეთოდებსაც. კერძოდ, სიმბოლოების რეგისტრის შესაცვლელად **String** ობიექტი იყენებს ორ მეთოდს:

ToUpperCase() – ტექსტი გადაჰყავს ასომთავრულ რეგისტრში;

ToLower Case() – ტექსტი გადაჰყავს სტრიქონულ რეგისტრში.

მიუხედავად იმისა, რომ აღნიშნული მეთოდები *(ფუნქციები)* პარამეტრებს არ საჭიროებს, ფრჩხილების გამოყენება მაინც აუცილებელია.

გავეცნოთ სხვა მეთოდების დანიშნულებასაც:

substring() ტექსტიდან გამოჰყოფს საჭირო ნაწილს. დავიხსომოთ, რომ ტექსტის ინდექსაცია იწყება 0-დან. ვთქვათ, გვაქვს ცვლადი

alpha = ”**ABCDEFGHIJKLMNOPQRSTUVWXYZ**”;

alpha.substring (0.4) დაგვიბრუნებს **ABCD** მნიშვნელობას,

alpha.substring (10,12**)** დაგვიბრუნებს **KL**-ს,

alpha.substring (6,7) დაგვიბრუნებს **G**-ს,

alpha.substring (0,26) დაგვიბრუნებს მთელ ალფაბეტს,

alpha.substring (6,6) კი გვიბრუნებს ცარიელ სტრიქონს.

indexOf() მოგვიძებნის მოცემულ ტექსტში სიმბოლოების იმ დიაპაზონს *(ორი რიცხვის სახით),* რომელიც უკავია საძებნ სიტყვას. თუ ეს საძებნი სიტყვა ტექსტში რამდენჯერმე გვხვდება, მეორე პარამეტრში შეიძლება მივუთითოთ, მერამდენე სიმბოლოდან უნდა დაიწყოს ძებნის პროცესი. მაგალითად, შემდეგი გამოსახულება სიტყვა “მთას” **temp** სტრიქონულ ობიექტში მოგვიძებნის მე-20 სიმბოლოდან:

loc=temp.indexOf(“მთა”, 19);

აქვე მივუთითოთ, რომ ძებნისას გაითვალისწინება სიმბოლოების რეგისტრი.

lastIndexOf() მეთოდი წინასაგან იმით განსხვავდება, რომ საჭირო ფრაგ-მენტების ძიებას ტექსტში იგი ბოლოდან იწყებს.

მასივები

მასივები, ბევრი სხვა ელემენტებისაგან განსხვავებით, **JavaScript**-ში უნდა გამოცხადდეს მათ გამოყენებამდე. მაგალითად:

score = **new Array**(30);

ინდექსირება აქაც ნოლიდან იწყება, ამიტომ ბოლო ელემენტის ინდექსი იქნება 29. ამის შემდეგ შესაძლებელია მნიშვნელობების მინიჭება ცალკეული ელემენტებისთვის. მაგალითად:

score[0] = 5;

score[25] = 9;

რიცხვების გარდა, მასივი შეიძლება შეიცავდეს სტრიქონებს, ობიექტებს და მონაცემთა სხვა ტიპებსაც. **length** თვისება, ცხადია, მასივებსაც ახასიათებთ.

ახლა გავეცნოთ სტრიქონული ცვლადის დაყოფის **split**() მეთოდს. დაყოფა ხდება რაიმე სიმბოლოს მიხედვით. განცალკევებული ნაწილების მნიშვნელობა კი მიენიჭება მასივის ელემენტებს. მაგალითად:

test=”მაკრატელი”;

parts=**test**.**split**(“ა”); ოპერატორების შესრულების შედეგად იქმნება სამელემენტიანი მასივი:

parts[0]=”მ”;

parts[1]=”კრ”;

parts[2]=”ტელი”;

და, პირიქით, **join**() მეთოდი მასივის ელემენტებს ერთ სტრიქონში გააერთიანებს:

Fullname = **parts.join**(“ა”);

თუ გამაერთიანებელი სიმბოლოს მითითება საჭირო არ გახლავთ, მაშინ აუცილებელია, პარამეტრად ვუჩვენოთ *ძძიძე*.

მასივის ელემენტებს სორტირებისთვის განკუთვნილია **sort**() მეთოდი. მაგალითად, თუ გვაქვს მასივი:

names[0]=”**Fred**”;

names[1]=”**George**”;

names[2]=”**Alex**”;

შემდეგი ოპერატორი შეგვიქმნის ახალ, ალფაბეტის მიხედვით მოწესრიგებულ მასივს:

sortednames=names.sort();

დავუბრუნდეთ ზემოთ მოყვანილ *მორბენალი სტრიქონის პროგრამას.* პირველ რიგში, უნდა მივიღოთ გადაწყვეტილება, რა შეტყობინება გამოგვყავს. ამ მიზნით ვიყენებთ **msg** სტრიქონულ ცვლადს. შემდეგ, განვსაზღვრავთ შეტყობინებების გამყოფი არის შემცველობას:

spacer=”. . . . . .”;

გვჭირდება კიდევ ერთი, რიცხვითი ცვლადი **pos**, რომელიც განსაზღვრავს იმ სიმბოლოს ნომერს, სადაც ხდება სტრიქონის “გაჭრა”. მისი საწყისი მნიშვნელობა ნულია.

მორბენალი შეტყობინების შექმნა ხორციელდება **scrollMessage()** ფუნქციის მეშვეობით.

როგორც კი **pos** ცვლადის მნიშვნელობა **msg** ცვლადის სიგრძეს გადააჭარბებს, იგი კვლავ ნულის ტოლ მნიშვნელობას იღებს და ყველაფერი თავიდან იწყება.

ამავე პროგრამაში გამოიყენება კიდევ ერთი მეთოდი **window**.**setTimeout**(), რომელიც განსაზღვრავს დროის პერიოდს **window**-ობიექტის განახლებისათვის.

პირობითი ოპერატორები

if ოპერატორი

გამოყენების მაგალითები:

if (**a** == 1) **window**.**alert** (“მოიძებნა ერთი ერთეული!”);

```
if (a==1) {
  window.alert (“მოიძებნა ერთეული!”);
  a=0;
}
```

ჩამოვთვალოთ **JavaScript**-ში გამოყენებული პირობითი ოპერატორები:

== *ტოლია* (და არა =)
!= *განსხვავდება*
< <= > >=

პირობების კომპაქტურად ჩასაწერად იყენებენ ლოგიკურ, ანუ ბულის ოპერატორებს. ესენია:

|| – ლოგიკური “ან” ოპერატორი,
&& – ლოგიკური “და” ოპერატორი,
! – უარყოფის ოპერატორი.

მოვიყვანოთ გამოყენების მაგალითები:

if (**phone** == ” “ || **email** == “ “) **window.alert** (*” შეცდომაა!”*);

if (**phone** == ” “ && **email** == “ “) **window.alert** (*” შეცდომაა!”*);

JavaScript იყენებს **else** ოპერატორსაც:

```
if ( a == 1) {
window.alert (”მოიძებნა 1 ერთეული!”);
a = 0;
}
else {
alert ( “ a ცვლადის არასწორი მნიშვნელობაა” + a );
}
```

პროგრამირების თანამედროვე ენებში დასაშვებია ვისარგებლოთ ასეთი კონსტრუქციითაც:

value = (**a** == 1) ? 1: 0;

იგი ტოლფასია შემდეგი ფრაგმენტის:

```
if ( a == 1)
value = 1;
else
value = 0;
```

if ოპერატორების წყებას ხშირად ცვლიან **switch** ოპერატორით. მოვიყვანოთ მაგალითი:

```
switch (button) {
case "მომდევნო გვერდი" :
window.location="next.html";
break;
case "წინა გვერდი" :
window.location="prev.html";
break;
case "საწყისი გვერდი" :
window.location="home.html";
break;
case "უკან დაბრუნება" :
window.location="back.html";
break;
default :
window.location="არა, მეგობარო, ამ ღილაკზე ხელის დაჭერით ვერ
                 გავფრინდებით!";
}
```

button ცვლადის თითოეული წინასწარ ცნობილი მნიშვნელობისთვის სრულდება განსაზღვრული მოქმედება *(მოცემულ შემთხვევაში გამოიძახება შესაბამისი **Web**-ფურცელი)* და შემდეგ **break** ოპერატორით ხდება კოდის ფრაგმენტის ბოლოში გადასვლა, **default** ოპერატორს კი გამოჰყავს დუმილით გათვალისწინებული **Web**-ფურცელი.

button ცვლადს მნიშვნელობა შეიძლება დიალოგის რეჟიმშიც მივანიჭოთ, რისთვისაც ვიყენებთ **prompt()** ოპერატორს:

```
button = window.prompt ("საით გავსწიოთ?");
```

მოვიყვანოთ **Web**-ფურცლის სრული კოდის მაგალითი, რომელშიც გამოყენებული იქნება **window.prompt** და **switch** ოპერატორები:

```
<HTML>
<HEAD>
<TITLE>მონაცემების შემოწმება</title>
<STYLE>
H2, P  {font-family: LitNusx}
```

```
</style>
</head>
<BODY>
<H2>მივცეთ მასებს საკუთარი არჩევანის გაკეთების უფლება!</h2>
<HR>
<SCRIPT LANGUAGE="JavaScript">
where = window.prompt (”დღეს რომელ საიტს ვესტუმროთ?”);
switch (where) {
      case “posta” :
window.location = “http://www.posta.ge”;
break;
      case “Microsoft” :
window.location = “http://www.microsoft.com”;
break;
      case “Yahoo” :
window.location =“http://www.yahoo.com”;
break;
      default :
window.location = “http://www.ftv.fr.”;
}
</script>
</body>
</html>
```

ციკლები

JavaScript-ენა რამდენიმე სახის ციკლს იყენებს:

- *ციკლი* ***for***

მოვიყვანოთ **for** ციკლის გამოყენების მაგალითი:

```
for ( i = 1;  i < 10;  i ++ ) {
document.write (“გამოგვყავს სტრიქონი ნომერი “, i, “<BR>”);
}
```

აღვნიშნოთ, რომ ციკლი *(მოცემულ შემთხვევაში სტრიქონის გამოყვანა)* 9-ჯერ შესრულდება.

- *ციკლი* ***while***

```
while ( total < 10 ) {
n ++;
total += values[n];
}
```

მოცემულ შემთხვევაში **values** მასივის წევრების შეკრება მანამდე გაგრძელდება, სანამ ჯამი ნაკლები იქნება 10-ზე.

იმავე შედეგს მოგვცემდა **for**-ოპერატორიანი შემდეგი ციკლიც:

```
for ( n = 0;  total < 10;  n ++ ) {
total += values[n];
}
```

- *ციკლი* ***do … while***

do … while ციკლი **while**-ციკლის ვარიანტს წარმოადგენს:

```
do {
n ++;
total += values[n];
}
while ( total < 10 );
```

სხვაობა ის გახლავთ, რომ **do … while** კონსტრუქციაში ციკლი ნებისმიერ შემთხვევაში ერთხელ მაინც სრულდება, რასაც **while**-ციკლში ადგილი არა აქვს.

- *ციკლი* ***for … in***

ამ ციკლის გამოყენება ფრიად მოსახერხებელია ობიექტების თვისებებსა და მასივების ელემენტებზე ოპერაციების ჩასატარებლად. **for … in** ციკლის მეშვეობით შესაძლებელია, ეკრანზე ავსახოთ, ვთქვათ, **navigator**-ობიექტის თვისებები:

```
for ( i in navigator ) {
document.write ( "თვისება: " + i );
document.write ("მნიშვნელობა " + navigator[i] );
}
```

ჩანს, რომ საჭირო აღარ არის **i** ცვლადისათვის საწყისი და საბოლოო მნიშვნელობების მინიჭება.

უსასრულო ციკლი. ციკლის შეწყვეტა-გავრძელება

ციკლის შემცველი კოდის დაწერისას ყურადღებით უნდა ვიყოთ, რათა შეცდომით უსასრულო ციკლში არ აღმოვჩნდეთ. ზოგჯერ ამგვარ ციკლს სპეციალურად ქმნიან - იგი შეწყდება მხოლოდ რაიმე პირობის შესრულებისას.

მოვიყვანოთ უსასრულო ციკლის მაგალითი, რომლის შეწყვეტა **break** ოპერატორის მეშვეობით მოხდება მაშინ, როცა მასივის რომელიმე ელემენტი 1-თან ტოლობის პირობას დააკმაყოფილებს:

```
while ( true) {
n ++;
```

```
if ( value[n] == 1 ) break;
}
```

ზოგჯერ რაიმე პირობის შესრულების შემდეგ მოითხოვება, ციკლი ისე გაგრძელდეს, რომ მოხდეს ციკლის ბოლომდე დარჩენილი ოპერატორების გამოტოვება. ამ მიზნით გამოიყენება **continue** ოპერატორი:

```
for ( i = 1;  i < 21;  i ++ )  {
if (score [i] == 0 ) continue;
document.write (“სტუდენტის ნომერია “, i, “ შეფასებაა: “, score [i],
“\n”);
}
```

20 სტუდენტიდან მხოლოდ მათ შესახებ დაიბეჭდება ინფორმაცია, რომლებმაც ჩააბარეს ტესტური გამოცდები.

დასასრულ, მოვიყვანოთ მაგალითი ჩვენ მიერ კომპიუტერთან დიალოგის რეჟიმში ნაჩვენები სახელების დანომრილი სიის სახით ეკრანზე გამოტანისა ერთ-ერთი ზემოთ განხილული ციკლის დახმარებით:

```
<HTML>
<HEAD>
<TITLE>ციკლის გამოყენების მაგალითი</title>
<STYLE>
H2, P  {font-family: LitNusx}
</style>
</head>
<BODY>
<H2>  ციკლის გამოყენების მაგალითი </h2>
<P> შეიტანეთ რამდენიმე სახელი. ისინი ეკრანზე დანომრილი სიის სახით
  აისახებიან.
</p>
<HR>
<SCRIPT LANGUAGE="JavaScript">
names =  new Array();
i = 0;
do {
next =  window.prompt (”შეიტანეთ  შემდეგი  სახელი”);
if ( next > “ “ ) names[i] = next;
i = i + 1;
}
while ( next > “ “ );
  document.write ( “<H2>” + “შეტანილია  “ +  (names.length) + “ სახელი”  +
                   “</h2>”);
  document.write (“<OL>“ );
  for (i in  names ) {
```

```
  document.write ("<LI>" +  names[i] + "<BR>" );
  }
  document.write ("</ol>" );
</script>
</body>
</html>
```

განსაკუთრებული ყურადღება მივაქციოთ სცენარში **HTML**-ოპერატორების აწყობის წესს.

ობიექტები

ჩვენ უკვე შეგვექმნა წარმოდგენა ობიექტებზე, მათ თვისებებსა და მეთოდებზე. ვიცით, რომ **JavaScript**-ში საქმე გვაქვს 3 სახის ობიექტებთან. სანამ დაწვრილებით გავეცნობოდეთ თითოეულ მათგანს, ვნახოთ, თუ როგორ ხდება ობიექტების შექმნა.

JavaScript ამ მიზნით იყენებს სპეციალურ ფუნქციებს, ე.წ. **კონსტრუქტორებს**. მაგალითად, ჩვენ შეგვიძლია შევქმნათ სტრიქონული ცვლადის სახის მქონე ობიექტი (*მიღებულია გამოთქმა – "ობიექტის ეგზემპლარი")* **String** ჩაშენებულ ფუნქციაზე დაყრდნობით:

myname = new String("ესეც ასე**!");**

new გასაღებური სიტყვა **JavaScript**-ს აცნობებს, რომ საჭიროა **String** ობიექტის ახალი ეგზემპლარის შექმნა. ეს იქნება სტრიქონული ცვლადი **myname** სახელით და **"**ესეც ასე**!"** მნიშვნელობით.

ამგვარივე წესით ახალი ეგზემპლარები შეიძლება შევქმნათ არა მარტო **String, Date, Array** და სხვა ჩაშენებული, არამედ ე.წ. **მომხმარებელთა ობიექტებისთვისაც**.

გამონაკლისს წარმოადგენს **Math** ჩაშენებული ობიექტი *(იხ. ქვემოთ).*

ობიექტები ხასიათდება ერთი ან მეტი თვისებით *(ატრიბუტით).*

თვისება გახლავთ ობიექტში შენახული, რაიმე მნიშვნელობის მქონე ცვლადი.

ჩვენ უკვე ვიცით, როგორ მივმართოთ თვისებას, მაგალითად, მასივის სიგრძეს:

names.length (names მასივის სახელია).

შევნიშნოთ, რომ თვისება, თავის მხრივ, შეიძლება თვითონაც წარმოადგენდეს ობიექტს. მაგალითად, მასივის თითოეული ობიექტი ამ მასივისთვის წარმოადგენს სპეციალური ტიპის თვისებას, რომელიც აღინიშნება ინდექსით.

ამრიგად, მასივის ელემენტის სიგრძე, ვთქვათ, **names[7].length,** ასე ვთქვათ**,** თვისების თვისების როლში გვევლინება.

მეთოდი გახლავთ ობიექტის თვისებათა ფუნქცია ანუ ოპერატორების ერთობლიობა, რომელთა შესრულების შედეგი ამავე ობიექტშივე შეინახება როგორც ერთ-ერთი თვისება.

მეთოდის გამოძახება ფუნქციის გამოძახების ანალოგიურად ხდება. მაგალითად:

value.toUpperCase();

*(**value** სტრიქონული ტიპის ცვლადის მნიშვნელობა მეთოდის შესრულების შემდეგ მხოლოდ პატარა ასოებით იქნება გამოსახული.)*

მეთოდის მიერ დაბრუნებული მნიშვნელობა *(შეგახსენებთ, მეთოდი ფუნქცია გახლავთ!)* შეიძლება რომელიმე ცვლადს მივანიჭოთ:

finish = Math.round(num);

*(**Math** ჩაშენებული ობიექტის **round** მეთოდი ამრგვალებს **num** ცვლადის მნიშვნელობას, რომელიც ენიჭება **finish** ცვლადს.)*

აღსანიშნავია, რომ **with** საკვანძო სიტყვის მეშვეობით შესაძლებელია კოდის გამარტივება:

```
with (lastname) {
window.alert ( “გვარის სიგრძეა: “ + length);
toUpperCase();
}
```

*(**length** თვისება და **toUpperCase()** მეთოდი კავშირდება **lastname** ობიექტთან.)*

Math ობიექტი

ზემოთ აღვნიშნეთ, რომ **Math** ჩაშენებული ობიექტისთვის საჭირო არ არის ახალი ეგზემპლარების შექმნა. ამ ობიექტის თვისებები მათემატიკური კონსტანტებია, ხოლო მეთოდები – მათემატიკური ფუნქციები.

გავეცნოთ **Math** ობიექტთან დაკავშირებულ, რამდენიმე ხშირად გამოყენებულ მეთოდს:

Math.ceil() – რიცხვი მრგვალდება მეტობით;

Math.floor() – რიცხვი მრგვალდება ნაკლებობით;

Math.round() – რიცხვი მრგვალდება უახლოეს მთელ რიცხვამდე.

რომელიმე ათობით ნიშნამდე *(მაგალითად, მეასედამდე)* რიცხვის დასამრგვალებლად შეიძლება ასე მოვიქცეთ:

```
function round(num) {
return Math.round (num*100)/100;
}
```

იგულისხმება, რომ **num** ცვლადს მინიჭებული აქვს რაიმე მნიშვნელობა. **round** ფუნქციის გამოძახების შემდეგ აღნიშნული მნიშვნელობა ორ ათობით თანრიგამდე დამრგვალდება.

მსგავს გადაწყვეტას შეიძლება მივმართოთ **1–a** დიაპაზონში შემთხვევითი რიცხვის გენერირებისათვის:

```
function rand(num) {
return  Math.floor ( Math .random()*num) + 1;
}
```

Math.random() მეთოდით ხდება 0–1 დიაპაზონში შემთხვევითი რიცხვის მიღება, **Math.floor()** მეთოდით შესაბამისი ნამრავლის ნაკლებობით დამრგვალება. დაბოლოს, **rand** ფუნქციას უბრუნდება მიღებული სიდიდის ერთით გადიდებული მნიშვნელობა.

დავწეროთ შემთხვევითი რიცხვების გენერატორის პროდუქტის "შემთხვევითობაზე" შემოწმების კოდი:

```
<HTML>
<HEAD>
<TITLE>შემთხვევითი რიცხვების გენერატორი</title>
<STYLE>
H2, P  {font-family: LitNusx}
</style>
</head>
<BODY>
<H2> გენერატორის შემოწმება </h2>
<P> რამდენად შემთხვევითია შემთხვევითი რიცხვების გენერატორით
  მიღებული რიცხვები? გამოვთვალოთ 100 000 ასეთი რიცხვის საშუალო
  მნიშვნელობა.
<HR>
<SCRIPT LANGUAGE="JavaScript">
total = 0;
  for (i = 0;  i < 100000; i++) {
  num = Math.random();
total += num;
if (( i/10000-Math.round(i/10000)) ==0)
window.status = "generirebulia " + i + " ricxvi" ;
  }
  average = total/100000;
  average = Math.round (average*1000) / 1000;
  document.write ("<H2> შემთხვევითი რიცხვების საშუალო
არითმეტიკული ტოლია: " + average + "</h2>");
  window.status = "გენერირებულია " + i + " რიცხვი" ;
</script>
```

```
</body>
</html>
```

Date ობიექტი

ჩვენ უკვე გავეცანით ამ ჩაშენებულ ობიექტს. მისი შექმნა სხვა ობიექტების ანალოგიურად ხდება. დასაშვებია ამ დროს ობიექტისთვის მნიშვნელობის განსაზღვრაც:

```
birthday = new Date();
birthday = new Date("Jan 20 2002 11:00:0");
birthday = new Date(5, 26, 2002);
birthday = new Date(5, 26, 2002, 11, 0, 0);
```

აღსანიშნავია, რომ **Date** ობიექტს არც ერთი თვისება არ ახასიათებს, რის გამოც შექმნილი ობიექტისთვის მნიშვნელობის მისანიჭებლად ან მინიჭებული მნიშვნელობის გასაგებად იყენებენ ქვემომოყვანილ მეთოდებს:

ა) **setDate() –** განსაზღვრავს დღის აღმნიშვნელ რიცხვს;
setMonth() – განსაზღვრავს თვეს;
setYear() – განსაზღვრავს წელიწადს;
setTime() – განსაზღვრავს პერიოდს მილიწამებში 1970 წლის 1 იანვრიდან;
setHours(), setMinutes და **setSeconds** დროის შესაბამისი სიდიდეების განსაზღვრისთვის გამოიყენებიან.

ბ) რაც შეეხება თარიღის ტიპის ობიექტებიდან ინფორმაციის მიღებას, ამ მიზნით გამოიყენება მსგავსი მეთოდები, რომლებშიც **set** თავსართი შეცვლილია **get-**ით**.**

მაგალითად, უკვე გამოცხადებული **Date** ტიპის მქონე **holiday** ობიექტისთვის წლის მნიშვნელობის განსაზღვრა და იმავე მნიშვნელობის შესახებ ინფორმაციის მიღება მოხდება შემდეგი ოპერატორების მეშვეობით:

```
holiday. setYear(2002);
holiday. getYear();
```

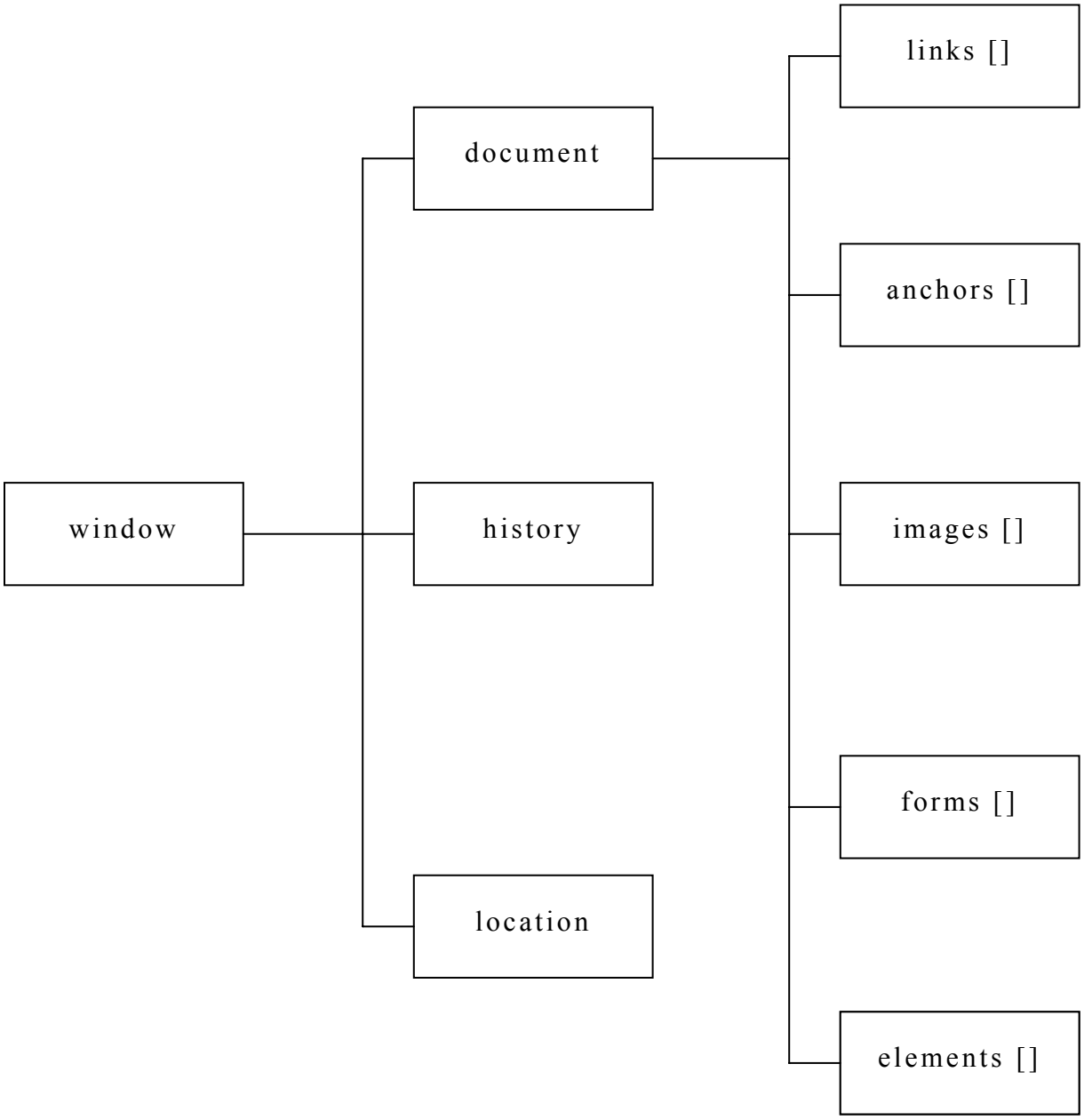
Date ობიექტის მნიშვნელობის ჩაწერის ფორმატის შესაცვლელად გამოიყენება შემდეგი ორი მეთოდი:

Date.parse() – ეს მეთოდი **Date** ტიპის ობიექტის მნიშვნელობის ტექსტური სახით ჩაწერას ცვლის მილიწამების რიცხვით, რომლებიც ათვლილია 1/1/1970 თარიღიდან.

Date.UTC() – ახორციელებს უკუგარდაქმნას.

ბროუზერის ობიექტების მოდელთან მუშაობა

ბროუზერის ობიექტების მართვა **JavaScript**-ის დიდი ღირსება გახლავთ. ეს ობიექტები შემდეგი იერარქიული სტრუქტურის სახით შეიძლება წარმოვადგინოთ:



შევნიშნოთ, ჩვენ განვიხილავთ დოკუმენტის ე.წ. ნულოვანი დონის ობიექტურ მოდელს. ინტერნეტისთვის სტანდარტების შემმუშავებელ **W3C** კონსორციუმს **(W3C – World Wide Web Consortium)** დამტკიცებული აქვს **DOM1** და **DOM2** დონეთა შესატყვისი სტანდარტებიც.

ზემოთ, ნახაზზე ასახულია ბროუზერის მხოლოდ უმნიშვნელოვანესი ობიექტები.

იერარქიული სტრუქტურის სათავეში იმყოფება **window**-ობიექტი. ჩვენ უკვე ვიცნობთ მის ზოგიერთ თვისებასა და მეთოდს. ესენია: **window.status** თვისება და **window.alert(), window.confirm(), window.prompt()** მეთოდები.

დასაშვებია, ბროუზერს ერთდროულად გავახსნევინოთ რამდენიმე ფანჯარა. აღსანიშნავია, რომ ფრეიმებიც და შრეებიც **window**-ობიექტის სახესხვაობებს წარმოადგენს.

გადავიდეთ **document**-ობიექტზე. მისი სხვა სახელწოდებებია **Web**-დოკუმენტი და **Web**-ფურცელი.

დავუშვათ, ეკრანზე ერთდროულად გამოგვყავს რამდენიმე ფანჯარა და გვსურს, რომელიმე მათგანისთვის გამოვიყენოთ ჩვენთვის უკვე ცნობილი **document.write()** მეთოდი. ცხადია, ასეთ შემთხვევაში აუცილებელი გახდება ფანჯრის სახელის დაკონკრეტება - **window.document.write().** აღვნიშნოთ, რომ მსგავსი როლი აკისრია **document.writeln()** მეთოდსაც, მხოლოდ მისი შესრულების შემდეგ ეკრანზე ინფორმაცია ახალი სტრიქონიდან გამოდის.

გავეცნოთ დოკუმენტის სხვა მნიშვნელოვან თვისებებსაც:

URL – გვიჩვენებს ბროუზერში ჩატვირთული მიმდინარე **Web**-ფურცლის მისამართს. ცხადია, ჩვენ მიერ ამ თვისების შეცვლა არ შეიძლება, მაგრამ თუკი საჭიროა ბროუზერში სხვა ფურცლის გამოყვანა, ვიყენებთ **window.location** ობიექტს *(იხ. ქვემოთ).*

title – შეიცავს მიმდინარე **Web**-ფურცლის სათაურს **<TITLE> </title>** დესკრიპტორული წყვილის შიგნით.

referrer – გვიჩვენებს წინა ფურცლის მისამართს, რომელზეც, როგორც წესი, ფიგურირებს მიმდინარე ფურცელზე დაყრდნობა *(ჰიპერკავშირი).*

lastModified – შეიცავს **Web**-ფურცლის ბოლო კორექტირების თარიღს. ეს მონაცემი ინახება სერვერზე და ბროუზერს გადაეგზავნება **Web**-ფურცელთან ერთად *(აქვე შევნიშნოთ, რომ ზოგი სერვერი ამ სერვისს არ უზრუნველყოფს).* ერთ-ერთ პროგრამაში ჩვენ გავეცანით, თუ როგორ შეიძლება **Web**-ფურცლის ბოლო კორექტირების თარიღის ეკრანზე გამოყვანა.

document-ობიექტში, **window**-ისაგან განსხვავებით, **open** და **close** მეთოდები დოკუმენტის თუ ფანჯრის გაღება-დახურვას კი არ ემსახურება, არამედ **open**-ით ხდება დოკუმენტის გასუფთავება და მისი მომზადება **write (writeln)** მეთოდებით ტექსტური ნაკადის ეკრანზე გამოყვანისთვის. რეალურად კი ეს პროცესი მაშინ იწყება, როცა **JavaScript**-ინტერპრეტატორი კოდში **close** მეთოდს იპოვის.

document.open ბრძანებაში შესაძლებელია ეკრანზე გამოსაყვანი მონაცემების ტიპის განსაზღვრაც. ქსელში გადასაგზავნი დოკუმენტი უმეტესწილად **HTML**-ტიპისაა *(შეესატყვისება **text/html** ტიპს).*

დაყრდნობები, ანკერები

document-ობიექტს შეიძლება ჰქონდეს შვილობილი ობიექტებიც. განვიხილოთ ორი მათგანი: **anchor** და **link**.

ანკერი **HTML**-დოკუმენტის იმ ადგილს მონიშნავს, რომელზეც შესაძლებელია დაყრდნობის (**link**) მეშვეობით გადავიდეთ. მოვიყვანოთ ანკერ-ობიექტის მაგალითი:

<A NAME="top">

ამ ანკერზე გადასვლა კი განხორციელდება შემდეგი დაყრდნობა-ობიექტის მეშვეობით:

<A HREF="#top">

დაყრდნობა-ობიექტით გადასვლა შესაძლებელია არა მარტო მიმდინარე **Web**-ფურცლის ფარგლებში, არამედ სხვა ფურცლის ნებისმიერ უბანზეც.

რადგანაც დაყრდნობები დოკუმენტში საკმაოდ ბევრი შეიძლება იყოს, მათი მართვისთვის მიზანშეწონილია **links** მასივის გამოყენება. ასევე, ანკერების სამართავად განკუთვნილია **anchors** მასივი.

შესაბამისი თვისებით შეიძლება მივიღოთ ინფორმაცია ამ მასივების სიგრძის შესახებ:

document.links.length და **document.anchors.length.**

links მასივის თითოეული ელემენტი ხასიათდება შემდეგი თვისებებით: დაყრდნობის ნომერი, სახელწოდება, **Web**-ფურცლის მისამართი.

თუ გვაინტერესებს, რომელი ფურცლის მისამართი ფიგურირებს **links** მასივის, ვთქვათ, პირველ ელემენტში, რომელიმე ცვლადს ამგვარად მივანიჭებთ შესაბამის მნიშვნელობას:

link1 = links[0].href

location ობიექტი

window-ობიექტის შვილობილია **location**-ობიექტიც. ახალი **Web**-ფურცლის ჩატვირთვა მხოლოდ ამ ობიექტის **href** თვისების გამოყენებით ხორციელდება. მაგალითად:

window.location.href = "http://home.netscape.com";

თუ საჭიროა, შეგვიძლია **URL**-მისამართის ნაწილის შესახებაც მივიღოთ ინფორმაცია. მაგალითად, მისამართის პროტოკოლური ნაწილი *(უმეტეს შემთხვევაში ეს გახლავთ **http**:)* ინახება **location.protocol** თვისებაში.

მართალია, **location.href** თვისება იმავე **URL**-მისამართს შეიცავს, რომელიც არის **document.URL** თვისების მნიშვნელობა, მაგრამ, რადგანაც უკანასკნელის შეცვლა არ შეიძლება, ახალი **Web**-ფურცლის გამოსაძახებლად მიმართავენ მხოლოდ **location**-ობიექტს.

location.reload მეთოდით დოკუმენტი ბროუზერში ხელახლა ჩაიტვირთება.

history ობიექტი

history-ობიექტიც **window**-ობიექტის შვილობილების რიცხვში შედის. იგი იმ მასივის სახით ინახება, რომლის თითოეული ელემენტი შეიცავს უკვე ნანახი **Web**-ფურცლების **URL**-მისამართს. მიმდინარე ფურცლისთვის გათვალისწინებულია მასივის პირველი ელემენტი – **history[0].**

history-ობიექტი 4 თვისებით ხასიათდება:

history.length – ერთი სეანსის მანძილზე ჩათვალიერებული ფურცლების რიცხვი;

history.current – მიმდინარე ფურცლის **URL**-მისამართი;

history.next – ფურცლის **URL**-მისამართი, რომელზეც მოვხვდებით ბროუზერის ღილაკების პანელში მყოფ **Forward** ღილაკზე დაწკაპუნების შემდეგ;

history.previous – შესაბამის ფურცელზე გადავალთ, თუ **Back** ღილაკზე დავაწკაპუნებთ.

რაც შეეხება მისამართების მასივში ნებისმიერი სასურველი ფურცლის მისამართის არჩევასა და გადასვლის განხორციელებას, ეს მიიღწევა **history.go(n)** და **history.go(-n)** მეთოდების გამოყენებით. მაგალითად, შესაძლებელია ეკრანზე გამოვიყვანოთ ნახატები – ერთ მათგანზე ასახული იქნება მარჯვნივ, ხოლო მეორეზე - მარცხნივ მიმართული ისრები. დავწეროთ კოდი, რომლითაც ამ ისრებზე დაწკაპუნებისას **history.go(+1)** და **history.go(-1)** მეთოდებით მიიღწევა **Forward** და **Back** ღილაკების გამოყენების ეფექტი:

```
<HTML>
<HEAD>
<TITLE>Back და Forward ღილაკების  ანალოგების  შექმნა</title>
<STYLE>
H2, P  {font-family: LitNusx}
</style>
</head>
<BODY>
<H2>Back და Forward ღილაკების  ანალოგები</h2>
<HR>
<P>მოვინახულოთ უკვე უკვე ნანახი ფურცლები!
<HR>
<A HREF="javascript:history.go(-1);">
      <IMG border=0 src="left.gif">
</a>
<A HREF="javascript:history.go(1);">
      <IMG border=0 src="right.gif">
</a>
<HR>
```

```
</script>
</body>
</html>
```

მივაქციოთ ყურადღება – დაყრდნობებში გამოყენებულია **javascript:URL** და **history.go** მეთოდების კომბინაცია. ამრიგად, შეიძლება **JavaScript**-ის ოპერატორებს **<SCRIPT> <script>** წყვილის გარეშეც მივმართოთ. ნახატების ასახვა კი ხდება **HTML** ენის საშუალებებით.

მომხმარებლის ობიექტების შექმნა

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**-ობიექტებს. ცხადია, ფუნქციის სახელადაც **Card**-ს ვირჩევთ. ასევე, ლოგიკურია ერთნაირი ან მსგავსი სახელები ჰქონდეს ობიექტის თვისებებსა და კონსტრუქტორის შესაბამის პარამეტრებს. მხოლოდ 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. რაც შეეხება ახალ ობიექტებს *(ანუ ობიექტის ეგზემპლარებს)*, თითოეულს ექნება ნებისმიერი ინდივიდუალური სახელი. ობიექტის ეგზემპლარი ასე შეიძლება შევქმნათ:

holmes = new Card (“შერლოკ ჰოლმსი”, “221**B** 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”;

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;
```

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));
</script>
```

```
</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:

```
<HTML>
<HEAD> <TITLE> პირადი ბარათები </title>
<STYLE>
H2, P  {font-family: LitNusx}
</style>
<SCRIPT LANGUAGE="JavaScript">
    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 ("<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", "37-37-37", "39-11-12");
lia= new Card ("ლია ბერიძე", "რუსთაველის  25", "93-23-34", "36-45-55");
tea= new Card ("თეა გიგაური", "წერეთლის  47", "34-34-34", "95-23-89");
```

```
// 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 ( );

<u>ხდომილობების დამუშავება</u>

კვლავ დავუბრუნდეთ ხდომილობების დამუშავების საკითხს. ვიცით, რომ ამა თუ იმ ხდომილობის შემდეგ შეიძლება შეიცვალოს პროგრამის შესრულების მიმდინარეობა.

<u>ხდომილობათა დამმუშავებელი</u> ეწოდება სცენარს, რომელსაც შეუძლია ხდომილობების დაფიქსირება და განსაზღვრული მოქმედებების შესრულება.

ხდომილობის მაგალითად შეიძლება მოვიყვანოთ თაგვის მაჩვენებლის მოთავსება **Web**-ფურცლის რომელიმე ობიექტზე – **MouseOver,** ხოლო ამ ხდომილობის დამმუშავებლის სახელი იქნება **onMouseOver.** ანალოგიური წესით იქმნება სხვა ხდომილობების და მათი დამმუშავებლების სახელწოდებანი.

მართალია, ხდომილობების აღმოჩენა და დამუშავება **JavaScript**-ის პრეროგატივა გახლავთ, მაგრამ ეს პროცესი არ მოითხოვს <**SCRIPT**> დესკრიპტორული წყვილის გამოყენებას – ხდომილობების დამმუშავებელს **HTML** ტეგში ათავსებენ:

< A HREF=”http://posta.ge/”
onMouseOver=”window.alert (‘ვესტუმროთ ფოსტას?’**);”>***დააწკაპუნეთ აქ*
</**a**>

შევნიშნოთ, რომ დასაშვებია “ ” და ‘ ’ ბრჭყალებმა ადგილები გაცვალოს, მაგრამ ერთმანეთში ჩადგმისას ორივე სახის ბრჭყალების გამოყენებაა საჭირო.

როცა ხდომილობის დამუშავება რამდენიმე ოპერაციის შესრულებას მოითხოვს, დასაშვებია, ხდომილობების დამმუშავებელში ისინი ერთმანეთისაგან

წერტილ-მძიმეებით განვაცალკეოთ. მაგრამ, საერთოდ, უმჯობესია ფუნქციით სარგებლობა, რომელიც განლაგდება <**HEAD**> უბანში და, ვთქვათ, ასეთი სახით გამოიძახება:

<**A HREF=”# bottom” onMouseOver=”Dolt ();”**> *თაგვის მიმთითებელი აქ მოათავსეთ!* </**a**>

აღვნიშნოთ, რომ ფუნქცია ნებისმიერ ადგილას შეიძლება გამოცხადდეს. ამასთან, დასაშვებია მისი, როგორც მეთოდის, გამოძახება ისეთი ობიექტებისათვის, რომლებისთვისაც შეიძლება ადგილი ჰქონდეს ამა თუ იმ ხდომილობას. მოვიყვანოთ მაგალითი:

```
function mousealert()  {
   alert  (“თქვენ დააწკაპუნეთ ღილაკზე”);
}
 document.onMouseDown =  mousealert;
```

event ობიექტი

event არის **JavaScript**-ში ჩაშენებული სპეციალური ობიექტი, რომლის თვისებები ამა თუ იმ ხდომილობის მომენტში ღებულობს სიტუაციის შესაბამის მნიშვნელობებს, რის შემდეგაც **event**-ობიექტი პარამეტრის სახით გადაეცემა ხდომილობის დამმუშავებელს. ჩამოვთვალოთ **event**-ობიექტის თვისებები:

- **type** – ხდომილობის ტიპი მაგალითად, **mouseover**.
- **target** – მიზნობრივი ობიექტი ხდომილობისათვის *(დოკუმენტი, კავშირი და სხვ.).*
- **which** – დაჭერილი კლავიატურის კლავიშის ან თაგვის ღილაკის ნომერი.
- **modifiers** – ხდომილობის გამომწვევი მართვის კლავიშების (*მაგალითად*: **Alt**, **Shift** ან **Ctrl**) ნომერი.
- **data** გახლავთ **drag&drop** ხდომილობისას გადაადგილებული მონაცემების სია.
- **pageX** და **pageY** – თაგვის მაჩვენებლის მდებარეობის განმსაზღვრელი კოორდინატები *(კოორდინატთა სათავე იმყოფება ფურცლის მარცხენა ზედა მხარეში).*
- **layerX** და **layerY** – იგივე მონაცემები შრისათვის.
- **screenX** და **screenY** – იგივე მონაცემები ეკრანისთვის.

თაგვთან დაკავშირებული ხდომილობების დამმუშავებელნი

ჩვენ უკვე ვიცნობთ **onMouseOver** ხდომილობის დამმუშავებელს. **OnMouseOut** მისი საპირისპიროა – იგი გამოიძახება ობიექტის ზონიდან თაგვის მაჩვენებლის გაყვანისას.

OnMouseMove ხდომილობის დამმუშავებელი დუმილით გამორთულია. საჭიროების შემთხვევაში მისი გააქტიურება ხდება ე.წ. ხდომილობის დაფიქსირების მეთოდით *(იხ. ქვემოთ).*

onClick დამმუშავებლის გამოძახება ობიექტზე დაწკაპუნებისას ხდება. მოვიყვანოთ მისი გამოყენების მაგალითი:

<**A HREF** = http://posta.ge **onClick**=”**alert***(‘თქვენ ტოვებთ ამ საიტს!’)*;”> *ფოსტის დასათვალიერებლად დააწკაპუნეთ აქ!*
</**a**>

,,ფოსტის დასათვალიერებლად დააწკაპუნეთ აქ” ტექსტზე დაწკაპუნების შემდეგ გამოდის შეტყობინება *,,თქვენ ტოვებთ ამ საიტს!”* და ჩვენ ისღა დაგვრჩენია, დავაწკაპუნოთ **OK** ღილაკზე, რასაც მოჰყვება ახალი **web**-ფურცლის გამოძახება.

კოდის ზედა ფრაგმენტი შეიძლება იმგვარად გარდავქმნათ, რომ დაწკაპუნების შემდეგაც გვქონდეს გადაფიქრების შესაძლებლობა:

<**A HREF**=”**http**://**posta**.**ge**” **onClick**=”**return** (**window.confirm** (‘დარწმუნებული ხართ?’));”> *დააწკაპუნეთ აქ* </**a**>

მოცემულ მაგალითში **alert**()-ის ნაცვლად გამოიყენება **return**() ფუნქცია, რომელიც, წინამორბედისაგან განსხვავებით, საშუალებას გვაძლევს, **Cancel** ღილაკზე დაწკაპუნებით უარი ვთქვათ საიტის დატოვებაზე.

JavaScript-ს შეუძლია დააფიქსიროს ღილაკზე ხელის დაჭერის და აშვების ხდომილობებიც და დაამუშაოს ისინი **onMouseDown** და **onMouseUp** საშუალებებით. ამ დამმუშავებლებისათვის *(ცხადია, **onClick**-სთვისაც)* **which** თვისებით შეიძლება განისაზღვროს, რომელ კლავიშზე მოხდა ხელის დაჭერა – მარცხენას შეესაბამება რიცხვი ,,1”, ხოლო მარჯვენას – ,,2”.

მოვიყვანოთ კოდის ფრაგმენტი, რომელშიც სხვადასხვა ღილაკზე ხელის დაჭერის შემთხვევებისთვის გათვალისწინებულია ეკრანზე სხვადასხვა შეტყობინების გამოყვანა:

```
function mousealert (e) {
whichone = (e.which == 1)? “მარცხენა” : ”მარჯვენა”;
message=”თქვენ დააწკაპუნეთ თაგვის ” + whichone + ” ღილაკზე”;
alert(message);
}
document.onMouseDown = mousealert;
```

onLoad ხდომილობის დამმუშავებელი

document-ობიექტის სერვერიდან გადმოტვირთვის დამთავრების შემდეგ შეიძლება შესაბამისი ხდომილობის დამმუშავებლის გამოძახება. მას <**BODY**> დესკრიპტორში ათავსებენ. მაგალითად:

<**BODY onLoad**=”**alert** (‘ჩატვირთვა დამთავრდა’);”>

ამ დამმუშავებლის გამოძახება დოკუმენტის ეკრანზე გამოსვლის შემდეგ ხდება. ამის გამო აზრი ეკარგება მასში **document.write** და **document.open** ოპერატორების გამოყენებას (*ასეთ შემთხვევაში აღნიშნული დოკუმენტი ეკრანიდან გაქრებოდა).*

დაყრდნობის აღწერის დამატება

ამა თუ იმ დაყრდნობაზე თაგვის მიმთითებლის მოთავსებისას ხშირად მიმართავენ იმ მოსალოდნელი ქმედების აღწერას, რაც შედეგად მოჰყვება დაწკაპუნებას. მაგალითად, შეიძლება სტატუსის სტრიქონში იმ საიტის შესახებ გამოვიყვანოთ მეტ-ნაკლებად ვრცელი ინფორმაცია, რომელზეც ვაპირებთ გადასვლას. ამ შემთხვევაში ვიყენებთ **onMouseOver** და **onMouseOut** ხდომილობების დამმუშავებლებს. უკანასკნელი გვაწვდის სტატუსის სტრიქონში ძველი ინფორმაციის აღდგენის საშუალებას იმ მომენტისათვის, როცა თაგვის მიმთითებელი სხვა ზონაში გადაინაცვლებს. როგორც წესი, ეს გახლავთ მიმდინარე **web**-ფურცლის **URL** მისამართი.

სტატუსის სტრიქონში დაყრდნობის აღმწერი ტექსტის გამოყვანა-ამოგდებისათვის უმჯობესია ფუნქციების გამოყენება. ქვემომოყვანილ მაგალითში ამ მიზნების განხორციელებას ემსახურება **describe(text)** და **clearstatus()** ფუნქციები:

```
<HTML>
<HEAD>
<TITLE>ჰიპერკავშირების აღწერა</title>
<SCRIPT LANGUAGE = "JavaScript">
function describe (text) {
window.status = text;
return true;
}
function clearstatus () {
window.status = “ “;
}
</script>
<STYLE>
H2, P, UL {font-family: LitNusx}
</style>
</head>
```

```
<BODY>
<H2> ჰიპერკავშირების აღწერა </h2>
<P> თაგვის მაჩვენებელი მოათავსეთ ჰიპერკავშირებზე მათი აღწერების
    გამოსაყვანად!
</p>
<UL>
<LI><A HREF = “order.html”
    onMouserOver = “describe(‘შეუკვეთეთ საქონელი’); return true;”
    onMouseOut = “clearstatus()”;>
საქონლის შეკვეთა</a>
<LI><A HREF = “email.html”
    onMouserOver = “describe(‘წერილის გაგზავნა’); return true;”
    onMouseOut = “clearstatus()”;>
ელექტრონული ფოსტა</a>
<LI><A HREF = “adm.html”
    onMouserOver = “describe(‘წინადადებები’); return true;”
    onMouseOut = “clearstatus()”;>
ადმინისტაციისადმი მიმართვა</a>
</ul>
</body>
</html>
```

describe ფუნქციაში **return true** ოპერატორი საშუალებას იძლევა, სტატუსის სტრიქონში ადგილი არ ჰქონდეს **URL** მისამართის ნაცვლად **text** შეტყობინების გამოყვანას.

ლიტერატურა

1. Освой самостоятельно JavaScript за 24 часа. Майкл Монкур, «Вильямс» , 2002.

2. Э. Кингсли-Хью, К. Кингсли-Хью, JavaScript 1.5, Учебний курс, «Питер» , 2001.

3. А. Гончаров. Самоучитель HTML, «Питер» , 2001.

4. Учебный курс «Компьютерные сети» , Microsoft Press. Санкт-Петербург, 1999.

ზოგიერთი ტერმინის განმარტება

HTML (HyperText Markup Language – *ჰიპერტექსტის მონიშვნის ენა***) –** **Web**-დოკუმენტების მონიშვნის *(გაწყობის, დაფორმატების)* ენა.

Web-ფურცელი - **Web**-სერვერზე შენახული დოკუმენტი *(უმეტესწილად **HTML** ფორმატის მქონე),* რომელიც სერვერიდან ჩამოიტვირთება კლიენტის **Web**-ბროუზერში. შეიძლება განთავსდეს ლოკალურ კომპიუტერზეც.

Web-ბროუზერი – პროგრამა, რომლის მეშვეობითაც ხორციელდება **WWW**-ში ინფორმაციის მოძიება და ჩათვალიერება.

Internet Explorer – კორპორაცია **Microsoft**-ის მიერ შექმნილი **Web**-ბროუზერი.

Navigator – კორპორაცია **Netscape Communications**-ის მიერ შექმნილი **Web**-ბროუზერი. ხშირად მას **Netscape**-საც უწოდებენ.

Web-კვანძი - **WWW**-დოკუმენტების კრებული. აქვს საწყისი ფურცელი, რომლიდანაც გადავდივართ კრებულის სხვა ფურცლებზე.

WWW (World Wide Web) – ინტერნეტის ყველაზე პოპულარული სამსახური, რომელიც მეტად აადვილებს ინფორმაციის ძებნის პროცესს და საერთოდ, ინტერნეტის სამყაროში მოგზაურობას.

ჰიპერტექსტი - ჩვეულებრივ ტექსტზე უფრო მეტი ინფორმაციული და ფუნქციური მონაცემების შემცველი დოკუმენტი.

ელემენტი - **HTML**-ის კონსტრუქცია – კონტეინერი, რომელიც შეიცავს ამა თუ იმ წესით დასაფორმატებულ *(ან რაიმე სხვა გზით დასამუშავებელ)* მონაცემებს.

აპლეტი (**Applet**) - პროგრამა, რომელიც **Web**-ფურცლის ჩათვალიერებისას დინამიურად მიუერთდება **HTML**-კოდს ფაილის სახით.

სცენარი (Script) - პროგრამა, რომელიც უმეტეს შემთხვევაში დაწერილია **JavaScript**-ის ერთ-ერთ ვერსიაში და წარმოადგენს **Web**-ფურცლის მაფორმირებელ **HTML**-კოდში ჩართულ კომპონენტს.

ფრეიმი – ეკრანის უბანი **Web**-ფურცლის ჩათვალიერების შესაძლებლობით. ეკრანის სტრუქტურის განსაზღვრა *(ვერტიკალზე ან ჰორიზონტალზე მისი დაყოფა ფრეიმებად)* ხორციელდება **cols** და **rows** ატრიბუტების მეშვეობით.

GIF – ნახატი ფაილების ყველაზე უფრო პოპულარული ფორმატი **Web**-სივრცეში.

შინაარსი