საქართველოს ტექნიკური უნივერსიტეტი

ე. ასაბაშვილი, თ. სტურუა, ზ. წვერაიძე

WEB-სცენარების დაპროექტება JavaScript ენის გამოყენებით

რეკომენდებულია სტუ-ის
სარედაქციო-საგამომცემლო საბჭოს
მიერ 29.07.2011, ოქმი № 3

თბილისი
2011

დღესდღეობით **JavaScript** ენა **Web**-სცენარების დასაწერი ყველაზე გავრცელებული ენა არის. სახელმძღვანელო, ენის ელემენტარული საფუძვლებიდან დაწყებული და რთული პრაქტიკული საკითხებით დამთავრებული, საკითხების ფართო სპექტრს მოიცავს. დეტალურად არის განხილული **JavaScript**-ისა და **HTML**-ის ურთიერთდამოკიდებულება: მონაცემთა ტიპები, ოპერაციები, გამოსახულებანი და ოპერატორები, ხდომილობები, ობიექტები და ობიექტთა თვისებები, სხვადასხვა ვიზუალური ეფექტები და სხვა. ასევე, მოყვანილია **Web**-გვერდების შექმნის მრავალი მაგალითი და შესასრულებლად გამზადებული პროგრამათა ტექსტები **JavaScript**-სცენარების გამოყენებით.

დამხმარე სახელმძღვანელო განკუთვნილია სტუდენტების, მაგისტრებისა და **Web**-გვერდების შექმნით დაინტერესებული სხვა სპეციალისტებისათვის.

რეცენზენტები: სრული პროფესორი ზ. ბოსიკაშვილი
სრული პროფესორი ზ. ბაიაშვილი

ISBN 978-9941-14-990-0
http://www.gtu.ge/publishinghouse/

**Verba volant,
scripta manent**

შესავალი

JavaScript-ი არის **Netscape Communication Corporation**-ის მიერ შექმნილი კომპიუტერული ენა, რომლის მეშვეობითაც იქმნება მცირე ზომის პროგრამები ე.წ. სცენარები. **Web**-კვანძებზე (**Web-site**) ამონათებული მენიუ, მოძრავი ტექსტები და მისი შინაარსის ცვლა – **JavaScript** ენის მეშვეობით არის შექმნილი. ვინაიდან ამ ენაზე დაწერილი პროგრამების შესრულება შესაძლებელია ყველა ძირითადი ინფორმაციის მძებნელი – ბრაუზერების (**browser** – მიმომხილველი) მიერ, მას იყენებს ვებ-პროგრამისტების უმრავლესობა.

კომპიუტერული ენა ეს არის ინსტრუქციების კრებული, რომელიც მიუთითებს კომპიუტერს შეასრულოს ზოგიერთი მოქმედება, რომელთა შორისაც შეიძლება იყოს ეკრანზე ტექსტის გამოტანა, გამოსახულების გადაადგილება ან სხვა. როგორც წესი ინსტრუქციები, რომელთა თანამიმდევრობასაც **კოდს** ვუწოდებთ, მუშავდება ზედა სტრიქონიდან ქვევით. დამუშავებაში იგულისხმება ის, რომ კომპიუტერი განიხილავს კოდს, რომელსაც პროგრამისტი წერს, არკვევს რა მოქმედებები მოეთხოვება და მხოლოდ ამის შემდეგ ანხორციელებს მათ. კოდის დამუშავების პროცესს მისი **შესრულება** ეწოდება.

დაპროგრამირების ენა, როგორც სხვა ენა განკუთვნილია კომუნიკაციისათვის, ანუ კავშირის დასამყარებლად მოლაპარაკესა და მსმენელს შორის. დაპროგრამებაში მოლაპარაკე არის პროგრამისტი, ხოლო მსმენელი – ენის ინტერპრეტატორი, კომპიუტერული პროგრამა, რომელსაც ესმის ენა და რომელიც ამ მოქმედებებს ისე ასრულებს, როგორც მან გაიგო. ინტერპრეტატორი **JavaScript**-ის კოდს გარდაქმნის მანქანურ კოდად, კოდის შესრულების მომენტშივე.

JavaScript ენა არის **OOP** (**Object Orientated Programming**, ობიექტ-ორიენტირებული დაპროგრამების ენა). ეს ნიშნავს, რომ მისი მეშვეობით შესაძლებელია მცირე ობიექტების აგება, რომლებიდანაც შემდეგ მთლიანი აეწყობა.

მაშინ, როცა **Java** დამოუკიდებელი ენა გახდა, **JavaScript**-ს არ შეუძლია თავისთავად არსებობა, ის **Web**-გვერდის შიგნით უნდა

იმყოფებოდეს, ხოლო **Web**-გვერდისათვის აუცილებელია მისი იმ ბრაუზერში ხილვა, რომელსაც **JavaScript** ენის "ესმის" (მაგალითად, **Internet Explorer**).

უპირველეს ყოვლისა უნდა გვახსოვდეს, რომ **JavaScript**-ი არ არის **HTML**-ი, თუმცა მათ ბევრი მსგავსი წესი აქვთ. **JavaScript**-ი თავსდება **HTML** დოკუმენტის შიგნით, სადაც ის ტექსტის სახით ინახება **HTML** დოკუმენტთან ერთად.

JavaScript-ის ენაზე პროგრამების შესაქმნელად საკმარისია ჩვენ კომპიუტერში იყოს ჩატვირთული **Notepad**-ი ან მისი მსგავსი ტექსტური რედაქტორი. იმისათვის, რომ შექმნილი კოდი გავუშვათ ტესტირებაზე, როგორც უკვე აღვნიშნეთ, აუცილებელია **Web**-ბრაუზერი, მაგალითად, **Internet Explorer**-ი (ინტერნეტის მიმომხილველი), ოღონდ ამ კონკრეტულ შემთხვევაში არა ნაკლებ მე-4 ვერსიისა.

JavaScript-ი არის ინტერპრეტირებადი და არა კომპილირებადი ენა. ვინაიდან კომპიუტერს არ ესმის **JavaScript** ენა, მას სჭირდება რაიმე, რაც მოახდენს **JavaScript**-ის ინსტრუქციების ინტერპრეტირებას და გარდაქმნის მათ მისთვის გასაგებ ბრძანებებად. კომპიუტერს ესმის მხოლოდ მანქანური კოდი, რომელიც არის ორობითი რიცხვების (ნულებისა და ერთიანების) თანამიმდევრობა. როდესაც ბრაუზერს მივაწვდით **JavaScript** ენაზე დაწერილ ინსტრუქციებს, ის მათ სპეციალურ პროგრამას გადასცემს, რომელსაც ინტერპრეტატორი ეწოდება და რომელიც **JavaScript**-ის კოდს გარდაქმნის მანქანურ კოდად, შესრულების მომენტშივე.

JavaScript-ი არ არის ერთადერთი ინტერპრეტირებადი ენა, არსებობს სხვებიც. მაგალითად, **VBScript** და **Perl**. **JavaScript**-ზე არჩევანის შეჩერების ძირითადი არგუმენტი არის მისი ფართო გავრცელება და ხელმისაწვდომობა. გარდა ამისა, ის ძალზე მოქნილი ენა არის და მისი გამოყენება არ შემოიფარგლება **Web**-გვერდების შექმნის შესაძლებლობით.

საცდელად დავწეროთ და შევასრულოთ ელემენტარული პროგრამა **JavaScript**-ზე.

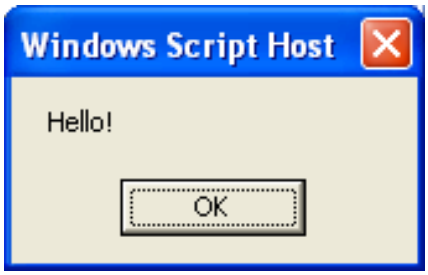
გავხსნათ ტექსტური რედაქტორი **Notepad** და შევიტანოთ შემდეგი ტექსტი:

```
WScript.echo ("Hello! ")
```

ჩვენ **JavaScript** ენაზე დავწერეთ ელემენტარული ერთი სტრიქონისაგან შემდგარი პროგრამა. შევინახოთ ეს პროგრამა მაგალითად, **Mag.js** სახელით. ფაილის სახელი შეიძლება ნებისმიერი იყოს, ხოლო მისი გაფართოება **.js** მიუთითებს, რომ ის შეიცავს **JavaScript** ენაზე დაწერილ გამოსახულებას.

ფაილის შესანახად უნდა შესრულდეს მენიუს ბრძანება **File ⇨ Save As**. გახსნილი ფანჯრის **Save as type** ჩამოშლად ველში ავირჩიოთ **All Files**, ხოლო **File name** ველში კლავიატურიდან შევიტანოთ ფაილის სახელი **.js** გაფართოებით და **Save** ღილაკით შევინახოთ ის.

პროგრამის შესასრულებლად შენახული პროგრამის ფაილის ნიშნაკზე ორჯერ დავაწკაპუნოთ მაუსით. შედეგად ეკრანზე გაიხსნება **Windows Script Host** დიალოგური ფანჯარა წარწერით „**Hello!**“ და **OK** ღილაკით. **OK** ღილაკზე დაწკაპუნებით ეკრანზე ამონათებული ფანჯარა დაიხურება.



ახლა მოვახდინოთ ჩვენი პროგრამის მოდიფიცირება. გავხსნათ **Notepad** და დავწეროთ შემდეგი ტექსტი:

```
<HTML>
<SCRIPT>
alert("Hello! ")
</SCRIPT>
</HTML>
```

ეს პროგრამა შევინახოთ ფაილში **Mag.htm** სახელით. ამ პროგრამის შესრულების შედეგად გაიხსნება **Web**-ბრაუზერი (**Microsoft Internet Explorer, Opera** და სხვა), ხოლო მის ფონზე დიალოგური ფანჯარა წარწერით „**Hello!**“ და **OK** ღილაკით.

ამგვარად, ზემოთ მოყვანილი ორივე პროგრამის შესრულების ეფექტი თითქმის ერთნაირია. საქმე იმაშია, რომ ამ ორი, თითქმის ერთნაირი პროგრამის შესრულება ხდება ენის სხვადასხვა ინტერპრეტატორის მიერ. ამრიგად, სხვადასხვა ინტერპრეტატორი აღიქვავს ერთი და იგივე ენის სხვადასხვა რედაქციას.

1. JavaScript-ის საფუძვლები

განვიხილოთ, თუ სად უნდა დაიწეროს პროგრამა და როგორ მოხდეს მისი შესრულებაზე გაშვება . პირველ ეტაპზე ვისარგებლოთ ყველაზე უბრალო და მარტივი მეთოდით: **JavaScript** ენაზე დაწერილი პროგრამის ინტერპრეტატორად გამოვიყენოთ **Web**-ბრაუზერი, ხოლო პროგრამების რედაქტორად ტექსტური რედაქტორი **Notepad**. გავხსნათ ტექსტური რედაქტორი და ჩავწეროთ მასში შემდეგი სახის პროგრამა:

```
<HTML>
<HEAD><TITLE>Example</TITLE></HEAD>
<SCRIPT>
x=5
y=x+3
alert(y)
</SCRIPT>
</HTML>
```

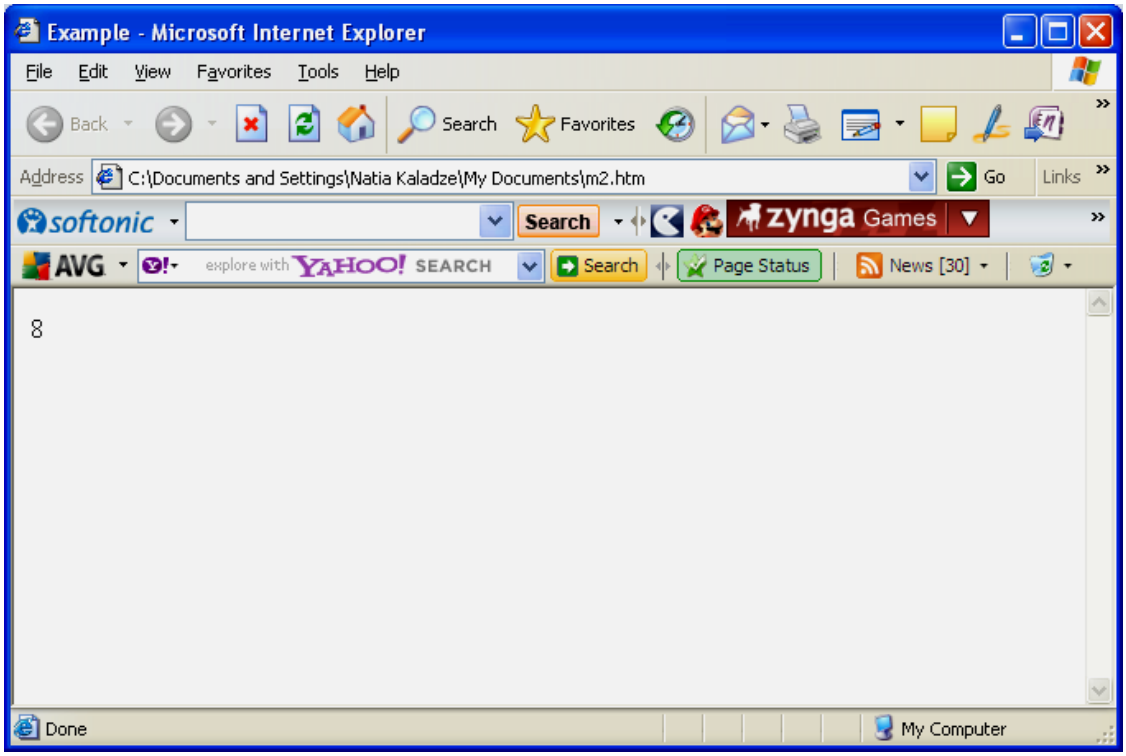
ეს არის **HTML**-დოკუმენტი. ეს ფაილი შევინახოთ დისკზე გაფართოებით **.htm** ან **.html**. ამ ფაილში ჩვენ დავწერეთ **HTML** ენის ტეგები, ხოლო **JavaScript** ენაზე გამოსახულების ჩაწერა მოვახდინეთ **<SCRIPT>** და **</SCRIPT>** ტეგებს შორის. სასურველია, რომ თითოეულ

სტრიქონში არ ჩავწეროთ ერთზე მეტი გამოსახულება. ახალ სტრიქონზე გადასასვლელად სტრიქონის ჩაწერა დავამთავროთ **<Enter>** კლავიშზე ხელის დაჭერით..

პროგრამის შესრულების შედეგის ეკრანზე გამოსატანად გამოიყენება **alert()** მეთოდი. მას შედეგი გამოაქვს დიალოგურ ფანჯარაში, რომელზეც გამოტანილი იქნება ფრჩხილებში მოთავსებული გამოსახულების მნიშვნელობა (ჩვენ შემთხვევაში – 8).

თუ გვსურს პასუხი არ გამოვიტანოთ დიალოგურ ფანჯარაში, არამედ გამოვიტანოთ უშუალოდ **Web**-ბრაუზერის ფანჯარაში, მაშინ **alert(y)**-ის ნაცვლად უნდა ჩავწეროთ შემდეგი გამოსახულება:

document.writeln(y)



განვიხილოთ **JavaScript** ენაზე, შემდეგ სტილში დაწერილი პროგრამა:

```
<SCRIPT LANGUAGE="javascript">
document.write("<FONT COLOR='RED'>წითელი ფერის ტექსტი
                </FONT>")
</SCRIPT>
```

მოცემულ შემთხვევაში ეკრანზე ამონათებული ტექსტი წითელი ფერის იქნება. **document**-ი არის **object**-ი (ობიექტი), სიტყვა **write** (წერა) გამოყოფილი წერტილით, იწოდება **method**-ად (ობიექტის მეთოდად). ამრიგად, სკრიპტი ამბობს "აიღეთ ობიექტი (უკვე არსებული) და მიაწერეთ მას რაღაც". ფრჩხილებში მოთავსებულ ტექსტს ეწოდება **instance** (მეთოდის მაგალითი), ის გადმოგვცემს იმას, რაც ხდება როცა მეთოდი ზემოქმედებს ობიექტზე. ფრჩხილებს შორის მოთავსებული ტექსტი ბრჭყალებშია. ის წარმოადგენს უბრალო **HTML**-ს, ჩვენ შემთხვევაში ეს არის ბრძანება **<FONT>**-ი, რომელიც ტექსტს წითლად გადააკეთებს. ყურადღება მიაქციეთ იმას, რომ შემდეგ ერთმაგი ბრჭყალებია. ორმაგის შემთხვევაში **JavaScript**-ი გადაწყვეტდა, რომ ეს სტრიქონის დასასრულია და მაშინ მხოლოდ ტექსტის ნაწილი იქნებოდა ობიექტისკენ მიმართული, რაც შეცდომაა.

შენიშვნა. ორმაგი ბრჭყალების შიგნით ყოველთვის ისმევა - ერთმაგი.

თუ შეიქმნა პროგრამის რედაქტირების საჭიროება, მაშინ პროგრამის ნიშნაკზე მაუსის მარჯვენა კლავიშით დაწკაპუნების გზით გავხსნათ კონტექსტური მენიუ და ავირჩიოთ **Open With ⇨ Notepad** ბრძანება. თუ გვსურს ამ პროგრამის გახსნა რედაქტირებისათვის პირდაპირ **Microsoft Internet Explorer Web**-ბრაუზერიდან, მაშინ საჭიროა შესრულდეს მენიუს **View ⇨ Source** ან **Web**-ბრაუზერის ფანჯრის ცარიელ ადგილზე გახსნილი კონტექსტური მენიუს **View Source** ბრძანება. პროგრამა გაიხსნება ახალ ფანჯარაში. მისი ხელმეორედ შესრულებაზე გასაშვებად აუცილებელია, ჯერ შევინახოთ ჩასწორებული პროგრამა **File ⇨ Save ⇨ HTML Source** ბრძანებით და შემდეგ შევასრულოთ მენიუს **View ⇨ Refresh** ბრძანება ან ხელი დავაჭიროთ კლავიატურის **<F5>** კლავიშს.

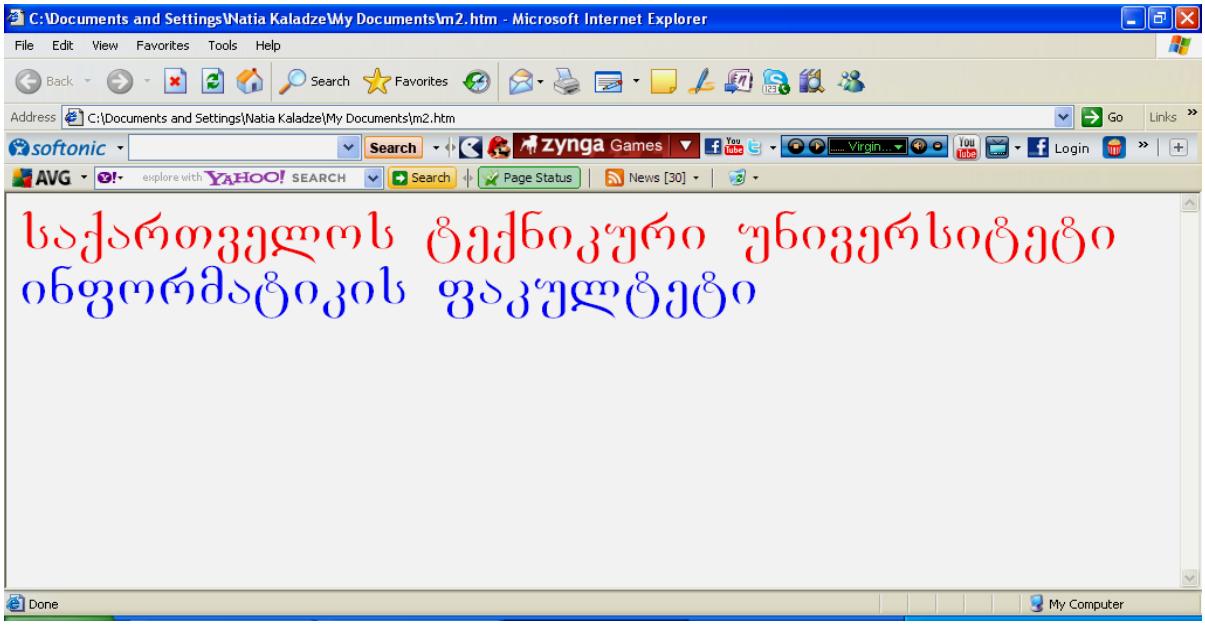
თუ გვსურს ამ პროგრამის გახსნა რედაქტირებისათვის პირდაპირ **Opera Web**-ბრაუზერიდან საჭიროა შესრულდეს მენიუს **View ⇨ Source** ან **Web**-ბრაუზერის ფანჯრის ცარიელ ადგილზე გახსნილი კონტექსტური მენიუს **Source** ბრძანება ან გამოვიყენოთ კლავიატურის **<Ctrl>+<U>** კლავიშთა კომბინაცია. ხოლო რედაქტირების შემდეგ მისი შენახვა უნდა განხორციელდეს მენიუს **File ⇨ Save As...** ბრძანებით ან **<Ctrl>+<S>** კლავიშთა კომბინაციით.

JavaScript ენაზე პროგრამების დაწერა აგრეთვე შეიძლება **Web** - საიტების დასამუშავებელი სპეციალური, მაგალითად, **Microsoft FrontPage** ან **Macromedia Dreamweaver**, პროგრამების საშუალებითაც.

დავალება: გარდაქმენით სკრიპტი ისე, რომ გამონათდეს ტექსტის ორი, წითელ და ლურჯ ჩანაწერიანი სტრიქონი. ამისათვის, დაგჭირდებათ **JavaScript**-ის რამდენიმე ბრძანების ჩაწერა:

```
<SCRIPT LANGUAGE="javascript">
document.write("<FONT FACE='AcadNusx'><FONT SIZE='22'><FONT
        COLOR ='RED'>საქართველოს ტექნიკური
        უნივერსიტეტი</FONT><BR>")
document.write("<FONT FACE='AcadNusx'><FONT SIZE='22'><FONT
        COLOR ='BLUE'>ინფორმატიკის ფაკულტეტი</FONT>")
</SCRIPT>
```

მიიღება:



შენიშვნა. თითოეული ბრძანება აუცილებლად უნდა განთავსდეს ერთ სტრიქონში!

2. მონაცემების შეტანა და გამოტანა

JavaScript ენაში მონაცემების შეტანა-გამოტანისათვის გამოიყენება ძალზე მწირი საშუალებანი. ეს სრულებით გამართლებულია, ვინაიდან ეს ენა შექმნილია **Web**-გვერდების სცენარის შესაქმნელად. რადგანაც **JavaScript** ენაზე დაწერილი სცენარის **HTML** კოდთან ინტეგრირება კარგად ხორციელდება, ამიტომ მონაცემების შეტანა-გამოტანისათვის თავისუფლად შეიძლება გამოვიყენოთ **HTML**-ის საშუალებები, კერძოდ: **alert()**, **prompt()** და **confirm()**.

alert. მოცემული მეთოდი საშუალებას გვაძლევს გამოვიტანოთ დიალოგური ფანჯარა მოცემული შეტყობინებით და **OK** ღილაკით. ამ ბრძანებას აქვს შემდეგი სახე:

alert (შეტყობინება)

შეტყობინება შეიძლება იყოს ნებისმიერი ტიპის მონაცემი: სიმბოლოების მიმდევრობა მოთავსებული ორმაგ ან ერთმაგ ბრჭყალებში, რიცხვი, ცვლადი, ან გამოსახულება.

გამოტანილი დიალოგური ფანჯრის შემდეგ შეწყდება ყოველგვარი მოქმედება და შეუძლებელი იქნება გადასვლა ადრე გახსნილ ფანჯრებზე, ვიდრე მაუსით არ დავაწკაპუნებთ **OK** ღილაკზე. ასეთი ტიპის ფანჯარას **მოდალური** ეწოდება.

confirm. მოცემული მეთოდი საშუალებას გვაძლევს გამოვიტანოთ დიალოგური ფანჯარა მოცემული შეტყობინებით და ორი ღილაკით **OK** და **Cancel**. მას აქვს შემდეგი სახე:

confirm (შეტყობინება)

შეტყობინება შეიძლება იყოს ნებისმიერი ტიპის მონაცემი: სიმბოლოების მიმდევრობა მოთავსებული ორმაგ ან ერთმაგ ბრჭყალებში, რიცხვი, ცვლადი, ან გამოსახულება.

გამოტანილი დიალოგური ფანჯარაც არის მოდალური და შეუძლებელი იქნება გადასვლა ადრე გახსნილ ფანჯრებზე, ვიდრე მაუსით არ დავაწკაპუნებთ **OK** ან **Cancel** ღილაკზე.

alert მეთოდისაგან განსხვავებით ეს მეთოდი აბრუნებს ლოგიკურ სიდიდეს, რომლის მნიშვნელობა დამოკიდებულია იმაზე, თუ რომელ ღილაკზე დავაწკაპუნეთ მაუსით. თუ ეს იყო **OK** ღილაკი, მაშინ შედეგი იქნება **true** (ჭეშმარიტი), ხოლო თუ **Cancel**, მაშინ **false** (მცდარი).

prompt. მოცემული მეთოდი საშუალებას გვაძლევს გამოვიტანოთ დიალოგური ფანჯარა მოცემული შეტყობინებით, აგრეთვე ტექსტური ველით, რომელშიც მომხმარებელს შეუძლია შეიტანოს მონაცემები და ორი ღილაკით **OK** და **Cancel**. მას აქვს შემდეგი სახე:

prompt (შეტყობინება, შესატანი მონაცემების ველის მნიშვნელობა)

შეტყობინება შეიძლება იყოს ნებისმიერი ტიპის მონაცემი: სიმბოლოების მიმდევრობა მოთავსებული ორმაგ ან ერთმაგ ბრჭყალებში, რიცხვი, ცვლადი, ან გამოსახულება.

alert და **confirm** მეთოდისაგან განსხვავებით ეს მეთოდი შეიცავს ორ პარამეტრს, შეტყობინებასა და მნიშვნელობას, რომელიც შესატანი მონაცემების ველში უნდა გამოჩნდეს გულისხმობის პრინციპით. თუ მაუსით დავაწკაპუნეთ **OK** ღილაკზე, მაშინ მეთოდი დააბრუნებს მნიშვნელობას, რომელიც შესატანი მონაცემების ველში იქნება ჩაწერილი, ხოლო თუ **Cancel** მაშინ **false** (მცდარი) მნიშვნელობას.

თუ შესატანი მონაცემების ველში არავითარ პარამეტრს არ ჩავწერთ, მაშინ ფანჯარაში გულისხმობის პრინციპით გამოჩნდება ჩანაწერი – **undefined** (განუსაზღვრელი), ხოლო თუ გვსურს არაფერი არ ეწეროს ამ ველში, მაშინ პარამეტრად ჩავწეროთ ცარიელი სტრიქონი " ".

გამოტანილი დიალოგური ფანჯარაც არის მოდალური და შეუძლებელი იქნება ადრე გახსნილ ფანჯრებზე გადასვლა, ვიდრე მაუსით არ დავაწკაპუნებთ **OK** ან **Cancel** ღილაკზე.

3. მონაცემთა ტიპები

დაპროგრამების ნებისმიერ ენაში ძალზე მნიშვნელოვანია მონაცემთა ტიპის ცნება. ის მონაცემები, რომელთა დამუშავება მიმდინარეობს პროგრამის მიერ შეიძლება მივაკუთვნოთ სხვადასხვა ტიპს და ამ მახასიათებლის მიხედვით მონაცემებზე შეიძლება განხორციელდეს სხვადასხვა ოპერაცია. **JavaScript** ენაში გამოიყენება შემდეგი ტიპის მონაცემები:

• სტრიქონული ანუ სიმბოლური (**string**) ეს არის სიმბოლოების მიმდევრობა მოთავსებული ერთმაგ ან ორმაგ ბრჭყალებში;

• რიცხვითი (**number**) – რიცხვი, ციფრების მიმდევრობა, რომლის წინაც შეიძლება მითითებული იყოს რიცხვის ნიშანი (”+” ან ”-”); დადებითი რიცხვის წინ ”+” ნიშანის მითითება არ არის აუცილებელი; მთელი და წილადის ნაწილი ერთმანეთისაგან გამოყოფილი უნდა იყოს წერტილით. რიცხვი ბრჭყალების გარეშე ჩაიწერება;

• ლოგიკური ანუ ბულის (**boolean**) – შეუძლია მიიღოს მხოლოდ ორი მნიშვნელობა: **true** (ჭეშმარიტი) ან **false** (მცდარი); ჩვეულებრივ ეს მნიშვნელობები მიიღება ორი გამოსახულების შედარების ან ლოგიკური ოპერაციების (ლოგიკური უარყოფა – **NOT**, ლოგიკური ”და” – **AND**, ლოგიკური ”ან” – **OR**) ჩატარების შედეგად. ლოგიკური ტიპის მონაცემებს ხშირად ბულის ტიპის მონაცემების სახელწოდებით მოიხსენიებენ, რომელიც ინგლისელი მათემატიკოსის ჯონ ბულის საპატივცემულოდ ეწოდა. მან ლოგიკური სიდიდეებისათვის შექმნა ე.წ. ბულის ალგებრა.

• ობიექტი (**object**) – პროგრამული ობიექტი, განსაზღვრული თავისი თვისებებით. კერძოდ, მასივიც აგრეთვე მიეკუთვნება ობიექტს;

• ფუნქცია (**function**) – ფუნქციის ანუ პროგრამული კოდის განსაზღვრა, რომლის შესრულებამაც შეიძლება დააბრუნოს რაიმე მნიშვნელობა;

• **Null** – რაიმე მნიშვნელობის არ არსებობა.

განსაკუთრებული მნიშვნელობა ენიჭება რიცხვითი და სტრიქონული მონაცემების გარჩევას. თუ ციფრებისაგან ჩაწერილი მონაცემი მოთავსებულია ბრჭყალებში, მაშინ იგი მიეკუთვნება სტრიქონული ტიპის მონაცემს და მასზე არითმეტიკული ოპერაციების

ჩატარება არ შეიძლება. აქვე უნდა შევნიშნოთ, რომ სტრიქონს "", რომელიც არ შეიცავს არც ერთ სიმბოლოს, ეწოდება ცარიელი, ხოლო სტრიქონი, რომელშიც არის ერთი ჰარი (ინტერვალი) მაინც (მაგალითად, " "), არ არის ცარიელი.

JavaScript ენაზე პროგრამის დაწერის პროცესში პროგრამისტმა ყურადღება უნდა მიაქციოს მონაცემთა ტიპებს. წინააღმდეგ შემთხვევაში, ინტერპრეტატორი შეცდომას არ დააფიქსირებს და ეცდება მითითებული ოპერაციის შესასრულებლად ეს მონაცემი ამა თუ იმ ტიპს მიაკუთვნოს, .

სიმბოლურ მონაცემებში ხშირად სპეციალური სიმბოლოები გამოიყენება:

\n – ახალი სტრიქონი;

\t – ტაბულაცია;

\f – ახალი გვერდი;

\b – ჩასმა;

\r – **<Enter>** კლავიში.

იმ შემთხვევაში, თუ ტექსტში საჭიროა სპეციალური სიმბოლოების გამოტანა, გამოიყენება სიმბოლო ‘‘\’’ (საწინააღმდეგოდ დახრილი ხაზი).

მაგალითად, თუ გვსურს ბრჭყალებთან ერთად გამოვიტანოთ შემდეგი სახის ტექსტი: **სააქციო საზოგადოება "აზოტი"**, მაშინ ეს ტექსტი ასე უნდა ჩაიწეროს: **"სააქციო საზოგადოება \"აზოტი\""**.

იგივე ამოცანა შეიძლება გადაწყდეს, თუ გარე და შიგა ბრჭყალებად გამოვიყენებთ სხვადასხვა ტიპის ბრჭყალებს (ერთმაგი და ორმაგი). მაგალითად, **‘სააქციო საზოგადოება "აზოტი"’**.

შენიშვნა:

1. სტრიქონული მონაცემების შემოსაზღვრისათვის გამოყენებული ბრჭყალები ერთი და იმავე ტიპის უნდა იყოს და დაწყვილებული.

2. ერთი ტიპის ბრჭყალებში მოთავსებული სტრიქონის შიგნით შეიძლება მხოლოდ სხვა ტიპის ბრჭყალები გამოვიყენოთ (წინააღმდეგ შემთხვევაში ინტერპრეტატორი მოგვცემს შეტყობინებას შეცდომის შესახებ, ან არასწორად აღიქვამს მონაცემებს).

4. შეტყობინება შეცდომების შესახებ

ძირითადად შეცდომების ორი ტიპი არსებობს: სინტაქსური და სცენარული. სინტაქსური შეცდომა ნიშნავს ჩანაწერში ტექსტის, ასოს გამოტოვების ან უნებლიე ბეჭდით შეცდომას. ხოლო სცენარული - ბრძანებების ადგილების გადანაცვლებით ან არასწორი ჩასმით გამოწვეულ შეცდომებს.

არსებობს შვიდი მეტნაკლებად გავრცელებული ტიპური შეცდომა. ესენია: განუსაზღვრელი ცვლადების, ასოთა რეგისტრის, გახსნილი და დახურული ფიგურული ასევე, მრგვალი ფრჩხილების რაოდენობრივი შეუსაბამობა, კონკატენაციის დროს გამოტოვებული პლუს ნიშნები, ტოლობის შემოწმების ნაცვლად ცვლადის მნიშვნელობის მინიჭება, მეთოდების გამოყენება თვისებების ნაცვლად ან პირიქით.

შეცდომების გასწორებაში არსებობს დამხმარე პროგრამები. ამ პროცესს "**debugging**"-ი («შეცდომების აღმოფხვრა») ეწოდება. ასევე, შესაძლებელია, ხელით ჩასწორებაც, რასაც ბევრი მომხმარებელი უპირატესობას ანიჭებს მისი სიმარტივის გამო.

ამბობენ, რომ შეცდომების გასწორების საუკეთესო მეთოდი შეცდომების არ დაშვებაა, მაგრამ მისი მინიმუმამდე დაყვანა შესაძლებელია ტექსტური რედაქტორის მინდვრების გარეშე რეჟიმის გამოყენებით. გარდა ამისა, თითოეული ბრძანებისათვის ცალკე სტრიქონის გამოყოფა შეცდომის სწრაფი დაფიქსირების საშუალებას მოგვცემს. ამონათებული შეტყობინება ყოველთვის მიგვანიშნებს სად, რომელ სტრიქონშია დაშვებული შეცდომა და რაშია პრობლემა.

შეცდომის სტრიქონი უნდა გადაითვალოს **HTML** დოკუმენტის ყველაზე ზედა სტრიქონიდან და არა **JavaScript**-ის პირველი სტრიქონიდან. ქვემოთ მოყვანილ მაგალითში შეცდომა დაშვებულია მე-9 სტრიქონში. ეს არის სინტაქსური შეცდომა, ვინაიდან მაგალითი (**instance**) არ მთავრდება იმავე სტრიქონში სადაც ის დაიწყო. ნათელია, რომ ფრჩხილი შემდეგ სტრიქონზეა გადასული.

```
<HTML>
<HEAD>
<TITLE></TITLE>
</HEAD>
<BODY>

<SCRIPT LANGUAGE="javascript">
document.write("text for the page"
)
</SCRIPT>
</BODY>
</HTML>
```

მაგრამ რატომ არის შეცდომა მე-9 სტრიქონში და არა მე-8-ში? ვნახოთ დოკუმენტი დანომრილი სტრიქონებით.

```
(სტრიქონი 1)  <HTML>
(სტრიქონი 2)  <HEAD>
(სტრიქონი 3)  <TITLE></TITLE>
(სტრიქონი 4)  </HEAD>
(სტრიქონი 5)  <BODY>
(სტრიქონი 6)
(სტრიქონი 7)  <SCRIPT LANGUAGE="javascript">
(სტრიქონი 8)  document.write("text for the page"
(სტრიქონი 9)  )
(სტრიქონი 10) </SCRIPT>
(სტრიქონი 11) </BODY>
(სტრიქონი 12) </HTML>
```

შენიშვნა. შეცდომის საპოვნელად დათვალეთ ყველა, ცარიელი სტრიქონიც.

5. მრავალჯერადი შეტყობინებები

JavaScript-ი ლოგიკური ენაა და მოითხოვს რომ ყველაფერი მიმდინარეობდეს თანამიმდევრულად, ერთიმეორის მიყოლებით. დავუშვათ, რომ პროგრამაში დაშვებულია 10 შეცდომა.

შეტყონინებები შეცდომის შესახებ ამონათდება თანამიმდევრულად ერთიმეორის მიყოლებით. შესაძლებელია რომ პირველმა შეცდომამ გამოიწვია ყველა დანარჩენი. ამიტომ გასწორება უნდა დავიწყოთ პირველი შეცდომიდან. ყოველი გასწორების შემდეგ კი გავუშვათ სკრიპტი შემოწმებაზე.

ხშირია სცენარის ისეთი შეცდომაც, რომელიც არ არის განსაზღვული ან მისი განსაზღვრა რთულია. რაც მიანიშნებს იმაზე, რომ სკრიპტში რაღაც შეთანხმებული არ არის. ეცადეთ რომელიმე სტრიქონი არ ჩაწეროთ საჭიროზე ადრე და თუ საქმე ამაში არ არის, მაშინ წაშალეთ ეს სტრიქონი, მისი დაბრუნება ხომ ყოველთვის შეიძლება. **JavaScript** არის რეგისტრზე დამოკიდებული ენა. ეს ნიშნავს, რომ სიმბოლოს რეგისტრის შეცვლა (ასომთავრულის შეცვლა პატარა ასოებით და პირიქით) ცვლადის სახელში იწვევს სხვა ცვლადის მიღებას. მაგალითად, **variable**, **Variable** და **vaRiable** – სხვადასხვა ცვლადია.

ბეჭდვის დროს ბანალური შეცდომებიც ხდება, ამიტომ ტექსტს ყურადღებით დააკვირდით. შეეცადეთ შემდეგ მაგალითში მოცემულ კოდში იპოვოთ ასოთა რეგისტრთან დაკავშირებული სამი შეცდომა:

```
var myName = “Paata”;
If (myName = = “paata”)
Alert (myName.toUppercase());
```

1. პირველი შეცდომა - მეორე სტრიქონში **if**-ის ნაცვლად ჩაწერილია **If**. მიუხედავად ამისა **JavaScript**-ი არ გვამცნობს რომ ასო აკრეფილია არასწორ რეჟიმში. სამაგიეროდ **Internet Explorer**-ი აცხადებს: ”**Object expected**” (მიმდინარეობს ობიექტის მოლოდინი), ხოლო **Netscape Navigator**-ი გვეტყვის, რომ ”**If is not defined**” (**If** ცვლადი არ არის განსაზღვრული);
2. მეორე შეცდომა არ ეკუთვნის **JavaScript**-ის სინტაქსისს, ეს ლოგიკური შეცდომაა. **JavaScript** ენაში ”**Paata**” და ”**paata**” არ არის ერთი და იგივე, ამიტომ **myName = = “Paata**-ს ექნება **false** მნიშვნელობა. ასეთ შეცდომას თან არ სდევს არანაირი შეტყობინება, უბრალოდ სცენარი არ იმუშავებს ისე როგორც დავგეგმეთ.

3. მესამე შეცდომა დაკავშირებულია ობიექტ **String**-ის მეთოდ **toUpperCase()**-თან, რომელიც ინახება **myName** ცვლადში. მოყვანილ მაგალითში, როცა **toUppercase()**-ში "**c**" ჩაწერილია პატარა ასოთი, **IE** შეგვატყობინებს, "**Object doesn't support this property or method**" (ობიექტი მხარს არ უჭერს ამ თვისებას ან მეთოდს), ხოლო **Netscape Navigator**, რომ "**myName.toUpperCase is not a function**" (**myName.toUpperCase**-ი არ წარმოადგენს ფუნქციას).

აქედან გამომდინარე ვასკვნით, რომ რეგისტრის გამოყენების დროს ყურადღებით უნდა ვიყოთ.

6. ცვლადები და მინიჭების ოპერატორი

ცვლადი შეიძლება მონაცემთა შესანახ კონტეინერად ჩაითვალოს. ცვლადში შენახულ მონაცემს ამ ცვლადის მნიშვნელობა ეწოდება. ყოველ ცვლადს გააჩნია სახელი. ცვლადის სახელი უნდა იწყებოდეს ასოთი ან ქვედა ხაზით და ასოებისა და ციფრების გარდა შეიძლება შეიცავდეს მხოლოდ ქვედა ხაზს ყოველგვარი ჰარისა (ინტერვალი) და სასვენი ნიშნების გარეშე. ცვლადის სახელად არ შეიძლება გამოყენებული იყოს ენის დარეზერვირებული (გამხსნელი) სიტყვები. დარეზერვირებული სიტყვები მოცემულია 1-ელ ცხრილში.

ცხრილი 1

abstract	default	for	Interface	reset	throws
boolean	delete	function	Long	return	transient
break	do	goto	Native	short	true
byte	double	if	New	static	try
case	else	Implements	Null	super	typeof
catch	extends	import	Package	switch	var
char	false	in	Private	synchronized	void
class	final	instanceof	Protected	this	while
const	finally	int	Public	throw	with
continue	float				

ზოგჯერ ცვლადის სახელის პირველ სიმბოლოდ იყენებენ ასოს, რომელიც ამ ცვლადის მონაცემთა (მნიშვნელობის) ტიპს მიუთითებს: **c** – სტრიქონული (**character**), **n** – რიცხვითი (**number**), **b** – ლოგიკური (**boolean**), **o** – ობიექტი (**object**), **a** – მასივი (**array**).

პროგრამაში ცვლადის შექმნა რამდენიმე ხერხით შეიძლება:

• მნიშვნელობის მინიჭების ოპერატორის საშუალებით, ფორმატით:

<ცვლადის-სახელი> = <მნიშვნელობა>

მინიჭების ოპერატორი აღინიშნება ტოლობის ”=” სიმბოლოთი.

მაგალითად,

MyName="Levan"

• გამხსნელი სიტყვის **var** (**variable** – ცვლადი) გამოყენებით, ფორმატით:

var <ცვლადის-სახელი>

ამ შემთხვევაში შექმნილ ცვლადს არა აქვს არავითარი მნიშვნელობა, მაგრამ მნიშვნელობა შეიძლება მიიღოს მინიჭების ოპერატორის დახმარებით.

მაგალითად,

```
var MyName
MyName="Levan"
```

• გამხსნელი სიტყვის **var**-ისა და მინიჭების ოპერატორის საშუალებით, ფორმატით:

var <ცვლადის_სახელი> = <მნიშვნელობა>

მაგალითად,

var MyName="Levan"

პროგრამის კოდის სტრიქონი გამხსნელი სიტყვა **var**-ით იძლევა ე. წ. ცვლადის ინიციალიზაციას და თითოეული ცვლადისათვის გამოიყენება მხოლოდ ერთხელ. ცვლადის ტიპი განისაზღვრება იმ მნიშვნელობის ტიპით, რომელიც მას მიენიჭება. დაპროგრამების სხვა ენებისაგან განსხვავებით ცვლადის ინიციალიზაციის დროს არ ხდება ამ ცვლადის ტიპის აღწერა. ცვლადს შეიძლება მიენიჭოს სხვადასხვა ტიპის მნიშვნელობები და არაერთხელ შეუძლია მისი შეცვლა.

ერთი გამხსნელი სიტყვა **var**-ი შეიძლება ერთდროულად გამოყენებულ იქნეს რამდენიმე ცვლადის ინიციალიზაციისათვის, როგორც მინიჭების ოპერატორთან ერთად, ასევე მის გარეშეც.

ცვლადები, რომლებიც განსაზღვრულია მოცემულ პროგრამაში, არის გლობალური ცვლადები. ეს ნიშნავს, რომ მათი მნიშვნელობის გამოყენება შეიძლება ყველგან, როგორც მოცემულ პროგრამაში, აგრეთვე სხვა ფაილებიდან გამოძახებულ პროგრამებშიც. ამ ცვლადებზე მიმართვა შეიძლება ფუნქციის კოდის შიგნითაც განხორციელდეს.

გლობალური ცვლადის გარდა არსებობს ლოკალური ცვლადები. ისინი შეიძლება ფუნქციის კოდის შიგნით იყოს შექმნილი. გარე პროგრამაში და ფუნქციის კოდის შიგნით ცვლადი შეიძლება განისაზღვროს ერთი და იგივე სახელით. ამ შემთხვევაში ფუნქციის კოდის შიგნით გამხსნელი სიტყვა **var**-ით განსაზღვრული ცვლადი იქნება ლოკალური: მათი მნიშვნელობის შეცვლა ფუნქციის შიგნით არაფრით არ აისახება გარე პროგრამაში განსაზღვრული იმავე სახელის მქონე ცვლადის მნიშვნელობაზე. ამასთან, ლოკალური ცვლადის მნიშვნელობა მიუწვდომელია გარე პროგრამისათვის.

7. ოპერატორები

ოპერატორები გამოსახულების მისაღებად გამოიყენება. ოპერატორი იყენებს ერთ ან ორ მონაცემს, რომლებსაც ამ შემთხვევაში ოპერანდები ეწოდებათ. ოპერატორი აბრუნებს მნიშვნელობას და ეს გამოსახულება შეიძლება მივანიჭოთ ცვლადს.

ერთ მინიჭების ოპერატორს, ‘‘=’’ ჩვენ უკვე გავეცანით.

კომენტარი. კომენტარის ოპერატორი საშუალებას გვაძლევს გამოვყოთ პროგრამის ფრაგმენტი, რომელიც ინტერპრეტატორის მიერ არ სრულდება და გამოიყენება მხოლოდ პროგრამის შინაარსის ასახსნელად.

JavaScript ენაში გამოიყენება ორი ტიპის კომენტარის ოპერატორი:

• // – ამ ოპერატორის მარჯვნივ განლაგებული სიმბოლოების ერთი სტრიქონი არის კომენტარი;

• **/*...*/** – ყველაფერი, რაც მოთავსებულია **/*** და ***/** სიმბოლოებს შორის არის კომენტარი; ამ ოპერატორის საშუალებით შეიძლება რამდენიმე სტრიქონი გამოვყოთ კომენტარის სახით.

<u>არითმეტიკული ოპერატორები.</u> არითმეტიკული ოპერატორები **JavaScript**-ში შეიძლება გამოყენებული იყოს ნებისმიერი ტიპის მონაცემებთან. ეს ოპერატორებია (ცხრილი 2):

ცხრილი 2

ოპერატორი	დასახელება	მაგალითი
+	შეკრება	X+Y
-	გამოკლება	X-Y
*	გამრავლება	X*Y
/	გაყოფა	X/Y
%	მოდულით გაყოფა	X%Y
++	1-ით გაზრდა	X++
--	1-ით შემცირება	X--

პირველი ოთხი ოპერატორი ყველასთვის ცნობილია. ოპერატორი მოდულით გაყოფის შედეგი იქნება პირველი რიცხვის მეორეზე გაყოფის ნაშთი. მაგალითად, 11%3 შედეგია 2. ოპერატორი **X++** ექვი-ვალენტურია **X+1** გამოსახულების, ხოლო **X--** კი **X-1**-ის. ”++” და ”--” ოპერატორებს შესაბამისად ეწოდებათ ინკრემენტი და დეკრემენტი. სიმბოლო ”-” გამოიყენება აგრეთვე რიცხვის ნიშნის შესაცვლელად.

მაგალითები:

```
y=5          // y ცვლადის მნიშვნელობა ტოლია 5-ის
x=y+3        // x ცვლადის მნიშვნელობა ტოლია 8-ის (5+3)
x++          // x ცვლადის მნიშვნელობა ტოლი გახდა 9-ის (8+1)
x--          // x ცვლადის მნიშვნელობა ტოლი გახდა 8-ის (9-1)
5-3          // მნიშვნელობა ტოლია 2-ის
-3           // უარყოფითი რიცხვი
```

შეკრების ოპერატორის გამოყენება შეიძლება სტრიქონული ცვლადების დროსაც. ამ ოპერაციას ეწოდება სტრიქონული ცვლადების

გაერთიანება ანუ კონკატენაცია. ყველა სხვა არითმეტიკული ოპერატორის სტრიქონებთან გამოყენება გვაძლევს **NaN** შედეგს, რაც ნიშნავს, რომ რიცხვი შედეგი არ არის.

მაგალითებია

```
x=Tecnical      // x ცვლადის მნიშვნელობა ტოლია "Tecnical"-სი
y=University    // y ცვლადის მნიშვნელობა ტოლია "University"-სი
z=x+" "+y       // z ცვლადის მნიშვნელობა ტოლია "Tecnical
                   University"-სი
z=x+5           // z ცვლადის მნიშვნელობა ტოლი გახდა "Tecnical5"-ის
n="20"+5        // n ტოლი იქნება "205"-ის და არა 25-ის ან "25"
```

ლოგიკური მონაცემების შემთხვევაში ინტერპრეტატორს ლოგიკური მნიშვნელობები რიცხვითში გადაჰყავს (**true** 1-ში, **false** 0-ში), ასრულებს გამოთვლებს და აბრუნებს რიცხვით შედეგს. იგივე ხდება იმ შემთხვევაში, როდესაც ერთი ოპერანდი ლოგიკურია, ხოლო მეორე – რიცხვითი.

თუ ერთი ოპერანდი სტრიქონული ტიპისაა, ხოლო მეორე – ლოგიკური, მაშინ ინტერპრეტატორს ლოგიკური ოპერანდი გადაჰყავს სტრიქონულში და შედეგიც სტრიქონული ტიპის იქნება.

მინიჭების დამატებითი ოპერატორები. ჩვეულებრივი მინიჭების ოპერატორის გარდა კიდევ ხუთი დამატებითი ოპერატორი არსებობს, რომლებიც თავისთავში მოიცავს მინიჭებისა და არითმეტიკული ოპერატორების ქმედებებს (იხ ცხრილი 3).

ეს ოპერატორები საშუალებას იძლევა ეკონომიურად ჩავწეროთ მსგავსი გამოსახულებანი.

ცხრილი 3

ოპერატორი	მაგალითი	ეკვივალენტური გამოსახულება
+=	**X+=Y**	**X=X+Y**
-=	**X-=Y**	**X=X-Y**
***=**	**X*=Y**	**X=X*Y**
/=	**X/=Y**	**X=X/Y**
%=	**X%=Y**	**X=X%Y**

შედარების ოპერატორები. პროგრამებში ხშირად გვიხდება რაიმე პირობის შესრულების შემოწმება. შემოწმების შედეგიდან გამომდინარე შემდგომი გამოთვლების პროცესი შეიძლება გაგრძელდეს სხვადასხვა მიმართულებით. შესამოწმებელი პირობის ფორმირება ხდება შედარების ოპერატორის (იხ. ცხრილი 4) საფუძველზე და ამ შედარების შედეგი არის ლოგიკური მნიშვნელობა, ანუ თუ პირობა სრულდება შედეგია **true**, ხოლო საწინააღმდეგო შემთხვევაში **false**.

ცხრილი 4

ოპერატორი	დასახელება	მაგალითი
==	ტოლია	**X==Y**
!=	არ უდრის	**X!=Y**
>	მეტია	**X>Y**
>=	მეტია ან ტოლი	**X>=Y**
<	ნაკლებია	**X<Y**
<=	ნაკლებია ან ტოლი	**X<=Y**

ერთმანეთს შეიძლება შევადაროთ რიცხვითი, სტრიქონული და ლოგიკური მნიშვნელობები. რიცხვების შედარება ხდება ჩვეულებრივად, არითმეტიკული წესების გამოყენებით, ხოლო სტრიქონების, სიმბოლოთა **ASCII** კოდების შედარების გზით, დაწყებული სტრიქონის მარცხენა ბოლოდან. ლოგიკური მნიშვნელობების შედარება ხდება ისევე როგორც რიცხვების **1**-ის და **0**-ის.

აქვე უნდა შევნიშნოთ, რომ შედარების ოპერატორები შეიძლება გამოყენებულ იქნეს სხვადასხვა ტიპის მონაცემთანაც. თუ რიცხვი უნდა შევადაროთ სტრიქონს ან ლოგიკურ მონაცემს, მაშინ ეს უკანასკნელი ინტერპრეტატორს გადაჰყავს რიცხვითში. თუ ერთმანეთს დარდება ლოგიკური მონაცემი და სტრიქონი, აქ საქმე ცოტა რთულადაა. იმ შემთხვევაში თუ სტრიქონი შეიცავს მხოლოდ რიცხვებს (ციფრებსა და ასოებს არა), მხოლოდ ჰარებს (ინტერვალებს) ან არის ცარიელი სტრიქონი, მაშინ ოპერანდები დაიყვანება რიცხვით ტიპზე. ამასთან, ცარიელი სტრიქონი ("") ან სტრიქონი შემდგარი მხოლოდ ჰარებისაგან გარდაიქმნება ნულად (0). ყველა სხვა შემთხვევაში, ყველა შედარების

ოპერატორის (გარდა ”**!=**” ოპერატორისა) შედეგი არის **false**, ხოლო ”**!=**” ოპერატორის შემთხვევაში კი **true**.

<u>ლოგიკური ოპერატორი.</u> ლოგიკური მონაცემები, რომელიც ელემენტარული შედარების ოპერატორის დახმარებით მიიღება, შეიძლება გაერთიანდეს უფრო რთული გამოსახულების სახით. ამისათვის გამოიყენება ლოგიკური (ბულის) ოპერატორები – ლოგიკური უარყოფა, ლოგიკური ”და” და ”ან” (იხ. ცხრილი 5).

ცხრილი 5

ოპერატორი	დასახელება	მაგალითი
!	უარყოფა	**!X**
&&	”და”	**X&&Y**
||	”ან”	**X||Y**

უარყოფის ოპერატორი ”**!**” მხოლოდ ერთ ოპერანდთან გამოიყენება და ცვლის ამ ოპერანდის მნიშვნელობას საწინააღმდეგოთი: თუ **X**-ს ჰქონდა მნიშვნელობა **true**, მაშინ **!X** იქნება **false** და პირიქით.

მე-6 ცხრილში მოყვანილია ლოგიკური ”და”-სა და ”ან”-ის მნიშვნელობები ორი ოპერანდის შემთხვევაში.

ცხრილი 6

X	**Y**	**X&&Y**	**X||Y**
true	**true**	**true**	**true**
true	**false**	**false**	**true**
false	**true**	**false**	**true**
false	**false**	**false**	**false**

ოპერატორები ”**&&**” და ”**||**” ხშირად მოიხსენიება შესაბამისად როგორც ლოგიკური ნამრავლი და ლოგიკური ჯამი, ხოლო მათემატიკაში კონიუნქცია და დეზიუნქცია.

8. პირობითი გადასვლის ოპერატორები

პროგრამაში იმისდა მიხედვით სრულდება თუ არა რაიმე წინასწარ მოცემული პირობა, გამოთვლითი პროცესი შეიძლება განხორციელდეს

ამა თუ იმ მიმართულებით. ამ მიზანს ემსახურება პირობითი გადასვლის **if** და **switch** ოპერატორები.

<u>if ოპერატორი.</u> პირობითი გადასვლის **if** ოპერატორი საშუალებას იძლევა მოვახდინოთ პირობითი გამოსახულების "თუ ..., მაშინ ..., თუ არა და ..." სტრუქტურის რეალიზაცია.

if ოპერატორის სინტაქსი შემდეგია:

if (პირობა)
{კოდი, რომელიც სრულდება, თუ პირობა შესრულდა}
else

{კოდი, რომელიც სრულდება იმ შემთხვევაში, თუ პირობა არ შესრულდა}

ფიგურულ ფრჩხილებში თავსდება კოდის ბლოკი – რამდენიმე გამოსახულება. თუ ბლოკში გამოყენებულია მხოლოდ ერთი გამოსახულება, მაშინ ფიგურული ფრჩხილები შეიძლება არ გამოვიყენოთ. ამ კონსტრუქციის ის ნაწილი რომელიც განსაზღვრულია სიტყვით **else** არ არის აუცილებელი. ამ დროს ამ ოპერატორს ექნება შემდეგი სახე:

if (პირობა)
{კოდი, რომელიც სრულდება, თუ პირობა შესრულდა}

ფიგურული ფრჩხილების აბზაცით ჩაწერა გემოვნების საკითხია და არ არის აუცილებელი, მაგრამ საჭიროა პროგრამა ისე დაიწეროს, რომ იყოს თვალსაჩინო და მკაფიო სტრუქტურის, რომლის დროსაც მარტივად შემოწმდება ფრჩხილების განლაგების სისწორე. ზემოთ ნაჩვენები სტრუქტურის გარდა, ხშირად იყენებენ პირობითი ოპერატორის შემდეგი ტიპის ჩანაწერს:

```
if (პირობა) {
  კოდი, რომელიც სრულდება, თუ პირობა შესრულდა
} else {
  კოდი, რომელიც სრულდება, თუ პირობა არ შესრულდა
}
```

პირობითი ოპერატორის კონსტრუქცია ერთმანეთში ჩადგმული პირობითი ოპერატორების არსებობის საშუალებას იძლევა. ამ დროს მას აქვს უფრო რთული სტრუქტურა:

```
if  (პირობა 1) {
  კოდი, რომელიც სრულდება, თუ პირობა 1 შესრულდა
} else { if  (პირობა 2) {
  კოდი, რომელიც სრულდება, თუ პირობა 2 შესრულდა
  } else {
  კოდი, რომელიც სრულდება, თუ პირობა 2 არ შესრულდა
  }
}
```

მაგალითები:

1. **I** (ასაკი) ცვლადის მნიშვნელობის მიხედვით ეკრანზე გამონათდება დიალოგური ფანჯარა სხვადასხვა შეტყობინებით:

```
if (I<18)
  {alert("თქვენ ძალზე ახალგაზრდა ხართ არჩევნებში
                  მონაწილეობის მისაღებად")}
else
   {alert("დაადასტურეთ თქვენი გადაწყვეტილება არჩევნებში
                  მონაწილეობის მიღების შესახებ")}
```

2. **I** (ასაკი) ცვლადის მნიშვნელობის მიხედვით გამოდის დიალოგური ფანჯარა მხოლოდ ერთი შეტყობინებით:

```
if (I<18)
  {alert("თქვენ ძალზე ახალგაზრდა ხართ არჩევნებში
                  მონაწილეობის მისაღებად")}
```

ან

```
if (I<18) alert("თქვენ ძალზე ახალგაზრდა ხართ არჩევნებში
                  მონაწილეობის მისაღებად")
```

<u>ოპერატორი switch.</u> რამდენიმე პირობის არსებობის შემთხვევაში სავსებით შესაძლებელია IF ოპერატორის გამოყენება, მაგრამ უფრო მოსახერხებელი და თვალსაჩინოა ოპერატორი **switch** (გადამრთველი). იგი განსაკუთრებით მოსახერხებელია მაშინ, როდესაც შესამოწმებელი პირობები არ არის ურთიერთგამომრიცხავი.

switch ოპერატორის სინტაქსი შემდეგია:

```
switch (გამოსახულება) {
  case ვარიანტი 1;
    კოდი
    [break]
  case ვარიანტი 2;
    კოდი
    [break]
  . . .
   [default:
  კოდი]
}
```

ოპერატორ **switch**-ის პარამეტრმა "**გამოსახულება**" შეიძლება მიიღოს სტრიქონული, რიცხვითი და ლოგიკური მნიშვნელობა. გამხსნელი სიტყვები **break** და **default** არ არის სავალდებულო, რისთვისაც ისინი მოთავსებულია კვადრატულ ფრჩხილებში. თუ მითითებულია ოპერატორი **break**, მაშინ დანარჩენი პირობების შემოწმება არ მოხდება, ხოლო თუ მითითებულია ოპერატორი **default**, მაშინ მის შემდეგ მდგომი სრულდება მხოლოდ იმ შემთხვევაში, როდესაც "**გამოსახულების**" მნიშვნელობა არ შეესაბამება არც ერთ ვარიანტს. თუ ოპერატორ **switch**-ში გათვალისწინებულია ყველა შესაძლო ვარიანტი, მაშინ ოპერატორი **default** შეიძლება არ გამოვიყენოთ.

ოპერატორ **switch** მუშაობს შემდეგნაირად: ჯერ გამოითვლება მრგვალ ფრჩხილებში მოთავსებული "**გამოსახულების**" მნიშვნელობა. მიღებული მნიშვნელობა შედარდება "**ვარიანტი 1**" მნიშვნელობას. თუ ისინი ერთმანეთს არ ემთხვევა, მაშინ მოხდება გადასვლა შემდეგ ვარიანტზე, ხოლო თუ ისინი ერთმანეთს დაემთხვევა, მაშინ მიმდევრობით იწყება იმ კოდების შესრულება, რომლებიც შეესაბამება მოცემულ ვარიანტს. ამასთან, თუ არ არის მითითებული ოპერატორი **break**, მაშინ შესრულდება მომდევნო ვარიანტების კოდებიც, ვიდრე არ შეხვდება ოპერატორი **break**.

მაგალითები:

1. ოპერატორი **switch** ოპერატორ **break**-ის გარეშე:

```
x=2
switch (x) {
        case 1:
            alert (1)
        case 2:
            alert (2)
        case 3:
            alert (3)
}
```

მოცემულ მაგალითში მეორე და მესამე ვარიანტი იმუშავებს.

2. ოპერატორი **switch** ოპერატორი **break**-ით:

```
x=2
switch (x) {
        case 1:
            alert (1)
            break
        case 2:
            alert (2)
            break
        case 3:
            alert (3)
            break
}
```

მოცემულ მაგალითში მხოლოდ მეორე ვარიანტი იმუშავებს.

3. დავუშვათ, ცვლადი **lang** შეიცავს ენის დასახელებას, რომელიც მომხმარებელმა აირჩია და შეიტანა ფორმის ველში.

```
switch (lang) {
  case "ინგლისური"
            window.open ("english.htm")
            break
  case "ფრანგული"
            window.open ("french.htm")
            break
  case "გერმანული"
            window.open ("german.htm")
            break
```

```
        default:
          alert ("მოცემულ ენაზე დოკუმენტი არ გვაქვს")
}
```

აქ გამოსახულება **window.open()** ხსნის ბრაუზერის ახალ ფანჯარას და მასში ჩატვირთავს ფრჩხილებში მითითებულ **HTML**-დოკუმენტს.

4. იგივე ამოცანა ჩავწეროთ **if** ოპერატორის გამოყენებით:

```
if (lang == "ინგლისური")
                window.open ("english.htm")
          else {
if (lang ==  "ფრანგული")
                window.open ("french.htm")
          else {
if (lang ==   "გერმანული")
                window.open ("german.htm")
          else
                alert ("მოცემულ ენაზე დოკუმენტი არ გვაქვს")
}
}
```

აღნიშნული კოდი შესაძლებელია შემდეგნაირადაც ჩაიწეროს:

```
if (lang == "ინგლისური") window.open ("english.htm")
else { if (lang ==  "ფრანგული") window.open ("french.htm")
else { if (lang ==   "გერმანული") window.open ("german.htm")
else alert ("მოცემულ ენაზე დოკუმენტი არ გვაქვს")
        }
        }
```

9. ციკლის ოპერატორები

ციკლის ოპერატორი უზრუნველყოფს პროგრამული კოდის ბლოკის მრავალჯერად შესრულებას მანამ, ვიდრე არ შესრულდება გარკვეული პირობა. **JavaScript**-ში გამოიყენება ციკლის სამი ოპერატორი: **for**, **while** და **do-while**.

<u>ოპერატორი for.</u> **for** ოპერატორს ხშირად უწოდებენ ციკლის მთვლელიან ოპერატორს, თუმცა მასში მთვლელის გამოყენება არ არის აუცილებელი.

ამ ოპერატორის სინტაქსი შემდეგია:

```
for ( [საწყისი გამოსახულება] : [პირობა] : [განახლების
      გამოსახულება] )
{
    კოდი
}
```

ან

```
for ( [საწყისი გამოსახულება] : [პირობა] : [განახლების
      გამოსახულება] ){
    კოდი
}
```

ყველაფერს, რაც არის მოთავსებული ოპერატორ **for**-ის მარჯვენა მხარეს მდებარე მრგვალ ფრჩხილებში, ეწოდება ციკლის ოპერატორის სათაური, ხოლო ფიგურულ ფრჩხილებში - ციკლის ტანი.

ციკლის ოპერატორის სათაურში "**საწყისი გამოსახულება**" სრულდება მხოლოდ ერთხელ, ოპერატორის შესრულების დაწყების დროს. მეორე პარამეტრი არის ციკლის ოპერატორის შესრულების პირობა. იგი პირობითი გადასვლის **if** ოპერატორის პირობის ანალოგიურია. მესამე პარამეტრი შეიცავს გამოსახულებას, რომელიც სრულდება ფიგურულ ფრჩხილებში მოთავსებული კოდის ყველა გამოსახულების შესრულების შემდეგ.

ციკლის ოპერატორი მუშაობს შემდეგნაირად: პირველად სრულდება "**საწყისი გამოსახულება**", შემდეგ მოწმდება "**პირობა**". თუ ეს პირობა ჭეშმარიტია, მაშინ მთავრდება ციკლის ოპერატორის შესრულება, ხოლო წინააღმდეგ შემთხვევაში სრულდება ფიგურულ ფრჩხილებში მოთავსებული ციკლის ტანი. ამის შემდეგ სრულდება "**განახლების გამოსახულება**" და ამით მთავრდება ციკლის პირველი იტერაცია. შემდეგ ისევ მოწმდება "**პირობა**" და ყველაფერი იწყება თავიდან.

უმეტეს შემთხვევაში "**საწყისი გამოსახულების**" სახით ცვლადზე მინიჭების ოპერატორი გამოიყენება. ცვლადის სახელი და მინიჭებული მნიშვნელობა შეიძლება იყოს ნებისმიერი. ამ ცვლადს უწოდებენ ციკლის მთვლელს. ციკლის მთვლელი შეიძლება იყოს როგორც ზრდადი, ასევე კლებადი.

ციკლიდან დროზე ადრე გამოსასვლელად გამოიყენება ოპერატორი **break** (წყვეტა). თუ გამოთვლით პროცესს შეხვდება ეს ოპერატორი, იგი მაშინვე შეწყვეტს ყოველგვარ გამოთვლებს ციკლის ტანში. ჩვეულებრივ ოპერატორი **break** გამოიყენება რაიმე დამატებითი პირობის შემოწმების დროს, რომლის შესრულებასაც უნდა მოყვეს ციკლის დამთავრება. ოპერატორ **break**-ის გამოყენებით ციკლის ოპერატორის ტიპური სტრუქტურა შემდეგია:

```
for ( [საწყისი გამოსახულება] : [პირობა 1] : [განახლების
        გამოსახულება] )
{
    კოდი
    if (პირობა 2) {
კოდი
        break
    }
    კოდი
}
```

გამოთვლების სამართავად ციკლის ოპერატორში აგრეთვე გამოიყენება ოპერატორი **continue** (გაგრძელება). როგორც ოპერატორი **break**, ოპერატორი **continue** გამოიყენება ციკლის ტანში პირობითი გადასვლის ოპერატორთან ერთად, ოღონდ მისგან განსხვავებით იგი წყვეტს მომდევნო კოდების შესრულებას, შემდეგ სრულდება "**განახლების გამოსახულება**" და გამოთვლით პროცესს აბრუნებს ციკლის ოპერატორის დასაწყისში, სადაც მოწმდება პირობა. ციკლის ოპერატორის ტიპური სტრუქტურა ოპერატორ **continue**-ს გამოყენებით შემდეგია:

```
for ( [საწყისი გამოსახულება] : [პირობა 1] : [განახლების
        გამოსახულება] )
```

```
{
        კოდი
        if  (პირობა 2) {
კოდი
              continue
        }
        კოდი
}
```

მაგალითები:

1. გამოვთვალოთ მთელი დადებითი რიცხვების ჯამი 1-დან 25-მდე:

```
var s = 0
for ( i = 1;  i <= 25;  i ++ ) {
   s = s + i
}
```

2. გამოვთვალოთ **x** ხარისხად **y**, სადაც **x**, **y** – მთელი დადებითი რიცხვებია. ამ ამოცანის ალგორითმი მარტივია, უნდა გამოითვალოს **x*x*x* ... *x**, სადაც **x** თანამარავლად შეგვხვდება **y**-ჯერ.

```
var z = x
for ( i = 2;  i <= y;  i ++ ) {
   z = z * x
}
```

3. გამოვთვალოთ **n**-ის ფაქტორიალი. ფაქტორიალი აღინიშნება **n!**-ით. როცა **n** ტოლია **0** ან **1**-ის, მაშინ **n!** უდრის **1**-ს, დანარჩენ შემთხვევაში **n!=2*3*4* ... * n**.

```
var z = 1
if  ( n > 1 )  {
   for ( i = 2;  i <= n;  i ++ ) {
         z = z * i
   }
}
```

<u>ოპერატორი while.</u> **while** (მანამ, ვიდრე) ოპერატორს უფრო მარტივი სტრუქტურა აქვს, ვიდრე ოპერატორ **for**-ს. მისი სინტაქსი შემდეგია:

```
while  ( პირობა )
```

```
{
        კოდი
}
```

ამ ოპერატორის შესრულების დროს პირველად მოწმდება "**პირობა**", რომელიც მოთავსებულია ციკლის სათაურში მრგვალ ფრჩხილებში. თუ პირობა ჭეშმარიტია, მაშინ სრულდება ოპერატორის ტანში ფიგურულ ფრჩხილებში განთავსებული კოდი. წინააღმდეგ შემთხვევაში კოდი არ შესრულდება. კოდის შესრულების შემდეგ გამოთვლითი პროცესი უბრუნდება ციკლის სათაურს და ისევ მოწმდება პირობა, და ა. შ.

while ოპერატორში განახლების გამოსახულება მოთავსებული უნდა იყოს ციკლის ტანში. ამასთან, უნდა გვახსოვდეს, რომ თუ "**პირობა**" თავიდანვე იყო მცდარი, მაშინ კოდი არ შესრულდება.

განვიხილოთ რამდენიმე ამოცანის ამოხსნის მაგალითი, რომელიც ჩვენ უკვე ამოვხსენით ციკლის **for** ოპერატორის დახმარებით.

მაგალითები:

1. გამოვთვალოთ **x** ხარისხად **y**.

```
var z = x
i = 2
while ( i <= y ) {
   z = z * x
   i++
}
```

2. გამოვთვალოთ **n** -ის ფაქტორიალი.

```
var z = 1
if  ( n > 1 )  {
   i = 2
   while ( i <= n  ) {
         z = z * i
         i ++
   }
}
```

ზემოთ მოყვანილ მაგალითებში გამოყენებულია ციკლის მთვლელი. მათი ინიცირება ხდებოდა ციკლის ოპერატორამდე, ხოლო მათი განახლება ხდება ციკლის ტანში.

გამოთვლითი პროცესის მართვისათვის ოპერატორ **while**-შიც გამოიყენება ოპერატორები **break** და **continue**.

<u>ოპერატორი do-while.</u> **do-while** (შეასრულე მანამ, ვიდრე) ოპერატორი არის ორი ოპერატორის კონსტრუქცია, რომელიც ერთად გამოიყენება. მისი სინტაქსი შემდეგია:

```
do {
            კოდი
}
while (პირობა)
```

while ოპერატორისაგან განსხვავებით **do-while** ოპერატორში პირობისაგან დამოუკიდებლად კოდი ერთხელ მაინც შესრულდება. "**პირობა**" მოწმდება კოდის შესრულების შემდეგ. თუ იგი ჭეშმარიტია, მაშინ კვლავ სრულდება **do** ოპერატორის შემდეგ მდგომი კოდი.

განვიხილოთ მაგალითები, რომელიც ჩვენ უკვე ამოვხსენით ციკლის **for** და **while** ოპერატორების დახმარებით.

მაგალითები:

1. გამოვთვალოთ **x** ხარისხად **y**.

```
var z = x
i = 2
do {
        z = z * x
        i++
}
while ( i <= y )
```

2. გამოვთვალოთ **n** -ის ფაქტორიალი.

```
var z = 1
if  ( n > 1 )  {
   i = 2
   do {
         z = z * i
         i ++
   }
   while ( i <= n  )
}
```

10. ფუნქციები

ქვეპროგრამა ეს არის ლოგიკურად დამთავრებული კოდი, რომელსაც აქვს სახელი და რომლის საშუალებითაც ის შეიძლება გამოვიძახოთ პროგრამის ნებისმიერი ადგილიდან, რამდენჯერაც გვინდა. **JavaScript**-ში ფუნქცია არის ქვეპროგრამა, რომელიც შესასრულებლად შეიძლება გამოვიძახოთ სახელის მიხედვით. ყველა ფუნქცია იყოფა ორ სტანდარტულ და მომხმარებლის მიერ განსაზღვრულ ჯგუფად:. სტანდარტული ფუქციები არის ენის ნაწილი და შეიძლება გამოყენებულ იქნეს წინასწარი აღწერის გარეშე, მაგრამ შეუძლებელია მისი კოდის რედაქტირება ან დათვალიერება. რაც მათ შესახებ შეიძლება გავიგოთ – ეს არის მათი მოქმედების, პარამეტრებისა და დასაბრუნებელი მნიშვნელობების აღწერა. **JavaScript**-ში გამოიყენება სტანდარტული ფუნქციების ფართო სპექტრი.

პროგრამიდან ფუნქციის გამოსაძახებელი გამოსახულება უნდა ჩაიწეროს შემდეგი ფორმატით:

ფუნქციის სახელი (პარამეტრები)

თუ ფუნქციას ესაჭიროება პარამეტრები, მაშინ ისინი უნდა მივუთითოთ მრგვალ ფრჩხილებში, რომელიც ერთმანეთისაგან მძიმით უნდა იყოს გამოყოფილი. ფუნქციას შეიძლება ჰქონდეს ან არ ჰქონდეს პარამეტრები. უკანასკნელ შემთხვევაში მრგვალ ფრჩხილებში არაფერი არ მიეთითება.

ინფორმაციას ფუნქციათა გამოყენების შესახებ დაწვრილებით ქვემოთ განვიხილავთ.

<u>სტანდარტული ფუნქციები.</u> **JavaScript**-ში გამოიყენება ქვემოთ ჩამოთვლილი სტანდარტული ფუნქციები:

parseInt (სტრიქონი, ათვლის სისტემის ფუძე) – მითითებულ სტრიქონს გარდაქმნის მთელ რიცხვად, მითითებულ ათვლის სისტემაში (8, 10 ან 16); თუ არ არის მითითებული ფუძე, მაშინ იგულისხმება ათვლის ათობითი სისტემა.

მაგალითები:

parseInt ("2.5") ***შედეგი = 2***

parseInt ("-16.254") ***შედეგი = -16***

parseInt ("15 ლარი") ***შედეგი = 15***

parseInt ("ფასი 150 ლარი") ***შედეგი = NaN***

parseFloat (სტრიქონი, ათვლის სისტემის ფუძე) – მითითებულ სტრიქონს გარდაქმნის მცურავმძიმიან რიცხვად, მითითებულ ათვლის სისტემაში (8, 10 ან 16); თუ არ არის მითითებული ფუძე, მაშინ იგულისხმება ათვლის ათობითი სისტემა.

მაგალითები:

parseFloat ("2.5") ***შედეგი = 2.5***

parseFloat ("-16.254"") ***შედეგი = -16.254***

parseFloat ("15.3 ლარი") ***შედეგი = 15.3***

parseFloat ("ფასი 150 ლარი") ***შედეგი = NaN***

isNaN (მნიშვნელობა) – შედეგი არის **true** (ჭეშმარიტი), თუ პარამეტრად მითითებული მნიშვნელობა არ არის რიცხვი, წინააღმდეგ შემთხვევაში – **false** (მცდარი).

მაგალითები:

isNaN (123) ***შედეგი false***

isNaN ("123") ***შედეგი false***

isNaN ("tel. 123456) ***შედეგი true***

isNaN (true) ***შედეგი false***

isNaN (false) ***შედეგი false***

isNaN ("მიხო") ***შედეგი true***

eval (სტრიქონი) – გამოითვლება სტრიქონში მითითებული გამოსახულების მნიშვნელობა; გამოსახულება ჩაწერილი უნდა იყოს **JavaScript** ენაზე (არ უნდა შეიცავდეს **HTML**-ის ტეგებს).

მაგალითები:

```
var y = 5
var x = "if ( y  < 10 )  { y = y + 2 }"
eval (x)                                  შედეგი x ტოლია 7-ის
```

escape (სტრიქონი) – შედეგი არის **%XX** სახის სტრიქონი, სადაც **XX** – მითითებული სიმბოლოს **ASCII**-კოდია; ასეთ სტრიქონს კიდევ **escape**-მიმდევრობასაც უწოდებენ. **ASCII**-კოდი არის მოცემული სიმბოლოს

კოდი თექვსმეტობით სისტემაში, რომლის წინ მითითებულია სიმბოლო ‘‘%’’. მაგალითად, ჰარი (ინტერვალი) **escape**-მიმდევრობაში წარმოდგება როგორც **%20**.

unescape (სტრიქონი) – ახდენს პირუკუ გარდაქმნას.

typeOf (ობიექტი) – სიმბოლური სტრიქონის სახით აბრუნებს მითითებული ობიექტის ტიპს. მაგალითად, ‘‘**boolean”**, ‘‘**function”** და სხვა.

<u>მომხმარებლის ფუნქციები.</u> მომხმარებლის ფუნქციები ეს ის ფუნქციებია, რომელიც შეიძლება შექმნას მომხმარებელმა თავისი ამოცანის ამოსახსნელად. პირველად უნდა მოხდეს ფუნქციის აღწერა, რომელიც იწყება გამხსნელი სიტყვით **function**. ფუნქციის აღწერას აქვს შემდეგი სინტაქსი:

function ფუნქციის სახელი (პარამეტრები)
{
კოდი
}

ხშირად ფუნქციის აღწერის ჩაწერა შემდეგი ფორმითაც ხდება:

function ფუნქციის სახელი (პარამეტრები) {
კოდი
}

ან

function ფუნქციის სახელი (პარამეტრები) { კოდი }

ფუნქციის სახელის შერჩევა ისევე ხდება, როგორც ცვლადის სახელის შერჩევა. ფუნქციის სახელს აუცილებლად მოსდევს მრგვალი ფრჩხილები. თუ ფუნქციას ესაჭიროება პარამეტრები, მაშინ ისინი მითითებულია ამ ფრჩხილებში და გამოყოფილია ერთმანეთისაგან მძიმით. თუ ფუნქციას არ ესაჭიროება პარამეტრები, მაშინ მრგვალ ფრჩხილებში არ მიეთითება არაფერი. ფუნქციის პროგრამული კოდი (ტანი) მოთავსებული არის ფიგურულ ფრჩხილებში.

ფუნქციის განსაზღვრის დროს, მისი პარამეტრების სია შეიცავს ფორმალურ პარამეტრებს – იდენტიფიკატორებს. პარამეტრებად არ შეიძლება გამოყენებული იყოს კონკრეტული მნიშვნელობები და

გამოსახულებები. ამ პარამეტრების დაკონკრეტება ხდება ფუნქციის გარე პროგრამიდან გამოძახების შემდეგ.

თუ საჭიროა, რომ ფუნქციამ დააბრუნოს რაიმე მნიშვნელობა, მაშინ მის ტანში გამოიყენება მნიშვნელობის დაბრუნების ოპერატორი **return**, რომლის მარჯვნივ მიეთითება დასაბრუნებელი ცვლადი ან გამოსათვლელი გამოსახულება. ოპერატორი **return** ფუნქციის ტანში შეიძლება გამოყენებულ იქნეს რამდენიმეჯერ.

იმისათვის, რომ შესრულდეს მოცემული ფუნქცია, პროგრამაში უნდა ჩაიწეროს ფუნქციის გამოსაძახებელი გამოსახულება. მას აქვს შემდეგი სინტაქსი:

ფუნქციის სახელი (პარამეტრები)

ფუნქციის სახელი სრულად (რეგისტრიც) უნდა ემთხვეოდეს ადრე განსაზღვრული ფუნქციის სახელს. ფუნქციის განსაზღვრაში მოცემული პარამეტრები ფუნქციის გამოძახების დროს იცვლება კონკრეტული მნიშვნელობებით, ცვლადებით ან გამოსახულებით.

JavaScript-ში ფუნქციის განსაზღვრაში და მის გამოძახებაში პარამეტრების რაოდენობა შესაძლებელია ერთმანეთს არ ემთხვეოდეს. თუ ფუნქციის განსაზღვრაში გვაქვს მაგალითად, სამი პარამეტრი, ხოლო მის გამოძახებაში მითითებულია მხოლოდ ორი, მაშინ ბოლო პარამეტრს მიენიჭება მნიშვნელობა **null** და პირიქით, ზედმეტი პარამეტრები ფუნქციის გამოძახებაში იქნება იგნორირებული.

პროგრამა შეიძლება შეიცავდეს როგორც ფუნქციის აღწერას, ასევე მისი გამოძახების გამოსახულებას. ამასთან, პროგრამაში მათი ჩაწერის მიმდევრობას არა აქვს მნიშვნელობა. ფუნქცია შეიძლება ჩაწერილი იყოს როგორც პროგრამის დასაწყისში, ასევე პროგრამის ბოლოს და **.js** გაფართოების მქონე ცალკე ფაილის სახითაც.

ფუნქციის გამოძახების ბრძანება შეიძლება გამოყენებულ იქნეს ნებისმიერ გამოსახულებაში როგორც ოპერანდი.

მაგალითები:

1. ვთქვათ საჭიროა მართკუთხედის ფართობის გამოსათვლელი ფუნქციის განსაზღვრა. ამ ფუნქციას დავარქვათ **Srectangle**.

function Srectangle (width, height) {

```
S = width * height
return S
}
```

იგივე ფუნქცია შეიძლება ჩაიწეროს უფრო მოკლედ:

```
function Srectangle (width, height) {
return width * height
}
```

განვიხილოთ ფუნქციაზე მიმართვისა და ცვლადების მოქმედების არესთან დაკავშირებული სიტუაციები:

ა. ცვლადი **S** განსაზღვრულია ფუნქციის ტანში:

```
function Srectangle (width, height) {
S = width * height
return S
}
z = Srectangle (2, 3)     /* z-სა და S-ის მნიშვნელობა ტოლია 6-ის */
```

ბ. ცვლადი **S** განსაზღვრულია გარე პროგრამაში:

```
function Srectangle (width, height) {
var S = width * height
return S
}
var S = 2
z = Srectangle (3, 4+2)        /* z-ის მნიშვნელობა ტოლია 18-ის,
                                  ხოლო S-ის დარჩება 2-ის ტოლი */
```

გ. ლოკალური ცვლადისათვის:

```
function Srectangle (width, height) {
var S = width * height
var x = 12
return S
}
w = 4
h = 5
z = Srectangle (w, h)     /* z-ის მნიშვნელობა ტოლია 20-ის, ხოლო
                             S და x გარე პროგრამაში განსაზღვრული არ არის */
```

დ. ფუნქციის პარამეტრების სახელი ემთხვევა გარე პროგრამისა და ფუნქციის გამოძახების პარამეტრებს:

```
function Srectangle (width, height) {
var S = width * height
width = width + 10
return S
}
width = 3
height = 4
z = Srectangle (width, height)      /* z-ის მნიშვნელობა ტოლია 12-
                                       ის, ხოლო width-ის 3-ის */
```

ე.ფუნქციაზე მიმართვის დროს პარამეტრების რაოდენობა არ ემთხვევა ერთმანეთს:

```
function Srectangle (width, height) {
S = width * height
return S
}
z = Srectangle (2)    /* მეორე პარამეტრის უქონლობის გამო  z-ისა
                         და S-ის მნიშვნელობა ტოლია 0-ის */
```

2. **n!** ფაქტორიალის გამოთვლის ფუნქცია ციკლის ოპერატორის გამოყენებით:

```
function factorial (n) {
if ( n <= 1 ) return 1
R = 2
for ( i = 3; i <= n; i++ ) {
   R = R * i
}
return R
}
var m = 10
x = factorial (m)        /* x-ის მნიშვნელობა ტოლია 3628800 */
```

3. **n!** ფაქტორიალის გამოთვლის ფუნქცია რეკურსიის გამოყენებით (ფუნქციის განსაზღვრა შეიძლება შეიცავდეს ამავე ფუნქციის გამოძახებას – ე. წ. ფუნქციის რეკურსიული განსაზღვრა):

```
function factorial (n) {
      if ( n <= 1 ) {
      return 1
```

```
    }
    return n * factorial (n-1)
}
```

იგივე შეიძლება ჩაიწეროს მოკლედ:

```
function factorial (n) {
if ( n <= 1 ) return 1
return n * factorial (n-1)
}
```

11. სტანდარტული (ჩაშენებული) ობიექტები

ობიექტი არის პროგრამული ერთეული, რომელსაც აქვს რაიმე თვისება. ობიექტზე შეიძლება ვიმსჯელოთ მისი თვისების მნიშვნელობით და ფუნქციონირების აღწერით. **JavaScript**-ის სტანდარტული (ჩაშენებული) ობიექტი მომხმარებლისთვის მიუწვდომელია, ხოლო მისი გამოყენება მარტივი. **JavaScript**-ზე დაწერილი სცენარების მეშვეობით **Web**-გვერდების მართვის განხორციელება ხდება **HTML**-დოკუმენტის ან თვით პროგრამა ბრაუზერის ობიექტთა თვისებების გამოყენებითა და შეცვლით.

სტანდარტულ (ჩაშენებულ) ობიექტს აქვს ფიქსირებული სახელი და თვისებები. ობიექტების ყველა თვისება იყოფა ორ სახეობად: უბრალოდ თვისებად და მეთოდად. თვისებები ჩვეულებრივი ცვლადის ანალოგიურია. მათ აქვთ სახელი და მნიშვნელობა. ობიექტთა ზოგიერთი თვისება ხელმისაწვდომია მხოლოდ წაკითხვისათვის. ეს ნიშნავს, რომ მათი მნიშვნელობის შეცვლა არ შეიძლება. სხვები ხელმისაწვდომია ჩაწერისათვისაც – მათი შეცვლა შესაძლებელია მინიჭების ოპერატორის გამოყენებით. მეთოდები ფუნქციის ანალოგიურია, მათ შეიძლება ჰქონდეთ ან არ ჰქონდეთ პარამეტრები.

იმისათვის, რომ გავიგოთ ობიექტის თვისება აუცილებელია მივუთითოთ ამ ობიექტის სახელი და თვისების სახელი, რომლებიც ერთმანეთისაგან წერტილით იქნება გამოყოფილი:

```
ობიექტის-სახელი.თვისება
```

უნდა აღინიშნოს, რომ ობიექტს შეიძლება არც ჰქონდეს თვისება.

ჩვენ შეიძლება ვაიძულოთ ობიექტი შეასრულოს ესა თუ ის მისთვის დამახასიათებელი მეთოდი. ამ დროს ამბობენ, რომ ობიექტის მიმართ გამოყენებული არის მეთოდი. შესაბამისი გამოსახულების სინტაქსი შემდეგია:

ობიექტის-სახელი.მეთოდი (პარამეტრი)

აქვე შევნიშნოთ, რომ ობიექტს შეიძლება არც ჰქონდეს მეთოდები.

თვისებათა სინტაქსი ჩვეულებრივი ცვლადისაგან განსხვავდება იმით, რომ მათ აქვთ შედგენილი სახელი და ამასთან, ზოგიერთი თვისების მნიშვნელობის შეცვლა არ შეიძლება. მეთოდები ჩვეულებრივი ფუნქციისაგან განსხვავდება მხოლოდ შედგენილი სახელით.

ჩვენ შეიძლება ზემოქმედება მოვახდინოთ ობიექტზე მხოლოდ თვისებებისა და მეთოდების საშუალებით.

JavaScript-ში მათემატიკური გამოთვლები, სტრიქონებისა და თარიღების რთული დამუშავება, აგრეთვე მასივების შექმნა სრულდება შესაბამისი სტანდარტული (ჩაშენებული) ობიექტების საშუალებით. **Web**-საიტების დამუშავების დროს განსაკუთრებით მნიშვნელოვანია **String** (სტრიქონების დამუშავება), **Array** (მასივები), **Math** (მათემატიკური ფორმულები და მუდმივები) და **Date** (თარიღებთან მუშაობა) ობიექტები.

სტანდარტულ (ჩაშენებულ) ობიექტებს, როგორც უკვე აღვნიშნეთ, აქვს ფიქსირებული დასახელება. ობიექტებს, რომელთა სახელები ემთხვევა მათ ფიქსირებულ დასახელებას, ეწოდებათ სტატიკური. ამასთან შეგვიძლია შევქმნათ სტატიკური ობიექტების ეგზემპლარები (ასლები), რომელთაც მივანიჭებთ სხვა სახელებს. სტატიკური ობიექტების ეგზემპლარები არის ობიექტები პროგრამაში, რომლებსაც გააჩნიათ ყველა იგივე თვისებები და მეთოდები. მასთან ერთად შეგვიძლია გამოვიყენოთ სტატიკური ობიექტებიც.

სტანდარტულ (ჩაშენებულ) ობიექტებს სხვა თვისებებთან ერთად აქვთ **prototype** (პროტოტიპი) თვისება, რომლის საშუალებითაც ობიექტის უკვე არსებულ ეგზემპლარებს ემატება ახალი თვისებები და მეთოდები. ეს ახალი თვისებები და მეთოდები წინასწარ კარგად უნდა

იყოს გააზრებული და მხოლოდ ამის შემდეგ ჩაწერილი პროგრამული კოდის სახით.

ობიექტი String (სტრიქონი). ობიექტი **String**-ი გამოიყენება სტრიქონის დამუშავების დროს. იგი შეუცვლელია, როდესაც საჭიროა ერთი სტრიქონიდან ამოიჭრას რაღაც ნაწილი, მოიძებნოს ერთ სტრიქონში მეორე ქვესტრიქონი, დაიყოს ერთი სტრიქონი რამდენიმე ნაწილად და სხვა.

ობიექტ **String**-ის საშუალებით შეიძლება სტრიქონი შეიქმნას, როგორც სტრიქონული ობიექტი. მაგრამ უმეტეს შემთხვევაში ამისათვის საკმარისია გამოვიყენოთ ჩვეულებრივი ცვლადი და სტრიქონული მნიშვნელობის მინიჭების ოპერატორი. ამ შემთხვევაში ინტერპრეტატორი მაინც შექმნის სტრიქონული ობიექტის ეგზემპლარს (ასლს), რომლის თვისება და მეთოდი ხელმისაწვდომია პროგრამული კოდით.

სტრიქონული ობიექტის შექმნა. სტრიქონული ობიექტის შესაქმნელად შემდეგი სახის გამოსახულება გამოიყენება:

ცვლადის სახელი = new String ("სტრიქონული მნიშვნელობა")

აქ ცვლადის სახელი ასრულებს სტრიქონულ ობიექტზე მითითების როლს. მაგალითად, გამოსახულება **mystring = new String ("Hello!")** ქმნის სტრიქონულ ობიექტს **mystring** მნიშვნელობით **"Hello!"**.

თუმცა სტრიქონული ობიექტი შეიძლება შეიქმნას ჩვეულებრივი მინიჭების ოპერატორის გამოყენებითაც:

ცვლადის სახელი = "სტრიქონული მნიშვნელობა"

ან

var ცვლადის სახელი = "სტრიქონული მნიშვნელობა"

სტრიქონული ობიექტის თვისებებსა და მეთოდებზე მიმართვა ხორციელდება ასეთი გამოსახულებით:

სტრიქონი . თვისება
String. თვისება

სტრიქონი . მეთოდი ([პარამეტრები])

String. მეთოდი ([პარამეტრები])

ზოგიერთ მეთოდს შეიძლება არ ჰქონდეს პარამეტრი, რაც კვადრატული ფრჩხილების საშუალებითაა მითითებული. აქ "**სტრიქონი**" შეიძლება იყოს სტრიქონულ ობიექტზე მიმართვა, სტრიქონული ცვლადი, გამოსახულება, რომელიც დააბრუნებს სტრიქონს, ან უბრალოდ სტრიქონული მნიშვნელობა.

როდესაც გამოიყენება გამხსნელი სიტყვა **String** ობიექტის სახელად, ეს ნიშნავს, რომ ჩვენ გვაინტერესებს სტატიკური სტრიქონული ობიექტის თვისებები და მეთოდები, ანუ საერთო თვისებები და მეთოდები.

ქვემოთ მოყვანილია სტრიქონული ობიექტის თვისება **length**-ის (სტრიქონის სიგრძე) გამოყენების სხვადასხვა ხერხი:

```
mystring = "ABCDEFG"
mystring . length          /* მნიშვნელობა ტოლია 7-ის */
"ABCDEFG" . length         /* მნიშვნელობა ტოლია 7-ის */
function fstring ( ) { return "abcde" }  /* ფუნქცია აბრუნებს
                                            სტრიქონს "abcde" */
fstring ( ) . length       /* მნიშვნელობა ტოლია 5-ის */
x = ""                     /* ცარიელი სტრიქონი */
x . length                 /* მნიშვნელობა ტოლია 0-ის */
```

სტრიქონული ობიექტის მეთოდები გამოიყენება სტრიქონების სინტაქსური დამუშავებისა და დაფორმატებისათვის.

თვისება String.

length – სტრიქონის სიგრძე ან სიმბოლოთა რაოდენობა სტრიქონში (ჰარების ანუ ინტერვალების ჩათვლით).

prototype – თვისება, რომელიც საშუალებას იძლევა დავამატოთ ახალი თვისება და მეთოდი ყველა შესაქმნელ სტრიქონულ ობიექტს, თუ უკვე არსებული საკმარისი არ იქნება.

მაგალითი:

მოყვანილ მაგალითში ჩვენ ვქმნით ახალ მეთოდს ყველა სტრიქონული ობიექტისათვის. ამ მეთოდის არსი განისაზღვრება მომხმარებლის ფუნქციით.

```
function myFunc ( ) {
return "fgh"
}
String . prototype . myName ( ) = myFunc
mystring = "abcde" . myName ( )        /* mystring-ის მნიშვნელობა
                                          ტოლი იქნება "abcdefgh" */
```

სტრიქონების დამუშავების String მეთოდები.

1. **charAt (ინდექსი)** – აბრუნებს სიმბოლოს, რომელსაც სტრიქონში უკავია მითითებული პოზიცია. მისი სინტაქსია:

სტრიქონი . charAt (ინდექსი)

შედეგი არის ერთსიმბოლოიანი ან ცარიელი სტრიქონი. პარამეტრი **(ინდექსი)** არის რიცხვი. პირველი სიმბოლოს ინდექსი ნულის ტოლია.

მაგალითად,

```
"abcde" . charAt (1)                   /* მნიშვნელობა ტოლია "b"*/
"abcde" . charAt (15)                  /* მნიშვნელობა ტოლია " " */
mystring = "abcde"
mystring . charAt (mystring . length - 1)    /* ბოლო სიმბოლოს
                                          მნიშვნელობა ტოლია "e" */
```

2. **charCodeAt ([ინდექსი])** – სტრიქონის მითითებულ პოზიციაში მდგომ სიმბოლოს გარდაქმნის რიცხვით კოდში. მისი სინტაქსია:

სტრიქონი . charCodeAt ([ინდექსი])

შედეგად მიიღება რიცხვი (კოდი).

მაგალითად,

```
"abcde" . charCodeAt ( )     /* მნიშვნელობა ტოლია 97 */
"abcde" . charCodeAt (0)     /* მნიშვნელობა ტოლია 97 */
"abcde" . charCodeAt (1)     /* მნიშვნელობა ტოლია 98 */
"abcde" . charCodeAt (25)    /* მნიშვნელობა ტოლია NaN */
```

3. **fromCharCode (ნომერი1 [, ნომერი2 [, ... ნომერიN]])** – აბრუნებს სიმბოლოთა სტრიქონს, რომლის კოდიც მითითებულია პარამეტრის სახით. მისი სინტაქსია:

სტრიქონი . fromCharCode (ნომერი1 [, ნომერი2 [, ... ნომერიN]])

შედეგად მიიღება სტრიქონი.

მაგალითად,

```
String . fromCharCode (97 , 98 , 99 )          /* მნიშვნელობა ტოლია
                                                  "abc"  */
```

4. **concat (სტრიქონი)** – სტრიქონების კონკატენაცია (გაერთიანება). მისი სინტაქსია:

სტრიქონი1 . concat (სტრიქონი2)

შედეგად მიიღება სტრიქონი.

მაგალითად,

```
"abc" . concat ("de" )          /* მნიშვნელობა ტოლია "abcde" */
```

5. **indexOf (საძიებო სტრიქონი [, ინდექსი])** – სტრიქონში ეძებს პარამეტრში მითითებულ საძიებო სტრიქონს და აბრუნებს იმ პოზიციის ნომერს, სადაც პირველად შეხვდა აღნიშნული სტრიქონი. მისი სინტაქსია:

სტრიქონი . indexOf (საძიებო სტრიქონი [, ინდექსი])

შედეგი რიცხვია – პოზიციის ნომერი. დასაბრუნებელი რიცხვის ათვლა იწყება 0-დან. თუ სტრიქონში საძიებო სტრიქონი არ აღმოჩნდა შედეგი იქნება -1. ძებნა ცარიელ სტრიქონში ყოველთვის -1-ია. ცარიელი სტრიქონის ძებნის შედეგი 0-ია. მეორე პარამეტრი არ არის აუცილებელი და იგი მიუთითებს თუ რომელი პოზიციიდან დაიწყოს ძებნა.

მაგალითად,

```
x = "Abcde abcde"
x . indexOf ("cde" )          /* მნიშვნელობა ტოლია 2 */
x . indexOf ("A" )            /* მნიშვნელობა ტოლია 0 */
x . indexOf ("g" )            /* მნიშვნელობა ტოლია -1 */
x . indexOf ("cde" , 3 )      /* მნიშვნელობა ტოლია 8 */
```

6. **lastIndexOf (საძიებო სტრიქონი [, ინდექსი])** – სტრიქონში ეძებს პარამეტრში მითითებულ საძიებო სტრიქონს და აბრუნებს იმ პოზიციის ნომერს, სადაც პირველად შეხვდა აღნიშნული სტრიქონი. ამასთან ძებნა იწყება სტრიქონის ბოლოდან, ხოლო პოზიციის ნომრის ათვლა თავიდან. მისი სინტაქსია:

სტრიქონი . lastIndexOf (საძიებო სტრიქონი [, ინდექსი])

შედეგი რიცხვია – პოზიციის ნომერი.

მაგალითად,

```
x = "Abcde abcde"
x . lastIndexOf ("cde" )          /* მნიშვნელობა ტოლია 8 */
x . lastIndexOf ("b" )            /* მნიშვნელობა ტოლია 7 */
x . indexOf ("b" )                /* მნიშვნელობა ტოლია 1 */
```

7. **localeCompare (სტრიქონი)** – ახდენს სტრიქონების შედარებას **Unicode** კოდებში. მისი სინტაქსია:

სტრიქონი 1 . localeCompare (სტრიქონი 2)

შედეგი არის რიცხვი. თუ შესადარებელი სტრიქონები ერთნაირია შედეგი ნულის ტოლია. თუ **სტრიქონი 1** ნაკლებია **სტრიქონი 2**-ზე, შედეგი უარყოფითი რიცხვია, წინააღმდეგ შემთხვევაში – დადებითი. სტრიქონების შედარება ხდება მისი შემადგენელი სიმბოლოების კოდების ჯამის შედარებით.

8. **slice (ინდექსი 1 [, ინდექსი 2])** – ახდენს საწყისი სტრიქონიდან ქვესტრიქონის გამოყოფას დაწყებული **ინდექსი 1**-დან **ინდექსი 2**-მდე ბოლო სიმბოლოს გამოკლებით. მისი სინტაქსია:

სტრიქონი 1 . slice (ინდექსი 1 [, ინდექსი 2])

შედეგად მიიღება სტრიქონი. მოცემული მეთოდი არ ცვლის საწყის სტრიქონს. თუ მეორე პარამეტრი მითითებული არ არის, მაშინ გამოიყოფა ქვესტრიქონი **ინდექსი 1**-დან სტრიქონის ბოლომდე. თუ მეორე პარამეტრი უარყოფითია, მაშინ ბოლო ინდექსის ათვლა იწყება სტრიქონის ბოლოდან.

მაგალითად,

```
x = "Abcde abcde"
```

x . slice (3, 8) /* მნიშვნელობა ტოლია **"de ab"** */

x . slice (2, -2) /* მნიშვნელობა ტოლია **"cde abc"** */

9. **substr (ინდექსი [, სიგრძე])** – ახდენს საწყისი სტრიქონიდან **ინდექსი**-დან დაწყებული მითითებული სიგრძის ქვესტრიქონის გამოყოფას. მისი სინტაქსია:

სტრიქონი . substr (ინდექსი [, სიგრძე])

შედეგად მიიღება სტრიქონი. მოცემული მეთოდი არ ცვლის საწყის სტრიქონს. თუ მეორე პარამეტრი მითითებული არ არის, მაშინ გამოიყოფა ქვესტრიქონი **ინდექსი**-დან სტრიქონის ბოლომდე. თუ მეორე პარამეტრი უარყოფითია, მაშინ ბოლო ინდექსის ათვლა იწყება სტრიქონის ბოლოდან.

მაგალითად,

x = "abcde abcde"

x . substr (3, 5) /* მნიშვნელობა ტოლია **"de ab"** */

10. **substring (ინდექსი1, ინდექსი2)** – ახდენს საწყისი სტრიქონიდან ქვესტრიქონის გამოყოფას. მისი სინტაქსია:

სტრიქონი1 . substring (ინდექსი1, ინდექსი2)

შედეგი სტრიქონია. მოცემული მეთოდი არ ცვლის საწყის სტრიქონს. ინდექსების მიმდევრობას მნიშვნელობა არა აქვს. უმცირესი ითვლება საწყის ინდექსად.

მაგალითად,

x = "abcdefgh"

x . substring (3, 8) /* მნიშვნელობა ტოლია **"defgh"** */

x . substring (4, 8) /* მნიშვნელობა ტოლია **"efgh"** */

x . substring (8, 4) /* მნიშვნელობა ტოლია **"efgh"** */

11. **split (გამყოფი [, შეზღუდვა])** – აბრუნებს საწყისი სტრიქონიდან მიღებული ელემენტების მასივს. მისი სინტაქსია:

სტრიქონი . split (გამყოფი [, შეზღუდვა])

შედეგად მიიღება სტრიქონი. პირველი პარამეტრი არის სიმბოლოების სტრიქონი, რომელიც შემდგომ გამოიყენება სტრიქონის

ელემენტებად დაყოფისათვის. მეორე არა აუცილებელი პარამეტრი – მიღებული მასივის ელემენტების რაოდენობაა.

მაგალითად,

x = "abecdefgheijkl"

x . split ("e") /*მნიშვნელობა – მასივი, რომლის ელემენტებია **"ab", "cd", "fgh", "ijkl"** */

x . split ("e",2) /* მნიშვნელობა – მასივი, რომლის ელემენტებია **"ab", "cd"** */

12. **toLocaleLowerCase ( ), toLowerCase ( )** – გადაჰყავს სტრიქონი ქვედა რეგისტრში. მისი სინტაქსია:

სტრიქონი. toLocaleLowerCase ( ), სტრიქონი. toLowerCase ( )

შედეგად მიიღება სტრიქონი.

მაგალითად,

x = "AbcDefghiJK"

x . toLocaleLowerCase ( ) /* მნიშვნელობა ტოლია **"abcdefghijk"** */

x . toLowerCase ( ) /* მნიშვნელობა ტოლია **"abcdefghijk"** */

13. **toLocaleUpperCase ( ), toUpperCase ( )** – გადაჰყავს სტრიქონი ზედა რეგისტრში. მისი სინტაქსია:

სტრიქონი.toLocaleUpperCase ( ), სტრიქონი. toUpperCase ( )

შედეგი სტრიქონია.

მაგალითად,

x = "AbcDefgh"

x . toLocaleUpperCase ( ) /* მნიშვნელობა ტოლია **"ABCDEFGH"** */

x . toUpperCase ( ) /* მნიშვნელობა ტოლია **"ABCDEFGH"** */

სტრიქონების დაფორმატების String მეთოდები. როგორც ცნობილია, **Web**-გვერდებზე ტექსტის შექმნა და დაფორმატება ჩვეულებრივ **HTML**-ის ტეგების საშუალებით ხდება. მაგალითად,

იმისათვის, რომ **Web**-გვერდზე გამოვიტანოთ სტრიქონი "**Hello!**" მუქი შრიფტით **HTML** კოდში უნდა ჩავწეროთ შემდეგი ინსტრუქცია:

<B> Hello! </B>

იმისათვის, რომ იგივე სტრიქონი მოვამზადოთ **JavaScript**-ის საშუალებებით, უნდა ჩავწეროთ შემდეგი გამოსახულება:

"Hello! ".bold ( )

ჩანაწერის გამუქების მიზნით აქ გამოიყენება მეთოდი **bold ( )**, ხოლო სტრიქონის ეკრანზე გამოსატანად ობიექტ **document**-ისათვის უნდა შესრულდეს მეთოდი **write ( )**. შესაბამის კოდს ექნება შემდეგი სახე:

```
< HTML >
< SCRIPT >
x = "Hello! " . bold ( )
document . write ( )
< /SCRIPT >
< /HTML >
```

String მეთოდს აქვს შემდეგი სინტაქსი:

სტრიქონი . მეთოდი (პარამეტრი)

დაფორმატების მეთოდების უმეტესობას არა აქს პარამეტრები. ქვემოთ მოყვანილია მათი სია:

anchor ("anchor-სახელი")
blink ( )
bold ( )
fixed ( )
fontcolor (ფერის მნიშვნელობა)
fontsize (რიცხვი 1-დან 7-მდე)
italics ( )
link (მდებარეობა ანუ URL)
big ( )
small ( )
strike ( )
sub ( )
sup ( )

ქვესტრიქონის ჩამატებისა და შეცვლის ფუნქციები. სტრიქონების დამუშავების დროს ხშირად საჭიროა ქვესტრიქონის ჩამატება ან შეცვლა. ქვესტრიქონის წაშლა შეიძლება განვიხილოთ როგორც შეცვლის კერძო შემთხვევა, ანუ მითითებული ქვესტრიქონის შეცვლა ცარიელი

სტრიქონით. ობიექტის დამუშავების **String** მეთოდების დახმარებით შეიძლება დაიწეროს ამ ამოცანის გადასაწყვეტი პროგრამა. ვინაიდან ეს ამოცანა ხშირად გვხვდება, მიზანშეწონილია პროგრამა გაფორმდეს ფუნქციის სახით.

პირველ რიგში განვიხილოთ საწყის სტრიქონში სტრიქონის ჩასმის ფუნქცია. დავარქვათ მას მაგალითად, **insstr**. მოცემულ ფუნქციას უნდა მიეწოდოს სამი პარამეტრი: საწყისი სტრიქონი **s1**, ჩასამატებელი სტრიქონი **s2** და ჩასამატებელი პოზიციის ინდექსი **n**.

```
function insstr ( s1, s2, n ) {
return s1 . slice ( 0, n ) + s2 + s1 . slice ( n )
}
```

ახლა განვიხილოთ ფუნქცია, რომელიც საწყის სტრიქონში ყველა შესაცვლელ ქვესტრიქონს ვცვლით სხვა ქვესტრიქონით. ამ ფუნქციას დავარქვათ **replacestr**. მას უნდა მივაწოდოთ სამი პარამეტრი: საწყისი სტრიქონი **s1**, შესაცვლელი ქვესტრიქონი **s2** და **s3** ქვესტრიქონი, რომლითაც უნდა მოხდეს ყველა **s1**-ში შემავალი **s2** ქვესტრიქონის შეცვლა.

```
function replacestr ( s1, s2, s3 ) {
   var s = ""
   while ( true ) {
         i = s1 . indexOf ( s2 )
         if ( i >= 0 ) {
         s = s + s1 . substr ( 0, i ) + s3
         s1 = s1 . substr ( i + s2 . length )
         } else break
   }
   return s + sx
}
```

შევნიშნოთ, რომ თუ შესაცვლელი ქვესტრიქონი **s2** ცარიელი სტრიქონია, მაშინ ციკლი უსასრულოდ გაგრძელდება, რადგან ცარიელი სტრიქონი შედის ნებისმიერი სტრიქონის შემადგენლობაში. ჩაციკვლის თავიდან აცილების მიზნით ცოტათი უნდა შეიცვალოს ფუნქციის ტანის კოდი:

```
function replacestr ( s1, s2, s3 ) {
   if ( s2 == "" )
   s2 = " "
   var s = ""
   while ( true ) {
         i = s1 . indexOf ( s2 )
         if ( i >= 0 ) {
         s = s + s1 . substr ( 0, i ) + s3
         s1 = s1 . substr ( i + s2 . length )
         } else break
   }
   return s + sx
}
```

ზემოთ განხილული ფუნქციები შეიძლება შევინახოთ ტექსტურ ფაილში **.js** გაფართოებით და გამოვიყენოთ ნებისმიერ ძირითად პროგრამაში.

12.ობიექტი Array (მასივი)

მასივი არის მონაცემთა მოწესრიგებული ნაკრები. იგი შეიძლება წარმოვადგინოთ ერთსვეტიანი ცხრილის სახით, რომელიც შეიცავს გარკვეული რაოდენობის სტრიქონებს. ასეთი ცხრილის უჯრედში შეიძლება იყოს ნებისმიერი ტიპის მონაცემი, მათ შორის მასივებიც. ამ ბოლო შემთხვევაში გვაქვს მრავალგანზომილებიანი მასივი. მასივში ელემენტების რაოდენობას ეწოდება მასივის სიგრძე. მასივის ელემენტებს შეიძლება პროგრამაში მივმართოთ მისი რიგითი ნომრით (ინდექსით). მასივის ელემენტების ნუმერაცია ნულით იწყება, პირველი ელემენტის ინდექსია 0, ხოლო ბოლოსი – ერთი ერთეულით ნაკლები, ვიდრე მასივის სიგრძე.

თუ გამოყენებულ მონაცემებს შორის არის ჯგუფი, რომლის დამუშავება უნდა მოხდეს ერთი და იგივე სახით, მაშინ უმჯობესია მოხდეს მათი მასივის სახით ორგანიზება.

მასივის შექმნა. არსებობს მასივის შექმნის რამდენიმე წესი. ნებისმიერ შემთხვევაში უპირველეს ყოვლისა იქმნება მასივის ახალი ობიექტი გამხსნელი სიტყვის **new**-ს გამოყენებით:

მასივის სახელი = new Array ([მასივის სიგრძე])

აქ მასივის სიგრძის მითითება აუცილებელი არ არის. თუ მასივის სიგრძე არ არის მითითებული, მაშინ იქმნება ცარიელი მასივი. წინააღმდეგ შემთხვევაში იქმნება მასივი მითითებული სიგრძით, თუმცა მათ აქვთ **null**-ის ტოლი მნიშვნელობა (ანუ არ აქვთ მნიშვნელობა).

პირველად, შეგვიძლია შევქმნათ ცარიელი მასივი და შემდეგ მინიჭების ოპერატორის გამოყენებით მასივის ელემენტებს მივანიჭოთ მნიშვნელობები. მასივის ელემენტის მისათითებლად მასივის სახელის გვერდით ელემენტის ინდექსი იწერება:

მასივის სახელი [ინდექსი]

Array ობიექტს აქვს **length** თვისება, რომლის მნიშვნელობა არის მასივის სიგრძე. ამ თვისების შედეგის მისაღებად უნდა ჩავწეროთ შემდეგი გამოსახულება:

მასივის სახელი . length

მაგალითი:

შევქმნათ მასივი **earth**, რომლის ელემენტები შეიცავს ჩვენი პლანეტის ზოგიერთ ინფორმაციას. ამ მასივის ელემენტები სხვადასხვა ტიპისაა (სტრიქონული და რიცხვითი).

```
earth = new Array ( 4 )                 /* 4 ელემენტიანი მასივი */
earth [0] = "პლანეტა"
earth [1] = "24 საათი"
earth [2] = 6378
earth [3] = 365.25
earth. length                           /* მნიშვნელობა ტოლია 4-ის */
```

მასივის შექმნის სხვა საშუალება მდგომარეობს უშუალოდ მასივის ელემენტების განსაზღვრაში.

მაგალითი:

```
earth= new Array ("პლანეტა", "24 საათი", 6378, 365.25 )
```

ამ შემთხვევაში **JavaScript**-ი ავტომატურად ქმნის ინდექსებს მასივის ელემენტებისათვის. მასივის შექმნის მესამე წესი არის მასივის თითოეული ელემენტისათვის ობიექტის თვისების მსგავსად სახელის მინიჭება.

მაგალითი:

```
earth = new  Array  ( )                    /* ცარიელი მასივი */
earth.xtype = "პლანეტა"
earth.xday = "24 საათი"
earth.radius = 6378
earth.period = 365.25
```

ამ შემთხვევაში ელემენტზე მიმართვა ხორციელდება როგორც ობიექტის თვისებაზე, მაგალითად, **earth**.**radius**. შესაძლებელია ასევე შემდეგი მიმართვაც: **earth ["radius"]**. მაგრამ ასეთ მასივში ელემენტებზე მიმართვა ინდექსით არ შეიძლება.

მრავალგანზომილებიანი მასივები. ზემოთ განხილული მასივები იყო ერთგანზომილებიანი. თუ რომელიმე ერთგანზომილებიანი მასივის ელემენტად ისევ მასივს შევქმნით, მაშინ მივიღებთ ორგანზომილებიან მასივს. ასეთი მასივის ელემენტებზე მიმართვა შემდეგნაირად ხდება:

მასივის სახელი [I დონის ინდექსი] [II დონის ინდექსი]

თუ მასივი უფრო მეტი განზომილებისაა, მაშინ შესაბამისად ემატება ინდექსები.

მაგალითი:

```
M = new Array ( )
M [0] = new Array ("M 1.1", "M 1.2", "M 1.3" )
M [1] = new Array ("M 2.1", "M 2.2" )
M [2] = new Array ("M 3.1", "M 3.2", "M 3.3", "M 3.4")
```

იმისათვის, რომ მივმართოთ რომელიმე ელემენტს, საჭიროა ჩავწეროთ:

```
M [2] [1]                  /* მნიშვნელობა ტოლი იქნება "M 3.2" */
```

ზემოთ მოყვანილი მაგალითის მსგავსად შეიძლება მრავალგანზომილებიანი მასივების შექმნა.

მასივის კოპირება. ზოგჯერ საჭიროა მასივის ასლის შექმნა, რათა მოხდეს საწყისი მნიშვნელობების შენახვა და მოხდეს ამ ელემენტების შემდგომი მოდიფიცირებისაგან დაცვა. ამასთან, მასივის კოპირებისათვის საკმარისი არ არის მხოლოდ სხვა ცვლადისათვის მისი მინიჭება. ახალი ცვლადისა და მინიჭების ოპერატორის გამოყენებით ვქმნით მხოლოდ ძველ მასივზე ახალ მიმართვას, და არა ახალ მასივს.

მაგალითი:

```
a = new Array ( 5, 2, 4, 3 )
x = a                    /* მიმართვა a მასივზე */
a [2] = 25               /* მე-3 ელემენტის მნიშვნელობის შეცვლა */
x [2]                    /* მნიშვნელობა ტოლია 25-ის */
```

ამ მაგალითში **a** და **x** მასივები ერთმანეთს ემთხვევა.

იმისათვის, რომ მასივის კოპირება მოხდეს, ანუ ახალი მასივი შეიქმნას, რომლის ელემენტები საწყისი მასივის ელემენტების ტოლია, საჭიროა ვისარგებლოთ ციკლის ოპერატორით, რომელშიც ახალი მასივის ელემენტებს მიენიჭება საწყისი მასივის ელემენტების მნიშვნელობები.

მაგალითი:

```
a = new Array ( 5, 2, 4, 3 )
x = new Array ( )
for ( i = 0; i < a . length; i ++ )
{ x [i] =a [i] }
```

თვისება Array.

length – მასივის სიგრძე ან ელემენტთა რაოდენობა მასივში. მთელი რიცხვია. მისი სინტაქსია:

```
მასივის სახელი . length
```

prototype – თვისება საშუალებას იძლევა დავამატოთ ახალი თვისება და მეთოდი ყველა შექმნილ მასივს, თუ არსებული საკმარისი არ იქნება.

მაგალითი:

მასივის ელემენტების ჯამის გამოთვლა. განვსაზღვროთ **aSum ( )** ფუნქცია, რომელიც აბრუნებს რიცხვითი მასივის ელემენტების ჯამს.

```
function aSum ( xarray ) {
   var s = 0
   for ( i = 0; i <= xarray . length - 1; i ++ ) {
         s = s + xarray [i]
   }
   return s
}
myarray = new Array ( 2, 3, 4 )
Array . prototype . Sum = aSum
myarray . Sum ( myarray )
```

ეს მაგალითი არის უბრალოდ იმის ილუსტრაცია, თუ როგორ შეიძლება **prototype** თვისების გამოყენება. მასივის ელემენტების ჯამის გამოსათვლელად საკმარისია ჩაიწეროს შემდეგი გამოსახულება:

```
s = aSum ( myarray )
```

Array მეთოდები. ობიექტ **Array**-ის მეთოდები განკუთვნილია მასივის სტრუქტურებში შენახული მონაცემების მართვისათვის.

1. **concat** (მასივი) – მასივის კონკატენაცია, ორი მასივის მესამე მასივად გაერთიანება. მისი სინტაქსია:

```
მასივის სახელი1 . concat  (მასივი2)
```

შედეგად მიიღება მასივი. მოცემული მეთოდი არ ცვლის საწყის მასივებს.

მაგალითი:

```
a1 = new Array ( 1, 2, "a" )
a2 = new Array ("b", "c", "d", "e" )
a3 = a1 . concat ( a2 )          /* შედეგი არის მასივი ელემენტე-
                                    ბით 1, 2, "a", "b", "c", "d", "e"  */
```

2. **join (გამყოფი)** – მასივის ელემენტებისაგან ქმნის სტრიქონს, სადაც ელემენტები ერთმანეთისაგან გამოყოფილია "**გამყოფით**". მისი სინტაქსია:

```
მასივის სახელი . join  (გამყოფი)
```

შედეგად მიიღება სიმბოლოების სტრიქონი.

მაგალითი:

```
a = new Array ( 1, 2, "a", "b", "c" )
a . join ( “,” )          /* შედეგი არის სტრიქონი 1, 2, a, b, c  */
a = new Array ( 1, 2, "a", "b", "c" )
a . join ( “ ” )          /* შედეგი არის სტრიქონი 1 2 a b c  */
```

3. **pop** ( ) – შლის მასივის ბოლო ელემენტს და აბრუნებს მის მნიშვნელობას. მისი სინტაქსია:

მასივის სახელი . pop ( )

შედეგად მიიღება მასივის წაშლილი ელემენტის მნიშვნელობა. მოცემული მეთოდი ცვლის საწყის მასივს.

4. **push (მნიშვნელობა/ობიექტი)** – მოცემულ მნიშვნელობას მასივის ბოლო ელემენტად ამატებს და აბრუნებს მასივის ახალ სიგრძეს. მისი სინტაქსია:

მასივის სახელი1 . push (მნიშვნელობა/ობიექტი)

შედეგად მიიღება რიცხვი. მოცემული მეთოდი ცვლის საწყის მასივს.

5. **shift ( )** – შლის მასივის პირველ ელემენტს და აბრუნებს მის მნიშვნელობას. მისი სინტაქსია:

მასივის სახელი . shift ( )

შედეგად აბრუნებს მასივის წაშლილი ელემენტის მნიშვნელობას. მოცემული მეთოდი ცვლის საწყის მასივს.

6. **unshift (მნიშვნელობა/ობიექტი)** – მოცემულ მნიშვნელობას ამატებს მასივის პირველ ელემენტს. მისი სინტაქსია:

მასივის სახელი1 . unshift (მნიშვნელობა/ობიექტი)

შედეგად არ აბრუნებს არაფერს. მოცემული მეთოდი ცვლის საწყის მასივს.

7. **reverse ( )** – ცვლის მასივის ელემენტების მიმდევრობას საწინააღმდეგო მიმდევრობით. მისი სინტაქსია:

მასივის სახელი . reverse ( )

შედეგი იქნება მასივი. მოცემული მეთოდი ცვლის საწყის მასივს.

მაგალითი:

```
a1 = new Array ( 1, 2, "a", "b" )
a . reverse ( )                 /* შედეგი არის მასივი ელემენტებით
                                "a", "b", 1, 2  */
```

8. **slise (ინდექსი1 [, ინდექსი2])** – ეს მეთოდი მოცემულ დიაპაზონში არსებული ელემენტებისაგან საწყისი მასივის ინდექსებით ქმნის მასივს. მისი სინტაქსია:

მასივის სახელი . slise (ინდექსი1 [, ინდექსი2])

მიღებული ოპერაციის შედეგი იქნება მასივი. მოცემული მეთოდი არ ცვლის საწყის მასივს. მეორე პარამეტრი აუცილებელი არ არის. თუ მოცემულია მხოლოდ ერთი პარამეტრი, მაშინ შედეგად მიღებული მასივის ელემენტები საწყისი მასივის ელემენტები იქნება, დაწყებული **ინდექსი1**-დან მასივის ბოლომდე. თუ მოცემულია ორივე პარამეტრი, მაშინ საწყისი მასივიდან აიღება ელემენტები **ინდექსი1**-დან **ინდექსი2**-მდე ამ ბოლო ელემენტის გამოკლებით.

მაგალითი:

```
a = new Array ( 1, 2, "a", "b", "c" )
a . reverse ( 1, 3 )  /* შედეგი არის მასივი ელემენტებით 2,  "a" */
a . reverse ( 3 )     /* შედეგი არის მასივი ელემენტებით "b", "c" */
```

9. **sort ([შედარების ფუნქცია])** – შედარების ფუნქციის დახმარებით მასივის ელემენტების მოწესრიგება (დალაგება). მისი სინტაქსია:

მასივის სახელი . sort ([შედარების ფუნქცია])

ოპერაციის შედეგი იქნება მასივი. მოცემული მეთოდი ცვლის საწყის მასივს. პარამეტრი აუცილებელი არ არის. პარამეტრად შეიძლება მითითებული იყოს ელემენტების მოწესრიგების საკუთარი ფუნქციის სახელი.

10. **splice (ინდექსი, რაოდენობა [, ელემ1 [, ელემ2 [, ... ელემN]]])** – მასივიდან ამოშლის რამდენიმე ელემენტს და აბრუნებს ამოშლილი ელემენტებისაგან შედგენილ მასივს ან ცვლის ელემენტების მნიშვნელობას. მისი სინტაქსია:

მასივის სახელი . splice (ინდექსი, რაოდენობა [, ელემ1 [, ელემ2 [, ... ელემN]]])

მიღებული ოპერაციის შედეგი იქნება მასივი. მოცემული მეთოდი ცვლის საწყის მასივს. პირველი ორი პარამეტრი აუცილებელია. პირველი პარამეტრი არის პირველი წასაშლელი ელემენტის ინდექსი, ხოლო მეორე – წასაშლელი ელემენტების რაოდენობა.

მაგალითი:

1. **a = new Array (1, 2, "a", "b", "c", "d")**
   x = a . splice (1, 3) /* **x** მასივის ელემენტებია – **2, "a", "b"** */
   /* **a** მასივის ელემენტებია – **1, "c", "d"** */
2. **a = new Array (1, 2, "a", "b", "c", "d")**
   x = a . splice (1, 3, "e", "f", "g")
   /* **x** მასივის ელემენტებია – **2, "a", "b"** */
   /* **a** მასივის ელემენტებია – **1, "e", "f", "g", "c", "d"** */

11. **toLocaleString ( ), toString ( )** – მასივის ელემენტებს გარდაქმნის სიმბოლურ სტრიქონად. **toLocaleString ( )** მეთოდით გარდაქმნის ალგორითმი დამოკიდებულია ბრაუზერის ვერსიაზე.

რიცხვითი მასივების დამუშავების ფუნქციები. ხშირად საჭიროა რიცხვითი მონაცემების სტატისტიკური მახასიათებლების გამოთვლა: ყველა მონაცემების ჯამი, საშუალო, მაქსიმალური და მინიმალური მნიშვნელობა.

1. არაცარიელი მასივის ყველა ელემენტთა ჯამის გამოსათვლელი ფუნქცია:

```
function S ( aN ) {
var S = aN [0]
for ( var i = 1; i <= aN . length – 1; i ++ ) {
                S += aN [i]
}
return S
}
```

ცხადია, საშუალო არითმეტიკულის გამოსათვლელად უნდა ვისარგებლოთ გამოსახულებით **S (aN) / aN . length**.

2. მასივის ელემენტებს შორის მინიმალური ელემენტის მნიშვნელობის პოვნის ფუნქცია:

```
function Nmin ( aN ) {
var Nmin = aN [0]
for ( var i = 1; i <= aN . length – 1; i ++ ) {
                if ( aN [i] < Nmin )
                        Nmin= aN [i]
}
return Nmin
}
```

3. მასივის ელემენტებს შორის მაქსიმალური ელემენტის მნიშვნელობის პოვნის ფუნქცია:

```
function Nmax ( aN ) {
var Nmax = aN [0]
for ( var i = 1; i <= aN . length – 1; i ++ ) {
                if ( aN [i] < Nmax )
                        Nmax= aN [i]
}
return Nmax
}
```

4. შევქმნათ ერთი ფუნქცია, რომლის საშუალებითაც შეიძლება ყველა ზემოთ ჩამოთვლილი სტატისტიკური მახასიათებლები გამოვთვალოთ და რომელიც შედეგად ამ მნიშვნელობებს მასივის სახით დააბრუნებს:

```
function statistic ( aN ) {
if ( aN == 0 | | aN == null | | aN == " " )
   return new Array ( 0, 0, 0, 0 )
var S = aN [0]
var Nmin = aN [0]
var Nmax = aN [0]
for ( var i = 1; i <= aN . length – 1; i ++ ) {
                        S += aN [i]
                if ( aN [i] < Nmin )
                        Nmin= aN [i]
                if ( aN [i] < Nmax )
                        Nmax= aN [i]
```

```
}
return new Array ( S, S / aN . length, Nmin, Nmax )
}
```

13. ობიექტი Number (რიცხვი)

Web-გვერდების დამუშავების დროს მათემატიკური ოპერაციები შედარებით იშვიათად გამოიყენება, ვიდრე სტრიქონული. ჩვეულებრივ ისინი დაკავშირებულია გვერდების კოორდინატების ცვლილებასთან, მაგრამ, გვხვდება უფრო რთული შემთხვევებიც. ამოცანებში საქმე გვაქვს რიცხვებთან. რიცხვებთან დაკავშირებით ჩვენ უკვე ვისაუბრეთ ლექციაში, რომელიც ეძღვნებოდა მონაცემთა ტიპებს. ახლა განვიხილოთ რიცხვები უფრო დაწვრილებით.

რიცხვები JavaScript-ში. **JavaScript**-ში რიცხვები შეიძლება იყოს ორი ტიპის: მთელი და მცურავმძიმიანი. მთელ რიცხვებს წილადი ნაწილი არა აქვს და გამყოფ წერტილს არ შეიცავს.

ამასთან, **JavaScript**-ში რიცხვები შეიძლება ათვლის სხვადასხვა სისტემაში იყოს წარმოდგენილი, კერძოდ: 10 (ათობითი), 16 (თექვსმეტობითი), 8 (რვაობითი).

რიცხვების თექვსმეტობითი ფორმით ჩაწერა **0x** (ან **0X**) პრეფიქსით იწყება, სადაც პირველი სიმბოლო არის ნული, ხოლო შემდეგ მოჰყვება თექვსმეტობითი ციფრების სიმბოლოები: **0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f** (ასოები შეიძლება ჩაიწეროს ნებისმიერ რეგისტრში).

რიცხვების რვაობითი ფორმით ჩაწერა ნულით იწყება, ხოლო შემდეგ მოჰყვება ციფრები **0**-დან **7**-მდე. არითმეტიკულ გამოსახულებაში რიცხვები შეიძლება ჩაწერილი იყოს ათვლის ნებისმიერ სისტემაში, მაგრამ შედეგი ყოველთვის ათვლის ათობით სისტემაში მიიღება.

Number ობიექტის შექმნა. რიცხვები, ცვლადებისა და მინიჭების ოპერატორის დახმარებით, შეიძლება ჩვეულებრივი წესით შეიქმნას. მაგრამ ობიექტის გამოყენებით რიცხვის შექმნას ზოგიერთი სასარგებლო თვისებები და მეთოდები აქვს, რომელიც ზოგჯერ შეიძლება გამოგვადგეს. ობიექტი **Number** შემდეგი სახის გამოსახულების დახმარებით იქმნება:

ცვლადი = new Number (რიცხვი)

რიცხვითი ობიექტების თვისებებსა და მეთოდებზე მიმართვა შემდეგი გამოსახულებით ხდება:

რიცხვი . თვისება
Number. თვისება
რიცხვი . მეთოდი ([პარამეტრი])
Number. მეთოდი ([პარამეტრი])

თვისება Number

1. **MAX_VALUE** – მუდმივა, რომლის მნიშვნელობა **JavaScript**-ში გამოყენებული უდიდესი დასაშვები რიცხვის მნიშვნელობის ტოლია (**1.7976931348623157e+308**).

2. **MIN_VALUE** – მუდმივა, რომლის მნიშვნელობა **JavaScript**-ში გამოყენებული უმცირესი დასაშვები რიცხვის მნიშვნელობის ტოლია (**5e-324**).

3. **NEGATIVE_INFINITY** – რიცხვი, რომლის მნიშვნელობა ნაკლებია, ვიდრე **Number . MIN_VALUE**-ზე.

4. **POSITIVE_INFINITY** – რიცხვი, რომლის მნიშვნელობა მეტია, ვიდრე **Number . MAX_VALUE**-ზე.

5. **NaN** – მუდმივა, რომლის მნიშვნელობაა **NaN**, მისი საშუალებითაც **JavaScript**-ი გვამცნობს, რომ მოცემული მონაცემი რიცხვი არ არის.

6. **prototype** – თვისება, რომელიც საშუალებას იძლევა ყველა შექმნილ რიცხვს დავამატოთ ახალი თვისება და მეთოდი, თუ უკვე არსებული საკმარისი არ იქნება.

მეთოდი Number. **Number** ობიექტს აქვს რამდენიმე მეთოდი, მაგრამ ჩვენ მხოლოდ ოთხი მათგანი განვიხილოთ.

1. **toExponential (რაოდენობა)** – არის რიცხვი ექსპონენციალური სახით. მისი სინტაქსია:

რიცხვი . toExponential (რაოდენობა)

შედეგად მიიღება სტრიქონი. პარამეტრი მთელი რიცხვია და მიუთითებს რამდენი ციფრი მიუთითოს ათობითი წერტილის შემდეგ.

მაგალითი:

```
x = new Number (456)
x . toExponential ( 3 )                    /*  4.560e+2  */
```

2. **toFixed (რაოდენობა)** – არის რიცხვი ათობითი წერტილის შემდეგ ფიქსირებული რაოდენობის ციფრებით. მისი სინტაქსია:

რიცხვი . toFixed (რაოდენობა)

შედეგად მიიღება სტრიქონი. პარამეტრი მთელი რიცხვია და მიუთითებს რამდენი ციფრი მიუთითოს ათობითი წერტილის შემდეგ.

მაგალითი:

```
x = new Number (25.65)
x . toFixed  ( 1 )                         /*  25.7  */
```

3. **toPrecision (სიზუსტე)** – არის რიცხვი მოცემული ნიშნადი ციფრების საერთო რაოდენობით. მისი სინტაქსია:

რიცხვი . toPrecision (სიზუსტე)

ამ მეთოდის შედეგი არის სტრიქონი. პარამეტრი მთელი რიცხვია და მიუთითებს სულ რამდენი ციფრი უნდა იყოს ათობითი წერტილის წინ და შემდეგ.

მაგალითი:

```
x = new Number (135.456)
x . toPrecision ( 7 )          /*  135.4560  */
x . toPrecision ( 3 )          /*  135  */
x . toPrecision ( 1 )          /*  1e2  */
x . toPrecision ( 0 )          /* შეტყობინება შეცდომის შესახებ */
```

4. **toString ([ფუძე])** – გადაჰყავს რიცხვი ათვლის მითითებულ სისტემაში. მისი სინტაქსია:

რიცხვი . toString ([ფუძე])

შედეგი არის სტრიქონი. პარამეტრი მთელი რიცხვია და მიუთითებს ათვლის სისტემის ფუძეს, რომელშიც რიცხვი უნდა გადაიყვანოს. თუ პარამეტრი მითითებული არ არის, მაშინ 10 იგულისხმება.

14. ობიექტი Math (მათემატიკა)

ობიექტი **Math** განკუთვნილია ზოგიერთი მათემატიკური კონსტანტების (მაგალითად, რიცხვი π) შესანახად და ტიპური მათემატიკური ფუნქციების საშუალებით რიცხვების გარდაქმნისათვის. **Math** ობიექტის თვისებებსა და მეთოდებზე მიმართვა შეიძლება განხორციელდეს შემდეგი გამოსახულებით:

Math. თვისება

Math. მეთოდი (პარამეტრი)

თვისებები Math. **Math** ობიექტის თვისებებს აქვთ თავისი მნიშვნელობების სახით მათემატიკური კონსტანტები.

E	ეილერის მუდმივა
LN10	რიცხვ 10-ის ნატურალური ლოგარითმის მნიშვნელობა
LN2	რიცხვ 2-ის ნატურალური ლოგარითმის მნიშვნელობა
LOG10	ექსპონენტის (რიცხვი e) ათობითი ლოგარითმის მნიშვნელობა
LOG2	ექსპონენტის ათობითი ლოგარითმის მნიშვნელობა
PI	π რიცხვის მნიშვნელობა
SQRT1_2	1/2-დან კვადრატული ფესვის მნიშვნელობა
SQRT	2-დან კვადრატული ფესვის მნიშვნელობა

მაგალითი:

წრეწირის სიგრძის გამოთვლა.

```
var R = 10
circus = 2 * R * Math . PI
```

მეთოდები Math.

- **abs (რიცხვი)** – რიცხვის მოდულის (აბსოლუტური სიდიდე) გამოთვლა;
- **acos (რიცხვი)** – რიცხვის არკკოსინუსი;
- **asin (რიცხვი)** – რიცხვის არკსინუსი;
- **atan (რიცხვი)** – რიცხვის არკტანგენსი;
- **atan2 (x,y)** – წერტილის კუთხე პოლარულ კოორდინატებში;
- **ceil (რიცხვი)** – ამრგვალებს რიცხვს მეტობით უახლოეს მთელამდე;
- **cos (რიცხვი)** – რიცხვის კოსინუსი;

- **exp (რიცხვი)** – ***e*** რიცხვი ხარისხად პარამეტრი;

- **floor (რიცხვი)** - ამრგვალებს რიცხვს ნაკლებობით უახლოეს მთელამდე;

- **log (რიცხვი)** – რიცხვის ნატურალური ლოგარითმი;

- **max (რიცხვი1, რიცხვი2)** – მაქსიმუმი ორ რიცხვს შორის;

- **min (რიცხვი1, რიცხვი2)** – მინიმუმი ორ რიცხვს შორის;

- **pow (რიცხვი1, რიცხვი2)** – **რიცხვი1** ხარისხად **რიცხვი2**;

- **random** ( ) – შემთხვევითი რიცხვი 0-სა და 1-ს შორის;

- **round (რიცხვი)** – რიცხვის დამრგვალება უახლოეს მთელ რიცხვამდე;

- **sin (რიცხვი)** – რიცხვის სინუსი;

- **sqrt (რიცხვი)** – კვადრატული ფესვი რიცხვიდან;

- **tan (რიცხვი)** – რიცხვის ტანგენსი.

თუ ტრიგონომეტრიული ფუნქციის გამოთვლის დროს პარამეტრი მოცემულია გრადუსებში, მაშინ პარამეტრი უნდა ჩაიწეროს ასე:

Math . PI * x / 180

ზოგიერთი მათემატიკური ამოცანის ამოხსნის ფუნქციები.

JavaScript-ი შეიძლება გამოვიყენოთ არა მარტო ვებ-გვერდების მართვისათვის, არამედ სხვადასხვა გამოთვლითი და მათემატიკური ამოცანის ამოსახსნელად. ამიტომ განვიხილოთ რამდენიმე მარტივი და სასარგებლო გამოთვლების ხასიათის პროგრამა:

1. **კვადრატული განტოლების ამოხსნა.** ამ ტიპის ამოცანას ჯერ კიდევ სკოლის პერიოდიდან ვიცნობთ. საჭიროა ვიპოვოთ კვადრატული განტოლების $ax^2 + bx + c = 0$ ნამდვილი ფესვები x_1 და x_2.

ცხადია, რომ ამ ფუნქციისათვის გარედან მისაწოდებელი პარამეტრები a, b და c კოეფიციენტები იქნება, ხოლო დასაბრუნებელი მნიშვნელობა - მასივი. თუ განტოლებას ამონახსნი არ ექნება, მაშინ ეს მასივი ცარიელი იქნება. თუ განტოლებას ორი ფესვი აქვს, მაშინ ისინი ჩაიწერება როგორც დასაბრუნებელი მასივის ელემენტები, ხოლო თუ განტოლებას ერთი ფესვი აქვს, მაშინ იგი ჩაიწერება მასივის პირველ ელემენტად, ხოლო მეორე ელემენტი ცარიელი (**null**) იქნება.

ქვემოთ მოყვანილია ამ პროგრამის ერთ-ერთი ვარიანტი:

```
function beg ( a, b, c)  {
var aret = new Array ( )
var D = b * b – 4 * a * c
if ( a == 0 )  {
   if ( ! (b == 0 ) )  {
          aret [0] = - c / b
          aret [1] = null
   }
return aret
}
   if ( D == 0 )  {
          aret [0] = - b / 2 / a
          aret [1] = aret [0]
   }
   if ( D > 0 )  {
          aret [0] = ( - b – Math . sqrt ( D ) ) / 2 / a
          aret [1] = ( - b + Math . sqrt ( D ) ) / 2 / a
   }
return aret
}
```

ფუნქციის შესრულება შევამოწმოთ რამდენიმე მაგალითზე:

```
beg ( 0, 2, 6 )          /* მასივი ( -3, null) */
beg ( 1, -2, 1 )         /* მასივი ( 1, 1 ) */
beg ( 3, 4, -2.5 )       /* მასივი ( -1.797054997187545,
                         0.4637216638542114 ) */
beg ( 2, 0, 5 )          /* ცარიელი მასივი */
```

2. **ინტეგრალის გამოთვლა.** ერთი ცვლადის $f(x)$ ფუნქციიდან ინტეგრალი a-დან b-მდე, შეიძლება გავიგოთ როგორც იმ ფიგურის ფართობი, რომელიც შემოსაზღვრულია $y = f(x)$ მრუდით, x ღერძით და ღერძის პარალელური ორი წრფით, რომლებიც გადის a და b წერტილებზე. აქვე უნდა აღინიშნოს, რომ ინტეგრალის გამოთვლის დროს ერთი გამონაკლისი უნდა დავუშვათ. ეს არის ის, რომ თუ მრუდი x ღერძის ქვემოთ გადის, მაშინ ფართობი უარყოფითი იქნება.

ასეთი ფიგურის ფართობის გამოსათვლელად საკმარისია ეს ფიგურა მცირე ზომის ფიგურებად დავყოთ, რათა გამოითვალოს

თითოეულის ფართობი და შეიკრიბოს. ელემენტარულ ფიგურად შეიძლება ავიღოთ მართკუთხედი ან ტრაპეცია. უფრო დიდი სიზუსტისათვის ასეთ ფიგურად ტრაპეცია გამოდგება.

ამ ფუნქციისათვის გარედან მისაწოდებელი პარამეტრები იქნება $f(x)$ ფუნქცია, რომელშიც x ცვლადი ლათინური პატარა ასოს სახით ჩაიწერება და ორი a და b, შესაბამისად ინტეგრირების ინტერვალის საწყისი და საბოლოო კოეფიციენტები იქნება. დასაბრუნებელი მნიშვნელობა არის ფართობი **S**. ქვემოთ მოყვანილია ამ პროგრამის ერთ-ერთი ვარიანტი:

```
function integral (expression, a,b)  {
var x, y1, y2, n, length, dx, S = 0
length = Math . abs (b – a)
n = 100
dx = length / n
x = a
y1 = eval (expression)
x = a + dx
y2 = eval (expression)
S = (y1 + y2) * dx / 2
for ( i = 2; i <= n; i ++ )  {
   y1 = y2
   x = x + dx
   y2 = eval (expression)
   S += (y1 + y2) * dx / 2
}
return S
}
```

ამ ფუნქციის მუშაობის შემოწმება მოვახდინოთ შემდეგ მონაცემებზე:

```
integral ("x", 0, 1 )                /*  0.5  */
integral ("x * x", 0, 1 )            /*  0.33335  */
integral ("x * x", 0, 10 )    /*  333.349999  */
```

თუ გამოთვლების სიზუსტე არ გვაკმაყოფილებს, მაშინ ეს შეიძლება გამოვასწოროთ ინტერვალის უფრო მცირე ინტერვალებად დაყოფის გზით. თუ ინტერვალის სიგრძე 2-ს აღემატება, მაშინ

სპეციალური ფორმულით უნდა მოხდეს ამ ინტერვალის დაყოფის შერჩევა. შედეგად ჩვენი პროგრამა ასეთ სახეს მიიღებს:

```
function integral (expression, a,b)  {
if ( !expression | | !b&&! a )
   return 0
if ( a == b )
   return 0
var x, y1, y2, n, length, dx, S = 0
length = Math . abs (b – a)
y1 = Math . min (a, b)
b = Math . max (a, b)
a = y1
n = 100
if (length > 2) n = Math . round ( 100 * Math . log (length + 1) )
dx = length / n
x = a
y1 = eval (expression)
x = a + dx
y2 = eval (expression)
S = (y1 + y2) * dx / 2
for ( i = 2; i <= n; i ++ )  {
   y1 = y2
   x = x + dx
   y2 = eval (expression)
   S += (y1 + y2) * dx / 2
}
return S
}
```

ფუნქციის მუშაობის შევამოწმოთ შემდეგ მონაცემზე:

```
integral ("x * x", 0, 10 )          /*  333.33749999  */
```

3. **წარმოებულის გამოთვლა.** ერთი ცვლადის ფუნქციის წარმოებული წერტილში, შეიძლება გავიგოთ როგორც ამ გამოსახულების მნიშვნელობის ცვლილების სიჩქარე, როდესაც x-ის მნიშვნელობა a ტოლია. ამისათვის, საჭიროა გამოითვალოს გამოსახულების მნიშვნელობა ორ, მოცემულ წერტილთან ახლო მდგომ წერტილში, ხოლო

შემდეგ ამ მნიშვნელობათა სხვაობა უნდა გაიყოს ამ ორ წერტილს შორის მანძილზე.

ამ ფუნქციისათვის გარედან მისაწოდებელი პარამეტრები $f(x)$ ფუნქცია იქნება, რომელშიც x ცვლადი ჩაიწერება ლათინური პატარა ასოს სახით და a წერტილი, რომელშიც წარმოებულის მნიშვნელობა უნდა გამოითვალოს. პროგრამულ კოდს შემდეგი სახე ექნება:

```
function Dydx (expression, a)  {
if ( !expression )
   return 0
var x, y, dx
dx = 0.000025
x = a – dx
y = eval (expression)
x = a + dx
return ( eval (expression) – y ) / 2 / dx )
}
```

შესამოწმებელი მაგალითი:

```
Dydx ("x", 1 )                    /*  0.9999999  */
Dydx ("x * x", 1 )          /*  1.9999999  */
```

4. **ექსტრემუმის პოვნა.** $f(x)$ ფუნქციის ექსტრემუმი ეს არის ისეთი (x_0, y_0) წერტილის პოვნა, რომელშიც $f(x)$ ფუნქცია ღებულობს მაქსიმალურ ან მინიმალურ მნიშვნელობას. ეს წერტილი შეიძლება წარმოებულის საშუალებით გამოითვალოს (ფუნქციის წარმოებული მოცემულ წერტილში ნულის ტოლია).

ამ ფუნქციისათვის გარედან მისაწოდებელი პარამეტრები $f(x)$ ფუნქცია იქნება, რომელშიც x ცვლადი ჩაიწერება ლათინური პატარა ასოს სახით, მეორე და მესამე პარამეტრები, რომელშიც ექსტრემუმი უნდა გამოითვალოს, ხოლო მეოთხე პარამეტრი მიუთითებს შედეგის სიზუსტეს. საძიებო კოდს შემდეგი სახე ექნება:

```
function extremum (expression, a, b, d )  {
if ( !d ) d = 0.001
var x, y1, y2, i
x = a
y1 = Dydx ( expression, a )
```

```
i = 1
while ( ( x <= b ) && ( i <= 1000 ) )  {
y2 = Dydx ( expression, x + d )
if ( ( y1 < 0 ) && ( y2 >= 0 ) | | ( y2 < 0 ) && ( y1 >= 0 ) )  {
x = x + d * Math . abs ( y1 ) / ( Math . abs ( y1 ) + Math . abs ( y2 ) )
return new Array ( x, eval (expression) )
}
x += d
y1 = y2
i ++
}
return new Array ( )
}
```

შესამოწმებელი მაგალითი:

```
extremum ("x * x - 1", -1, 1, 0.01 )          /*  (7.528699e-16, -1)  */
```

15. ობიექტი Date (თარიღი)

პროგრამათა უმრავლესობაში საჭიროა თარიღისა და დროის ასახვა, დღეების რაოდენობის გამოთვლა და სხვა. ხშირად ზოგიერთი პროგრამების მართვა თარიღისა და დროის მიხედვით ხდება. ამასთან, უნდა გვახსოვდეს, რომ არსებობს დროის სარტყელი და დროის სეზონური ცვლილებები (ზაფხულისა და ზამთრის დრო).

იმისათვის, რომ მოვახდინოთ ორგანიზაციისა და ფიზიკური პირის მოქმედების კოორდინაცია დროში, ჩვენი პლანეტის სხვადასხვა წერტილში შემოღებულ იქნა დროის ათვლის სისტემა. იგი უკავშირდება მერიდიანს, რომელიც გადის დიდი ბრიტანეთის ქალაქ გრინვიჩში მდებარე ასტრონომიულ ობსერვატორიაზე. ამ დროით ზონას უწოდებენ საშუალო დროს გრინვიჩის მიხედვით (**Greenwich Mean Time** – **GMT**). ახლახან ამ აბრევიატურის გარდა გამოჩნდა კიდევ ერთი – **UTC** (**Universal Time Coordinated** – საყოველთაო კოორდინირებული დრო).

თუ ჩვენი კომპიუტერის დრო დაყენებულია სწორად, მაშინ დროის ათვლა **GMT** სისტემაში ხდება, მაგრამ ამოცანათა პანელზე ლოკალურ დროს აჩვენებს, რომელიც ჩვენი დროის სარტყელს შეესაბამება.

JavaScript-ზე დაწერილ პროგრამაში უბრალოდ არ შეიძლება ჩავწეროთ თარიღი (მაგ., 30.10.2007). დროისა და თარიღის შექმნა სპეციალურად **Date** ობიექტის გამოყენებით ხდება.

Date ობიექტის შექმნა. ობიექტი **Date** იქმნება შემდეგი გამოსახულების საშუალებით:

თარიღის ობიექტის სახელი = new Date ([პარამეტრი])

პარამეტრი აუცილებელი არ არის. ყურადღება უნდა მიექცეს იმას, რომ **თარიღის ობიექტის სახელი** თარიღის ობიექტი არის და არა სხვა რომელიმე ტიპის მნიშვნელობა.

თარიღის ობიექტებზე მანიპულაციისათვის გამოიყენება **Date** ობიექტის მრავალი მეთოდი. ამასთან გამოიყენება ასეთი სინტაქსი:

ცვლადი = თარიღის ობიექტის სახელი . მეთოდი ( )

ხოლო თარიღის ობიექტისათვის ახალი მნიშვნელობის მისანიჭებლად გამოიყენება შესაბამისი მეთოდი:

ცვლადი = თარიღის ობიექტის სახელი . მეთოდი (ახალი მნიშვნელობა)

თარიღის ობიექტის **new Date ( )** გამოსახულებით შექმნის დროს შეიძლება პარამეტრის სახით მივუთითოთ ის თარიღი და დრო, რომელიც ამ ობიექტზე უნდა დავაყენოთ. ეს შეიძლება ხუთი ხერხით გაკეთდეს:

new Date ("თვე, dd, yyyy hh:mm:ss")
new Date ("თვე, dd, yyyy")
new Date (yy, mm, dd, hh, mm, ss)
new Date (yy, mm, dd)
new Date (მილიწამი)

პირველ ორ შემთხვევაში პარამეტრი სტრიქონის სახით მიეთითება, რომელშიც თარიღისა და დროის კომპონენტებია მითითებული. ასოებით აღნიშნულია პარამეტრის შაბლონი. ყურადღება უნდა მიექციეს გამყოფებს – მძიმესა და ორ წერტილს. დროის მითითება აუცილებელი არ არის. ამ შემთხვევაში დრო ნულის ტოლი იქნება. თარიღის კომპონენტი აუცილებლად უნდა მიეთითოს. თვის დასახელება უნდა ჩაიწეროს ინგლისურად (აბრევიატურა არ შეიძლება). დანარჩენი

კომპონენტები მიეთითება რიცხვების სახით. თუ რიცხვი 10-ზე ნაკლებია, მაშინ შეიძლება ერთი ციფრით ჩაიწეროს. მესამე და მეოთხე შემთხვევაში თარიღისა და დროის კომპონენტები რიცხვების სახით ჩაიწერება და ერთმანეთისაგან მძიმით გამოიყოფა.

ბოლო შემთხვევის დროს უნდა ჩაიწეროს მთელი რიცხვი, რაც მიუთითებს მილიწამების რაოდენობას, რომელიც გასულია 1970 წლის 1 იანვრიდან (ანუ 00:00:00 მომენტიდან). მილიწამების რაოდენობის მიხედვით თარიღისა და დროის ყველა კომპონენტი გამოითვლება.

Date ობიექტის მეთოდები. თარიღის ობიექტში შენახულ თარიღზე და დროზე ინფორმაციის წასაკითხად ან შესაცვლელად ობიექტ **Date**-ის მეთოდები გამოიყენება. თარიღის ობიექტზე მეთოდის გამოსაყენებლად უნდა ჩაიწეროს:

თარიღის ობიექტის სახელი . მეთოდი ([პარამეტრი])

ყველა მეთოდის საკმაოდ დიდი რაოდენობა შეიძლება დაიყოს ორ კატეგორიად: მნიშვნელობის მიღების მეთოდები (მათ დასახელებას აქვს პრეფიქსი **get**) და ახალი მნიშვნელობის დადგენის მეთოდები (მათ დასახელებას აქვს პრეფიქსი **set**). თითოეულ კატეგორიაში გამოიყოფა მეთოდების ორი ჯგუფი – ლოკალური ფორმატისათვის და **UTC** ფორმატისათვის. ეს მეთოდები საშუალებას იძლევა თარიღისა და დროის ცალკეულ კომპონენტებზე (წელი, თვე, რიცხვი, კვირის დღე, საათი, წუთი, წამი, მილიწამი) ვიმუშაოთ.

იგივე დასახელების ფუნქციები, ოღონდ პრეფიქსი **set** შესაბამისი კომპონენტის ახალ მნიშვნელობას ადგენს.

მეთოდების გამოყენების დროს უნდა გვახსოვდეს, რომ არ უნდა გამოვიყენოთ გამოსახულება, სადაც დრო სხვადასხვა ფორმატში იქნება გამოყენებული. ამასთან, უნდა გვახსოვდეს, რომ თვეების, კვირის დღეების, საათის, წუთისა და წამის ნუმერაცია 0-ით იწყება. XX საუკუნის წლების ჩაწერა შესაძლებელია ორნიშნა რიცხვით, ხოლო 1900-ზე ნაკლები და 1999-ზე მეტი წლები აუცილებლად უნდა ჩაიწეროს ოთხნიშნა რიცხვით.

Date ობიექტის მეთოდები

მეთოდი	მნიშვნელობის დიაპაზონი	აღწერა
getFullYear ( )	1970-...	წელი
getYear( )	70-...	წელი
getMonth ( )	0-11	თვე (იანვარი = 0)
getDate ( )	1-31	რიცხვი
getDay ( )	0-6	კვირის დღე (კვირა = 0)
getHours ( )	0-23	საათი 24 საათიან ფორმატში
getMinutes ( )	0-59	წუთები
getSeconds ( )	0-59	წამები
getTime ( )	0-...	მილიწამები დაწყებული 1.1.70 00:00:00 **GMT**-დან
getMilliseconds ( )	0-...	მილიწამები დაწყებული 1.1.70 00:00:00 **GMT**-დან
getUTCFullYear ( )	1970-...	წელი **UTC**
getUTCMonth ( )	0-11	თვე **UTC** (იანვარი = 0)
getUTCDate ( )	1-31	რიცხვი **UTC**
getUTCDay ( )	0-6	კვირის დღე **UTC** (კვირა = 0)
getUTCHours ( )	0-23	საათი **UTC** 24 საათიან ფორმატში
getUTCMinutes ( )	0-59	წუთები **UTC**
getUTCSeconds ( )	0-59	წამები **UTC**
getUTCTime ( )	0-...	მილიწამები **UTC** 1.1.70 00:00:00 **GMT**-დან
getUTCMilliseconds( )	0-...	მილიწამები **UTC** 1.1.70 00:00:00 **GMT**-დან
getTimezoneOffset ( )	0-...	სხვაობა წუთებში **GMT**-სა და **UTC**-ს შორის
Date.parse (“dateStr”)		თარიღიანი სტრიქონის გარდაქმნა რიცხვად მილიწამებში
Date . UTC (მნიშვ.)		თარიღიანი სტრიქონის გარდაქმნა **GMT** რიცხვად
toDateString ( )		სტრიქონი თარიღით (დროის გარეშე) ბრაუზერის ფორმატში
toGMTString ( )		სტრიქონი თარიღით და დროით გლობალურ ფორმატში
toLocaleDateString ( )		სტრიქონი თარიღით (დროის გარეშე) ლოკალურ ფორმატში

მეთოდი	მნიშვნელობის დიაპაზონი	აღწერა
toLocaleDateString ( )		სტრიქონი თარიღით და დროით ლოკალურ ფორმატში
toLocaleTimeString ( )		სტრიქონი დროით (თარიღის გარეშე) ლოკალურ ფორმატში
toString ( )		სტრიქონი თარიღით და დროთ ბრაუზერის ფორმატში
toTimeString ( )		სტრიქონი დროით (თარიღის გარეშე) ბრაუზერის ფორმატში
toUTCString ( )		სტრიქონი თარიღით და დროით ლოკალურ ფორმატში

მაგალითი:

1. განვსაზღვროთ თარიღი, რომელიც ერთი კვირის შემდეგ დადგება მიმდინარე თარიღთან მიმართებაში:

```
week = 1000 * 60 * 60 * 24 * 7      /* მილიწამების რაოდენობა
                                       კვირაში */
mydate = new Date ( )               /* მიმდინარე თარიღი */
mydate_ms = mydate . getTime ( )
mydate_ms += week
mydate . setTime (mydate_ms)
newdate = mydate . toLocaleString ( )   /* ახალი თარიღი
                                           სტრიქონის სახით */
```

2. განვსაზღვროთ დღეების რაოდენობა ორ თარიღს შორის. მაგალითად, 2007 წლის 10 თებერვალსა და 5 მარტს შორის. ამისათვის, ჯერ უნდა შევქმნათ ორი თარიღის ობიექტი:

```
date1 = new Date (2007, 01, 10)
date2 = new Date (2007, 02, 05)
```

სხვაობა უნდა გაიყოს ერთ დღეში მილიწამების რაოდენობაზე:

```
days = (Date . parse (date2) – Date . parse (date1) ) /1000/60/60/24
```

3. ზოგჯერ პროგრამაში საჭიროა დროის შეყოვნება წინასწარ განსაზღვრული ინტერვალით:

```
function pause (mSec)  {
clock = new Date ( );
justMinute = clock . getTime ( );
```

```
while (true)  {
just = new Date ( );
if ( just . getTime ( ) - justMinute . mSec ) break;
}
}
```

16. ობიექტი Boolean (ლოგიკური)

ობიექტი **Boolean** იქმნება შემდეგი გამოსახულების დახმარებით:

ცვლადი = new Boolean (ლოგიკური მნიშვნელობა)

მას აქვს თვისება **prototype**, მეთოდი **toString ( )** და მნიშვნელობა **Of ( )**, რომლებიც აგრეთვე აქვს ობიექტებს **String** და **Number**.

17. ობიექტი Function (ფუნქცია)

ობიექტ Function-ის შექმნა. ზემოთ ჩვენ განვიხილეთ ფუნქციის განსაზღვრის წესი:

```
Function ფუნქციის სახელი (პარამეტრი)  {
                კოდი
}
```

არსებობს ფუნქციის შექმნის სხვა მეთოდიც, რომლითაც ფუნქცია იქმნება როგორც ობიექტ **Function**-ის ეგზემპლარი:

```
ფუნქციის სახელი = new  Function  (["პარ1"[,"პარN"],
                                "ოპერატორი1  [; ოპერატორიN] ")
```

ყველა პარამეტრის დასახელება წარმოადგენს სტრიქონულ სიდიდეს. ისინი ერთმანეთისაგან მძიმით გამოიყოფა. დამამთავრებელი სტრიქონი კი შეიცავს ფუნქციის ტანის კოდის ოპერატორებს, რომლებიც ერთმანეთისაგან წერტილ-მძიმითაა გამოყოფილი. ფუნქციის გამოძახება ჩვეულებრივი წესით ხდება:

ფუნქციის სახელი (პარამეტრები)

მაგალითები:

განვიხილოთ მართკუთხედის ფართობის გამოსათვლელი ფუნქციის სხვადასხვა სახით შექმნა:

```
Srectangle = new Function ("width", "height",  "var s = width * height; return s")
Srectangle (2, 3)                    /* შედეგი 6-ის ტოლია */

var expr = "var s = width * height; return s"
Srectangle = new Function ("width", "height", expr )
Srectangle (2, 3)                    /* შედეგი 6-ის ტოლია */

a = "width"
b = "height"
expr = "var s = width * height; return s"
Srectangle = new Function (a, b, expr )
Srectangle (2, 3)                    /* შედეგი 6-ის ტოლია */
```

თვისებები Function.

1. **arguments** – განსაზღვრავს ფუნქციისათვის გადაცემული პარამეტრების რაოდენობის მასივს. მასივის ელემენტების ინდექსაცია 0-ით იწყება. ეს თვისება გამოიყენება ფუნქციის ტანში, როდესაც საჭიროა მისი გამოძახების დროს გადაცემული პარამეტრების გაანალიზება. მისი სინტაქსია:

ფუნქციის სახელი . arguments

2. **length** – ფუნქციის განსაზღვრაში მითითებული პარამეტრების რაოდენობა. მისი სინტაქსია:

ფუნქციის სახელი . length

ეს თვისება გამოიყენება ფუნქციის ტანის გარეთაც.

3. **caller** – შეიცავს მითითებას ფუნქციაზე, რომლიდანაც მოცემული ფუნქცია იქნა გამოძახებული. თუ ფუნქცია არ არის გამოძახებული სხვა ფუნქციიდან, მაშინ ამ თვისების მნიშვნელობა **null**-ის ტოლია. მისი სინტაქსია:

ფუნქციის სახელი . caller

მეთოდები Function.

toString ( ) – შედეგად აბრუნებს ფუნქციის განსაზღვრას სტრიქონის სახით. მისი სინტაქსია:

ფუნქციის სახელი . toString ( )

ზოგჯერ ეს მეთოდი გამოიყენება პროგრამის გამართვის პროცესში დიალოგური ფანჯრის დახმარებით.

apply ([მიმდინარე ობიექტი [, პარამეტრების მასივი]])
call ([მიმდინარე ობიექტი [, პარ1 [, პარ2 [, ... , პარN]]])

ორივე მეთოდი გამოიყენება ფუნქციის გამოსაძახებლად და იძლევა ერთი და იგივე შედეგს. განსხვავდება მხოლოდ პარამეტრთა წარმოდგენის ფორმის მიხედვით. ამ მეთოდების თავისებურებას ის წარმოადგენს, რომ მათი საშუალებით ფუნქციის გამოძახება ხდება მასზე მითითების გზით. ობიექტზე მითითება გამოიყენება, მაშინ როდესაც გამოსაძახებელი ფუნქცია განსაზღვრულია როგორც მომხმარებლის ობიექტის მეთოდი.

18. ობიექტი Object

Object-ი არის ძირითადი ობიექტი, რომელსაც **JavaScript**-ის ყველა დანარჩენი ობიექტები ეფუძნება. პროგრამებში შეიძლება შეიქმნას საკუთარი ობიექტები. ეს შეიძლება სხვადასხვა წესით მოხდეს:

წესი 1

Function კონსტრუქტორის სახელი ([პარ1, ... [, პარN]) {
კოდი
}
ობიექტის სახელი = new კონსტრუქტორის სახელი (["პარ1", ... [,"პარN"])

წესი 2

ობიექტის სახელი = new Object ( )
ობიექტის სახელი . თვისება = მნიშვნელობა

წესი 3

ობიექტის სახელი = { თვისება1 : მნიშვ.1 {, თვისება2 : მნიშვ.2 {, ... N] }

ობიექტთა თვისებებსა და მეთოდებზე მიმართვა შემდეგი სინტაქსით ხორციელდება:

ობიექტზე მითითება . თვისება

ობიექტზე მითითება . მეთოდი ([პარამეტრები])

მაგალითად, დავუშვათ გვინდა შევქმნათ ობიექტი **Agent**, რომელიც შეიცავს ცნობებს თანამშრომელთა შესახებ: სახელი, გვარი, განყოფილება, ტელეფონი, ხელფასი და სხვა. ეს სტრუქტურა შეიძლება ობიექტის კონსტრუქტორის საშუალებით შეიქმნას:

function Agent (Name1, Name2, Dep, Tel, Many) {
this . Name1 = სახელი
this . Name2 = გვარი
this . Dep = განყოფილება
this . Tel = ტელეფონი
this . Many = ხელფასი
}

გამხსნელი სიტყვა **this** არის მიმართვა მიმდინარე ანუ კონსტრუქტორით განსაზღვრულ ობიექტზე. ყველა ეს ოპერატორი განსაზღვრავს ობიექტის თვისებას.

ახალი კონკრეტული ობიექტის შექმნა შემდეგნაირად შეიძლება: მაგალითად, გვსურს ახალი თანამშრომლის – **agent007**-ის დამატება:

agent007 = new Agent ("Jeims", "Bond", 5, "22-33-33", 5600)

ასეთი ობიექტების თვისებებსა და მეთოდებს იშვიათად მიმართავენ. მეთოდი **toString ( )** ჩვეულებრივ გამოიყენება პროგრამების გამართვის პროცესში. თვისება **hasOwnPropery (“თვისება”)** გამოიყენება რათა განისაზღვროს აქვს თუ არა მოცემული თვისება ამ ობიექტის ეგზემპლარს, რომელიც განსაზღვრულია მის კონსტრუქტორში (და არა **prototype**-ის საშუალებით). თუ აქვს, მაშინ შედეგად დააბრუნებს **true**-ს, წინააღმდეგ შემთხვევაში – **false**.

19. მომხმარებლის ობიექტები

JavaScript-ში შესაძლებელია მომხმარებელმა შექმნას საკუთარი ობიექტი. პრაქტიკული ამოცანის გადაწყვეტის პროცესში იგი არ წარმოადგენს აუცილებლობას, მაგრამ პროგრამისტის თვალსაზრისით, ობიექტი მონაცემთა ორგანიზაციისა და მისი დამამუშავებელი ფუნქციის მოხერხებული საშუალება არის. თუმცა, ამისათვის ობიექტის შექმნა ყოველთვის არ არის აუცილებელი.

ობიექტის შექმნა. **JavaScript**-ში ობიექტი შეიძლება რამდენიმე გზით შეიქმნას. ჩვენ განვიხილოთ სამი მათგანი.

პირველი ხერხი ეფუძნება ფუნქციას, რომლის ტანში ხდება შესაქმნელი ობიექტის ყველა თვისებისა და მეთოდის აღწერა. ვინაიდან, ეს ფუნქცია განმსაზღვრელ როლს ასრულებს ობიექტის შექმნაში, მას ობიექტის ფუნქცია-კონსტრუქტორს ან უბრალოდ კონსტრუქტორს უწოდებენ. კონსტრუქტორმა უნდა შემოიტანოს შესაქმნელი ობიექტის სახელი, მათი თვისებები და მეთოდები. აგრეთვე, მასში უნდა მოხდეს თვისებებისათვის საწყისი მნიშვნელობების მინიჭება.

ობიექტის ფუნქცია-კონსტრუქტორის სახელი ერთდროულად არის შესაქმნელი ობიექტის სახელიც. შესაქმნელი ობიექტის თვისებები და მეთოდები მოცემული უნდა იყოს ფუნქცია-კონსტრუქტორის ტანში მინიჭების ოპერატორის საშუალებით. ამასთან, ცვლადი-თვისების სახელი ჩაიწერება გამხსნელი სიტყვით **this** (ეს): **this** . ცვლადი.

მაგალითის სახით განვიხილოთ ფუნქცია-კონსტრუქტორი, რომლის საშუალებითაც იქმნება ობიექტი **car** (ავტომობილი) თვისებებით **name** (დასახელება), **model** (მოდელი) და **color** (ფერი):

```
function car ( name, model, color ) {
   this . name = name
   this . model = model
   this . color = color
}
```

ეს ფუნქცია განსაზღვრავს ობიექტ **car**-ს. **car** ობიექტის კონკრეტული ეგზემპლარის შესაქმნელად, საჭიროა შესრულდეს ამ ფუნქციის გამოძახება და გადაეცეს შესაბამისი პარამეტრების მნიშვნელობა.

როგორც ჩვენ უკვე ვიცით ობიექტის ეგზემპლარი იქმნება მინიჭების ოპერატორით და **new** გამხსნელი სიტყვით:

mycar = new car ("Mercedes-Benz", "C220", "white")

ჩვენ შევქმენით **mycar** ობიექტი, რომელიც არის **car** ობიექტის ეგზემპლარი. ასეთი ეგზემპლარები შეიძლება შეიქმნას რამდენიმე. **mycar** ობიექტის თვისების მნიშვნელობა შეიძლება შეიცვალოს პროგრამაში:

mycar . model /* მნიშვნელობა ტოლია **"C220"**-ის */
mycar . name /* მნიშვნელობა ტოლია **"Mercedes-Benz"**-ის */
mycar . model = "CLK220" /* მნიშვნელობა ტოლია **"CLK220"**-ის */

ობიექტი შეიძლება შეიქმნას აგრეთვე **new Object ( )** კონსტრუქტორის საშუალებით:

```
mycar = new Object ( )
mycar . name = "Mercedes-Benz"
mycar . model = "C220"
mycar . color = "whitev
```

დასაშვებია აგრეთვე ობიექტის განსაზღვრის შემდეგი კომპაქტური ჩანაწერიც:

mycar = { name : "Mercedes-Benz", model : "C220", color : "white" }

აქ თვისებათა ყველა განსაზღვრება მოთავსებულია ფიგურულ ფრჩხილებში. თვისების სახელი თავისი მნიშვნელობებისაგან გამოყოფილია ორწერტილით, ხოლო თვისებები – მძიმით.

ობიექტის შექმნის დროს ჩვენ შეგვიძლია მივანიჭოთ თვისებებს მნიშვნელობები გულისხმობის (გაჩუმების) პრინციპით, ანუ მნიშვნელობები, რომელიც ექნება თვისებებს, თუ ობიექტის ეგზემპლარის შექმნის დროს თვისების მნიშვნელობა ცნობილი არ არის (ანუ აქვთ მნიშვნელობა **null**, **0** ან " "). ეს სრულდება ლოგიკური ‘‘ან” (აღინიშნება ||-ით) ოპერატორის დახმარებით:

```
function car ( name, model, color ) {
   this . name = name || undefined
   this . model = model || undefined
   this . color = color || "black"
```

```
}
mycar = new  car ("Mercedes-Benz", " " )
mycar . name          /*  "Mercedes-Benz"  */
mycar . model                 /*  undefined  */
mycar . color         /*  "black"  */
```

თვისების დამატება. თუ წარმოიშვება აუცილებლობა, რომ არსებულ ობიექტს დაემატოს ახალი თვისება, მაშინ ეს უნდა მოხდეს **prototype** თვისების საშუალებით. მაგალითად, **mycar** ეგზემპლარს დავუმატოთ ახალი თვისება:

```
car . prototype . owner = "Kapanadze"
```

თუ საჭიროა ახალი თვისება დაემატოს მხოლოდ ერთ კონკრეტულ ობიექტს, მაშინ ეს შეიძლება განხორციელდეს უბრალოდ, მინიჭების ოპერატორის საშუალებით:

ობიექტის სახელი . ახალი თვისება = მნიშვნელობა

შემდეგ მაგალითში ჩვენ ვქმნით ობიექტს საერთოდ თვისების გარეშე, ხოლო შემდეგ მას ვუმატებთ ეგზემპლარებს და სხვა თვისებებს.

```
function car ( ) {
   mycar1 = new car ( )
   mycar2 = new car ( )
   mycar1 . name = "Mercedes-Benz"
   mycar2 . model = "C220"
mycar1 . name                  /* "Mercedes-Benz" */
mycar1 . model                 /* undefined */
mycar2 . name                  /* undefined */
mycar2 . model                 /* "C220"  */
```

ურთიერთდაკავშირებული ობიექტები. ობიექტში თვისების სახით შეიძლება იყოს მითითება სხვა ობიექტზე. ამ შემთხვევაში ორივე ობიექტი არის დაკავშირებული ერთმანეთთან: ერთ-ერთი მათგანი არის ქვეობიექტი ანუ მეორის თვისება. მაგალითად, შევქმნათ ობიექტი **photo**, რომელიც თავის თვისებად შეიცავს ავტომობილის დასახელებას და მისი სურათის ფაილის **URL**-მისამართს.

```
function car ( name, model, color )  {
   this . name = name
```

```
    this . model = model
    this . color = color
    this . photo = photo
}
function photo ( name, url ) {
    this . name = name
    this . url = url
}
```

შევქმნათ რამდენიმე კონკრეტული ობიექტი:

```
photo1 = new photo ("Mercedes-Benz", "/images/MB1.jpg" )
photo2 = new photo ("Mercedes-Benz", "/images/MB2.gif" )
```

ახლა შევქმნათ **car** ობიექტის ეგზემპლარები:

```
mycar = new  car ("Mercedes-Benz", "C220", "white", photo1 )
mycar . photo . url          /* "/images/MB1.jpg" */
mycar . model                /* "C220"  */

mycar = new  car ("Mercedes-Benz", "C220", "white", photo2 )
mycar . photo . url          /* "/images/MB2.gif" */
mycar . model                /* "C220"  */
```

მონაცემთა ბაზის შექმნა ობიექტების საშუალებით. მონაცემთა ბაზის შექმნა განვიხილოთ ავტოსადგომზე მდგომი ავტომობილებისათვის. ავტომობილების სია შეიძლება წარმოვადგინოთ ცხრილის სახით, სადაც სვეტებად არის ავტომობილის დასახელება (**name**), მოდელი (**model**), სახელმწიფო ნომერი (**regnom**), მფლობელი (**owner**) და ავტომობილის ფოტო (**photo**). ჩავთვალოთ, რომ ავტომობილის ფოტოზე არ ჩანს მისი ნომერი, ამიტომ იგი განსაზღვრავს ავტომობილის ტიპს (დასახელება, მოდელი და ფერი). მონაცემთა ბაზის შესაქმნელად შევქმნათ ორი ცხრილი. ერთი იქნება დამხმარე და შეასრულებს ავტომობილის ტიპის ცნობარის როლს, ხოლო მეორე იქნება ჩანაწერი კონკრეტული ავტომობილის შესახებ.

ცნობარი შეიცავს ზოგად ინფორმაციას ავტომობილების შესახებ: დასახელება, მოდელი და ფოტოსურათი (უფრო ზუსტად ამ ფაილის მისამართი). ჩვენს მონაცემთა ბაზაში ავტომობილების ცნობარი არის ობიექტი **ref**:

```
function ref ( name, model, url )  {
   this . name = name
   this . model = model
   this . url = url
}
```

შევავსოთ ცნობარი კონკრეტული ჩანაწერებით. ასეთი ჩანაწერების სიმრავლე განვსაზღვროთ როგორც მასივი (**aref**):

```
aref = new Array ( )
aref [0] = new ref ("Mercedes-Benz", "C220", "pict0 . gif" )
aref [1] = new ref ("Mercedes-Benz", "ML430", "pict1 . gif" )
aref [2] = new ref ("Opel", "Vectra", "pict2 . gif" )
aref [3] = new ref ("Toyota", "Prado", "pict3 . gif" )
aref [4] = new ref ("Volkswagen", "Golf", "pict4 . gif" )
```

ახლა შევქმნათ ავტოსადგომზე მდგომი ავტომობილების სია. ფორმალურად, ამ მასივის ელემენტები განისაზღვრება როგორც **car** ობიექტის ეგზემპლარები, რომელიც შეიცავს მითითებას ცნობარზე – **ref** ობიექტზე. პირველ რიგში დავწეროთ **car** ობიექტის ფუნქცია-კონსტრუქტორის კოდი:

```
function car ( regnum, owner, ref )  {
   this . regnum = regnum
   this . owner = owner
   this . ref = ref
}
```

ახლა შევქმნათ ავტოსადგომზე მდგომი ავტომობილების ჩანაწერები. მასივს დავარქვათ **acar**:

```
acar = new Array ( )
acar [0] = new ref ("ABD245", "კაპანაძე", aref [3] )
acar [1] = new ref ("FAF367", "მაისურაძე", aref [4] )
acar [2] = new ref ("JOJ711", "იაშვილი", aref [2] )
acar [3] = new ref ("SSS777", "ბერიძე", aref [1] )
acar [4] = new ref ("NAN178", "მეგრელაძე", aref [3] )
acar [5] = new ref ("KKK477", "გოგრიჭიანი", aref [4] )
acar [6] = new ref ("AAC882", "უდესიანი", aref [4] )
acar [7] = new ref ("JJJ999", "დოლიძე", aref [0] )
```

```
acar [8] = new ref ("ACM326", "გამსახურდია", aref [2] )
```

ჩვენს სიაში სულ ცხრა ავტომობილია.

მონაცემთა ბაზა შეიძლება აისახოს მონიტორის ეკრანზე ბრაუზერის ფანჯარაში. ამისათვის, პირველ რიგში აუცილებელია შეიქმნას **HTML**-პროგრამა ცხრილით. ცხრილი იქმნება <**TABLE**>, <**TR**>, <**TH**>, <**TD**> და სხვა ტეგებით.

ქვემოთ მოყვანილია **HTML**-კოდი, რომელიც შეიცავს მხოლოდ სცენარს. გამარტივების მიზნით მასში ცხრილის ფორმირება მხოლოდ სამი სვეტით ხდება:

```
<HTML>
<HEAD><TITLE> ავტომობილების მონაცემთა ბაზა </TITLE>
        </HEAD>
<SCRIPT>

/* კონსტრუქტორები */

function ref ( name, model, url )  {
   this . name = name
   this . model = model
   this . url = url
}
function car ( regnum, owner, ref )  {
   this . regnum = regnum
   this . owner = owner
   this . ref = ref
}

/* მონაცემთა ბაზა */

aref = new Array ( )
aref [0] = new ref ("Mercedes-Benz", "C220", "pict0 . gif" )
aref [1] = new ref ("Mercedes-Benz", "ML430", "pict1 . gif" )
aref [2] = new ref ("Opel", "Vectra", "pict2 . gif" )
aref [3] = new ref ("Toyota", "Prado", "pict3 . gif" )
aref [4] = new ref ("Volkswagen", "Golf", "pict4 . gif" )

acar = new Array ( )
acar [0] = new ref ("ABD245", "კაპანაძე", aref [3] )
acar [1] = new ref ("FAF367", "მაისურაძე", aref [4] )
```

```
acar [2] = new ref ("JOJ711", "იაშვილი", aref [2] )
acar [3] = new ref ("SSS777", "ბერიძე", aref [1] )
acar [4] = new ref ("NAN178", "მეგრელაძე", aref [3] )
acar [5] = new ref ("KKK477", "გოგრიჭიანი", aref [4] )
acar [6] = new ref ("AAC882", "უდესიანი", aref [4] )
acar [7] = new ref ("JJJ999", "დოლიძე", aref [0] )
acar [8] = new ref ("ACM326", "გამსახურდია", aref [2] )
strTab = "<TABLE BORDER = 1> <TR>"
strTab += "<TH> დასახელება </TH><TH> ნომერი </TH><TH>
          მფლობელი </TH></TR>"

/* ცხრილის სტრიქონების ფორმირება */

for ( i = 0; i <= acar . length – 1; i ++ )  {
strTab += "<TR><TD>" + acar [i] . ref . name
strTab += "</TD><TD>" + acar [i] . regnum
strTab += "</TD><TD>" + acar [i] . owner + "</TD></TR>"
}

strTab += "</TABLE>"
document . write (strTab)

</SCRIPT>
</HTML>
```

ჩვენ შეგვეძლო ცხრილში აგვესახა სხვა სვეტებიც. მაგალითად, ავტომობილის სურათისათვის საკმარისი იყო **strTab** სტრიქონში ჩაგვემატებინა შემდეგი ფრაგმენტი:

```
"<TD><IMG SRC = ‘" + acar [i] . ref . url + " ‘ ></TD>"
```

20. სპეციალური ოპერატორები

აქ აღწერილი ის ოპერატორები იქნება, რომლებიც იშვიათად გამოიყენება **JavaScript**-ში, მაგრამ როდისმე შეიძლება დაგვჭირდეს ისინი.

ბიტური ოპერატორები. ბიტური (თანრიგების მიხედვით) ოპერატორები გამოიყენება მთელი ტიპის მნიშვნელობებთან და შედეგიც არის

მთელი. მათი გამოყენების დროს ოპერანდები ჯერ გარდაიქმნება ორობით სისტემაში, რომელშიც რიცხვები ჩაიწერება ნულებით და ერთიანებით (32 თანრიგში). თითოეულ თანრიგს ბიტი ეწოდება. შემდეგ მოქმედებები ტარდება ბიტებზე და შედეგად მიიღება ბიტების ახალი მიმდევრობა. და ბოლოს შედეგი ისევ გარდაიქმნება მთელ რიცხვად. ეს ოპერატორებია:

ოპერატორი	დასახელება	მარცხენა ოპერანდი	მარჯვენა ოპერანდი
&	ბიტური "და"	მთელი რიცხვი	მთელი რიცხვი
|	ბიტური "ან"	მთელი რიცხვი	მთელი რიცხვი
^	ბიტური "ან"-ის უარყოფა	მთელი რიცხვი	მთელი რიცხვი
~	ბიტური "არა"	-	მთელი რიცხვი
<<	წანაცვლება მარცხნივ	მთელი რიცხვი	ბიტების რაოდენობა, რომელზეც მოხდება წანაცვლება
>>	წანაცვლება მარჯვნივ	მთელი რიცხვი	ბიტების რაოდენობა, რომელზეც მოხდება წანაცვლება
>>>	ნულებით შევსება მარჯვნივ წანაცვ-ლების დროს	მთელი რიცხვი	ბიტების რაოდენობა, რომელზეც მოხდება წანაცვლება

ოპერატორები **&**, |, ^ და ~ მოგვაგონებს ლოგიკურ ოპერატორებს. ოპერატორი, ბიტის მნიშვნელობას ცვლის საწინააღმდეგოთი: 0-ს 1-იანით, ხოლო 1-ს 0-იანით. დანარჩენი ოპერატორები ასე მუშაობს:

X	**Y**	**X&Y**	**X|Y**	**X^Y**
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

ობიექტის ოპერატორები.

ობიექტის თვისების წაშლის ოპერატორი (**delete**). ობიექტის თვისების, აგრეთვე მასივის ელემენტის წაშლა შეიძლება **delete**

ოპერატორის საშუალებით. ჩვეულებრივ ეს ოპერატორი გამოიყენება მასივის ელემენტის წასაშლელად:

delete ელემენტი

მასივის ელემენტის წაშლის დროს იშლება მისი ინდექსიც, მაგრამ დარჩენილი ელემენტები თავიანთ ძველ ინდექსებს ინარჩუნებს, ამასთან მასივის სიგრძეც არ იცვლება.

მაგალითი:

```
myarray = new Array ("a", "b", "c", "d" )
myarray . length                    /*  4  */
delete myarray [1]
myarray . length                    /*  4  */
myarray [0]                         /*  "a"  */
myarray [1]                         /*  undefined  */
myarray [2]                         /*  "c"  */
myarray [3]                         /*  "d"  */
```

თვისების არსებობის შესამოწმებელი ოპერატორი (**in**). ეს ოპერატორი საშუალებას იძლევა შევამოწმოთ ამა თუ იმ ობიექტს აქვს თუ არა რომელიმე თვისება ან მეთოდი. მისი მარცხენა ოპერანდი არის მითითება, რომელშიც იწერება ჩვენთვის საინტერესო თვისება ან მეთოდი სტრიქონის სახით, ხოლო მარჯვენა ოპერანდი – ობიექტია. მითითება მეთოდზე შეიცავს მხოლოდ მის დასახელებას ფრჩხილების გარეშე. თუ აღნიშნული თვისება ან მეთოდი არის მოცემულ ობიექტში, მაშინ შედეგი არის **true**, წინააღმდეგ შემთხვევაში – **false**. აქედან გამომდინარე შეიძლება **in** ოპერატორი გამოვიყენოთ პირობით გამოსახულებებში.

მაგალითად, თუ ობიექტი **document**-ს აქვს მეთოდი **write ( )**, მაშინ ამის დასამოწმებლად გამოსახულება ასე შეიძლება ჩავწეროთ:

“write” in document

შედეგი კი იქნება **true**.

ობიექტთა მოდელზე მიკუთვნების შესამოწმებელი ოპერატორი (**instanceof**). ეს ოპერატორი ამოწმებს ეკუთვნის თუ არა რომელიმე ობიექტი **JavaScript**-ის ობიექტთა მოდელს. მისი მარცხენა ოპერანდი

არის შესამოწმებელი მნიშვნელობა, ხოლო მარჯვენა არის მითითება ძირითად ობიექტზე, როგორიცაა **Array**, **String**, **Date** და სხვა. **instanceof** ოპერატორის შედეგი არის **true** ან **false**, ამიტომ მისი გამოყენება შეიძლება პირობით გამოსახულებებში.

მაგალითად, შესაქმნელი მასივი არის **Array** ობიექტის ეგზემპლარი, ხოლო ეს უკანასკნელი კი **Object** – ძირითადი ობიექტის ეგზემპლარი.

```
myarray = new Array ( )
myarray  instanceof  Array                /* true */
Array  instanceof  Object                 /* true */
myarray  instanceof  String               /* false */
```

კომპლექსური ოპერატორები.

პირობითი ოპერატორი (**?:**). ეს ოპერატორი პირობითი გადასვლის ოპერატორის **if ... else ...** შემოკლებულ ფორმას წარმოადგენს. ჩვეულებრივ იგი გამოიყენება მინიჭების ოპერატორთან ერთად. პირობითი გამოსახულების მნიშვნელობის მიხედვით ენიჭება ან ერთი ან მეორე მნიშვნელობა. პიორობითი ოპერატორის მნიშვნელობის სინტაქსი შემდეგია:

პირობა ? გამოსახულება1 : გამოსახულება2

მინიჭების ოპერატორთან ერთად პირობით ოპერატორს აქვს შემდეგი სახე:

ცვლადი = პირობა ? გამოსახულება1 : გამოსახულება2

პირობითი ოპერატორის შედეგი არის **გამოსახულება1**-ის მნიშვნელობა, თუ პირობა ჭეშმარიტია, ხოლო **გამოსახულება2**-ის მნიშვნელობა, თუ პირობა მცდარია.

მაგალითი:

```
d = new Date ( )
x = d . getDate ( )
typedate = ( x%2 == 0 ) && ( x > 1 ) ? "ლუწი" : "კენტი"
```

ტიპის განმსაზღვრელი ოპერატორი (**typeof**). ეს ოპერატორი გამოიყენება იმის შესამოწმებლად, ეკუთვნის თუ არა მნიშვნელობა შემდეგი ტიპიდან ერთ-ერთს: **String**, **number**, **boolean**, **object**, **function**

ან **undefined**. **typeof** ოპერატორის შესრულების შედეგი არის სტრიქონული ტიპის და იგი არის ზემოთ ჩამოთვლილი ტიპებიდან ერთ-ერთი. ერთადერთი ოპერანდი იწერება **typeof** გამხსნელი სიტყვის მარჯვნივ.

მაგალითი:

```
var x
n = 3.14
N = new Number (5.23)
s = “Hello!”
a = new Array ( 1, 2, 3, 4, 5 )
function f ( ) { }

typeof x            /* "undefined" */
typeof n            /* "number" */
typeof N            /* "object" */
typeof s            /* "string" */
typeof a            /* "object" */
typeof f            /* "function" */
```

21. ოპერატორთა პრიორიტეტები

ზემოთ ჩვენ აღვნიშნეთ, რომ გამოსახულებებში ოპერატორები სრულდება პრიორიტეტების მიხედვით. ერთი და იგივე პრიორიტეტის მქონე ოპერატორები მიმდევრობით სრულდება მარცხნიდან მარჯვნივ. მრგვალი ფრჩხილები, რომელთა საშუალებითაც შესაძლებელია გამოსახულებაში ოპერატორთა შესრულების ნებისმიერი რიგითობის უზრუნველყოფა, შეიძლება ჩაითვალოს უმაღლესი პრიორიტეტის მქონე ოპერატორად. ფრჩხილებს შეუძლიათ შექმნან ერთმანეთში ჩალაგებული სტრუქტურა. პირველ რიგში გამოსახულება სრულდება, რომელიც მოთავსებულია შიგა ფრჩხილებში. **JavaScript**-ის ინტერპრეტატორი გამოსახულების ანალიზს სწორედ ფრჩხილებში ჩალაგებული სტრუქტურის გამოკვლევით იწყებს. პრიორიტეტის მიხედვით მეორე ადგილზეა მასივის ინდექსების გამოთვლა და თვით მასივის ელემენტების განსაზღვრა.

ბოლო ადგილზეა მძიმე, როგორც პარამეტრების გამყოფი.

ოპერატორების პრიორიტეტების განაწილება

პრიორიტეტი	ოპერატორი	კომენტარი
1	**( )**	შიდა ფრჩხილებიდან გარე ფრჩხილებისაკენ
	[ ]	მასივი **HTML**-დოკუმენტის ინდექსის მნიშვნელობა
	function ( )	ფუნქციის გამოძახება
2	**!**	ლოგიკური ‘‘არა’’
	~	ბიტური ‘‘არა’’
	-	უარყოფა
	++	ინკრემენტი (1-ით გაზრდა)
	--	დეკრემენტი
	new	
	typeof	
	void	
	delete	ობიექტის ელემენტის წაშლა
3	*****	გამრავლება
	/	გაყოფა
	%	მოდულით გაყოფა (განაყოფის ნაშთი)
4	+	შეკრება (კონკანეტაცია)
	-	გამოკლება
5	<<	ბიტური წანაცვლებები
	>>	
	>>>	
6	<	ნაკლებია
	<=	ნაკლებია ან ტოლი
	>	მეტია
	>=	მეტია ან ტოლი
7	==	ტოლობა
	!=	უტოლობა
8	**&**	ბიტური ‘‘და’’
9	^	ბიტური ‘‘ან’’-ის უარყოფა
10	|	ბიტური ‘‘ან’’
11	**&&**	ლოგიკური ‘‘და’’
12	||	ლოგიკური ‘‘ან’’
13	**?**	პირობითი გამოსახულება (პირობითი ოპერატორი)

პრიორიტეტი	ოპერატორი	კომენტარი
14	=	მინიჭების ოპერატორი
	+=	
	-=	
	*=	
	/=	
	%=	
	<<=	
	>>=	
	&=	
	^=	
	|=	
15	,	მძიმე (პარამეტრების გამყოფი)

პრიორიტეტების გარდა აგრეთვე უნდა გავითვალისწინოთ, რომ რთული ლოგიკური გამოსახულებები, რომლებიც შედგება რამდენიმე, ერთმანეთთან ‘‘და’’ და ‘‘ან’’ ოპერატორებით დაკავშირებული მარტივი ოპერატორებისაგან, სრულდება ე. წ. მოკლე დამუშავების პრინციპით. ეს ნიშნავს, რომ მთელი გამოსახულების მნიშვნელობა შეიძლება განისაზღვროს ერთი ან რამდენიმე მარტივი გამოსახულების საშუალებით. მაგალითად, გამოსახულება **x&&y** გამოითვლება მარცხნიდან მარჯვნივ; თუ **x**-ის მნიშვნელობა ტოლია **false**-ის, მაშინ **y**-ის მნიშვნელობა შეიძლება არ გამოითვალოს, რადგანაც მთელი გამოსახულების მნიშვნელობა ნებისმიერ შემთხვევაში იქნება **false**-ის ტოლი.

22.სცენარების შექმნა

HTML-დოკუმენტების ელემენტებისა და თვით ბრაუზერის მართვისათვის, ახალი დოკუმენტის გენერაციისათვის, მომხმარებელთან დიალოგური ურთიერთობის ორგანიზაციისათვის, **HTML**-ში მონაცემების გამოთვლებისა და დამუშავების შესასრულებლად გათვალისწინებული იყო დაპროგრამების სპეციალურ ენებთან ინტეგრაცია. ამ ენებზე დაწერილ პროგრამებს, რომლებიც მუშაობს **HTML**-დოკუმენტების ობიექტებთან, სცენარები (**scripts)** ეწოდებათ. სცენარების სტანდარტული ენა არის **JavaScript**-ი. მისი ინტერპრეტირება

შეუძლია ყველა ბრაუზერს. ბრაუზერში ჩაშენებული **JavaScript**-ის ინტერპრეტატორი მომხმარებელს საშუალებას აძლევს გამოიყენოს ენის შესაძლებლობანი ბრაუზერსა და მოცემულ მომენტში მასში არსებულ ყველა რესურსებზე მიმართვისათვის. კერძოდ, ბრაუზერისა და მასში ჩატვირთული დოკუმენტების თვისებებთან. ყოველივე ეს სრულდება სცენარების საშუალებით.

სცენარების განთავსება. სცენარები შეიძლება დაიწეროს უშუალოდ **HTML**-დოკუმენტში, აგრეთვე ცალკეულ ტექსტურ ფაილებში, რომელთა გამოძახება **HTML**-დოკუმენტიდან ხდება. ყველაზე მარტივია სცენარი განთავსდეს უშუალოდ **HTML**-დოკუმენტში. აქვე შევნიშნოთ, რომ სცენარების განთავსება ცალკე ფაილში არც ისე რთულია, მაგრამ ხშირად არ არის გამართლებული ეკონომიკური თვალსაზრისით. აუცილებელია დაგროვდეს გარკვეული რაოდენობის მზა პროგრამები, რათა მოხდეს მათი გამოყენება შემდგომ პროექტებში ბიბლიოთეკის სახით.

HTML-დოკუმენტში სცენარი შეიძლება განთავსდეს რამდენიმე წესით. სცენარების განთავსების სტანდარტულ წესს წარმოადგენს მათი განთავსება კონტეინერულ **<SCRIPT>** ტეგში, ანუ **<SCRIPT>** და **</SCRIPT>** ტეგებს შორის. როდესაც ბრაუზერი შეხვდება **<SCRIPT>** ტეგს, მაშინ იგი ‘‘გაიგებს’’, რომ მის შემდეგ იწყება სცენარის კოდი. დამამთავრებელი **</SCRIPT>** ტეგი აჩვენებს ბრაუზერს, რომ სცენარის კოდი დამთავრდა. რაც მოთავსებულია ამ ტეგებს გარეთ ბრაუზერი აღიქვამს, როგორც **HTML**-კოდს. **<SCRIPT>** კონტეინერი **HTML**-დოკუმენტში შეიძლება შეგვხვდეს ნებისმიერ ადგილზე და არაერთხელ. კონტეინერულ **<SCRIPT>** ტეგი შეიძლება შეიცავდეს შემდეგ ატრიბუტებს:

- **LANGUAGE** – სცენარის ენა; მათი შესაძლო მნიშვნელობებია – "**JavaScript**", "**JScript**";

- **SRC** – მიუთითებს ფაილს (სახელი ან **URL**-მისამართი), რომელიც სცენარის კოდს შეიცავს. ეს ატრიბუტი იმ შემთხვევაში გამოიყენება, თუ სცენარი **HTML**-დოკუმენტში განთავსებული არ არის, არამედ განთავსებულია ცალკე ფაილში.

მაგალითი:

<SCRIPT LANGUAGE = "JavaScript">

```
. . .                                        /* სცენარის კოდი */
</SCRIPT>
<SCRIPT LANGUAGE = "JScript"  SRC = "myscripts . js">
      </SCRIPT>
```

თუ სცენარი განთავსებულია ცალკე ფაილში, მაშინ მასში **<SCRIPT>** და **</SCRIPT>** ტეგები არ იწერება. ჩვეულებრივ ცალკე ფაილებში განთავსდება ფუნქციათა ბიბლიოთეკა (ფუნქციის განსაზღვრა), აგრეთვე ერთი ან რამდენიმე საიტის რამდენიმე **HTML**-დოკუმენტში გამოყენებული სცენარი. გარე ფაილიდან ჩატვირთული სცენარი შეიძლება წარმოვადგინოთ როგორც უბრალო ჩანართი **HTML**-დოკუმენტში.

მაგალითი:

```
<HTML>
. . .
<SCRIPT>
function myfunc ( )  {
. . .
}
</SCRIPT>
<SCRIPT  SRC = "mylibrary . js" ></SCRIPT>
<SCRIPT SRC = "myprogram . js" ></SCRIPT>
. . .
</HTML>
```

აქ **HTML**-დოკუმენტში განთავსებულია სცენარის სამი განყოფილება (სექცია), რომელთაგან ორი ჩაიტვირთება ცალკე ფაილიდან. სცენარის პირველ განყოფილებაში მოცემულია, განსაზღვრულია რაღაც ფუნქცია.

სცენარი შეიძლება კიდევ ჩაიწეროს როგორც ერთმანეთისაგან წერტილ-მძიმით გამოყოფილი ბრჭყალებში ჩასმული ოპერატორების სტრიქონი. ასეთი ფორმით სცენარი ჩვეულებრივ გამოიყენება ხდომილობის დამუშავების სახით.

ფუნქციის გამოძახება და მისი განსაზღვრა შეიძლება განთავსდეს ნებისმიერი მიმდევრობით, თუ კი ისინი განთავსებულია ერთი და იგივე **<SCRIPT>** კონტეინერში. თუ ისინი განთავსებულია სხვადასხვა

კონტეინერში, მაშინ აუცილებელია რომ ფუნქციის განსაზღვრა წინ უსწრებდეს ამ ფუნქციის გამოძახების ბრძანებას. ანალოგიურად, თუ ფუნქციის განსაზღვრა განთავსებულია ცალკე ფაილში, მაშინ იგი უნდა ჩაიტვირთოს **HTML**-დოკუმენტში ამ ფუნქციის გამოძახების ბრძანებაზე ადრე. ქვემოთ მოყვანილია ორი მაგალითი **HTML**-დოკუმენტში სცენარების სწორი და არასწორი განთავსების შესახებ:

1.

სწორი	არასწორი
<HTML> **. . .** **<SCRIPT>** **function myfunc ( ) {** **. . .** **}** **</SCRIPT>** **. . .** **<SCRIPT>** **. . .** **myfunc ( )** **. . .** **</SCRIPT>** **. . .** **</HTML>**	**<HTML>** **. . .** **<SCRIPT>** **. . .** **myfunc ( )** **. . .** **</SCRIPT>** **. . .** **<SCRIPT>** **function myfunc ( ) {** **. . .** **}** **</SCRIPT>** **. . .** **</HTML>**

2.

სწორი	არასწორი
<HTML> **. . .** **<SCRIPT>** **function myfunc ( ) {** **. . .** **}** **. . .** **myfunc ( )** **. . .** **</SCRIPT>** **. . .** **</HTML>**	**<HTML>** **. . .** **<SCRIPT>** **. . .** **myfunc ( )** **. . .** **function myfunc ( ) {** **. . .** **}** **</SCRIPT>** **. . .** **</HTML>**

თუ აუცილებელია, რომ სცენარი ბრაუზერში ჩაიტვირთოს მანამ, ვიდრე ჩაიტვირთება **HTML**-დოკუმენტის ელემენტები, მაშინ საჭიროა იგი განთავსდეს **HTML**-კოდის ზედა ნაწილში. ამ შემთხვევაში სცენარს ჩვეულებრივ ათავსებენ **<HEAD>** კონტეინერში (დოკუმენტის სათაურში).

თუ საჭიროა, რომ სცენარი ჩაიტვირთოს **HTML**-დოკუმენტის ყველა ელემენტის ჩატვირთვის შემდეგ, მაშინ შესაძლებელია შემდეგი ორი ვარიანტი. პირველი, სცენარი შეიძლება განთავსდეს **HTML**-დოკუმენტის ბოლოში. მეორე, შეიძლება გამოვიყენოთ ხდომილობის ატრიბუტი **onload** კონტეინერულ **<BODY>** ტეგში, რომელიც იძლევა **HTML**-დოკუმენტის ძირითად ნაწილს. ბოლო შემთხვევის დროს **onload** ატრიბუტის მნიშვნელობა ჩვეულებრივ არის ფუნქციის სახელის შემცველი სტრიქონი. ამ ფუნქციის განსაზღვრა არის **<SCRIPT>** კონტეინერში, რომელიც განთავსებულია **HTML**-დოკუმენტის სათაურის **<HEAD>** კონტეინერში. ყურადღება უნდა მიექცეს იმას, რომ **onload** ხდომილობის ატრიბუტის კონტეინერულ **<BODY>** ტეგში გამოყენების დროს, ამ ხდომილობის დამუშავება სრულდება კონტეინერულ **<BODY>** ტეგში ყველა ელემენტების ჩატვირთვის დამთავრების შემდეგ და არა მათი ჩატვირთვის პროცესში.

მაგალითი:

```
<HTML>
<HEAD>
<SCRIPT>
function myfunc ( )  {
. . .
}
</SCRIPT>
</HEAD>
. . .
<BODY  onload = "myfunc ( ) " >
    ტეგები
</BODY>
. . .
</HTML>
```

ხდომილობის დამუშავება. **HTML**-დოკუმენტში სცენარების ერთ-ერთი ძირითადი დანიშნულებაა ხდომილობის დამუშავება, ისეთის როგორიცაა მაუსის კლავიშით დოკუმენტის ელემენტზე დაწკაპუნება, მაუსის მაჩვენებლის ელემენტზე დაყენება, მაჩვენებლის ელემენტიდან გადანაცვლება, კლავიშზე ხელის დაჭერა და სხვა. **HTML**-ის უმეტესობა ტეგებს აქვთ სპეციალური ხდომილობის განმსაზღვრელი ატრიბუტები, რომლებზეც შესაბამისი ელემენტები რეაგირებს. ყველა შესაძლო ხდომილობათა სია საკმაოდ დიდია და გათვალისწინებულია თითქმის ყველა შესაძლო შემთხვევა. ხდომილობათა დასახელებები საკმაოდ მარტივია, განსაკუთრებით თუ მომხმარებელმა ინგლისური ენა იცის. მაგალითად, მაუსის მარცხენა ღილაკით დაწკაპუნება – **onclick**; მონაცემთა შეტანის ველში ცვლილება – **onchange**; **HTML**-დოკუმენტის ელემენტზე მაუსის მაჩვენებლის განთავსება – **onmouseover**. ხდომილობათა სრულ სიას განვიხილავთ მოგვიანებით. ასეთი ატრიბუტ-ხდომილობების მნიშვნელობას **HTML**-ის ტეგებში არის ხდომილობის დამმუშავებელი ბრჭყალებში მოთავსებული სცენარის კოდის შემცველი სტრიქონი. ამ სცენარს აგრეთვე უწოდებენ ხდომილობის დამმუშავებელს. მაგალითად, შემდეგი **HTML**-კოდი განსაზღვრავს მეორე დონის სათაურს, რომელიც რეაგირებს მაუსის ღილაკის დაწკაპუნებაზე და რაც ასრულებს რომელიღაც **myfunc ( )** ფუნქციას:

<H2 onclick = "myfunc ( ) " > აქ დააწკაპუნეთ </H2>

ერთი და იგივე ელემენტისათვის შეიძლება განისაზღვროს რამდენიმე ხდომილობა, რომელზეც იგი მოახდენს რეაგირებას ანუ ერთი და იმავე ტეგისათვის შეიძლება მივუთითოთ რამდენიმე ატრიბუტი-ხდომილობა. ეს ატრიბუტები შეიძლება ჩაიწეროს ნებისმიერ რეგისტრში. ატრიბუტების მიმდევრობასაც არა აქვს მნიშვნელობა.

მაგალითი:

ქვემოთ მოცემულია გამოსახულებაზე მაუსის დაწკაპუნების დამმუშავებლის გაფორმების ორი ვარიანტი.

1. **<HTML>**
   <SCRIPT>
   function clickimage ( ) {
   alert ("Hello! ")

```
}
  </SCRIPT>
  <IMG SRC = "pict . jpg" onclick = "clickimage ( ) ">
</HTML>
```

2.
```
<HTML>
<IMG SRC = "pict . jpg" onclick = "alert ("Hello! ")">
</HTML>
```

თუ ხდომილობის დამმუშავებელი სცენარი მცირეა და იგი **HTML**-დოკუმენტში ერთხელ გამოიყენება, მაშინ მიზანშეწონილია მისი გაფორმება ატრიბუტ-ხდომილობის მნიშვნელობის სახით. სხვა შემთხვევაში კი სასურველია პირველი ვარიანტის გამოყენება.

სცენარების მმართველი ობიექტები. თუ ხდომილობა ხდება, მაშინ სრულდება მისი შესაბამისი სცენარი-დამმუშავებელი. ამის გარდა, სცენარი შეიძლება გაშვებულ იქნეს ისე, რომ არ იყოს კავშირში არავითარ ხდომილობასთან. ნებისმიერ შემთხვევაში სცენარმა რაღაც ქმედება უნდა შეასრულოს. მისი საქმიანობის საგანს წარმოადგენს ბრაუზერის ფანჯრისა და მასში ჩატვირთული **HTML**-დოკუმენტის ელემენტები. შესაბამისი ტეგების ატრიბუტების საშუალებით მოცემული დოკუმენტის ელემენტების პარამეტრები შეიძლება შეიცვალოს. უფრო მეტიც, ერთი ტეგი შეიძლება შეიცვალოს მეორით და ჩატვირთული **HTML**-დოკუმენტიც კი შეიძლება შეიცვალოს სხვა დოკუმენტით. ეს სცენარების საშუალებით ხდება.

HTML-დოკუმენტი ბრაუზერის ფანჯარაში აისახება. ბრაუზერის ფანჯარას შეესაბამება ობიექტი **window**, ხოლო ფანჯარაში ჩატვირთულ **HTML**-დოკუმენტს ობიექტი **document**. ეს ობიექტები თავის შემადგენლობაში შეიცავს სხვა ობიექტებს. კერძოდ, ობიექტი **document** შედის **window** ობიექტის შედგენილობაში. **HTML**-დოკუმენტის ელემენტებს შეესაბამება ობიექტები, რომლებიც შედის **document** ობიექტის შედგენილობაში. ობიექტთა მთელ სიმრავლეს აქვს იერარქიული სტრუქტურა, რომელსაც ობიექტური მოდელი ეწოდება.

ობიექტები შეიძლება იყოს ერთმანეთში ჩალაგებული. ობიექტი, რომელიც თავისთავში შეიცავს სხვა ობიექტს მშობლიური ეწოდება. ობიექტი, რომელიც შედის სხვა ობიექტის შედგენილობაში ეწოდება

შვილობილი ან ქვეობიექტი მეორე ობიექტის მიმართ. რომ მივუთითოთ კონკრეტული ობიექტი საჭიროა იერარქიული მიმდევრობით ჩამოვთვალოთ ის ობიექტები, რომლებიც მოცემულ ობიექტს მოიცავს:

ობიექტი1 . ობიექტი2 . ... . ობიექტიN

ობიექტის თვისებებსა და მეთოდებზე მიმართვას ექნება შემდეგი სინტაქსი:

ობიექტი1 . ობიექტი2 . ... . ობიექტიN . თვისება
ობიექტი1 . ობიექტი2 . ... . ობიექტიN . მეთოდი ( )

ხშირად რომელიმე ობიექტის ქვეობიექტს მის თვისებას უწოდებენ (ე. წ. რთული თვისება). ამ შემთხვევაში შეიძლება ითქვას, რომ ობიექტთა თვისება სამი ტიპის არსებობს:

- მარტივი თვისება;
- მეთოდი (თვისება-ფუნქცია);
- ობიექტები (რთული თვისება, რომლებსაც აქვთ თავიანთი თვისება).

ვინაიდან, ობიექტი **document** არის **window** ობიექტის ქვეობიექტი, ამიტომ ბრაუზერის მიმდინარე ფანჯარაში ჩატვირთულ **HTML**-დოკუმენტზე მითითებას ექნება სახე: **window.document**.**document** ობიექტს აქვს მეთოდი **write (სტრიქონი)**, რომლის საშუალებითაც მიმდინარე **HTML**-დოკუმენტში ჩაწერს სტრიქონს. ამ მეთოდის გამოყენება ასე ჩაიწერება: **window . document** . **write (სტრიქონი)**.

დოკუმენტის ობიექტურ მოდელში ობიექტები დაჯგუფებულია ეგრეთ წოდებულ კოლექციებში. კოლექცია შეიძლება განხილულ იქნეს როგორც საშუალედო ობიექტი, რომელიც შეიცავს ამ დოკუმენტის ობიექტებს. მეორე მხრივ, კოლექცია არის ობიექტთა მასივი, რომელიც დალაგებულია **HTML**-დოკუმენტში მისი შესაბამისი ელემენტების მოხსენიების რიგით. კოლექციაში ობიექტთა ინდექსაცია ნულიდან იწყება. კოლექციის ელემენტებზე მიმართვა მასივის მსგავსად ხორციელდება. კოლექციას აქვს თვისება **length** – ყველა ელემენტების რაოდენობა.

ასე მაგალითად, დოკუმენტში გრაფიკული გამოსახულებების ყველა ობიექტების კოლექციას უწოდებენ **images**, ყველა ფორმების

კოლექციას – **forms**, ყველა მითითებების კოლექციას – **links**. ეს არის კერძო ანუ თემატური კოლექციების მაგალითი. ამის გარდა, არის აგრეთვე დოკუმენტის ყველა ობიექტის კოლექცია, რომელსაც ეწოდება **all**. ერთი და იგივე ობიექტი შეიძლება შედიოდეს რომელიმე კერძო კოლექციაში, მაგრამ აუცილებლად შედის **all** კოლექციაში. ამასთან, ამ ობიექტის ინდექსი კერძო და **all** კოლექციაში შეიძლება იყოს სხვადასხვა.

დოკუმენტის ობიექტთა თვისებებზე მიმართვის ზოგადი წესი მდგომარეობს იმაში, რომ მითითებაში ნახსენები უნდა იყოს კოლექციის დასახელება. ამ წესიდან გამონაკლისს წარმოადგენს ფორმის ობიექტი. თუ დოკუმენტი ჩატვირთულია მიმდინარე ფანჯარაში, მაშინ **window** ობიექტის მითითება საჭირო არ არის და მითითება უნდა დავიწყოთ გამხსნელი სიტყვით **document**. დოკუმენტის ობიექტებზე მიმართვის რამდენიმე წესი არსებობს:

- **document . კოლექცია . ობიექტის-id;**
- **document . კოლექცია ["ობიექტის-id"];**
- **document . კოლექცია [ობიექტის ინდექსი].**

აქ **ობიექტის-id** – არის ტეგში **ID** ატრიბუტის მნიშვნელობა, რომელიც **HTML**-დოკუმენტში შესაბამის ელემენტს განსაზღვრავს. სიდიდე **ობიექტის ინდექსი** – მთელი რიცხვია, რომელიც მიუთითებს კოლექციაში ობიექტის რიგით ნომერს. თუ დოკუმენტის შექმნის მომენტში არ გამოგვიყენებია **ID** ატრიბუტი, მაშინ მათ ობიექტებზე მიმართვისათვის მხოლოდ ინდექსები უნდა ავიღოთ. აქვე შევნიშნოთ, რომ ზოგიერთ ძველ ტეგებს (მაგალითად,**<FORM>**, **<INPUT>**) აქვს ატრიბუტი **NAME**. ამ ატრიბუტის მნიშვნელობა შეიძლება გამოყენებულ იქნეს ობიექტზე მიმართვისათვის **ID** ატრიბუტის მნიშვნელობის დონეზე. ფორმის ობიექტს, ზემოთ აღწერილი წესების გარდა შეიძლება მივმართოთ **NAME** (ფორმის სახელი) ატრიბუტის მნიშვნელობით, თუ იგი მითითებულია **<FORM>** ტეგში, მაგრამ არა **ID** ატრიბუტის მნიშვნელობით:

document . ფორმის სახელი

როგორც ცნობილია ფორმის **<FORM>** ტეგი არის კონტეინერული ტეგი და შეიძლება სხვა ელემენტებსაც შეიცავდეს, როგორიცაა **<INPUT>**,

<BUTTON> და სხვა. ფორმის ელემენტებზე მიმართვა შეიძლება განხორციელდეს **all** კოლექციის საშუალებით. ამასთან, მათთვის არსებობს სპეციფიკური წესი:

document . ფორმის სახელი . elements [ელემენტის ინდექსი]

document . ფორმის სახელი . elements [ელემენტის id]

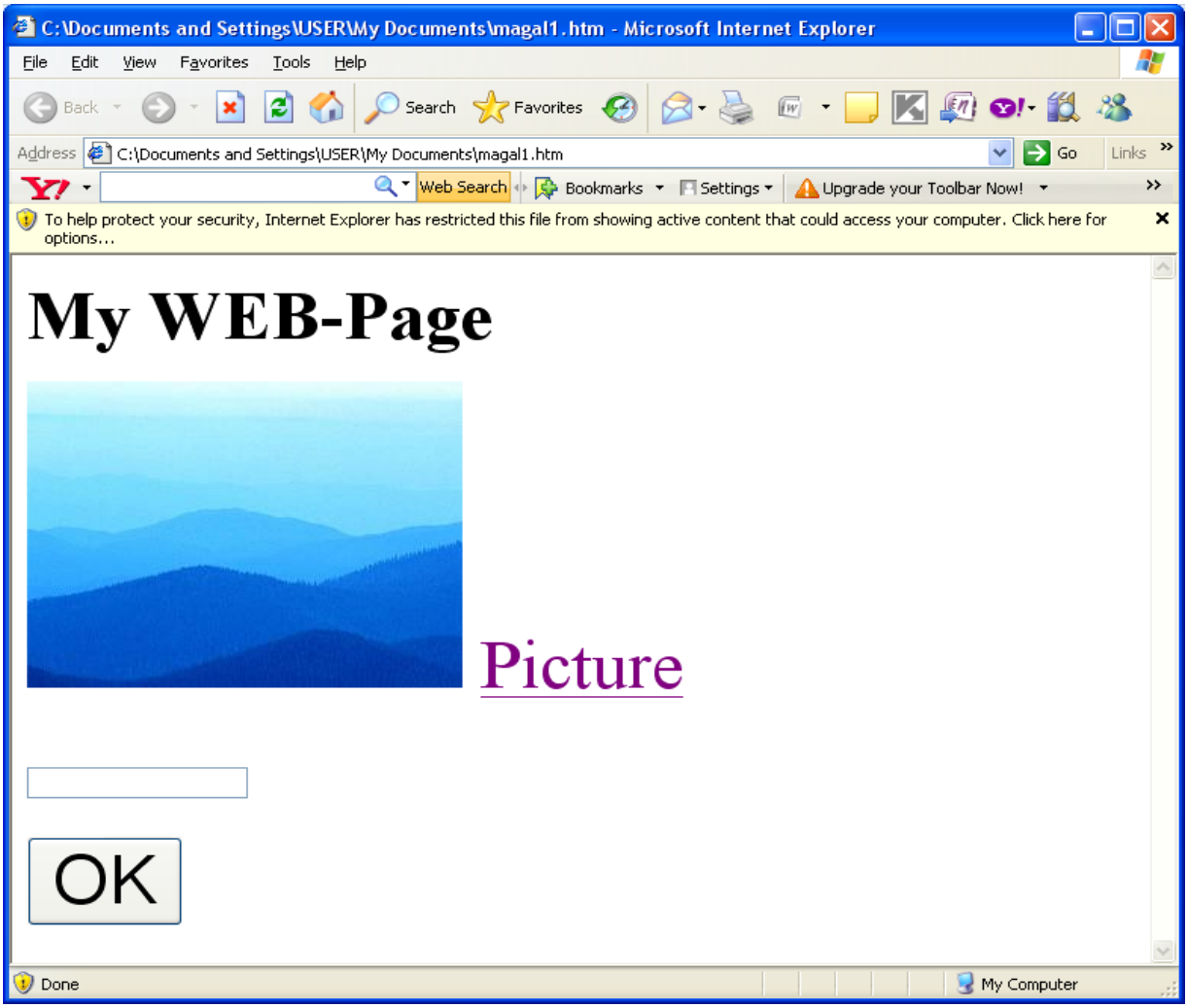
document . ფორმის სახელი . ელემენტის id

აქ **ელემენტის ინდექსი** – ფორმის ელემენტის რიგითი ნომერია და არა რიგითი ნომერი ყველა ელემენტების კოლექციაში. პირველი ელემენტის ინდექსია ნული. აქვე შევნიშნოთ, რომ **Internet Explorer**-ისათვის დასაშვებია კვადრატული ფრჩხილების ნაცვლად გამოვიყენოთ მრგვალი ფრჩხილები.

ქვემოთ მოყვანილი **HTML**-კოდის საშუალებით უბრალო **WEB**-გვერდის ფორმირება ხდება, რომელიც შეიცავს პირველი დონის სათაურს, გამოსახულებას, მითითებას და ფორმას ორი ელემენტით – მონაცემთა შესატანი ველით და ღილაკით.

```
<HTML>
<H1><FONT SIZE='40'> My WEB-Page </H1>
<IMG SRC = "C:\Documents and Settings\USER\My Documents\
My Pictures\Blue hills.jpg">
<A HREF = "C:\Documents and Settings\USER\My Documents\
My Pictures\Sunset.jpg" > Picture</a>
<FORM>
   <INPUT TYPE = "text" VALUE ="">
   <p>
   <BUTTON onclick = "my ( )"><FONT SIZE='30'>OK
</BUTTON>
</FORM>
</HTML>
```

ქვემოთ სურათზე მოცემულია მიღებული **WEB**-გვერდი:

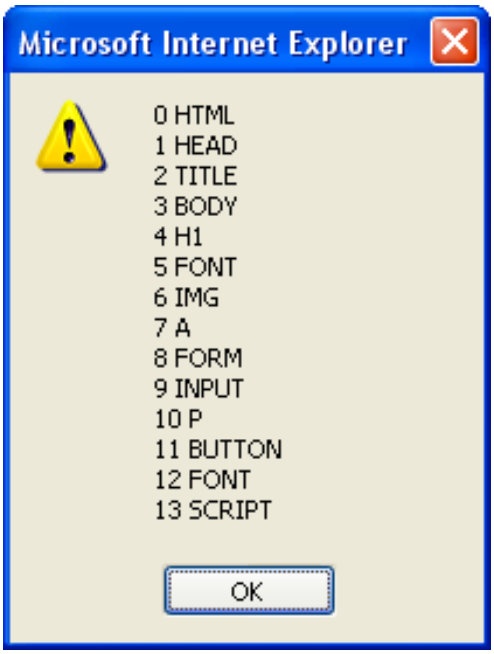


დავუშვათ, გვაინტერესებს ამ დოკუმენტის რომელი ტეგები შეესაბამება **all** კოლექციის **document** ობიექტის ელემენტებს. ამ კითხვაზე პასუხის გასაცემად დავწეროთ სცენარი და ჩავსვათ **HTML**-დოკუმენტის ბოლოში. ამ სცენარში გამოვიყენოთ **tagName** თვისება, რომელიც აბრუნებს იმ ტეგის დასახელებას, რომლის საშუალებითაც შეიქმნა შესაბამისი ობიექტი.

```
<HTML>
<H1><FONT SIZE='40'> My WEB-Page </H1>
<IMG SRC = "C:\Documents and Settings\USER\My Documents\
My Pictures\Blue hills.jpg">
<A HREF = "C:\Documents and Settings\USER\My Documents\
My Pictures\Sunset.jpg"> Picture</a>
<FORM>
  <INPUT TYPE = "text" VALUE ="">
  <p>
```

```
<BUTTON onclick = "my ( )"><FONT SIZE='30'>OK
</BUTTON>
</FORM>
<SCRIPT>
msg = " "
for ( i = 0; i < document . all . length; i ++ ) {
msg += i + " " + document . all [i] . tagName + "\n"
}
alert (msg)
</SCRIPT>
</HTML>
```

ამ **HTML**-კოდის შესრულების შედეგი იქნება ზემოთ მოყვანილი ფანჯარა, რომელსაც დაემატება აქვე მოყვანილი დიალოგური ფანჯარა, სადაც ჩამოთვლილია **HTML**-დოკუმენტში გამოყენებული ტეგები. მაგალითად, **document . all (5)** ობიექტს შეესაბამება **<IMG>** ტეგის მიერ შექმნილი **HTML**-დოკუმენტის ელემენტი. აქვე ყურადღება მიაქციეთ, რომ ტეგები **<HEAD>**, **<TITLE>** და **<BODY>** ცხადად არ მოიხსენიება, მაგრამ მათი შესაბამისი ობიექტებია ამ დოკუმენტის ობიექტურ მოდელში და შესაბამისად **all** კოლექციაში.



23. ხდომილობის ცნება

მომხმარებლის ყოველი მოქმედება (კლავიშზე ხელის დაჭერა, მაუსით დაწკაპუნება და სხვა) ახდენს რაიმე ხდომილობის ფორმირებას, ანუ შეტყობინებას რაიმე მოქმედებაზე. ოპერაციული სისტემა აანალიზებს ამ შეტყობინებას, რათა გაიგოს საიდან მოვიდა და რა გააკეთოს.

ხდომილობათა თვისებები. ხდომილობაზე შეტყობინების ფორმირება ხდება ობიექტის, ანუ ინფორმაციის შესანახი კონტეინერის

სახით. როგორც კი ხდომილობის ობიექტი შეიქმნება ბრაუზერი მის თვისებას მიანიჭებს მნიშვნელობას. მაგალითად, მაუსის დაწკაპუნების შესაბამისი ობიექტი შეიცავს მაუსის მაჩვენებლის კოორდინატებს და აგრეთვე ცნობას იმის შესახებ, თუ რომელი კლავიში დავაწკაპუნეთ. ამის გარდა, ხდომილობის ობიექტი ერთ-ერთ თავის თვისებაში შეიცავს მითითებას ელემენტზე, რომელთანაც არის დაკავშირებული მოცემული ხდომილობა (მაგალითად, ღილაკი, გამოსახულება შეტანის ველი და სხვა).

ხდომილობის ობიექტი მეხსიერებაში ინახება იმდენ ხანს, რამდენიც ესაჭიროება სცენარს მის დასამუშავებლად. ვიდრე სრულდება ხდომილობის დამმუშავებელი მანამდე ბრაუზერის მეხსიერებაში არის განთავსებული ხდომილობის ობიექტი. როგორც კი იგი დაამთავრებს მუშაობას ობიექტი ცარიელდება (უბრუნდება საწყის მდგომარეობას). დროის ნებისმიერ მომენტში არ არსებობს ერთზე მეტი ხდომილობის ობიექტი. ყველა ინიცირებული ხდომილობა ოპერაციული სისტემის მიერ ჩაიწერება ბუფერში და იმ მიმდევრობით სრულდება, რა მიმდევრობითაც მათი იქ ჩაწერა მოხდა.

ობიექტურ მოდელში გვაქვს ობიექტი **event**, რომელიც არის ობიექტ **window**-ს ქვეობიექტი. ის შეიცავს ინფორმაციას თუ რომელი ხდომილობა მოხდა, რომელმა ელემენტმა უნდა მოახდინოს მასზე რეაგირება და სხვა.

ქვემოთ მაგალითის სახით მოვიყვანოთ **HTML**-დოკუმენტი, რომელიც არ შეიცავს ხილულ ელემენტებს. მაუსის მარცხენა კლავიშზე დაწკაპუნება გამოიტანს დიალოგურ ფანჯარას, სადაც გამოტანილი იქნება **event** ობიექტის ზოგიერთი თვისება. თვისება **x** და **y** შეიცავს მაუსის მაჩვენებლის კოორდინატებს დაწკაპუნების მომენტში.

```
<HTML>
<BODY ID = "test">
</BODY>
<SCRIPT>
function test . onclick ( ) {
var str = " "
str += "x = " + window . event . x + "\n"
str += "y = " + window . event . y + "\n"
```

```
str += "თქვენ ხელი დააჭირეთ მაუსის მარცხენა კლავიშს"
alert (str)
}
</SCRIPT>
</HTML>
```

ამ პროგრამული კოდის შედეგი გამოტანილი იქნება ქვემოთ წარმოდგენილი დიალოგური ფანჯრის სახით:

ხშირად გამოიყენება თვისებები **button** და **srcElement**. თვისება **button** შედეგად აბრუნებს მთელ რიცხვით მნიშვნელობას, რომელიც მიუთითებს მაუსის რომელი კლავიში ან კლავიშები დააწკაპუნა მომხმარებელმა (0 – კლავიშები არ არის დაწკაპუნებული, 1 – დაწკაპუნებულია მარცხენა კლავიში, 2 – დაწკაპუნებულია მარჯვენა კლავიში, 3 - ერთდროულად დაწკაპუნებულია ორივე კლავიში).

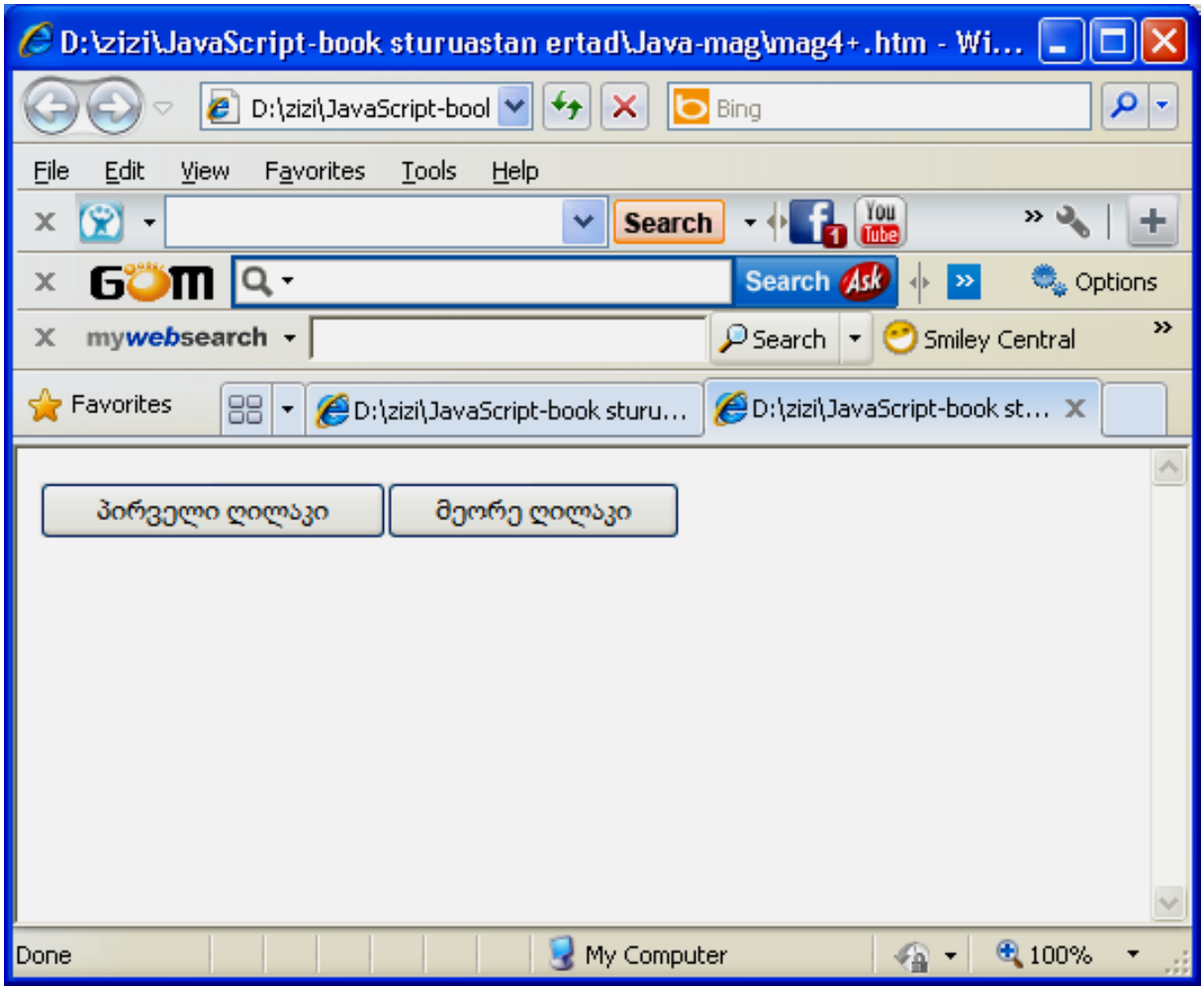
თვისება **srcElement** შედეგად ხდომილობის მაინიცირებელი **HTML**-დოკუმენტის ელემენტის ობიექტზე მითითებას აბრუნებს. ასეთი მითითებით შეიძლება გავიგოთ ან შევცვალოთ ამ ობიექტის თვისების მნიშვნელობა და მის მიმართ გამოვიყენოთ მისი ნებისმიერი მეთოდი. თვისების მნიშვნელობის შეცვლა **innerText** თვისებით ხორციელდება.

ქვემოთ მოყვანილია **HTML**-კოდის მაგალითი, რომლის საშუალებითაც ხდება დოკუმენტის ფორმირება, რომელიც შეიცავს ორ ღილაკს. მაუსით შეიძლება დაწკაპუნება ერთ-ერთ ღილაკზე ან თავისუფალ ადგილზე.

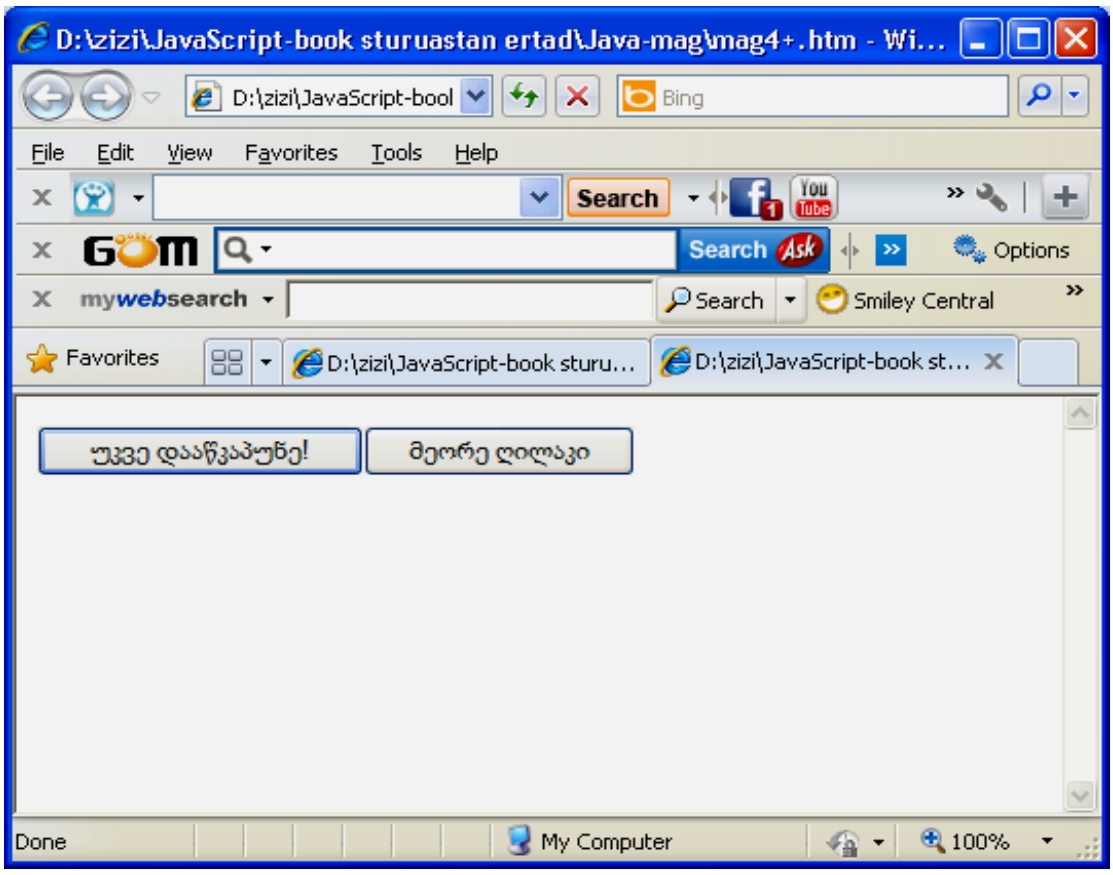
```
<HTML>
<BODY onclick = "changetext ( )">
<button> პირველი ღილაკი </button>
<button> მეორე ღილაკი </button>
</BODY>
<SCRIPT>
```

```
function changetext ( ) {
x = window . event . srcElement
x . innerText = "უკვე დააწკაპუნე"
}
</SCRIPT>
</HTML>
```

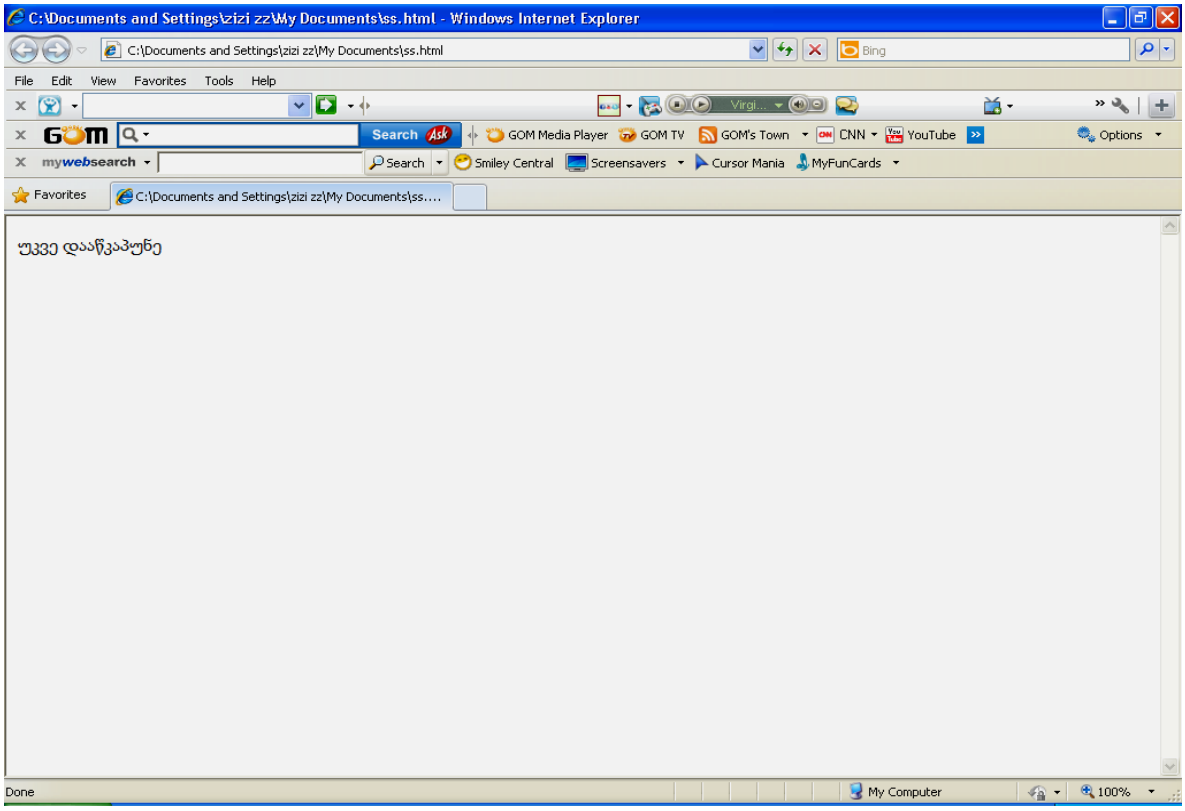
პროგრამის გაშვების შედეგად გამონათდება ფანჯარა:



პირველ შემთხვევაში ის ღილაკი, რომელზეც მოხდა დაწკაპუნება შეიცვლება სხვა, ”უკვე დააწკაპუნე!” ღილაკით (იხილეთ სურათი):



ხოლო თუ ფანჯრის ნებისმიერ სხვა ადგილზე დავაწკაპუნებთ, მაშინ მივიღებთ სურათზე გამოსახულ შედეგს:



იმისათვის, რომ ზემოთ მოყვანილმა კოდმა რეაგირება მოახდინოს მხოლოდ ღილაკზე მაუსით დაწკაპუნებაზე, საჭიროა ამ კოდის შემდეგნაირი მოდიფიცირება:

```
<HTML>
<BODY>
<button onclick = "changetext ( )"> პირველი ღილაკი </button>
<button onclick = "changetext ( )"> მეორე ღილაკი </button>
</BODY>
<SCRIPT>
function changetext ( ) {
x = window . event . srcElement
x . innerText = "უკვე დააწკაპუნე"
}
</SCRIPT>
</HTML>
```

ობიექტი **event**-ს აქვს მაუსთან დაკავშირებული რამდენიმე თვისება, რომელიც შეიცავს ხდომილობის მოხდენის მომენტში მაუსის მაჩვენებლის კოორდინატებს პიქსელებში. ამ ტიპების მრავალსახეობა გამოწვეულია ათვლის საწყისი წერტილების სხვადასხვაობით. ეს თვისებებია:

- **screenX**, **screenY** – მაუსის მაჩვენებლის კოორდინატები ეკრანის ზედა მარცხენა კუთხის მიმართ;

- **clientX**, **clientY** – მაუსის მაჩვენებლის კოორდინატები კლიენტის (ბრაუზერის სამუშაო არე) მიმართ არის, რომელშიც იმყოფება **HTML**-დოკუმენტი, ჩარჩოს, გადაფურცვლის ზოლის, მენიუსა და სხვა გარეშე;

- **offsetX**, **offsetY** – მაუსის მაჩვენებლის კოორდინატები ხდომილობის გამომწვევი ელემენტის ზედა მარცხენა კუთხის მიმართ;

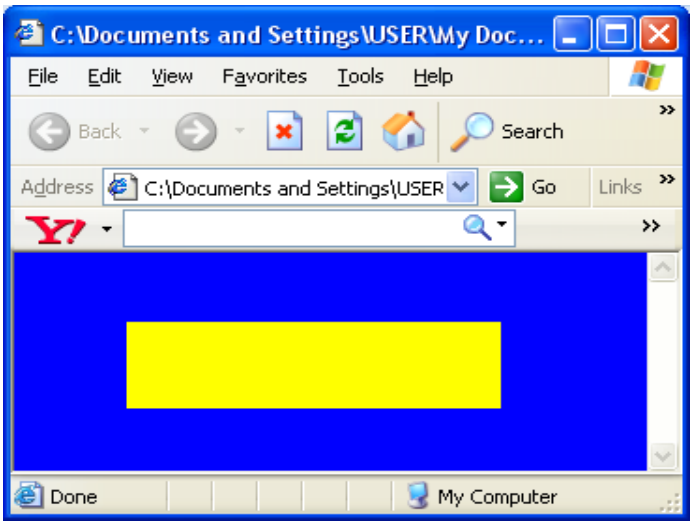
- **x**, **y** – მაუსის მაჩვენებლის კოორდინატები პირველი აბსოლუტური ან ფარდობითი პოზიციური კონტეინერის ზედა მარცხენა კუთხის მიმართ, რომელშიც ხდომილობის გამომწვევი ელემენტი იმყოფება. პოზიციური კონტეინერი არის ელემენტი, რომელიც მოცემულია რომელიმე კონტეინერული ტეგის (მაგალითად,**<BODY>**, **<DIV>**, **<H1>**) საშუალებით, რომელშიც გამოყენებულია **STYLE** ატრიბუტი, სადაც მითითებულია თვისება **position**. თუ ასეთი

კონტეინერი არ არის, მაშინ **x** და **y** თვისება, როგორც **clientX**, **clientY**, დააბრუნებს კოორდინატებს მთავარი დოკუმენტის მიმართ.

ეს თვისებები განვიხილოთ მაგალითზე:

```
<HTML>
<BODY ID = "Mybody"
<DIV ID = "DIV1" STYLE = "position:absolute; left:50; top:50;
    width:30; height:100; background-color:blue">
<DIV ID = "DIV1" STYLE = "position:relative; left:50; top:25;
    width:200; height:50; background-color:yellow">
</DIV>
</DIV>
</BODY>
<SCRIPT>
function document . onclick ( ) {
var e = window . event
var str = "ID = " + e . srcElement . id +
"\n screenX=" + e . screenX + ", screenY=" + e . screenY +
"\n clientX=" + e . clientX + ", clientY=" + e . clientY +
"\n offsetX=" + e . offsetX + ", offsetY=" + e . offsetY +
"\n x=" + e . x + ", y=" + e . y
alert (str)
}
</SCRIPT>
</HTML>
```

ამ კოდის შესრულებაზე გაშვებისა და მაუსის დაწკაპუნებით მივიღებთ შემდეგ შედეგს:



24. მუშაობა ფანჯრებთან და ჩარჩოებთან

როგორც ცნობილია, **HTML**-დოკუმენტი ბრაუზერის ფანჯარაში იტვირთება. შესაძლებელია რამდენიმე ფანჯრის გახსნა და მასში სხვადასხვა დოკუმენტის ჩატვირთვა, ასევე ერთი ფანჯრის რამდენიმე მართკუთხედის ფორმის არეებად დაყოფა, რომელსაც ჩარჩოები ეწოდებათ. ყოველ ასეთ ჩარჩოში ცალკეული დოკუმენტის ჩატვირთვა შეიძლება. ამასთან, არსებობს შესაძლებლობა მოხდეს ურთიერთდაკავშირება ამ ჩარჩოებს შორის. ამ დროს ბრაუზერში იქმნება ამ დოკუმენტის იერარქიული ობიექტური მოდელი, რომლის ყველაზე ზედა დონე არის ობიექტი **window**. მოცემული ობიექტის თვისებებსა და მეთოდებზე მიმართვას ასეთი სახე აქვს:

window . თვისება

window . მეთოდი ([პარამეტრები])

window ობიექტს აქვს სინონიმი **self**, რომელიც მიმდინარე დოკუმენტის შემცველ ფანჯარაზე მიმართვის დროს გამოიყენება. იდენტიფიკატორი **self**-ი გამოიყენება მრავალფანჯრიან ან მრავალჩარ-ჩოიან სისტემებში, როცა საჭიროა მივუთითოთ ფანჯარა დოკუმენტით, რომელშიც არის განთავსებული მოცემული სცენარი. სცენარის გაშვების დროს მიმდინარე დოკუმენტში განთავსებულ ობიექტზე მითითებებში **window** და **self** იდენტიფიკატორები შეიძლება გამოვტოვოთ.

window ობიექტს აქვს რიგი ქვეობიექტებისა. **event** თვისებას ჩვენ უკვე გავეცანით. მეორე ობიექტი, **location**, შეიცავს ქსელში მუშაობისა და მრავალჩარჩოიანი სტრუქტურის მქონე დოკუმენტებში მითითების შექმნისათვის სასარგებლო ინფორმაციას. ამის გარდა, **location** ობიექტის **href** თვისება გამოიყენება მიმდინარე ფანჯარაში დოკუმენტის ჩასატვირთად:

window . location. href = "URL – დოკუმენტის მისამართი"

Internet Explorer-ში შეიძლება გამოვიყენოთ მეთოდი **navigate ( )**:

window . navigate ("URL – დოკუმენტის მისამართი")

ახალი ფანჯრების შექმნა. ბრაუზერის მთავარი ფანჯარა იქმნება არა სცენარის დახმარებით, არამედ ავტომატურად, ბრაუზერის გაშვების

თანავე, აგრეთვე დოკუმენტის გახსნის დროს განსაზღვრული **URL**-მისამართით ან სხვა ფაილით.

HTML-ში დოკუმენტის გახსნა ახალ ფანჯარაში შეიძლება მითითების ტეგში **<A HREE = . . . > TARGET** ატრიბუტის საშუალებით. მაგალითად,

<A HREE = http://www.rambler.ru TARGET = "newWindow">Rumbler</A>

სცენარის საშუალებით შეიძლება შეიქმნას ნებისმიერი რაოდენობის ფანჯარა. ამისათვის გამოიყენება მეთოდი **open ( )**:

window . open ([პარამეტრები])

ამ მეთოდს გადაეცემა შემდეგი არააუცილებელი პარამეტრები:

- შესაქმნელ ფანჯარაში ჩასატვირთი დოკუმენტის მისამართი;
- ფანჯრის სახელი (როგორც ცვლადის სახელი);
- ფანჯრის თვისების აღწერის სტრიქონი (**features**).

თვისებათა სტრიქონში ჩაიწერება წყვილი **თვისება=მნიშვნელობა**, რომელიც ერთმანეთისაგან მძიმით გამოიყოფა. ქვემოთ ცხრილში მოყვანილია ფანჯრის თვისებათა სია, რომელიც **features** სტრიქონში გადაეცემა. მნიშვნელობა **yes** და **no** შეიძლება შეიცვალოს მათი რიცხვითი ეკვივალენტით შესაბამისად **1** და **0**-ით.

features სტრიქონში გადაცემული ფანჯრის თვისებები

თვისება	მნიშვნელობა	აღწერა
channel mode	**yes, no, 1, 0**	უჩვენებს **Channel**-ის მართვის ელემენტებს
directories	**yes, no, 1, 0**	რთავს კატალოგის ღილაკებს
ful1screen	**yes, no, 1, 0**	მთლიანად მოაბრუნებს ფანჯარას
height	**რიცხვი**	ფანჯრის სიმაღლე პიქსელებში
Left	**რიცხვი**	ჰორიზონტალურად ეკრანის მარცხენა კიდის მიმართ მდებარეობა პიქსელებში
location	**yes, no, 1, 0**	ტექსტური **Address** ველი
menubar	**yes, no, 1, 0**	ბრაუზერის სტანდარტული მენიუ
resizeable	**yes, no, 1, 0**	შეუძლია თუ არა მომხმარებელს ფანჯრის ზომების შეცვლა
scrollbar	**yes, no, 1, 0**	გადაფურცვლის ჰორიზონტალური და ვერტიკალური ზოლები
Status	**yes, no, 1, 0**	მდგომარეობის სტანდარტული სტრიქონი

toolbar	**yes, no, 1, 0**	რთავს ბრაუზერის ინსტრუმენტთა პანელს
Top	**რიცხვი**	ვერტიკალურად ეკრანის ზედა კიდის მიმართ მდებარეობა პიქსელებში
Width	**რიცხვი**	ფანჯრის სიგანე პიქსელებში

მაგალითი:

```
window . open ("mypage.htm", "NewWin", "height = 150, width = 300")
window . oipen ("mypage.htm")

strfeatures = "top = 100, left = 15, width = 400, height = 200,
      location = no, menubar = no"
window . open ("www. rambler. ru", "Web-Page", strfeatures)
```

მეთოდი **window . open ( )** აბრუნებს ფანჯრის ობიექტზე მითითებას. ეს მითითება შეიძლება შევინახოთ ცვლადში, რათა შემდგომ იგი გამოვიყენოთ, მაგალითად, ფანჯრის დახურვის დროს. ფანჯრის დახურვისათვის გამოიყენება მეთოდი **close ( )**. მაგრამ გამოსახულება **window . close ( )** ან **self . close ( )** ხურავს მთავარ ფანჯარას და არა დამატებითს, რომელიც ჩვენ შევქმენით **open ( )** მეთოდით. ზუსტად, ამ შემთხვევისათვისაა შექმნილი ფანჯრის მითითება. ეს მითითება საჭიროა შენახულ იქნეს გლობალურ ცვლადში, რადგან ხელმისაწვდომი იყოს მანამ, ვიდრე მთავარი დოკუმენტი ჩატვირთულია ბრაუზერში. მაგალითად:

```
var objwin = window . open ("mypage . htm", "My Page")
objwin . close ( )
```

window . open ( ) მეთოდი ხსნის ახალ დამოუკიდებელ ფანჯარას როგორც ბრაუზერის ეგზემპლარს. ამ დროს ბრაუზერის მთავარი ფანჯრის დახურვის შემდეგ ახალი ფანჯარა რჩება გახსნილი. დამოუკიდებელ ფანჯარას კიდევ უწოდებენ არამოდალურს (**modalless**). ამასთან, შეიძლება შეიქმნას მოდალური ფანჯარაც. ვიდრე გახსნილია მოდალური ფანჯარა, მომხმარებელს არ შეუძლია მიმართოს სხვა ფანჯრებს, მათ შორის მთავარსაც. ასე ჩვეულებრივ მუშაობს სტანდარტული დიალოგური ფანჯარა. მაგალითად, **alert ( )**, **prompt ( )** და **confirm ( )** მეთოდით შექმნილი ფანჯრები არის მოდალური. მოდალურ ფანჯარაში შეიძლება ჩაიტვირთოს ნებისმიერი დოკუმენტი.

მოდალური ფანჯრის შესაქმნელად გამოიყენება **showModalDialog ( )** მეთოდი. როგორც **open ( )** მეთოდს, ისიც პარამეტრის სახით ღებულობს დოკუმენტის (ფაილის) მისამართს, ფანჯრის მისამართს და თვისებათა სტრიქონს. მაგრამ ამ სტრიქონის ფორმატი სხვანაირია. კერძოდ, სტრიქონში პარამეტრები ერთმანეთისაგან გამოიყოფა წერტილ-მძიმით, ფანჯრის ზომები და მისი ზედა მარცხენა კუთხის კოორდინატები საჭიროებს საზომი ერთეულის (მაგალითად, **px** – პიქსელი) მითითებას. ამის გარდა, ეს მეთოდი არ აბრუნებს ფანჯრის ობიექტზე მითითებას, რადგან იგი არ არის საჭირო მოდალური ფანჯრისათვის.

ქვემოთ ცხრილში მოყვანილია **showModalDialog ( )** მეთოდით შექმნილი ფანჯრის თვისებათა სია, რომელიც გადაეცემა **features** სტრიქონში.

showModalDialog ( ) მეთოდით შექმნილი ფანჯრის თვისებები

თვისება	მნიშვნელობა	აღწერა
order	**thick, thin**	ფანჯრის გარშემო ჩარჩოს ზომა (სქელი/ვიწრო)
center	**yes, no, 1, 0**	მთავარი ფანჯრის მიმართ განთავსება ცენტრში
dialogHeight	**რიცხვი+საზ. ერთ.**	ფანჯრის სიმაღლე
dialogLeft	**რიცხვი+საზ. ერთ.**	ჰორიზონტალური კოორდინატი
dialogTop	**რიცხვი+საზ. ერთ.**	ვერტიკალური კოორდინატი
dialogWidth	**რიცხვი+საზ. ერთ.**	ფანჯრის სიგანე
Font	**სტილის ცხრილის სტრიქონი**	ფანჯრისათვის გაჩუმების პრინციპით განსაზღვრული სტილი
font-family	**სტილის ცხრილის სტრიქონი**	ფანჯრისათვის გაჩუმების პრინციპით განსაზღვრული შრიფტის სახეობა
font-size	**სტილის ცხრილის სტრიქონი**	ფანჯრისათვის გაჩუმების პრინციპით განსაზღვრული შრიფტის ზომა
font-style	**სტილის ცხრილის სტრიქონი**	ფანჯრისათვის გაჩუმების პრინციპით განსაზღვრული შრიფტის ტიპი
font-variant	**სტილის ცხრილის სტრიქონი**	ფანჯრისათვის გაჩუმების პრინციპით განსაზღვრული შრიფტის ვარიანტი
font-weight	**სტილის ცხრილის სტრიქონი**	ფანჯრისათვის გაჩუმების პრინციპით განსაზღვრული შრიფტის სისქე
Help	**yes, no, 1, 0**	ზედა პანელში **Help** ღილაკის ჩართვა

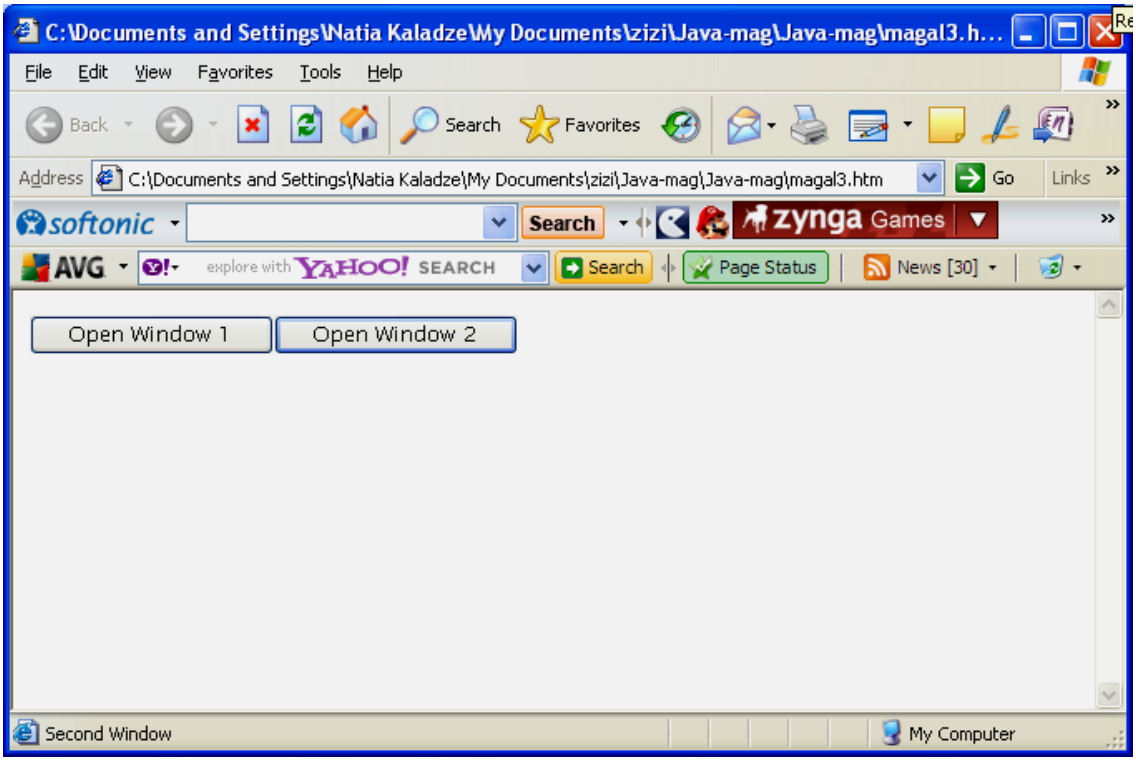
maximize	**yes, no, 1, 0**	ზედა პანელში **Maximize** ღილაკის ჩართვა
minimize	**yes, no, 1, 0**	ზედა პანელში **Minimize** ღილაკის ჩართვა

მაგალითი:

```
<HTML>
<BUTTON onclick = "return OpenWin1()">Open Window 1
    </BUTTON>
<BUTTON onclick = "return OpenWin2()">Open Window 2
    </BUTTON>
<SCRIPT>
var newWindow

function OpenWin1 () {
window . status = "Ferst Window"
strfeatures = "top=100, left=50, width=300, height=270, toolbar=no"
window . open ("magal1.htm", "win1", strfeatures)
}
function OpenWin2 () {
window . status = "Second Window"
strfeatures = "dialogWidth=500px; dialogHeight=320px;
    border=thin; help=no"
window . showModalDialog("magal2.htm", "win2", strfeatures)
}
function CloseWin1 () {
if (newWindow) {
newWindow . close ()
newWindow = null
window . status = ""
}
}
</SCRIPT>
</HTML>
```

პროგრამის გაშვების შემდეგ ამონათდება შემდეგი სახის ფანჯარა:



ფანჯარაში მოთავსებულ ღილაკებზე მაუსის დაწკაპუნებით კი გაიხსნება პირველი და მეორე მაგალითის ამსახველი ფანჯრები.

25. ჩარჩოები

ჩარჩო არის ბრაუზერის ფანჯრის მართკუთხედ არეს, რომელშიც შეიძლება ჩაიტვირთოს **HTML**-დოკუმენტი. ფანჯრის დაყოფა ჩარჩოებად ხდება **HTML**-კოდის საშუალებით, რომელიც განთავსებულია ცალკე **HTML**-ფაილში. ამ ფაილს ეწოდება ჩარჩოს დასაყენებელი ფაილი. მასში ჩაწერილია მხოლოდ ჩარჩოს მოსანიშნი ტეგები ანუ მათი ფარდობითი ურთიერთმდებარეობა, ზომები, მათში ჩასატვირთ დოკუმენტებზე მითითება და ჩარჩოების სხვა პარამეტრები. მათში არ უნდა იყოს სხვა ელემენტების ტეგები და ტექსტური ინფორმაცია. გამონაკლისი არის ტეგი <**META**>. ჩარჩოს დასაყენებელ ფაილში შეიძლება ჩაიწეროს დოკუმენტის ჩარჩოს სტრუქტურის შესაქმნელი სცენარი. ბრაუზერის ფანჯრის დაყოფა რამდენიმე ჩარჩოდ ხდება <**FRAMESET**> ტეგის საშუალებით. ამ ტეგის შიგნით ისმება <**FRAME**> ტეგი თავისი ატრიბუტებით, რომელიც მიუთითებს ჩარჩოს სახელსა და ამ ჩარჩოში ასასახ **HTML**-დოკუმენტის მისამართს.

მაგალითი:

```
<HTML>
<FRAMESET COLS ="30%, 70%">
<FRAME SRC ="magal1.htm" NAME="frame1">
<FRAME SRC ="magal2.htm" NAME="frame2">
</FRAMESET>
</HTML>
```

ეს მაგალითი წარმოადგენს ჩარჩოების ჰორიზონტალურ განლაგებას. თუ **COLS** ატრიბუტის მაგივრად გამოვიყენებთ **ROWS** ატრიბუტს, მაშინ ჩარჩოები განლაგდება ვერტიკალურად. თუ გამოვიყენებთ ერთმანეთში ჩალაგებულ **<FRAMESET>** ტეგს, მაშინ არსებული ჩარჩო შესაძლებელია დავყოთ კიდევ ორ ჩარჩოდ და ა. შ.

დამოკიდებულება ჩარჩოებსა და მთავარ ფანჯარას შორის. ფანჯრის ორ ჩარჩოდ დაყოფის დროს ბრაუზერის ობიექტურ მოდელში წარმოიქმნება ობიექტთა იერარქია, რომელშიც ბრაუზერის მთავარი ფანჯარა წარმოდგება მთავარი, მშობლიური ჩარჩოს სახით, ხოლო შექმნილი ორი ჩარჩო - შვილობილ ჩარჩოდ. ამ ჩარჩოებიდან ნებისმიერი მათგანის ისევ ორად დაყოფის შემთხვევაში, ეს უკანასკნელი შვილობილი ჩარჩო იქნება.

ბრაუზერის გაშვების შემთხვევაში მთავარი ფანჯრის **window** ობიექტი ავტომატურად ფორმირდება. თუ ამ ფანჯარაში იტვირთება დოკუმენტი ჩარჩოიანი სტრუქტურით, მაშინ მთავარი ფანჯარა იღებს მშობლიური ჩარჩოს სტატუსს ყველა სხვა ჩარჩოს მიმართ. მეორე მხრივ ყოველი **<FRAME>** ტეგი **<FRAMESET>** კონტეინერის შიგნით ქმნის თავის საკუთარ **window** ობიექტს, რომელშიც ჩაიტვირთება შესაბამისი დოკუმენტი. ყოველ ჩარჩოს შეესაბამება თავისი **document** ობიექტი. **document** ობიექტის თვალთახედვიდან კი მას შეესაბამება ერთადერთი **<FRAMESET>** კონტეინერი. თუმცა, მშობლიური ობიექტი მომხმარებლისათვის უხილავია, მაგრამ იგი არსებობს ობიექტურ მოდელში. მშობლიურ ფანჯარაზე მითითებისათვის გამოიყენება გამხსნელი სიტყვა **parent**.

ზოგჯერ საჭიროა მივიღოთ მშობლიურ ფანჯარაზე მიმართვის უფლება, რომელშიც უნდა ჩაიტვირთოს ახალი დოკუმენტი. ამისათვის, სცენარში საჭიროა ჩაიწეროს შემდეგი:

parent . location . href = "URL - დოკუმენტის მისამართი"

ხშირად საჭირო ხდება ერთი ჩარჩოდან მივმართოთ მეორე ჩარჩოს. ამ მიზნით ჩვენ ჯერ უნდა მივმართოთ მშობლიურ ფანჯარას, ხოლო შემდეგ – **frame2**-ს და მხოლოდ ამის შემდგომ **document** ობიექტს, რომელიც განთავსებულია მეორე ჩარჩოში:

parent . frame2 . document . write ("სტრიქონი")

მაგალითისთვის, განვიხილოთ თუ როგორ ხდება ერთი ჩარჩოდან მეორეში ელემენტის ცვლილება. ქვემოთ მოყვანილია მარცხენა ჩარჩოში ჩაწერილი კოდი:

```
<HTML>
One <BR>
Two
<H1 ID = "XXX">Three</H1>
</HTML>
```

ხოლო მარჯვენა ჩარჩოში ჩაწერილია შემდეგი დოკუმენტი:

```
<HTML>
<SCRIPT>
function change ()  {
parent.frame1.XXX.innerText = "Hello!"
}
</SCRIPT>
<H1 onclick = "change ()"> CLICK HERE <H1>
</HTML>
```

მარცხენა სურათზე მოყვანილია პროგრამის გაშვების შედეგად გამოტანილი ჩარჩოები, ხოლო მარჯვენა სურათზე ცვლილების შედეგი:

ელემენტების შესაცვლელად **innerText** თვისების გარდა შეიძლება **outerText**, **innerHTML** და **outerHTML** თვისებების მოხმარებაც.

ჩარჩოების გამოყენება მოსახერხებელია ნავიგაციური პანელების შესაქმნელად. ერთ ჩარჩოში განთავსდება მითითებები, ხოლო მეორეში გააქტიურებული მითითების შედეგად გამოძახებული დოკუმენტი.

მაგალითი:

ნავიგაციური პანელის შექმნა. ამ შემთხვევაში ბრაუზერის ფანჯარა გაყოფილია ორ ნაწილად, ერთი ასრულებს ნავიგაციური პანელის, ხოლო მეორე დოკუმენტის ასახვის როლს.

```
<HTML>
<FRAMESET COLS ="25%, 75%">
<FRAME SRC = "menu.htm" NAME="menu">
<FRAME SRC = "start.htm" NAME="main">
</FRAMESET>
</HTML>
```

აქ **start.htm** არის ის დოკუმენტი, რომელიც თავდაპირველად გამოჩნდება **main** ჩარჩოში, ხოლო **menu.htm** – ნავიგაციური პანელია:

```
<HTML>
<SCRIPT>
function load (url) {
parent . main . location . href = url;
}
</SCRIPT>
<BODY>
<A HREF = "javascript:load ('magal1.htm')"> ONE </A>
<A HREF = "magal2.htm" TARGET = "main"> TWO </A>
<A HREF = "magal3.htm" TARGET = "top"> THREE </A>
</BODY>
</HTML>
```

მოყვანილ მაგალითში პირველი-ორი დოკუმენტი ჩაიტვირთება ჩარჩოში, ხოლო მესამე ცალკე ფანჯრის სახით გაიხსნება, რაზეც **TARGET** ატრიბუტი მიუთითებს. კონკრეტულ ჩარჩოზე მიმართვა შეიძლება **<FRAME>** ტეგში, ატრიბუტ **NAME**-ში მნიშვნელობის სახით მითითებული ჩარჩოს ინდექსის ან სახელის , მეშვეობით:

window . frames [ინდექსი]

window . ჩარჩოს სახელი

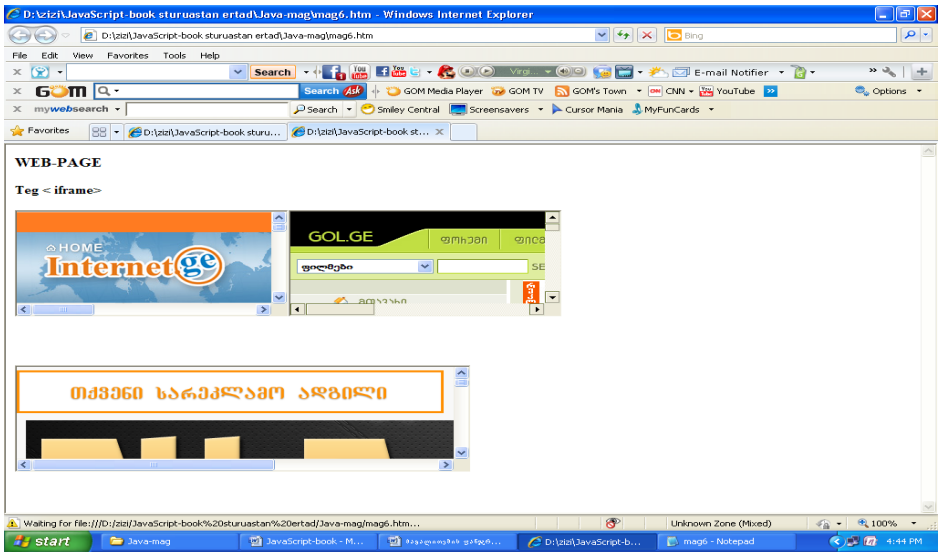
”მცურავი” ჩარჩო. ერთი **HTML**-დოკუმენტის ჩასმა მეორის ტანში, არა სერვერის, არამედ მომხმარებლის ბრაუზერის საშუალებით შესაძლებელია **<IFRAME>** კონტეინერული ტეგით:

<IFRAME> SRC = "დოკუმენტის მისამართი"></ **IFRAME>**

ამ ტეგით მოცემული ელემენტების პოზიციები შეიძლება განისაზღვროს **<STYLE>** ტეგის ან **STYLE** ატრიბუტით. გარეგნულად ეს ელემენტი გამოიყურება როგორც მართკუთხა არე (გადაფურცვლის ზოლის ან მის გარეშე), რომელშიც **HTML**-ფაილიდან აისახება დოკუმენტი. ასეთ ფანჯრებს ზოგჯერ ”მცურავ” ჩარჩოს უწოდებენ. ”მცურავ” ჩარჩოში ჩასატვირთ **HTML**-დოკუმენტებს შეიძლება ჰქონდეთ **HTML**-დოკუმენტების შესაბამისი სცენარი და სხვა საშუალებანი. ქვემოთ მოყვანილია სამი სხვადასხვა დოკუმენტის ”მცურავ” ჩარჩოში ჩატვირთვის მაგალითი:

```
<HTML>
<H3>   WEB-PAGE </H3>
<H4>   Teg < iframe> </H4>
<IFRAME SRC = "http://www.internet.ge" ></IFRAME>
<IFRAME SRC = "http://www.gol.ge"></IFRAME>
<p>
<IFRAME SRC = "http://www.file.ge" STYLE =
      "position:absolute; top:310; width:500; heght:250;
      SCROLLING = no"></IFRAME>
</HTML>
```

პროგრამის გაშვების შემდეგ მივიღებთ:



”მცურავი” ჩარჩო თითქმის ისევე იქცევა, როგორც ჩვეულებრივი ჩარჩო. თუ ის შექმნილია, მაშინ ის შეიძლება ვიპოვოთ **frames** კოლექციაში. მის თვისებებს შორის განვიხილოთ **align** თვისება, რომელიც ”მცურავი” ჩარჩოს დოკუმენტის შემცველი არსებული გარემოს მიმართ გათანაბრებას იწვევს. მისი შესაძლო მნიშვნელობებია:

- **absbottom** – ჩარჩოს ქვედა საზღვრის სწორება მის გარშემო მყოფი ტექსტის სიმბოლოთა სტრიქონის ქვედა ხაზის მიმართ;
- **absmiddle** – ჩარჩოს შუა ნაწილის სწორება მის გარშემო მყოფი ტექსტის **top** და **absbottom**-ს შორის ცენტრალური ხაზის მიმართ;
- **baseline** – ჩარჩოს ქვედა საზღვრის სწორება მის გარშემო მყოფი ტექსტის საბაზო ხაზის მიმართ;
- **bottom** – ემთხვევა **baseline**-ს (მხოლოდ **Internet Excplorer**-ისათვის);
- **left** – ჩარჩოს სწორება ელემენტ-კონტეინერის მარცხენა კიდით;
- **middle** – გარშემო მყოფი ტექსტის წარმოსახვითი ცენტრალური ხაზის სწორება ჩარჩოს წარმოსახვითი ცენტრალური ხაზის მიმართ;
- **right** – ჩარჩოს სწორება ელემენტ-კონტეინერის მარჯვენა კიდით;
- **texttop** – ჩარჩოს ზედა საზღვრის სწორება მის გარშემო მყოფი ტექსტის სიმბოლოთა სტრიქონის ზედა ხაზის მიმართ;
- **top** – ჩარჩოს ზედა საზღვრის სწორება მის გარშემო მყოფი ტექსტის ზედა საზღვრის მიმართ.

მოლივლივე ფანჯრები. მოლივლივე ფანჯარას არა აქვს მართვის ორგანო და იმ დოკუმენტზე განთავსდება, რომელშიც იგი შეიქმნა, მათ შორის დიალოგურ ფანჯარაზე. მაუსის სხვა ადგილზე დაწკაპუნება ან სხვა დანართის გამოძახება იწვევს ამ ფანჯრის დახურვას. იგი ასევე არ აისახება **windows**-ის ამოცანათა პანელზე. მოლივლივე ფანჯრის შექმნის შემდეგ მასში გამოჩნდება ტექსტი. ამასთან, ყურადღება უნდა მიექცეს იმას, რომ ეს ტექსტი სრულად განთავსდეს მოცემულ საზღვრებში, ვინაიდან მას არა აქვს გადაფურცვლის ზოლები. ესეთი ფანჯრების გამოყენება მოსახერხებელია კონტექსტური საცნობარო ინფორმაციის გამოსატანად.

მოლივლივე ფანჯარას შეესაბამება ობიექტი **popup**. იგი იქმნება **window . createPopup ( )** მეთოდით. მისი თვისებებია:

- **document** – თვისება, რომელსაც მნიშვნელობის სახით აქვს მოლივლივე ფანჯარაში არსებულ დოკუმენტზე მითითება; მისი საშუალებით შეიძლება როგორც ფანჯრის, ასევე მისი შიგთავსის რიგი პარამეტრების მომართვა:

ფანჯრის საზღვრების:

mypopup . document . body . style . border = "solid 4px black"

ფანჯრის ფონის ფერი:

mypopup . document . body . style . background = "yellow"

ფანჯარაში ტექსტის ფერი:

mypopup . document . body . style . color = "blue"

- **isOpen** – ვიდრე მოლივლივე ფანჯარა აისახება ეკრანზე, ამ თვისებას აქვს მნიშვნელობა **true**, წინააღმდეგ შემთხვევაში – **false**; გამოიყენება იმ სცენარების გამოსახულებაში, რომელიც სრულდება უკვე გახსნილი ფანჯრის შემთხვევაში.

popup ობიექტის მეთოდები:

- **show (left, top, width, height [, პოზიცია])** – ასახავს მოლივლივე ფანჯარას **popup** ობიექტის შექმნის შემდეგ **window . createPopup ( )** მეთოდის დახმარებით და მისი შევსებით. ფანჯრის გაქრობის შემდეგ მისი განმეორებითი გახსნა ხდება **show ( )** მეთოდით. პირველი ოთხი ახდენს ფანჯრის ზომებისა და პოზიციის დაფიქსირებას. პარამეტრი **left** და **top** განსაზღვრავს ფანჯრის მარცხენა ზედა კუთხის კოორდინატებს მონიტორის ეკრანის მიმართ და არა ბრაუზერის ფანჯრის მიმართ;

- **hide ( )** – შეიძლება დაიმალოს შექმნილი და გამოსახული მოლივლივე ფანჯარა.

მაგალითი:

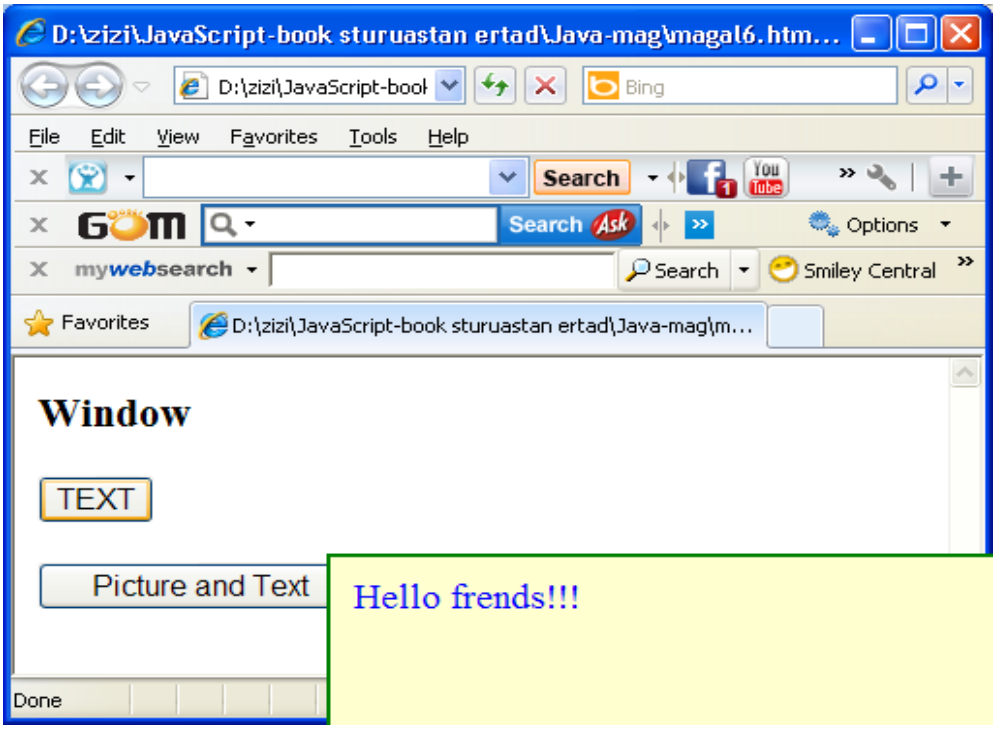
```
<HTML>
<H3> Window </H3>
<BUTTON onclick = "pop('Hello frends!!!')">TEXT</BUTTON>
<BR><BR>
<BUTTON onclick = "pop('<IMG SRC = C:\My Pictures\DSC5.gif>
    Picture')"> Picture and Text </BUTTON>
```

```
<SCRIPT>
function  pop (xcontent) {
var  mypopup = window . createPopup ( );
var  popupBody = mypopup . document . body
popupBody . style . border = "solid 2px green"
popupBody . style . padding = "5px"
popupBody . style . color = "blue"
popupBody . style . background = "ffffd0"
popupBody . innerHTML = "<p>" + xcontent + "</p>"
mypopup . show (130, 90, 400, 200, document . body)
}
</SCRIPT>
</HTML>
```

ზემოთ განხილულ მაგალითში **pop ( )** ფუნქციის ყოველი გამოძახების დროს ხელახლა იქმნება მოლივლივე ფანჯარა, რომლის შინაარსი შეიძლება შეიცვალოს. ვინაიდან ფანჯრის პოზიცია, ზომა და ფერი არ იცვლება, ამიტომ შეგვიძლია მისი შექმნის გამოსახულება და პარამეტრები გამოტანილი იყოს **pop ( )** ფუნქციის განსაზღვრის გარეთ და მას შემდეგი სახე ექნება:

```
<HTML>
<H3> Window </H3>
<BUTTON onclick = "pop('Hello frends!!!')">TEXT</BUTTON>
<BR><BR>
<BUTTON onclick = "pop('<IMG SRC = C:\My Pictures\DSC5.gif>
      Picture')"> Picture and Text </BUTTON>
<SCRIPT>
var  mypopup = window . createPopup ( );
var  popupBody = mypopup . document . body
popupBody . style . border = "solid 2px green"
popupBody . style . padding = "5px"
popupBody . style . color = "blue"
popupBody . style . background = "ffffd0"
function  pop (xcontent) {
popupBody . innerHTML = "<p>" + xcontent + "</p>"
mypopup . show (130, 90, 400, 200, document . body)
}
```

```
</SCRIPT>
</HTML>
```



გამონათებულ ფანჯარაში **Text** ღილაკზე მაუსის დაწკაპუნების შემდეგ გამოჩნდება დამატებითი ფანჯარა ჩანაწერით "**Hello frends**".

26. დოკუმენტის ელემენტების დინამიკური ცვლილება

ბრაუზერში ჩატვირთული **HTML**-დოკუმენტი შეიძლება შეიცვალოს სცენარის საშუალებით. არსებობს დინამიკური ცვლილების სამი ძირითადი საშუალება:

- **write ( )** მეთოდის საშუალებით;
- **HTML**-ტეგებისა და პარამეტრების შესაბამისი თვისებების მნიშვნელობათა შეცვლის გზით;
- **innerText**, **outerText**, **innerHTML** და **outerHTML** თვისებების შეცვლის გზით.

write ( ) მეთოდის გამოყენება. ობიექტ **document**-ის მეთოდი **write ( )** უკვე რამდენიმეჯერ განვიხილეთ. პარამეტრის სახით ის იღებს სტრიქონს. შედეგად დოკუმენტი და მისი ობიექტური მოდელი განახლდება. ამასთან, ფაილი საწყისი **HTML**-კოდით ცვლილების გარეშე

დარჩება. თუ საჭიროა მიმდინარე დოკუმენტის მთლიანად შეცვლა, მაშინ ჯერ უნდა განხორციელდეს ამ დოკუმენტის გასუფთავება **document . clear ( )** მეთოდი, ხოლო შემდეგ **document . write (სტრიქონი)**. თუმცა, მიმდინარე დოკუმენტის ასეთი კარდინალური ტრანსფორმაციის დროს უნდა ვიყოთ ფრთხილად. უფრო უსაფრთხო მეთოდი გახლავთ – ჯერ სცენარის საშუალებით მოვახდინოთ ახალი **HTML**-დოკუმენტის შიგთავსის მიმდინარე დოკუმენტში გენერირება, ხოლო შემდეგ გავაგზავნოთ **HTML**-კოდი ახალ ფანჯარაში ან მრავალჩარჩოიანი დოკუმენტის შემთხვევაში სხვა ჩარჩოში. უფრო ზუსტად, **write ( )** მეთოდს პარამეტრის სახით შეუძლია მიიღოს ნებისმიერი რაოდენობის სტრიქონი:

document . write (სტრიქონი1 [, სტრიქონი2 . . . [, სტრიქონიN]])

აქ კვადრატული ფრჩხილები მიუთითებს არააუცილებელ პარამეტრებზე. რამდენიმე პარამეტრის მითითების შემთხვევაში, ისინი ერთმანეთისაგან მძიმით გამოიყოფა.

write ( ) მეთოდის გარდა შეიძლება გამოყენებულ იყოს მეთოდი **writeln ( )**. იგი დოკუმენტის ყოველი სტრიქონის შემდეგ უმატებს ახალ ხაზზე გადასვლის უჩინარ სიმბოლოს.

ელემენტის ატრიბუტების მნიშვნელობის შეცვლა. **HTML**-დოკუმენტის ელემენტები მოცემულია ტეგების საშუალებით, მათ შორის უმეტესობას აქვს ატრიბუტი (პარამეტრები). დოკუმენტის ობიექტურ მოდელში ელემენტის ტეგებს შეესაბამება ობიექტები, ხოლო ატრიბუტებს – ამ ობიექტის თვისებები. უმეტეს შემთხვევაში ობიექტთა თვისებების დასახელება ემთხვევა ატრიბუტთა დასახელებებს, მაგრამ ამ უკანასკნელიდან განსხვავებით მათი ჩაწერა ხდება ქვედა რეგისტრში. იგივე ეხებათ სტილის ცხრილის პარამეტრებს. მაგრამ ზოგჯერ არსებობს გამონაკლისიც.

მაგალითად, გამოსახულების პოზიციონირებისათვის შეიძლება გამოვიყენოთ შემდეგი **HTML**-კოდი:

```
<IMG ID = "myimg" SRC = "picture . jpg"
STYLE = "position : absolute; top : 20; left : 50; z-index : 3">
```

იმისათვის, რომ სცენარში შევცვალოთ ელემენტის სტილის პარამეტრები, საჭიროა ობიექტ **style**-ის შესაბამის თვისებებს მიენიჭოს ახალი მნიშვნელობები:

```
document . all . myimg . style . top = 30
document . all . myimg . style . top = 100
document . all . myimg . style . zIndex = -2
```

z-index პარამეტრი სტილის ცხრილში ელემენტისათვის მიუთითებს ფენას (ფარდობითი მდებარეობა ზედა-ქვედა), რომელიც ობიექტურ მოდელში წარმოდგენილია **zIndex** თვისებით.

ელემენტთა ცვლილება. **HTML**-დოკუმენტის ელემენტების დინამიკური ცვლილების ყველაზე მოსახერხებელი საშუალება არის **innerText**, **outerText**, **innerHTML** და **outerHTML** თვისებები.

დოკუმენტის ელემენტთა ტეგები შეიძლება შეიცავდეს ტექსტს ან/და სხვა ტეგს. ზემოთ ჩამოთვლილი თვისებების საშუალებით შეიძლება მივიღოთ ელემენტის შიგთავსზე მიმართვა. ამ თვისებათა მნიშვნელობების შეცვლით შეიძლება შეიცვალოს თვით ელემენტი, ნაწილობრივ ან სრულად. მაგალითად, შეიძლება შეიცვალოს მხოლოდ წარწერა ღილაკზე, ან შეიძლება ღილაკი გარდავქმნათ გამოსახულებად ან **Flash**-ანიმაციად.

innerText თვისების მნიშვნელობა არის გაღებულ და დახურულ ტეგებს შორის მოთავსებული ტექსტური მნიშვნელობა. თუ ამ სტრიქონში შედის **HTML**-ტეგი, მაშინ ხდება მათი იგნორირება, ხოლო **outerText** თვისების შემთხვევაში მოიცავს ამ სტრიქონში შემავალ ტეგებსაც.

მაგალითის სახით განვიხილოთ **HTML**-კოდის შემდეგი ფრაგმენტი:

```
<DIV ID = "my">
<A HREE = 'magal1 . htm'>
<IMG SRC = 'picture . jpg'> EXILE<B> CLICK </B>
</A>
</DIV>
```

ამ შემთხვევაში **innerText** და **outerText** თვისებები **<DIV>** კონტეინერული ტეგით მოცემული ელემენტისათვის ერთი და იგივეა:

document . all . my . innerText

თუ ზემოთ მოყვანილი ფრაგმენტისათვის ჩავწერთ

document . all . my . innerText = "<BUTTON> CLICK</BUTTON>"

ის დოკუმენტში არ ქმნის ღილაკს, მხოლოდ გამოიტანს ტექსტის სტრიქონს.

უფრო ეფექტურია **innerHTML** და **outerHTML** თვისებები. ზემოთ მოყვანილი **HTML**-კოდის ფრაგმენტისათვის ადგილი ექნება შემდეგს:

document . all . my . innerHTML

მნიშვნელობა ტოლია:

<A HREE = 'magal1 . htm'>
<IMG SRC = 'picture . jpg'> EXILE<B> CLICK </B>
document . all . my . outerHTML

ამ შემთხვევაში კი მნიშვნელობა ტოლი იქნება მთლიანად ზემოთ მოყვანილი ფრაგმენტისა:

<DIV ID = "my">
<A HREE = 'magal1 . htm'>
<IMG SRC = 'picture . jpg'> EXILE<B> CLICK </B>
</A>
</DIV>

თუ სცენარში შესრულებული იქნება მაგალითად, გამოსახულება:

document . all . my . innerHTML = "<BUTTON>
CLICK</BUTTON>"

დოკუმენტში გამოსახულება და ტექსტი შეიცვლება ღილაკით, წარწერით **CLICK**.

თუ **innerHTML** და **outerHTML** თვისებები გამოყენებული იქნება არაკონტეინერულ ტეგებში, მაშინ მათი შედეგები ერთმანეთს დაემთხვევა.

მაგალითი: **HTML**-დოკუმენტის ელემენტების **innerText**, **outerText**, **innerHTML** და **outerHTML** თვისებების მნიშვნელობების მიღება.

<HTML>
<DIV ID = "my">
<A HREE = 'magal1 . htm'>
<IMG SRC = 'picture . jpg'> EXILE<B> CLICK </B>

```
</A>
</DIV>
<H2 ID = "hello"> Hello <I> reader</I></H2>>
<SCRIPT>
getproperties ("my")
getproperties ("hello")
hello

function getproperties ("xid")  {
var x = eval ("document . all . " + xid)
alert (" innerHTML: " + x . innerHTML + "/n outerHTML: " + x .
      outerHTML + "/n innerText: " + x . innerText + "/n outerText:
      " + x . outerText)
}
</SCRIPT>
</HTML>
```

27. გამოსახულების ჩატვირთვა

Web-გვერდების უმეტესობა შეიცავს გრაფიკულ ელემენტებს, რომლებიც გამოიყენება არა მარტო გვერდების გასაფორმებლად, არამედ ინფორმაციის შევსების მიზნითაც: საქონლის კატალოგების ილუსტრაცია, სქემები, ნახაზები, გეოგრაფიული რუკები, ფოტოგალერეა და სხვა. თუ გრაფიკული ფაილები დიდი და/ან მრავალია, მაშინ ბრაუზერში ასეთი გვერდების ჩატვირთვამ შეიძლება მოითხოვოს ძალიან დიდი დრო. ამიტომ, ხშირად ჯერ ახდენენ მცირე მოცულობის გამოსახულების ჩატვირთვას, ხოლო შემდეგ მათ საფუძველზე გრაფიკულ ფაილებზე მითითებები იქმნება, რომელიც მხოლოდ ამ მითითებაზე მაუსის დაწკაპუნების შემდეგ იტვირთება. აგრეთვე, შეიძლება **<IMG>** ტეგში **SRC** ატრიბუტის გარდა **LOWSRC** ატრიბუტი გამოვიყენოთ, რომელიც საშუალებას მოგვცემს ჯერ ჩავტვირთოთ დაბალი ამოხსნადობის მქონე გამოსახულება, ხოლო შემდეგ შევცვალოთ ის მაღალი ამოხსნადობის მქონე გამოსახულებით.

სცენარის საშუალებით შესაძლებელია მოვახდინოთ გამოსახულების წინასწარი ჩატვირთვის ორგანიზება ბრაუზერის კეშ-

მეხსიერებაში ისე, რომ ის არ აისახოს ეკრანზე. ეს განსაკუთრებით ეფექტურია გვერდის საწყისი ჩატვირთვის დროს. ვიდრე გამოსახულება იტვირთება მეხსიერებაში და არ ჩანს, მომხმარებელს შეუძლია დაათვალიეროს ტექსტური ინფორმაცია, ამასთან მას არ აღიზიანებს გრაფიკული ელემენტის ნელ-ნელა გამოჩენა. შედეგად გვერდის ჩატვირთვა მომხმარებელში არ იწვევს უარყოფით ემოციებს.

გამოსახულების წინასწარი ჩატვირთვის მიზნით ბრაუზერის მეხსიერებაში უნდა შეიქმნას მისი ობიექტი. ამისათვის კი საჭიროა სცენარში შემდეგი სახის გამოსახულების შესრულება :

myimg = new Image (სიგანე, სიმაღლე)

ობიექტის ფუნქცია-კონსტრუქტორის პარამეტრები განსაზღვრავს გამოსახულების ზომებს. ისინი უნდა შეესაბამებოდეს **<IMG>** ტეგის **WIDTH** და **HEIGHT** ატრიბუტების მნიშვნელობებს, რომელიც გამოიყენება წინასწარ ჩატვირთული გამოსახულების ასასახად. მეხსიერებაში გამოსახულების ობიექტის **myimg**-ის შესაქმნელად ბრაუზერს შეიძლება მივაწოდოთ გრაფიკული გამოსახულების სახელი ან **URL**-მისამართი. ეს კეთდება **src** თვისების საშუალებით:

myimg . src = "URL – გამოსახულების მისამართი"

მოცემული გამოსახულება ბრაუზერს უბრძანებს, რომ მითითებული გამოსახულება ჩატვირთოს კეშ-მეხსიერებაში და არ მოხდეს მისი ასახვა ეკრანზე. მათ გამოსაჩენად **<IMG>** ელემენტის **src** თვისებას მიენიჭება კეშ-მეხსიერებაში ასახული ობიექტის იგივე თვისების მნიშვნელობა. მაგალითად:

document . images [0] . src = myimg . src

მაგალითის სახით განვიხილოთ **HTML**-დოკუმენტი, რომელშიც გრაფიკული ელემენტების დასახელებათა სია და ერთი საწყისი გამოსახულებაა ასახული. სიის ელემენტზე მაუსის დაწკაპუნება იწვევს შესაბამისი გამოსახულების ეკრანზე გამონათებას. მოცემული სიის ყველა გრაფიკული ელემენტი წინასწარაა ჩატვირთული კეშ-მეხსიერებაში, ამიტომ სიიდან ამორჩევის შემდეგ სწრაფად მოხდება მათი ასახვა ეკრანზე. სიის შექმნა ხორციელდება **<SELECT>**

კონტეინერული ტეგის საშუალებით, რომელიც **<OPTION>** ტეგსაც შეიცავს.

```
<HTML>
<HEAD>
<SCRIPT>
var imgFile = mew Array ( )
imgFile [0] = "pict1 . jpg"
imgFile [1] = "pict2 . jpg"
imgFile [2] = "pict3 . jpg"
imgFile [3] = "pict4 . jpg"
var imgName = new Array ( )
imgName [0] = "picture1"
imgName [1] = " picture2"
imgName [2] = " picture3"
imgName [3] = " picture4"
var imgObj = new Array ( )
for ( i = 0; i < imgFile . length; i++ )  {
imgObj [i] = new Image (150, 100)
imgObj [i] . src = imgFile [i]
}
function imgshow (list)  {
var x = list . options [ list . selectedIndex] . value
document . all . img0 . src = eval ("imgObj ["+ x + "] . src")
}
var clist = "<SELECT onchange = 'imgshow (this)'>"
for ( i = 0; i < imgFile . length; i++ )  {
clist += "<OPTION VALUE =" + i + ">" + imgName [i]
}
clist += "</SELECT>"
document . writeln (clist)
</SCRIPT>
</HEAD>
<! საწყისი სურათი>
<IMG ID = "img0" SRC = "pict0 . jpg" WIDTH = 150 HEIGHT =
     100>
</HTML>
```

სტრიქონის ობიექტზე მიმართვად გადასაქცევად გამოიყენება ფუნქცია **eval ( )**. **HTML**-კოდი, რომელიც გამოსახულების სიას განსაზღვრავს და სცენარით გენერირდება, გამოიყურება შემდეგნაირად:

```
<SELECT onchange = 'imgshow (this) '>
<OPTION VALUE = 0>picture1
<OPTION VALUE = 0>picture2
<OPTION VALUE = 0>picture3
<OPTION VALUE = 0>picture4
</SELECT>
```

28. პროცესების მართვა დროში

ჩვენ შეგვიძლია პერიოდულად, დროის გარკვეული ინტერვალით, გავუშვათ **JavaScript** კოდი (მაგალითად, რომელიმე ფუნქცია). ამ დროს იქმნება გამოთვლითი პროცესის ერთდროული (პარალელური) შესრულების ეფექტი. მაგალითად, ჩვენ შეგვიძლია შესრულებაზე გავუშვათ რამდენიმე ფუნქცია, რომლებიც ეკრანზე გადაანაცვლებს რამდენიმე ობიექტს. ამ დროს შეიქმნება ობიექტების ერთდროული მოძრაობის ეფექტი. ზოგჯერ საჭიროა, რომელიმე ფუნქციის შესრულების წინ მოხდეს დროებითი შეყოვნება, რათა ადრე დაწყებულმა პროცესმა შეძლოს დასრულება. ზემოაღნიშნული მიეკუთვნება გამოთვლითი პროცესების დროში მართვის ამოცანას.

რომელიმე გამოსახულების ან ფუნქციის პერიოდულად (დროის მუდმივი ინტერვალით) შესრულების ორგანიზაციისათვის გამოიყენება **window** ობიექტის ფუნქცია **setInterval ( )**. ამ მეთოდს აქვს შემდეგი სინტაქსი:

setInterval (გამოსახულება, პერიოდი [, ენა])

პირველი პარამეტრი არის სტრიქონი, რომელიც შეიცავს გამოსახულებას (კერძოდ, ფუნქციის გამოძახება). მეორე პარამეტრი – მთელი რიცხვია, რომელიც მიუთითებს პირველ პარამეტრად მითითებული გამოსახულების შემდგომი შესრულების წინ შეყოვნების დროს, მილიწამებში. მესამე, არააუცილებელი პარამეტრი უჩვენებს ენას, რომელზეც ჩაწერილია გამოსახულება, სტანდარტულად – **JavaScript**.

შედეგად მიიღება რაიმე მთელი რიცხვი – დროის ინტერვალის იდენტიფიკატორი, რომელიც შემდეგში შეიძლება გამოყენებულ იქნეს ამ მეთოდით გაშვებული პროცესის შეწყვეტის მიზნით. ამისათვის, გამოიყენება მეთოდი **clearInterval (იდენტიფიკატორი)**, რომელიც პარამეტრის სახით ღებულობს **setInterval ( )** მეთოდით დაბრუნებულ, მთელ რიცხვით იდენტიფიკატორს.

იმისათვის, რომ გამოსახულება შესრულდეს დროებითი შეყოვნებით, გამოიყენება მეთოდი **setTimeout ( )** ამ მეთოდს აქვს შემდეგი სინტაქსი:

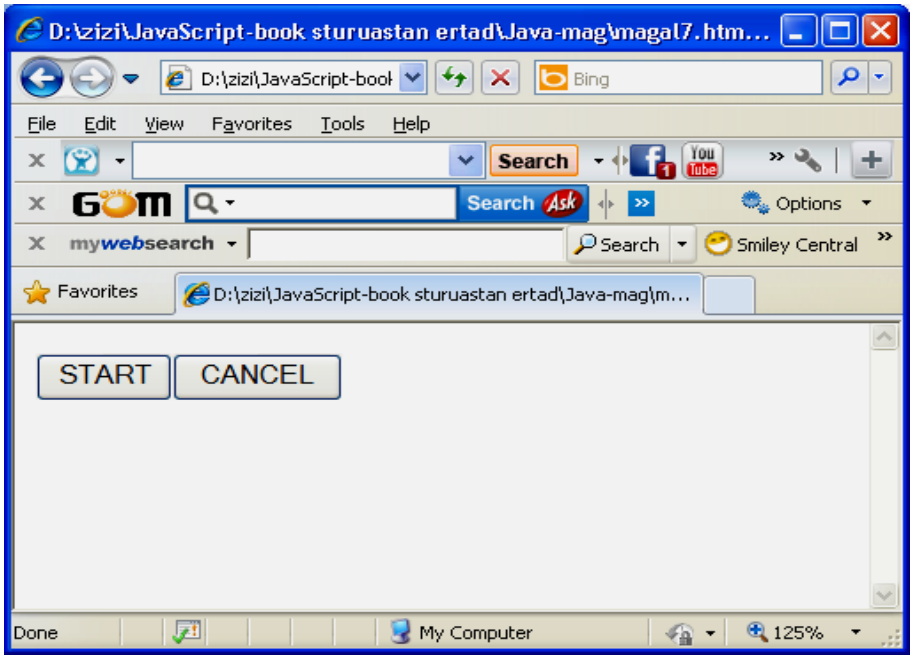
setTimeout (გამოსახულება, პერიოდი [, ენა])

პირველი პარამეტრი არის სტრიქონი, რომელიც შეიცავს გამოსახულებას (კერძოდ, ფუნქციის გამოძახება). მეორე პარამეტრი – მთელი რიცხვია, რომელიც მიუთითებს პირველ პარამეტრად მითითებული გამოსახულების შესრულების წინ შეყოვნების დროს მილიწამებში. მესამე, არააუცილებელი პარამეტრი უჩვენებს ენას, რომელზეც ჩაწერილია გამოსახულება; სტანდარტულად – **JavaScript**. შედეგად მიიღება რაიმე მთელი რიცხვი – დროის ინტერვალის იდენტიფიკატორი, რომელიც შემდეგში შეიძლება გამოყენებულ იქნეს ამ მეთოდით გაშვებული პროცესის შეყოვნების შეწყვეტის მიზნით. ამ მიზნით გამოიყენება მეთოდი **clearTimeout (იდენტიფიკატორი)**, რომელიც პარამეტრის სახით ღებულობს **setTimeout ( )** მეთოდით დაბრუნებულ, მთელ რიცხვით იდენტიფიკატორს.

მაგალითი:

```
<HTML>
<BUTTON ID = "start">START</BUTTON>
<BUTTON ID = "stop">CANCEL</BUTTON>
<SCRIPT>
var myproc
function start . onclick ( ) {
myproc = setTimeout ("window . open ('magal.htm')" , 5000)
}
function stop . onclick ( ) {
clearTimeout (myproc)
}
```

```
</SCRIPT>
</HTML>
```



ზემოთ მოყვანილ მაგალითში დოკუმენტში გამოტანილი იქნება ორი ღილაკი. **START** ღილაკზე მაუსით დაწკაპუნების შემდეგ 5 წამში გაიხსნება ახალი ფანჯარა და მასში ჩაიტვირთება **magal.htm** დოკუმენტი. მაგრამ ამ ფანჯრის გახსნამდე დოკუმენტის ჩატვირთვა შეიძლება შეჩერებულ იქნეს **CANCEL** ღილაკით:

29. Cookie მექანიზმთან მუშაობა

ბრაუზერში მომხმარებლის კომპიუტერის დისკზე მცირე მოცულობის ინფორმაციის შესანახად გათვალისწინებულია ე. წ. **Cookie** მექანიზმი. ჩვეულებრივ იგი გამოიყენება მომხმარებლის სახელისა და პაროლის (რომელიც შეგვყავს დაცული **web**-საიტის ფორმის ველში), აგრეთვე საიტის წინა მონახულების შესახებ ინფორმაციის შესანახად. საიტის ჩატვირთვის დროს ეს თარიღი შედარდება ავტორის მიერ დაყენებულ საიტის განახლების თარიღს. თუ ავტორის თარიღი უფრო "მეტია" გვიანია, ვიდრე პირველი თარიღი, მაშინ **web**-საიტზე გაჩნდება რაიმე აღნიშვნა, მაგალითად, ტექსტზე ან გამოსახულებაზე წარწერა "**new**" ან **"განახლებულია!"**. რა თქმა უნდა ყოველივე ეს შესაბამისი სცენარის საშუალებით სრულდება.

Internet Explorer ბრაუზერის **cookie** ჩანაწერს **Windows\Cookies** საქაღალდეში განთავსებულ ცალკეულ ტექსტურ ფაილებში **Cookie** მექანიზმი ინახავს. ასეთი ფაილის სახელი მომხმარებლისა და ამავე სერვერის დომენის, რომელზეც **Cookie**-ფაილი იქმნება, სახელის საფუძველზე იქმნება. ზოგიერთი ბრაუზერი უბრალოდ **cookie.txt** ფაილს ქმნის. ამ ფაილების გახსნა და მასში ცვლილებების შეტანა არ არის რეკომენდებული.

მონაცემები **Cookie**-ფაილში ორგანიზებულია ჩანაწერების სახით. ყოველი ასეთი ჩანაწერი შეიძლება წარმოვადგინოთ სტრიქონის სახით, რომელიც შემდეგ ელემენტებს შეიცავს:

- ჩანაწერის სახელი;
- ჩანაწერის შინაარსი;
- ჩანაწერის შენახვის ვადა;
- სერვერის დომენი, რომელმაც შექმნა ეს ჩანაწერი;
- ცნობა ჩანაწერებთან წვდომისათვის მიზანდასახული უსაფრთხო **http**-მიერთების აუცილებლობის შესახებ;
- დოკუმენტების განთავსება, რომლის მიხედვითაც ნებადართულია ჩანაწერებზე მიმართვა.

ჩანაწერის დამოკიდებულება დომენზე უზრუნველყოფს ე. წ. არააღდგენადი პაროლების (წყვილი **მომხმარებლის სახელი-პაროლი**) უსაფრთხო შენახვას, ვინაიდან ერთი დომენის სერვერის მიერ შექმნილ ჩანაწერს ვერ წაიკითხავს სხვა დომენით შექმნილი სერვერი.

შენახვის ვადას ბრაუზერი ითვალისწინებს ვადაგასული ფაილების ავტომატურად წასაშლელად, რათა არ მოხდეს **Cookie**-ფაილების მოცულობის გაზრდა. **Cookie**-ფაილების მოცულობა თითოეული დომენისათვის შემოსაზღვრულია 20 ჩანაწერით.

Cookie-ფაილებში **JavaScript**-ის საშუალებით მონაცემების ჩასაწერად გამოიყენება **Cookie**-მონაცემების შემცველი სტრიქონის მინიჭების გამოსახულების **document . cookie** თვისება. ამასთან, საჭიროა დავიცვათ სტრიქონის ფორმატი:

document . cookie = "cookieName = მონაცემები
[; expires = GMT დროის სტრიქონი]
[; path = გზა]

[; domain = დომენი]
[[; secure] "

კვადრატული ფრჩხილები მიუთითებს, რომ მოცემული პუნქტები არ არის აუცილებელი და შეიძლება გამოტოვებული იყოს.

განვიხილოთ **Cookie**-სტრიქონის ელემენტები:

- **cookieName** – **მონაცემები**. ყოველ **Cookie**-ჩანაწერს უნდა ჰქონდეს სახელი და სტრიქონული მნიშვნელობა, რომელიც შეიძლება ცარიელი სტრიქონიც იყოს. მაგალითად, თუ საჭიროა შევინახოთ სიტყვა „**Zura“** **Cookie**-ჩანაწერში **User_Name** სახელით, მაშინ შესაბამის გამოსახულებას ექნება სახე:

document . cookie = " User_Name = Zura "

ამ გამოსახულების შესრულების დროს ბრაუზერი მოცემული სახელით ეძებს ჩანაწერს. თუ ასეთ ჩანაწერს ვერ ნახულობს მოცემულ დომენში, მაშინ მას ქმნის ავტომატურად; ხოლო თუ ასეთი სახელით ჩანაწერი უკვე არსებობს, მაშინ ბრაუზერი მის მონაცემებს ცვლის ახლით. მონაცემები არ უნდა შეიცავდეს წერტილ-მძიმეს, მძიმეს და ჰარს. იმისათვის, რომ ჰარი შეიცვალოს შესაბამისი სიმბოლოთი (**%20**), მონაცემების სტრიქონი წინასწარ უნდა დამუშავდეს **escape ( )** ფუნქციით.

- **GMT** დროის სტრიქონი. **Cookie**-ჩანაწერის შენახვის თარიღი და დრო უნდა იყოს სტრიქონული ჩანაწერი გრინვიჩის დროით (**GMT**). თუ ეს დრო მითითებული არ არის, მაშინ ბრაუზერი თვლის რომ მოცემული **Cookie**-ჩანაწერი არის დროებითი და ფაილში აღარ ჩაიწერება.

- **path** – **Cookie**-ჩანაწერები, რომელიც იქმნება მომხმარებლის კომპიუტერის მიერ, აქვთ გაჩუმების პრინციპით მიღებული გზა. მაგრამ შეიძლება შეიქმნას **Cookie**-ს ასლი სხვა საქაღალდეში, რომლის გზაც მიეთითება ამ პარამეტრის სახით.

- **domain** – **Cookie**-მონაცემების გარკვეულ დოკუმენტთან ან დოკუმენტთა ჯგუფთან სინქრონიზაციისათვის ბრაუზერი არჩევს მიმდინარე დოკუმენტის დომენს და მას ათავსებს **Cookie**-ჩანაწერში, რომელიც შეესაბამება ამ დომენს. დომენის წარმოდგენის ფორმატი უნდა შეიცავდეს სულ მცირე ორ დონეს მაინც, მაგალითად, **google.ge**.

secure – ღებულობს ლოგიკურ მნიშვნელობას (**true** ან **false**). მომხმარებლის მხარეს (მომხმარებლის კომპიუტერში) **Cookie**-ჩანაწერების შექმნის მომენტში ეს პარამეტრი უნდა გამოვტოვოთ.

სცენარით მიღებული **Cookie**-მონაცემები არის ერთადერთი სტრიქონი – **document.cookie** - თვისების მნიშვნელობა. ცალკეული ელემენტების (პარამეტრების) მნიშვნელობების ამორჩევა ხდება ამ სტრიქონის მნიშვნელობის ანალიზის საფუძველზე ობიექტ **String**-ის მეთოდებით. გარდა ამისა, თუ ორი ან მეტი (20-მდე) ჩანაწერი შეესაბამება ერთი და იგივე დომენს, მაშინ ყველა ისინი მაინც არის ერთი სტრიქონია და ერთმანეთისაგან გამოიყოფა წერტილ-მძიმით და ჰარით.

30. ბრაუზერისა და დოკუმენტის ობიექტური მოდელი

ობიექტი Window

HTML-დოკუმენტში **<FRAMESET>** კონტეინერული ტეგის საშუალებით მოცემული ობიექტი **window** შეიცავს ყველა **frames** ჩარჩოს კოლექციას. ობიექტ **window**-ს აქვს თვისებები, მეთოდები, ხდომილობები და აგრეთვე, შვილობილი ობიექტები. ქვემოთ მოვიყვანოთ მათი სრული ჩამონათვალი და განვიხილოთ მხოლოდ მათ შორის პრაქტიკული თვალსაზრისით ყველაზე უფრო საჭირო.

თვისებები window.

- **parent** – გვაბრუნებს მიმდინარე ფანჯრიდან მშობლიურ ფანჯარაში;
- **self** – აბრუნებს მითითებას მიმდინარე ფანჯარაზე;
- **top** – აბრუნებს მითითებას მთავარ ფანჯარაზე;
- **name** – ფანჯრის დასახელება;
- **opener** – მიმდინარე ფანჯრით შექმნილი ფანჯარა;
- **closed** – იძლევა შეტყობინებას, თუ ფანჯარა დაკეტილია;
- **status** – ბრაუზერის მდგომარეობის სტრიქონში ნაჩვენებელი ტექსტი;
- **defaultStatus** – ბრაუზერის მდგომარეობის სტრიქონში გაჩუმების პრინციპით საჩვენებელი ტექსტი;

- **returnValue** – საშუალებას გვაძლევს განვსაზღვროთ ხდომილობისა და დიალოგური ფანჯრის დასაბრუნებელი მნიშვნელობები;

- **client** – მითითება, რომელიც აბრუნებს ბრაუზერის ნავიგატორის ობიექტს;

- **document** – **document** ფანჯრის ობიექტზე მითითება მხოლოდ წაკითხვისათვის;

- **event** – **event** გლობალურ ობიექტზე მითითება მხოლოდ წაკითხვისათვის;

- **history** – **history** ფანჯრის ობიექტზე მითითება მხოლოდ წაკითხვისათვის;

- **location** – **location** ფანჯრის ობიექტზე მითითება მხოლოდ წაკითხვისათვის;

- **navigator** – **navigator** ფანჯრის ობიექტზე მითითება მხოლოდ წაკითხვისათვის;

- **screen** – **screen** გლობალურ ობიექტზე მითითება მხოლოდ წაკითხვისათვის;

თვისება **parent** საშუალებას იძლევა მივმართოთ იერარქიულად ერთი საფეხურით მაღლა მდგომ ობიექტს. თუ გვსურს ორი საფეხურით მაღლა მდგომ ობიექტს მივმართოთ, უნდა გამოვიყენოთ **parent**. **parent** და ა. შ.

ყველაზე მაღლა მდგომ ობიექტზე მიმართვისათვის გამოიყენება თვისება **top**. ოღონდ **top**-ს არ შეუძლია მიმართოს მთავარ ჩარჩოს რამდენიმე ჩარჩოდ დაყოფის შემთხვევაში. თვისება **name** შეესაბამება **<FRAMESET>** ტეგის მიერ შექმნილ ჩარჩოს სახელს.

მეთოდები window.

- **open ( )** – ბრაუზერის ახალი ფანჯრის გახსნა;

- **close ( )** – ხურავს ბრაუზერის მიმდინარე ფანჯარას;

- **showHelp ( )** – დახმარების ფანჯარას აჩვენებს როგორც დიალოგურს;

- **showModalDialog ( )** – ახალ ფანჯარას აჩვენებს როგორც დიალოგურს (მოდალურს);

- **alert ( )** – აჩვენებს გაფრთხილების ფანჯარას შეტყობინებითა და **OK** ღილაკით;

- **prompt ( )** – აჩვენებს მოწვევის ფანჯარას შეტყობინებით, ტექსტური ველითა და **OK** და **Cancel** ღილაკებით;

- **confirm ( )** - აჩვენებს დამოწმების ფანჯარას შეტყობინებით და **OK** და **Cancel** ღილაკებით;

- **navigate ( )** – მითითებული მისამართით ჩაიტვირთება სხვა გვერდი;

- **blur ( )** – აალაგებს ფოკუსს მიმდინარე გვერდიდან; მისი შესაბამისი ხდომილობაა – **onblur**;

- **focus ( )** – დააყენებს გვერდს ფოკუსში; მისი შესაბამისი ხდომილობაა – **onfocus**;

- **scroll ( )** – ხსნის ფანჯარას მოცემულ სიგანესა და სიმაღლეზე;

- **setInterval ( )** – პროცედურას მიუთითებს შესრულდეს პერიოდულად წინასწარ მოცემული მილიწამების ინტერვალით;

- **setTimeout ( )** – უშვებს პროგრამას, გვერდის ჩატვირთვის შემდეგ წინასწარ მოცემული მილიწამების ინტერვალით;

- **clearInterval ( )** – ანულებს **setInterval ( )** მეთოდით ჩართულ ტაიმერს;

- **clearTimeout ( )** – ანულებს **setTimeout ( )** მეთოდით ჩართულ ტაიმერს;

- **execScript ( )** – ასრულებს სცენარის კოდს; გაჩუმების პრინციპით არის **JavaScript**.

ხდომილობა window.

- **onblur** – ფანჯრის გამოსვლა ფოკუსიდან;
- **onfocus** – ფანჯარა ხდება აქტიური;
- **onhelp** – მომხმარებლის მიერ **<F1>** კლავიშზე ხელის დაჭერა;
- **onresize** – მომხმარებლის მიერ ფანჯრის ზომების შეცვლა;
- **onscrokk** – მომხმარებლის მიერ ფანჯრის გადაფურცვლა;
- **onerror** – შეცდომა გადაცემის დროს;
- **onbeforeunload** – გვერდის მიერ ფანჯრის დატოვების შემთხვევაში, რათა მოხდეს მონაცემების შენარჩუნება;

- **onload** – გვერდი სრულად არის ჩატვირთული;

- **onunload** – უშუალოდ გვერდის მიერ ფანჯრის დატოვების შემთხვევაში.

ზემოთ ჩამოთვლილი ხდომილობიდან სამი სრულდება მომხმარებლის მოქმედების შედეგად. თუ გახსნილია ბრაუზერის რამდენიმე ფანჯარა, მომხმარებელს შეუძლია გააქტიუროს ნებისმიერი მათგანი ფოკუსის გადატანით ერთი ფანჯრიდან მეორეზე. ეს მოქმედებები ხორციელდება **onblur** და **onfocus** ხდომილობებით. იგივე მოქმედება შეიძლება გამოწვეულ იქნეს პროგრამულადაც **blur** და **focus** მეთოდების გამოყენებით. გვერდის ან მისი ელემენტის ჩატვირთვის დროს თუ ხდება შეცდომა, მაშინ ხდება **onerror** ხდომილობის ინიცირება. ჩვენ შეიძლება გამოვიყენოთ ეს ხდომილობა პროგრამაში რათა, მაგალითად, ვცადოთ კიდევ ერთხელ ჩავტვირთოთ გვერდი ან რაიმენაირად შევცვალოთ შემდგომი ქმედებები.

onload ხდომილობა ხდება მაშინ, როდესაც გვერდი მთლიანად ჩატვირთულია ფანჯარაში; ხდომილობა **onbeforeunload** – ვიდრე გვერდი დატოვებს ფანჯარას; ხდომილობა **onunload** – როდესაც გვერდმა დატოვა ფანჯარა, ახალი გვერდის ჩატვირთვის ან ბრაუზერის დახურვის წინ.

ობიექტ **window**-ს აქვს რამდენიმე შვილობილი ობიექტი, რომელთა გამოყენება შესაძლებელია მისი საშუალებით: **document**, **history**, **navigator**, **location**, **event** და **screen**.

31. ობიექტი document

ობიექტი **document** ობიექტური მოდელის იერარქიაში არის ცენტრალური ობიექტი და კოლექციებისა და თვისებების საშუალებით **HTML**-დოკუმენტზე მთელ ინფორმაციას მოიცავს. ის აგრეთვე გვაწვდის მრავალ მეთოდსა და ხდომილობას დოკუმენტთან სამუშაოდ. ვინაიდან ჩვენ უკვე განვიხილეთ ამ ობიექტებზე მიმართვის რამდენიმე ძირითადი ხერხი, აქ შემოვიფარგლებით მხოლოდ საცნობარო ინფორმაციით.

თვისება document.

თვისება	ატრიბუტი	დანიშნულება
activeElement		ახდენს აქტიური ელემენტის იდენტიფიკაციას
alinkColor	**ALINK**	გვერდზე აქტიური მითითების ფერი
bgColor	**BGCOLOR**	განსაზღვრავს ელემენტის ფონის ფერს
body		ტეგ <**BODY**>-ში განსაზღვრული დოკუმენტის არაცხად ძირითად ობიექტზე მითითება მხოლოდ წაკითხვისათვის
cookie		**cookie**-ჩანაწერის სტრიქონი. ამ თვისებისათვის ახალი მნიშვნელობის მინიჭებას მივყავართ, ბრაუზერის დახურვის შემდეგ, დისკზე **cookie**-ს ჩაწერამდე
domain		აყენებს ან აბრუნებს დოკუმენტის დომენს მისი დაცვისა ან იდენტიფიკაციისათვის
fgColor	**TEXT**	წინა პლანის ტექსტის ფერის დაყენება
lastModified		გვერდის ბოლო ცვლილების თარიღი, მისაწვდომია როგორც სტრიქონი
linkColor	**LINK**	გვერდზე არსებული ჯერ კიდევ მოუნახულებელი ჰიპერმიმართვების ფერი
location		დოკუმენტის სრული **URL**
parentWindow		აბრუნებს დოკუმენტის მშობლიურ ფანჯარას
readyState		განსაზღვრავს ჩასატვირთი ობიექტის მიმდინარე მდგომარეობას
referer		**URL** გვერდი, რომელმაც გამოიძახა მიმდინარე გვერდი
selection		**document**-ისათვის შვილობილი ობიექტი **selection**-ზე მითითება მხოლოდ წაკითხვისათვის
title	**TITLE**	ელემენტისათვის განსაზღვრავს იმ საცნობარო ინფორმაციას, რომელიც გამოიყენება ჩატვირთვის დროს ან ამოცურებად კარნახში
url	**URL**	კლიენტის დოკუმენტის ან <**META**> ტეგში **URL**-მისამართი
vlinkColor	**VLINK**	გვერდზე არსებული მონახულებული მითითებების ფერი

კოლექციები document.

- **all** – დოკუმენტის ძირითად ნაწილში ყველა ტეგებისა და ელემენტების კოლექცია;

- **anchors** – დოკუმენტში არსებული ყველა სანიშნობელის კოლექცია;

- **applets** – დოკუმენტში არსებული ყველა ობიექტის კოლექცია, მათ შორის მართვის ჩაშენებული ელემენტები, გრაფიკული ელემენტები, აპლეტები და სხვა ობიექტები;

- **embeds** – დოკუმენტში არსებული ყველა დანერგილი ობიექტის კოლექცია;

- **forms** – გვერდზე არსებული ყველა ფორმის კოლექცია;

- **frames** – **<FRAMESET>** ტეგში განსაზღვრული ყველა ჩარჩოს კოლექცია;

- **images** – გვერდზე გამოსახული ყველა გრაფიკული ელემენტის კოლექცია;

- **links** – გვერდზე არსებული ყველა მითითებისა და **<AREA>** ბლოკის კოლექცია;

- **plugins** – კიდევ ერთი დასახელება დოკუმენტში არსებული ყველა დანერგილი ობიექტის კოლექციისათვის;

- **scripts** – გვერდზე არსებული ყველა **<SCRIPT>** განყოფილების კოლექცია;

- **styleSheets** – დოკუმენტში განსაზღვრული სტილის ყველა კონკრეტული თვისების კოლექცია.

მეთოდები document.

- **clear** – ასუფთავებს გამოყოფილ მონაკვეთს;

- **close** – ხურავს ბრაუზერის მიმდინარე ფანჯარას;

- **createElement** – გამოყოფილი ტეგისათვის ქმნის ელემენტის ეგზემპლარს;

- **elementFromPoint** – აბრუნებს ელემენტს მოცემული კოორდინატებით;

- **execCommand** – ასრულებს ბრძანებას (ოპერაციას) გამოყოფილ ობიექტზე ან არეზე;

- **open** – ხსნის დოკუმენტს, როგორც ნაკადს **write** და **writeln** მეთოდების გამოყენებით შედეგების დასამუშავებლად;

- **queryCommandEnabled** – იძლევა შეტყობინებას დაშვებულია თუ არა მოცემული ბრძანება;

- **queryCommandIndeterm** – იძლევა შეტყობინებას, თუ მოცემულ ბრძანებას აქვს განუსაზღვრელი სტატუსი;

- **queryCommandState** – აბრუნებს ბრძანების მიმდინარე მდგომარეობას;

- **queryCommandSupported** – იძლევა შეტყობინებას მხარდაჭერილია თუ არა მოცემული ბრძანება;

- **queryCommandText** – აბრუნებს სტრიქონს, რომელთანაც მუშაობს ბრძანება;

- **queryCommandValue** – აბრუნებს დოკუმენტის ან ობიექტ **TextRange**-ისათვის განსაზღვრული ბრძანების მნიშვნელობას;

- **write** – მითითებულ ფანჯარაში არსებულ დოკუმენტში ჩაწერს ტექსტს და **HTML** კოდს;

- **writeln** – ჩაწერს ტექსტს და **HTML** კოდს, რომელიც მთავრდება ახალ ხაზზე გადასვლით.

ხდომილობა document.

- **onafterupdate** – წარმოიშვება მონაცემთა გადაცემის დამთავრების დროს;

- **onbeforeupdate** – წარმოიშვება გვერდის მიერ ფანჯრის დატოვების შემთხვევაში;

- **onclick** – ხდება მაუსის მარცხენა კლავიშზე დაწკაპუნების შემთხვევაში;

- **ondblclick** – ხდება მაუსის მარცხენა კლავიშზე ორჯერ დაწკაპუნების შემთხვევაში;

- **ondragstart** – ხდება, როდესაც მომხმარებელი იწყებს მაუსით ”გადათრევას”;

- **onerror** – შეცდომა გადაცემის დროს;

- **onhelp** – მომხმარებლის მიერ **<F1>** კლავიშზე ხელის დაჭერა;

- **onkeydown** – წარმოიშვება კლავიშზე ხელის დაჭერის შემთხვევაში;

- **onkeypress** – წარმოიშვება კლავიშზე ხელის დაჭერისა და მასზე ხელის დაჭერის მდგომარეობის შენარჩუნების შემთხვევაში;

- **onkeyup** – წარმოიშვება, როდესაც მომხმარებელი ხელს აიღებს კლავიშიდან;

- **onload** – წარმოიშვება დოკუმენტის მთლიანად ჩატვირთვის შემთხვევაში;

- **onmousedown** – ხდება მაუსის კლავიშზე ხელის დაჭერის შემთხვევაში;

- **onmousemove** – ხდება მაუსის მაჩვენებლის გადაადგილების შემთხვევაში;

- **onmouseout** – ხდება, როდესაც მაუსის მაჩვენებლი გადის ელემენტის საზღვრებს გარეთ;

- **onmouseover** – ხდება, როდესაც მაუსის მაჩვენებლი შედის ელემენტის საზღვრებში;

- **onmouseup** – ხდება, როდესაც მომხმარებელი ხელს აუშვებს მაუსის კლავიშს;

- **onresdystatechange** – წარმოიშვება **readystate** თვისების შეცვლის შემთხვევაში;

- **onselectstart** – ხდება, როდესაც მომხმარებელი პირველად უშვებს დოკუმენტის გამოყოფილ ნაწილს.

32. ობიექტი location

ობიექტი **location** შეიცავს ინფორმაციას მიმდინარე გვერდის **URL**-მისამართის (და მისი კომპონენტების) შესახებ, აგრეთვე იმ მეთოდებს, რომლებიც გვერდების განახლების საშუალებას იძლევა.

თვისებები location.

- **href** – სრული **URL**-მისამართი სტრიქონის სახით;

- **hash** – სტრიქონი, რომელიც **URL**-ში მოსდევს # სიმბოლოს, რომლის საშუალებითაც მიეთითება თუ რომელ ანკერთან უნდა გადავინაცვლოთ დოკუმენტის ჩატვირთვის დროს;

- **host** – **URL**-ის ‘‘ჰოსტ:პორტი’’ ნაწილი; პორტის მნიშვნელობა მხოლოდ მაშინ ინახება, როდესაც ის ცხადად არის მითითებული **URL**-მისამართში;

- **hostname** – **URL**-ის ‘‘ჰოსტ’’ ნაწილი;

- **pathname** – გზა ობიექტამდე ან ფაილამდე, რომელიც მდებარეობს მესამე ‘‘/’’ სიმბოლოს შემდეგ;

- **port** – **URL** პორტის ნომერი;

- **protocol** – პროტოკოლის განმსაზღვრელი საწყისი ნაწილი, რომლის შემდეგაც ორი წერტილი დგას, მაგალითად ‘‘**http:**’’;

- **search** – შეკითხვის სტრიქონი ან ‘‘**?**’’ სიმბოლოს შემდეგ **URL** მონაცემები;

მაგალითად, თუ ჩვენ ჩავტვირთეთ გვერდი **http://www.imedi.ge**, მაშინ **location . href**-ის მნიშვნელობა იქნება ამ გვერდის მისამართი.

თუ **href** თვისებას მივანიჭებთ ახალ მნიშვნელობას, მაშინ ჩვენ შეგვიძლია შევცვალოთ ბრაუზერში ნაჩვენები გვერდი, მაგალითად:

```
window . location . href = "http://www.google.ge"
```

მეთოდები location.

- **assign ( )** – ჩატვირთავს სხვა გვერდს; ეს მეთოდი **window.location. href** თვისების ეკვივალენტურია;

- **reload ( )** – განაახლებს მიმდინარე გვერდს;

- **replace ( )** – ჩატვირთავს გვერდს **URL**-მისამართით მითითებულ პარამეტრში და ცვლის მიმდინარე გვერდის **URL**-მისამართს (**location . href**).

ზოგიერთი **Web**-საიტი შეიძლება შეიცავდეს ისეთ გვერდებს, რომლებიც მომხმარებლების სანახავად და ნანახი გვერდების სიაში შესატანად არ არის განკუთვნილი. მაგალითად, საიტებზე გადანაცვლებამ შეიძლება მომხმარებელი მიიყვანოს რომელიმე საშუალედო გვერდზე, რომელიც მას შემდგომში მეტი აღარ დასჭირდება. ამასთან, სასურველია რომ მომხმარებელმა ვეღარ შეძლოს დაუბრუნდეს მას **Back** ღილაკის საშუალებითაც. მითითებულ გვერდზე გადასასვლელად იყენებენ მეთოდს **location.replace (URL-მისამართი)**, ამასთან ამ გვერდს პროგრამა სიაში არ შეიტანს, საიდანაც შეიძლება მასზე გადასვლა **Back** ღილაკის საშუალებით.

33. ობიექტი history

ობიექტი **history** შეიცავს ინფორმაციას გვერდების მისამართების შესახებ, რომელიც ბრაუზერმა მოინახულა მიმდინარე სეანსის დროს. ჩვენ შეგვიძლია ამ სიაში გადავინაცვლოთ სცენარის მიხედვით და ჩავტვირთოთ შესაბამისი გვერდები. ობიექტ **history**-ს აქვს მხოლოდ ერთი თვისება და სამი მეთოდი.

თვისება history.

length – მონახულებული გვერდების სიაში ელემენტების რაოდენობა.

ეს თვისება სცენარში კონტროლისათვის გამოიყენება, რათა არ მოხდეს სიის გარეთ გასვლა.

მეთოდები history.

- **back ( )** – სიიდან ჩატვირთავს წინა გვერდს;
- **forvard ( )** – სიიდან ჩატვირთავს მომდევნო გვერდს;
- **go ( )** – სიიდან ჩატვირთავს გვერდს ნომრით **n** (**0**-დან **history . length -1**-მდე) ან გვერდს მითითებული **URL**-მისამართით.

მაგალითი:

ჯერ იტვირთება პირველი გვერდი, ხოლო შემდეგ მეხუთე გვერდი, ამასთან მოწმდება იგი არსებობს თუ არა მონახულებული გვერდების სიაში:

```
window . history . go (1)
if (window . history . length > 4)
        window . history . go (5)
```

ცხადია, რომ მეთოდი **history.go (-1)**, **history.back ( )** მეთოდის ეკვივალენტურია, ხოლო მეთოდი **history.go (1)**, **history.forvard ( )**-ის.

34. ობიექტი navigator

ობიექტი **navigator** ინფორმაციას შეიცავს ბრაუზერის მწარმოებელის, მისი ვერსიისა და შესაძლებლობების შესახებ.

თვისებები navigator.

- **appCodeName** – ბრაუზერის კოდის დასახელება; მაგალითად, "**Mozilla**";

- **appName** – ბრაუზერის დასახელება; მაგალითად, "**Microsoft Internet Explorer**";

- **appVersion** – ბრაუზერის ვერსია;

- **cookieEnabled** –ბრაუზერში განსაზღვრავს **cookies** გამოყენების შესაძლებლობას კლიენტის მხრიდან; ღებულობს ლოგიკურ მნიშვნელობას;

- **userAgent** – ბრაუზერის დასახელება, რომელიც იგზავნება **http**-პროტოკოლის დახმარებით.

კოლექცია navigator.

- **mimeTypes** – ყველა ტიპის დოკუმენტებისა და ფაილების კოლექცია, რომელიც მხარდაჭერილია ბრაუზერის მიერ;

- **plugins** – გვერდზე ყველა დანერგილი ობიექტების კოლექცია.

მეთოდები navigator.

- **taintEnabled** – აბრუნებს **false** მნიშვნელობას, ჩართულია **Netscape Navigator**-თან თავსებადობისათვის;

- **javaEnabled** – იძლევა შეტყობინებას შესაძლებელია თუ არა მოცემულ ბრაუზერში **JavaScript** ენაზე დაწერილი სცენარის კოდის გაშვება; ღებულობს ლოგიკურ მნიშვნელობას.

35. ობიექტი event

ობიექტი **event** საშუალებას იძლევა მივიღოთ ინფორმაცია ბრაუზერში მომხდარი ხდომილობის შესახებ. ეს ინფორმაცია მოცემულია შემდეგ თვისებებში:

- **altKey** – შედეგად აბრუნებს <**Alt**> კლავიშის მდგომარეობას, როდესაც ხდომილობა ხდება;

- **button** – მაუსის კლავიში, რომელმაც გამოიწვია ხდომილობა;

- **cancelBubble** – ხდება მოცემული ხდომილობის აკრძალვა ობიექტური იერარქიის ზედა მიმართულებით;

- **clientX** – შედეგად აბრუნებს ელემენტის **x** კოორდინატას, გამოირიცხება ჩარჩოში ჩასმა, შეჭრები, გადაფურცვლის ზოლები და ა. შ.;

- **clientY** – შედეგად აბრუნებს ელემენტის **y** კოორდინატას, გამოირიცხება ჩარჩოში ჩასმა, შეჭრები, გადაფურცვლის ზოლები და ა. შ.;

- **CtrlKey** – <**Ctrl**> კლავიშის მდგომარეობა, როდესაც ხდება ხდომილობა;

- **fromElement** – შედეგად აბრუნებს ელემენტს, რომლიდანაც **onmouseover** და **onmouseout** ხდომილობისათვის მაუსის კურსორმა გადაინაცვლა;

- **keyCode** – დაჭერილი კლავიშის **ASCII** კოდი; შესაძლებელია ობიექტზე გადასაცემი მნიშვნელობის შეცვლა;

- **offsetX** – შედეგად აბრუნებს მაუსის მაჩვენებლის **x** კოორდინატას პიქსელებში მისი შემცველი ელემენტის მიმართ ხდომილობის წარმოქმნის დროს;

- **offsetY** – შედეგად აბრუნებს მაუსის მაჩვენებლის **y** კოორდინატას პიქსელებში მისი შემცველი ელემენტის მიმართ ხდომილობის წარმოქმნის დროს;

- **reason** – მიუთითებს, რომ მონაცემთა გადანაცვლება წარმატებით განხორციელდა ან რის გამო მოხდა მისი შეწყვეტა;

- **returnValue** – ხდომილობისათვის განსაზღვრავს დასაბრუნებელ მნიშვნელობას;

- **screenX** – შედეგად აბრუნებს მაუსის მაჩვენებლის ჰორიზონტალურ კოორდინატას ეკრანის მიმართ, როდესაც ხდომილობა ხდება;

- **screenY** – შედეგად აბრუნებს მაუსის მაჩვენებლის ვერტიკალურ კოორდინატას ეკრანის მიმართ, როდესაც ხდომილობა ხდება;

- **shiftKey** – განსაზღვრავს <**Shift**> კლავიშის მდგომარეობას, ხდომილობის წარმოშობის დროს;

- **srcElement** – შედეგად აბრუნებს ელემენტს, რომლიდანაც ხდომილობის გასვლა დაიწყო;

- **srcFilter** – შედეგად აბრუნებს ფილტრს, რომელმაც **onfilterchange** ხდომილობა გამოიწვია;

- **toElement** – შედეგად აბრუნებს ელემენტს, რომელზეც დადგა მაუსის მაჩვენებელი **onmouseover** და **onmouseout** ხდომილობის წარმოშობის დროს;

- **type** – შედეგად აბრუნებს ხდომილობის დასახელებას **on** წინსართის გარეშე სტრიქონის სახით;

- **x** – შედეგად მაუსის მაჩვენებლის **x** კოორდინატას შესაბამისად აბრუნებს მშობლიური ელემენტის ან ფანჯრის მიმართ;

- **y** – შედეგად მაუსის მაჩვენებლის **y** კოორდინატას შესაბამისად აბრუნებს მშობლიური ელემენტის ან ფანჯრის მიმართ;

event ობიექტის თვისებები ხდომილობის გასვლის მომენტში წარმოიშვება და მათი უმეტესობა განკუთვნილია მხოლოდ წაკითხვი-სათვის (არ შეიძლება მათი შეცვლა). თუმცა არის ორი თვისება **keyCode** და **returnValue**, რომელთა შეცვლაც შესაძლებელია. ჩვენ ადრე უკვე განვიხილეთ **event** ობიექტის გამოყენება.

36. ობიექტი screen

ობიექტი **screen** შეიცავს ინფორმაციას მომხმარებლის ეკრანის შესაძლებლობების შესახებ და შეიძლება გამოვიყენოთ, მაგალითად, შესაქმნელი ფანჯრის ზომების განსაზღვრისათვის და გადასაცემი გრაფიკის ამოხსნადობის ასარჩევად (აზრი არა აქვს ჩაიტვირთოს 32 ბიტიანი გამოსახულება, თუ მომხმარებლის მონიტორი და ვიდეოადაპტერი აღიქვავს მხოლოდ 256 ფერს). ობიექტ **screen**-ს აქვს შემდეგი თვისებები:

- **width** – შედეგად აბრუნებს მომხმარებლის ეკრანის სიგანეს (პიქსელებში);

- **height** – შედეგად აბრუნებს მომხმარებლის ეკრანის სიმაღლეს (პიქსელებში);

- **bufferDepth** – განსაზღვრავს, გამოიყენება თუ არა ბუფერი ‘‘მეორე ეკრანის’’ შესანახად;

- **colorDepth** – შედეგად აბრუნებს ინფორმაციას, რომელიც საშუალებას იძლევა გადავწყვიტოთ როგორ გამოვიყენოთ ფერები ეკრანზე; მომხმარებლის მოწყობილობების ან ვიდეობუფერის ერთ პიქსელზე ბიტების რაოდენობა;

- **updateInterval** – შედეგად მომხმარებლის ეკრანის განახლებებს შორის აბრუნებს დროის ინტერვალს ან აწვდის მას.

37. ობიექტი TextRange

ობიექტი **TextRange** (ტექსტური არე) ასახავს ტექსტის ნაკადის განყოფილებას, რომელმაც მოახდინა **HTML**-დოკუმენტის ფორმირება. შეიძლება გამოყენებულ იქნეს გვერდის შიგნით ტექსტის სამართავად.

თვისებები TextRange.

- **htmlText** – შედეგად აბრუნებს **TextRange**-ის შიგთავსს როგორც ტექსტს და **HTML** კოდს;

- **text** – უბრალო ტექსტი, რომელიც არის ელემენტ **TextRange**-ს ან **<OPTION>** ტეგის შიგნით.

მეთოდები TextRange.

- **collapse** – მიმდინარე არის ერთ წერტილში (დასაწყისში ან ბოლოში) თავს მოუყრის ტექსტურ არეს;

- **compareEndPoints** – ერთმანეთს ადარებს ორ ტექსტურ არეს და შედეგად აბრუნებს მნიშვნელობის მაჩვენებელ შედეგს;

- **duplicate** – შედეგად აბრუნებს **TextRange** არის ასლსს;

- **execCommand** – ასრულებს ბრძანებას (ოპერაციას) გამოყოფილზე ან არეზე;

- **expand** – აფართოებს ტექსტურ არეს მასში ახალი ნიშნაკების, სიტყვის, წინადადების დამატების გზით ან მიუთითებს რომელი არასრული ბლოკი შედის მასში სრულად;

- **findText** – განსაზღვრავს იმ ტექსტურ არეს, რომელიც მხოლოდ საძიებო ტექსტს შეიცავს;

- **getBookmark** – შედეგად აბრუნებს მნიშვნელობას, რომელიც შემდგომში დოკუმენტში მოცემული პოზიციის იდენტიფიცირებას მოახდენს;

- **inRange** – განსაზღვრავს მოცემული ტექსტური არე არის თუ არა მიმდინარე არის შიგნით;

- **isEqual** – განსაზღვრავს ტოლია თუ არა მოცემული და მიმდინარე ტექსტური არე;

- **move** – ცვლის ტექსტური არის საწყის და საბოლოო წერტილებს მასში სხვადასხვა ტექსტის ჩართვისათვის;

- **moveEnd** – აიძულებს ტექსტურ არეს შევიწროვდეს ან გაფართოვდეს მოცემულ საბოლოო წერტილამდე;

- **moveStart** – აიძულებს ტექსტურ არეს შევიწროვდეს ან გაფართოვდეს მოცემულ საწყის წერტილამდე;

- **moveToBookmark** – გადაწევს ტექსტური არის საზღვრებს მასში სხვა, ადრე **getBookmark**-ის საშუალებით განსაზღვრული არის ჩასართავად;

- **moveToElementText** – გადაწევს ტექსტური არის საზღვრებს მოცემულ ელემენტში ტექსტის ჩასართავად;

- **moveToPoint** – გადაწევს ტექსტური არის საზღვრებს და განალაგებს მას არჩეული წერტილის გარშემო;

- **parentElement** – შედეგად აბრუნებს ელემენტს, რომელიც არის ტექსტურ არეში შემავალი ყველა ელემენტის შვილობილი ელემენტი;

- **pasteHTML** – სვამს ტექსტს და/ან **HTML**-კოდს მიმდინარე ტექსტურ არეში;

- **queryCommandEnabled** – შეტყობინებას იძლევა მიღწევადია თუ არა მოცემული ბრძანება;

- **queryCommandIndeterm** – შეტყობინებას იძლევა, თუ მოცემულ ბრძანებას აქვს თუ არა განუსაზღვრელი სტატუსი;

- **queryCommandState** – შედეგად აბრუნებს ბრძანების მიმდინარე მდგომარეობას;

- **queryCommandSupported** – შეტყობინებას იძლევა, მხარდაჭერილია თუ არა მოცემული ბრძანება;

- **queryCommandText** – შედეგად აბრუნებს სტრიქონს, რომელთანაც ბრძანება მუშაობს;

- **queryCommandValue** – შედეგად ბრძანების მნიშვნელობას აბრუნებს, რომელიც განსაზღვრულია დოკუმენტისათვის ან ობიექტ **TextRange**-სათვის;

- **scrollIntoView** – მიმდინარე ტექსტურ არეს ბრაუზერის ფანჯრის ხილულ ნაწილში გადაიტანს;

- **select** – გააქტიურებს მიმდინარე ტექსტური არის ტოლ გვერდზე გამოყოფილ გამონათებულ მონაკვეთს;

- **setEndPoint** – მიმდინარე ტექსტური არის საწყის ან საბოლოო წერტილს გადაიტანს მოცემული არის დასაწყისში ან ბოლოში.

38. მარტივი ვიზუალური ეფექტები

გამოსახულების შეცვლა. Web-დიზაინში ხშირად წარმოიშვება ერთი გამოსახულების მეორეთი შეცვლის აუცილებლობა. გამოსახულების შეცვლის არსი მდგომარეობს <IMG> ტეგის SRC ატრიბუტის მნიშვნელობის სცენარის დახმარებით შეცვლაში. თუ <IMG> ტეგის ელემენტი, რომელიც გვაძლევს გამოსახულებას, არის HTML-დოკუმენტში, მაშინ ობიექტურ მოდელში გვაქვს ამ ელემენტის SRC თვისების მქონე ობიექტი. ამ თვისების მნიშვნელობა შეიძლება შეიცვალოს სცენარში. ამასთან, ბრაუზერის ფანჯარაში სცენარის მიერ ჩაიტვირთება შესაბამისი გრაფიკული ფაილი, თუ რა თქმა უნდა იპოვნის მას.

მაგალითი:

```
<HTML>
<IMG ID = "myimg" SRC = 'pict1.gif'
onclick = "document . all . myimg . src = 'pict2.gif' ">
</HTML>
```

მოყვანილ მაგალითში პირველ სურათზე მაუსის ერთხელ დაწკაპუნება გამოიწვევს მისი მეორე სურათით შეცვლას. გამოსახულების შეცვლა ამ მაგალითში მხოლოდ ერთხელ მოხდება. მაუსის შემდგომი დაწკაპუნება არავითარ ცვლილებას არ გამოიწვევს, ვინაიდან მეორე გამოსახულება იგივეთი ისევ იცვლება. იმისათვის, რომ განმეორებითმა დაწკაპუნებამ გამოიწვიოს წინა სურათის დაბრუნება, მოცემული სცენარი მნიშვნელოვნად უნდა შეიცვალოს: უნდა მოხდეს ცვლადი ტრიგერის (ე. წ. ალამი) შექმნა, რომელიც ორიდან მიიღებს ერთ-ერთის მნიშვნელობას. ალმის მიმდინარე მნიშვნელობით სცენარი განსაზღვრავს ორიდან რომელი გამოსახულების გამოტანა უნდა მოხდეს. გამოსახულების შეცვლასთან ერთად აუცილებლად უნდა მოხდეს ალმის მნიშვნელობის შეცვლაც.

```
<HTML>
<IMG ID = "myimg" SRC = 'pict1.gif' onclick = "imgchange ( ) ">
<SCRIPT>
var flag = false
function imgchange ( )  {
if (flag) document . all . myimg . src = "pict1.gif"
          else document . all . myimg . src = "pict2.gif"
flag = !flag
}
</SCRIPT>
</HTML>
```

ახლა განვიხილოთ მაგალითი, როდესაც Web-გვერდზე განთავსებულია მინიატურული გალერეა (**thumbnails** – ფართომასშტაბიანი გამოსახულების შემცირებული ასლები). შემცირებულ ასლზე მაუსის დაწკაპუნებით მისი გადიდება უნდა მოხდეს, ხოლო გადიდებულზე დაწკაპუნებით კი ისევ შემცირება. ამ ამოცანის გადასაწყვეტად საჭირო იქნება იმდენი ალამი რამდენი გამოსახულებაც გვექნება.

```
<HTML>
<SCRIPT>
var apict1 = new Array (" pict1.gif ", . . . )    /*  საწყისი ფაილების
                                                      სახელების მასივი */
var apict2 = new Array (" pict2.gif ", . . . )    /*  შემცვლელი
                                          ფაილების სახელების მასივი */
var aflag = new Array (apict1 . length)    /*  ალმების მასივი */
var xstr = " "
for ( i = 0; i < apict1 . length; i++)  {
xstr += ' <IMG ID = " i ' + i + ' " SRC = " ' + apict1 [i] + ' " onclick
      = " imgchange ( ) " >'
}
document . write (xstr)
function imgchange ( )  {
var xid = event . srcElement . id
var n = parseInt (xid . substr (1) )
if (aflag [n]) document . all [xid] . src = apict1 [n]
      else document . all [xid] . src = apict2 [n]
```

```
aflag [n] = !aflag [n]
}
</SCRIPT>
</HTML>
```

ყურადღება იმას უნდა მიექცეს, რომ **src** თვისებაზე მიმართვა ხდება **document.all [xid].src**, და არა **document.all.xid.src** ან **document.all ["xid"].src**. ეს აიხსნება იმით, რომ **xid** არის სტრიქონული ცვლადი, რომელიც შეიცავს **ID** იდენტიფიკატორის მნიშვნელობას და არ არის უშუალოდ ამ იდენტიფიკატორის მნიშვნელობა.

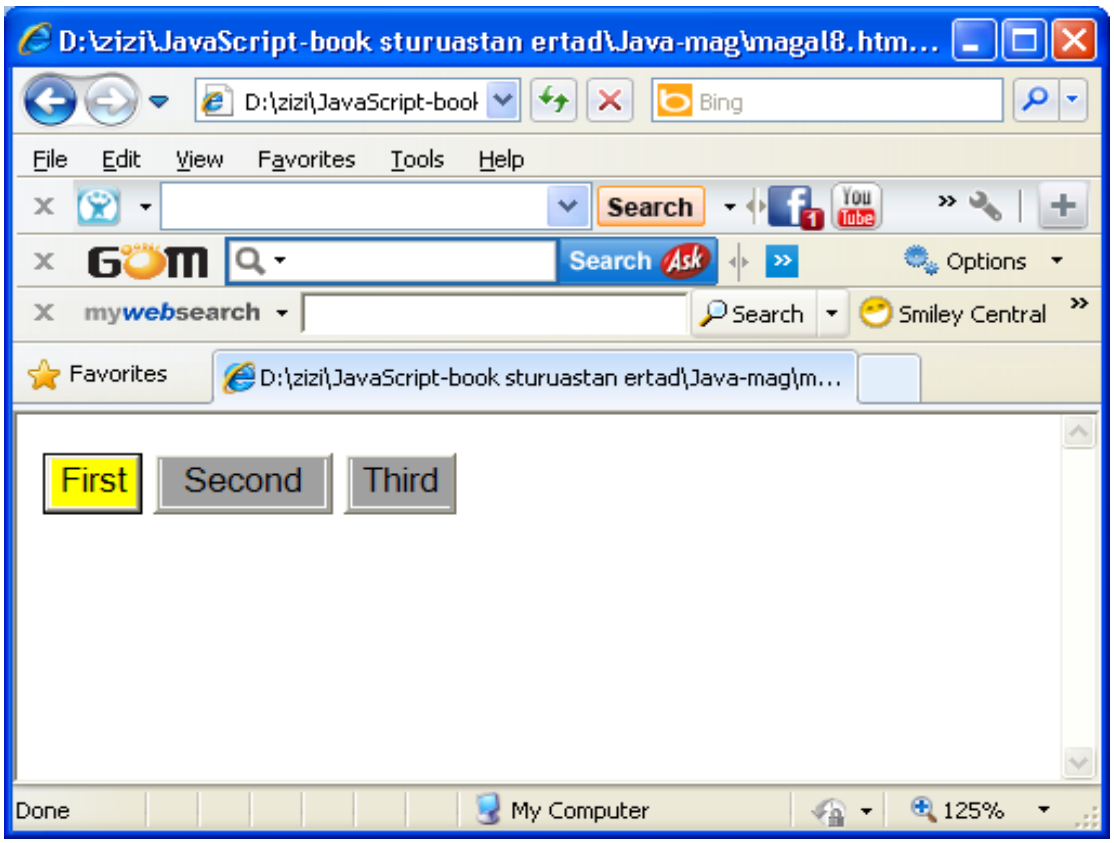
ღილაკებისა და ტექსტის გამონათება. განვიხილოთ ღილაკის ფერის შეცვლის ამოცანა, მასთან მაუსის მაჩვენებლის მიყვანის დროს. მაჩვენებლის ღილაკიდან გადაწევის შემთხვევაში მას უნდა დაუბრუნდეს თავდაპირველი ფერი. ეს არის ე. წ. ღილაკების გამონათება.

მაგალითი:

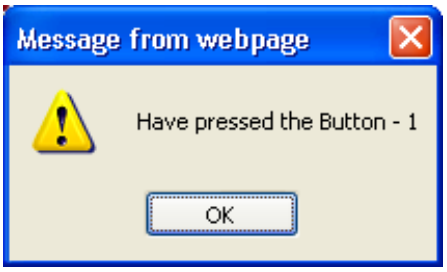
მოცემულია სამი ღილაკი კონტეინერული ფორმის **<FORM>** სახით. ამ კონტეინერთან მიმაგრებულია ხდომილობის დამამუშავებლები **onmouseover** (მაუსის მაჩვენებლის დაყენება ობიექტზე) და **onmouseout** (მაუსის მაჩვენებლის გადატანა ობიექტიდან). ამგვარად, ამ ხდომილობის ინიციატორი (მიმღები) შეიძლება იყოს ფორმის ნებისმიერი ელემენტი (ჩვენს მაგალითში ღილაკები). ჩვეულებრივ მდგომარეობაში ღილაკების ფერი ნაცრისფერია, მოცემული თექვსმეტობითი **a0a0a0** კოდით. მასთან მაუსის მაჩვენებლის მიყვანით ღილაკების ფერი იცვლება და ხდება ყვითელი (**yellow**).

```
<HTML>
<STYLE>
mystyle {font-weight:bold; background-color:a0a0a0}
</STYLE>
<FORM onmouseover = "colorchange ('yellow')"
onmouseout = "colorchange ('a0a0a0')">
<INPUT TYPE = "BUTTON" VALUE = "First" CLASS =
    "mystyle" onclick = "alert('You have pressed the Button - 1')">
<INPUT TYPE = "BUTTON" VALUE = "Second" CLASS =
    "mystyle" onclick = "alert('You have pressed the Button - 2')">
```

```
<INPUT TYPE = "BUTTON" VALUE = "Third" CLASS =
      "mystyle" onclick = "alert('You have pressed the Button - 3')">
</FORM>
<SCRIPT>
function colorchange  (color) {
if (event . srcElement . type == "button")
      event . srcElement . style . backgroundColor=color;
}
</SCRIPT>
</HTML>
```



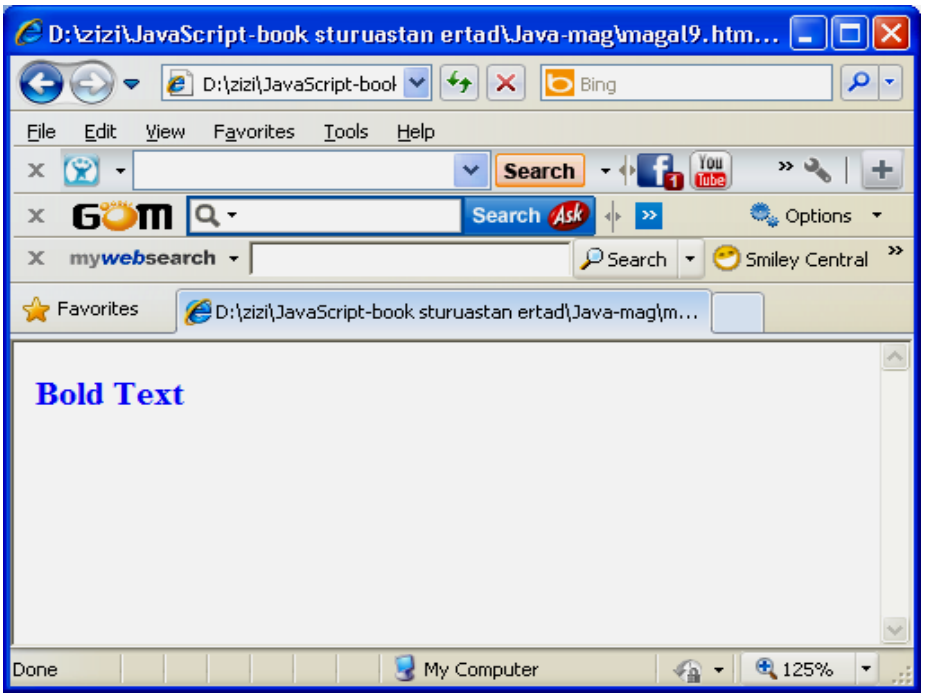
ფანჯარაში არსებულ რომელიმე ღილაკზე მაუსის დაჭერის შემდეგ ამონათდება შეტყობინების ფანჯარა, ხოლო თავად ღილაკი შეფერადდება ყვითელი ფერით. მაგალითად, პირველ ღილაკზე დაჭერისას მივიღებთ:



აქ **colorchange ( )** ფუნქციაში მოწმდება ხდომილობის ინიციატორი იყო თუ არა ობიექტი **button**. თუ ეს ასეა, მაშინ ღილაკის ფერი იცვლება, ხოლო წინააღმდეგ შემთხვევაში არა. ამ შემოწმების გარეშე შეიცვლებოდა არა მარტო ღილაკების ფერი, არამედ მთელი ფონიც. ანალოგიურად შეიძლება ნებისმიერი სხვა ელემენტის ფერის შეცვლაც, მაგალითად, ტექსტის ფრაგმენტის. თუ საჭიროა ტექსტის გამონათება, მაშინ ის მოთავსებული უნდა იყოს რომელიმე კონტეინერულ ტეგებს შორის, მაგალითად, ტეგებში <**P**>, <**B**>, <**I**> ან <**DIV**>. შემდეგ მაგალითში <**B**> ტეგში მოთავსებული ტექსტის ფერი მასზე მაუსის მაჩვენებლის მიყვანის შემთხვევაში ლურჯი ფერიდან იცვლება წითლით:

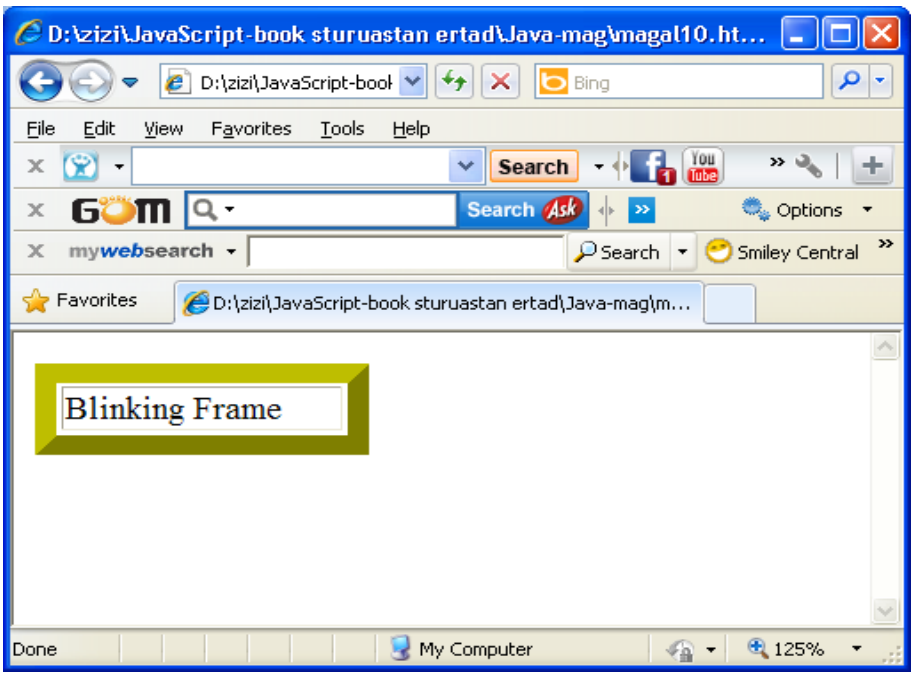
```
<HTML>
<B onmouseover = "colorch('red')" onmouseout =
      "colorch('blue')"
style = "color:blue">Bold Text</B>
<SCRIPT>
function colorch (color) {
event . srcElement . style . color=color;
}
</SCRIPT>
</HTML>
```

ეკრანზე გამონათებულ ჩანაწერს ექნება ლურჯი, მუქი შრიფტით ჩაწერილი ტექსტი.



მოციმციმე ჩარჩო. ჩვენ შეგვიძლია რაიმე ტექსტი მოვათავსოთ ჩარჩოში, რომელიც პერიოდულად იცვლის ფერს. ზოგჯერ ეს ეფექტი გამოიყენება მომხმარებლის ყურადღების მისაქცევად. ამისათვის, საჭიროა შეიქმნას ფუნქცია, რომელიც შეუცვლის ჩარჩოს ფერს და გადასცემს მას **setInterval ( )** მეთოდს პირველი პარამეტრის სახით. ამ მეთოდის მეორე პარამეტრი არის რიცხვი, რომელიც მიუთითებს ფუნქციის გამოძახების პერიოდულობას მილიწამებში.

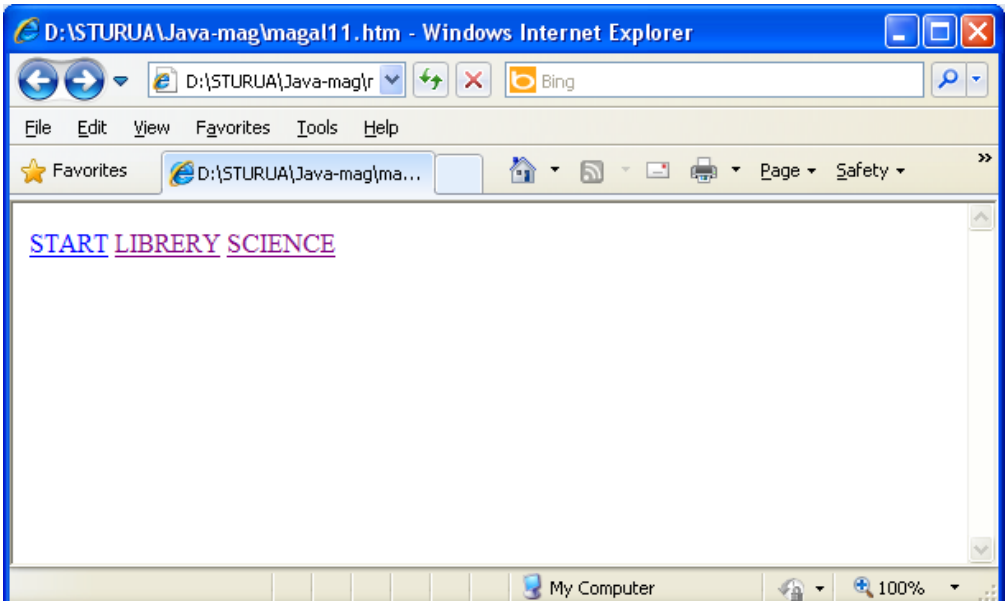
```
<HTML>
<TABLE ID="mytab" BORDER=1 WIDTH=150 style="border:10
      solid:yellow">
<TR><TD>Blinking Frame</TD></TR>
</TABLE>
<SCRIPT>
function flash ( ) {
if (!document . all)
return null;
if (mytab . style . borderColor == 'yellow') mytab . style .
      borderColor = 'red'
      else mytab . style . borderColor = 'yellow';
}
setInterval ("flash ( )" , 500);
</SCRIPT>
</HTML>
```



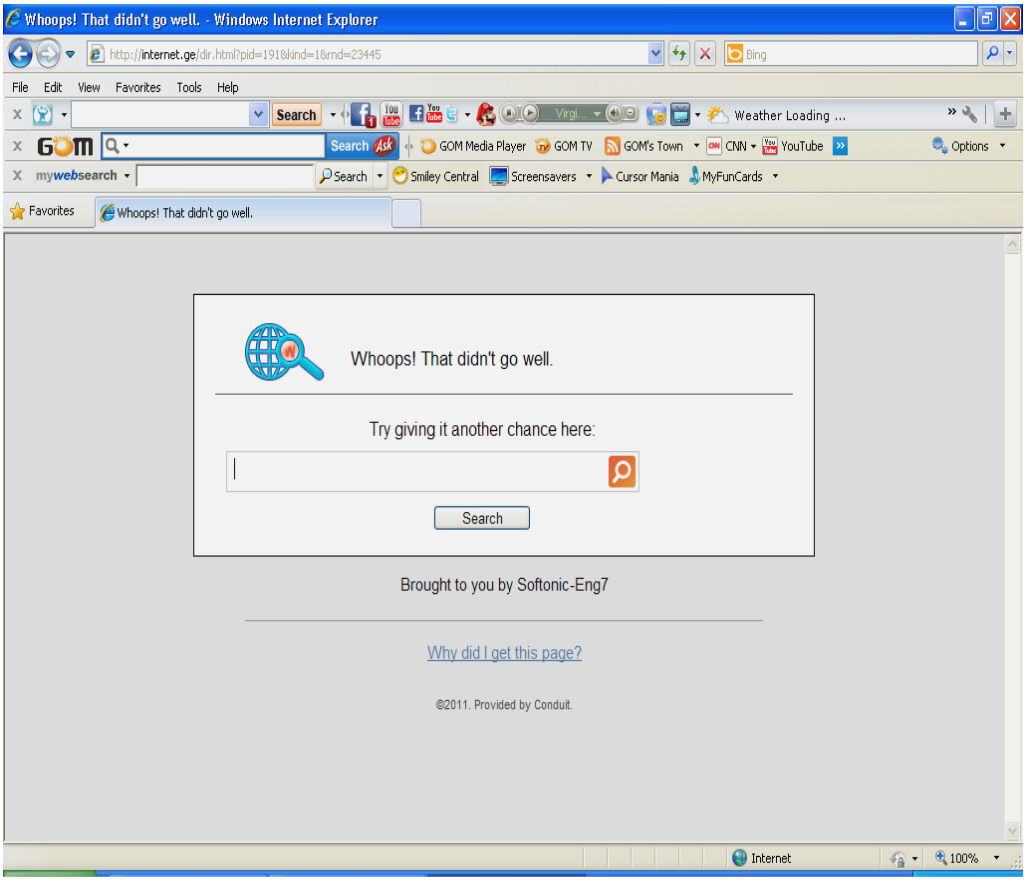
მოყვანილ მაგალითში ჩარჩოს ყვითელი ფერი 0,5 წამის პერიოდულობით იცვლება წითლით.

ფერებში გარდამავალი მითითებები. Web-გვერდის მითითებებზე მომხმარებლის ყურადღების მიპყრობა შეიძლება მათი ფერების დინამიკური ცვლილების მეშვეობით. ამოცანის არსი მდგომარეობს იმაში, რომ შემთხვევით მოხდეს მითითებების ფერების შერჩევა და მომართვა. ფერთა ეს სიმრავლე მოცემულია მასივის სახით.

```
<HTML>
<A HREF="htth://Internet.ge/#">START</A>
<A HREF="http://internet.ge/dir.html?pid=191&kind=1&
rnd=23445">LIBRERY</A>
<A HREF="http://internet.ge/dir.html?pid=113&kind=1&
rnd=149734">SCIENCE</A>
<SCRIPT>
aclrlink = new Array ( )
aclrlink [0] = 'yellow'
aclrlink [1] = '#80FF80'
aclrlink [2] = '#FFFF80'
aclrlink [3] = '#408000'
aclrvlink = new Array ( )
aclrvlink [0] = 'blue'
aclrvlink [1] = 'purple'
aclrvlink [2] = 'black'
aclrvlink [3] = 'red'
function colorchange ( ) {
link = Math . round ( (aclrlink . length+0.1) * Math . random ( ) )
vlink = Math . round ( (aclrvlink . length+0.1) * Math . random ( ) )
      document . linkColor = aclrlink [link]
      document . vlinkColor = aclrvlink [vlink]
}
setInterval ("colorchange ( )", 1000)
</SCRIPT>
</HTML>
```



Start ღილაკზე მაუსის დაწკაპუნებით გამონათდება შემდეგი სახის ფანჯარა:

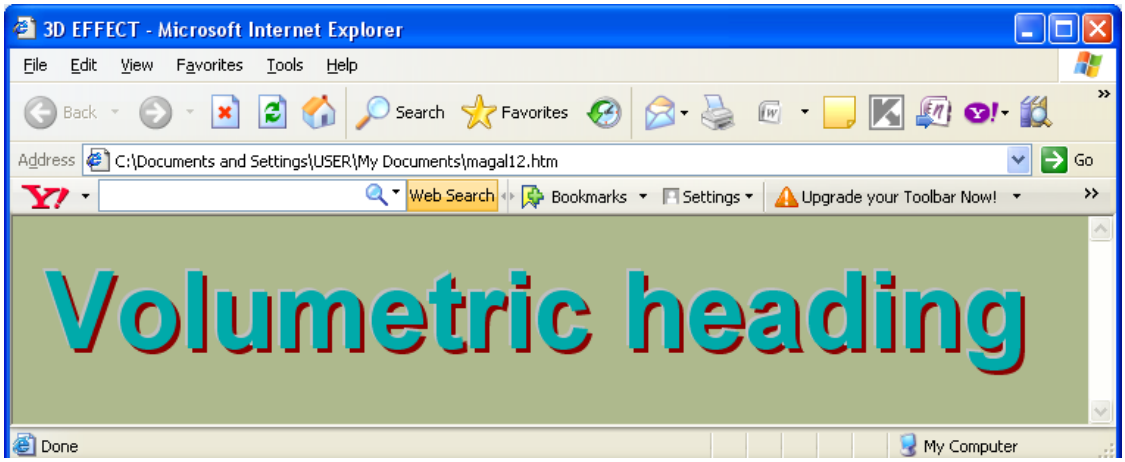


უნდა შევნიშნოთ, რომ ამ სცენარის **HTML**-დოკუმენტში ჩასმის შემდეგ დოკუმენტის ყველა მითითება დაიწყებს სხვადასხვა ფერებში გადასვლას, ვინაიდან ვცვლით **document** ობიექტის **linkColor** და **vlinkColor** თვისებებს.

მოცულობითი სათაურები. მოცულობითი სათაურები ძალზე ეფექტურია Web-გვერდების მომზადების დროს. მისი შექმნის იდეა მარტივია: საკმარისია შეიქმნას ერთნაირი რამდენიმე წარწერა და დაედოს ერთმანეთს მცირე წანაცვლებით. საჭიროა მინიმუმ ორი ასეთი წარწერა. ერთი მათგანი გამოიყენება ჩრდილის ეფექტის შესაქმნელად, ხოლო მეორე განთავსდება მასზე. შეიძლება გამოყენებულ იქნეს ასეთი მესამე წარწერაც ქვემოდან განათების ეფექტის შესაქმნელად. საუკეთესო ეფექტს იძლევა ფონის ფერის გათვალისწინებით ამ წარწერების ფერების შერჩევა. ქვემოთ მოყვანილ მაგალითში მოცულობითი სათაური მიღებულია სამი წარწერის ზედდებით. სტილების ცხრილი განისაზღვრება <P> ტეგით. მოცემულ ცხრილში განსაზღვრულია წარწერის შრიფტის ფერი და ზომა. წარწერის პოზიციები კი განისაზღვრება კონტეინერული <DIV> ტეგის STYLE ატრიბუტის პარამეტრებით.

```
<HTML>
<HEAD><TITLE> 3D EFFECT</TITLE><HEAD>
<STYLE>
P {font-family:sans-serif; font-size:72; font-weight:800;
    color:00aaaa}
P.highlight {color:silver}
P.shadow {color:darkred}
</STYLE>
<BODY BGCOLOR = aeb98d>
<DIV STYLE = "position:absolute; top:25; left:25">
<P CLASS = shadow>Volumetric heading</P>
</DIV>
<DIV STYLE = "position:absolute; top:20; left:20">
<P CLASS = highlight>Volumetric heading</P>
</DIV>
<DIV STYLE = "position:absolute; top:22; left:22">
<P>Volumetric heading</P>
</DIV>
</BODY>
</HTML>
```

ამ პროგრამის შესრულებით მიიღება შემდეგი შედეგი:

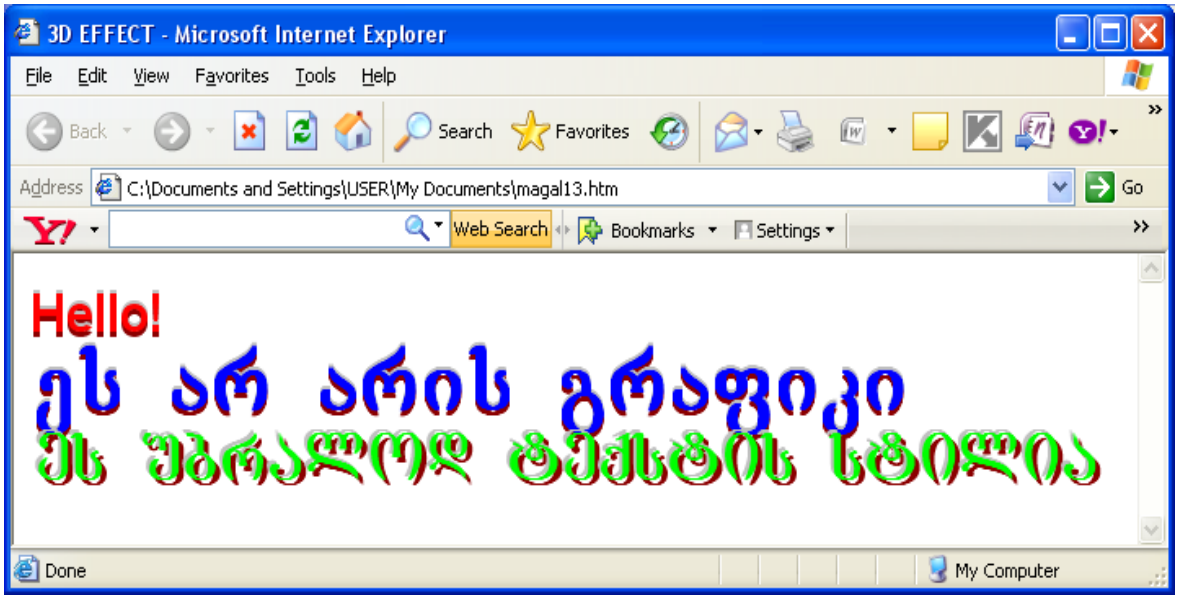


ახლა განვიხილოთ მაგალითი, სადაც გამოყენებული იქნება ფუნქცია, რომელიც ქმნის სათაურს მოცემული პარამეტრების მიხედვით.

```
<HTML>
<HEAD><TITLE> 3D EFFECT</TITLE><HEAD>
<SCRIPT>
function d3(text, x, y, tcolor, fsize, fweight, ffamily, zind) {
if (!text) return null
if (!ffamily) ffamily='Arial'
if (!fweight) fweight=800
if (!fsize) fsize=36
if (!tcolor) tcolor='00aaff'
if (!y) y=0
if (!x) x=0
var sd=5, hd=2
var xzind=" "
if (!zind) xzind=":z-Index:"+zind
var xstyle='font-family:' + ffamily + ';font-size:' + fsize + ';font-
      weight:' + fweight + ';color:' + tcolor + ';'
var xstr = '<DIV STYLE = "position:absolute; top:' + (y + sd) +
      ';left:' + (x + sd) + xzind + '">'
xstr += '<P style = "' + xstyle + 'color:darkred">' + text +
      '</P></DIV>'
xstr += '<DIV STYLE = "position:absolute; top:' + y + '; left:' + x +
      xzind + '">'
xstr += '<P style = "' + xstyle + 'color:silver">' + text +
      '</P></DIV>'
```

```
xstr += '<DIV STYLE = "position:absolute; top:' + (y + hd) + '; left:'
        + (x + hd) + xzind + '">'
xstr += '<P style = " ' + xstyle + 'color:' + tcolor + ' ">' + text +
        '</P></DIV>'
document . write (xstr)
}
d3 ("Hello!", 50, 15, 'red', 30, 800, 'arial')
d3 ("es ar aris grafiki", 50, 50, 'blue', 52, 800, 'acadnusx')
d3 ("es ubralod teqstis stilia", 10, 100, '00ff00', 40, 900, 'acadmtavr',
        -7)
</SCRIPT>
</HTML>
```

ამ პროგრამის შესრულებით შემდეგი შედეგი მიიღება:



ამ ფუნქციაში სტრიქონები იქმნება, რომელიც შემდეგ ჩაიწერება **HTML**-დოკუმენტში ტეგის სახით. ფუნქციის პარამეტრებს შორის ბოლო **z-Index**-ია, რომლის საშუალებითაც შეიძლება ფენის დონის განსაზღვრა, რომელშიც უნდა განთავსდეს სათაური. ელემენტი, რომლის **z-Index**-ის მნიშვნელობა მეტია, მდებარეობს ზემოთ, ვიდრე ელემენტი, რომლის **z-Index**-ის მნიშვნელობა ნაკლებია. ტოლი მნიშვნელობისათვის მიმდევრობა განისაზღვრება **HTML**-დოკუმენტში ტეგების მიმდევრობით. **sd** და **hd** პარამეტრებით განისაზღვრება ჩრდილისა და განათების არეალის ზომები.

39. ფილტრების გამოყენება

ფილტრების გამოყენებით შეიძლება მივიღოთ მრავალნაირი საინტერესო ვიზუალური ეფექტები. მაგალითად, სურათის ნელ-ნელა გამოჩენა (გაქრობა), ერთი გამოსახულების სხვა გამოსახულებად ნარნარად გარდაქმნა, გამჭირვალეობის ხარისხის მინიჭება და სხვა. უმეტეს შემთხვევაში **Web**-დიზაინერები მსგავსი ეფექტების მისაღწევად იყენებენ გამოსახულების დამუშავების გრაფიკულ რედაქტორებს, როგორიცაა **Adobe Photoshop**, **Macromedia Flash** და სხვა.

ფილტრი შეიძლება განვიხილოთ როგორც რაიმე ინსტრუმენტი გრაფიკული ფაილიდან აღებული და **HTML**-დოკუმენტში ტეგი **<IMG>**-ის საშუალებით ჩასმული გამოსახულების გარდაქმნისათვის. ამასთან, უნდა აღინიშნოს, რომ ფილტრები მხოლოდ **Internet Explorer**-ში მუშაობს. ქვემოთ განვიხილავთ რამდენიმე საინტერესო ფილტრს.

გამჭვირვალობა. გრაფიკული ობიექტის გამჭვირვალობა ნულიდან 100-მდე დიაპაზონში შეიძლება დავაყენოთ **alpha** ფილტრის საშუალებით. მნიშვნელობა 0 შეესაბამება გამოსახულების სრულ გამჭირვალობას, ანუ ის ხდება უჩინარი. ხოლო 100 კი სრულ გაუმჭირვალობას. ამიტომ უფრო სწორი იქნება **alpha**-ს ვუწოდოთ გაუმჭირვალობის ხარისხი. გამჭირვალე გრაფიკული ობიექტების ქვეშ ჩანს მის ქვემოთ მდებარე გამოსახულებანი. მათი ხილვის ხარისხი დამოკიდებულია ზედა ობიექტის გამჭირვალეობის ხარისხზე. ამის გარდა, გამჭირვალეობას აქვს გრადიენტული ფორმის რამდენიმე ვარიანტი. მაგალითად, გამჭირვალეობა თანდათან იზრდებოდეს ცენტრიდან კიდეებისაკენ.

როგორც სხვა ფილტრების ასევე **alpha**-ს მიცემა ხდება სტილის კასკადური ცხრილის საშუალებით და აქვს პარამეტრები. ქვემოთ მოყვანილ მაგალითში გრაფიკული გამოსახულებისათვის სტილი განისაზღვრება ატრიბუტ **STYLE**-ის საშუალებით:

```
<HTML>
<STYLE>
#myimg { position:absolute; top:10; left:50; filter:alpha (opacity=70,
    style=3)}
</STYLE>
```

```
<IMG ID = "myimg" SRC = "C:\Documents and Settings\USER\
    My Documents\ My Pictures\Blue hills.jpg">
</HTML>
```

აქ პარამეტრი **opacity** განსაზღვრავს გამჭვირვალობის ხარისხს (მთელი რიცხვი 0-დან 100-მდე), ხოლო პარამეტრი **style** იძლევა გამოსახულების გამჭვირვალობის განაწილების გრადიენტულ ფორმას და ღებულობს მთელ რიცხვით მნიშვნელობას 0-დან 3-მდე. ფილტრ **alpha**-ს სხვა პარამეტრებიც აქვს, რომლებიც განსაზღვრავს გამოსახულების იმ მართკუთხა არეს, რომლის მიმართაც გამოიყენება ფილტრი. გაჩუმების პრინციპით იგი გამოიყენება მთელი გამოსახულების მიმართ.

ფილტრი შეიძლება განისაზღვროს სტილის კასკადურ ცხრილში კონტეინერული ტეგის <STYLE>-ის შიგნით მითითების საშუალებით, რომელსაც შემდეგი სტრუქტურა აქვს:

```
#id_გამოსახულება { filter:ფილტრის სახელი(პარამეტრები)}
```

აქვე შევნიშნოთ, რომ თუ არ გამოვიყენებთ # სიმბოლოს, ფილტრი არ იმუშავებს. ამ შემთხვევაში ზემოთ მოყვანილ მაგალითს შემდეგი სახე ექნება:

```
<HTML>
<IMG ID = "myimg"  SRC = "C:\Documents and Settings\USER\
    My Documents\ My Pictures\Blue hills.jpg"
STYLE= "position:absolute; top:50; left:150; filter:alpha
    (opacity=70, style=3)">
</HTML>
```

სცენარში ფილტრის თვისებებზე მიმართვა შემდეგნაირად განისაზღვრება:

```
document . all . id_გამოსახულება . filters ["ფილტრის სახელი"] .
    პარამეტრი = მნიშვნელობა
```

ფილტრების გამოყენება შესაძლებელია არა მარტო გრაფიკული გამოსახულებების მიმართ, არამედ სხვა ელემენტების მიმართაც (მაგალითად, ტექსტების, ტექსტური ველის, ღილაკების მიმართ).

ტრანსფორმაცია. გამოსახულების გარდაქმნის ფილტრები საშუალებას გვაძლევს მოვახდინოთ გამოსახულების ნელ-ნელა გამოჩენა (გაუჩინარება), აგრეთვე ერთი გრაფიკული გამოსახულების მეორე

გამოსახულებად ტრანსფორმაცია. ეს არის ე. წ. დინამიკური ფილტრები. სტატიკური ფილტრებისაგან (**alpha**) განსხვავებით მათი გამოყენება აუცილებლად სცენარებთანაა დაკავშირებული. გრაფიკული ობიექტის გარდაქმნის არსი მდგომარეობს შემდეგში: ჯერ უნდა მოხდეს პირველი გამოსახულების დაფიქსირება, შემდეგ შესრულდეს ამ გამოსახულების შეცვლა სხვა გამოსახულებით და/ან მოხდეს იმავე გამოსახულების პარამეტრების შეცვლა, ხოლო ყოველივე ამის შემდეგ შესრულდეს ტრანსფორმაცია. ყველა ეს მოქმედება სრულდება სცენარში. გამოსახულების ფიქსაცია და ტრანსფორმაცია ხდება ფილტრების სპეციალური მეთოდების საშუალებით–შესაბამისად **apply ( )** და **play ( )**. სურვილის შემთხვევაში გარდაქმნის პროცესის შეჩერება შესაძლებელია **stop ( )** მეთოდის საშუალებით. გამოსახულების გარდაქმნისათვის გამოიყენება ფილტრი **revealtrans**. მას შემდეგი პარამეტრები აქვს:

- **duration** – გარდაქმნის ხანგრძლივობა წამებში (მცურავმძიმიანი რიცხვი);

- **transition** – გარდაქმნის ტიპი (მთელი რიცხვი 0-დან 23-მდე)

გამოსახულების გამოჩენის ეფექტის მისაღებად განვიხილოთ ფილტრ **revealtrans**-ის გამოყენების მაგალითი. ქვემოთ მოყვანილ მაგალითში, გამოსახულება ნელ-ნელა გამოჩნდება დოკუმენტის ჩატვირთვის, ანუ **onload** ხდომილობის შემდეგ:

```
<HTML>
<BODY onload = "transform ( )">
<IMG ID = "myimg" SRC = "C:\Documents and Settings\USER\
     My Documents\My Pictures\Blue hills.jpg"
STYLE = "position:absolute; top:10; left:50;  filter : revealtrans
     (duration=5, transition=0)">
</BODY>
<SCRIPT>
function transform ( ) {
document . all . myimg . style . visibility = "hidden"
myimg . filters ("revealtrans") . apply ( )
myimg . style . visibility = "visible"
myimg . filters ("revealtrans") . play ( )
}
</SCRIPT>
```

```
</HTML>
```

მოცემულ მაგალითში გამოსახულების გარდაქმნის ტიპისათვის გამოყენებულია **transition = 12**. მისი შეცვლით (0-დან 22-მდე) შეიძლება მივიღოთ გამოსახულების გარდაქმნის სხვადასხვა ეფექტი.

გარდაქმნის ტიპი და მისი ხანგრძლივობა შეიძლება მოცემული იყოს როგორც გამოსახულების სტილის განსაზღვრაში, ასევე სცენარშიც:

```
<HTML>
<BODY onload = "transform ( )">
<IMG ID = "myimg" SRC = "C:\Documents and Settings\USER\
     My Documents\ My Pictures\Blue hills.jpg"
STYLE = "position:absolute; top:10; left:50; filter:revealtrans">
</BODY>
<SCRIPT>
    function transform ( ) {
    document . all . myimg . style . visibility = "hidden"
    myimg . filters("revealtrans") . apply ( )
    myimg . style . visibility = "visible"
    myimg . filters("revealtrans") . transition = 12
    myimg . filters ("revealtrans") . play (3)
    }
</SCRIPT>
</HTML>
```

ცხადია, რომ გამოსახულების გასაქრობად უნდა მოვიქცეთ პირიქით, ჯერ იგი უნდა გავხადოთ ხილული და შემდეგ უხილავი.

ახლა განვიხილოთ უფრო ზოგადი შემთხვევა, ერთი გრაფიკული ობიექტის მეორედ ტრანსფორმაცია. წინა მაგალითისაგან განსხვავებით აქ უნდა მოხდეს საწყისი და ბოლო გამოსახულების განსაზღვრა:

```
<HTML>
<BODY onload = "transform ( )">
<IMG ID = "myimg" SRC = "C:\Documents and Settings\USER\
     My Documents\ My Pictures\Sunset.jpg"
STYLE = "position:absolute; top:10; left:50; filter:revealtrans
     (duration=5, transition=12)">
</BODY>
<SCRIPT>
function transform ( ) {
```

```
myimg . filters("revealtrans") . apply ( )
document . all . myimg . src = "C:\Documents and Settings\USER\
    My Documents\ My Pictures\Blue hills.jpg"
myimg . filters ("revealtrans") . play ( )
}
</SCRIPT>
</HTML>
```

შემდეგ მაგალითში ხდება გამოსახულების ტრანსფორმაცია, რომელიც სრულდება გრაფიკულ ობიექტზე მაუსის დაწკაპუნებით:

```
<HTML>
<IMG ID = "myimg" onload = "transform ( )" SRC =
    "C:\Documents and Settings\USER\My Documents\
    My Pictures\Blue hills.jpg"
STYLE = "position:absolute; top:10; left:50; filter:revealtrans
    (duration = 5, transition = 12)">
<SCRIPT>
function transform ( ) {
myimg . filters("revealtrans") . apply ( )
if (document . all . myimg . src . indexOf ("Blue hills") != 1)
document . all . myimg . src = ("C:\Documents and Settings\USER\
    My Documents\ My Pictures\Sunset.jpg")
else document . all . myimg . src = ("C:\Documents and Settings
    \USER\ My Documents\ My Pictures\Blue hills.jpg")
myimg . filters ("revealtrans") . play ( )
}
</SCRIPT>
</HTML>
```

ზოგჯერ შეიძლება გამოყენებულ იქნეს **Microsoft**-ის მიერ რეკომენდებული სხვა სინტაქსი. მისი თავისებურება იმაში მდგომარეობს, რომ სტილის კასკადურ ცხრილში მოცემულია მითითება სპეციალურ კომპონენტსა და ფილტრის სახელზე. მაგალითს ასეთი სინტაქსით ა შემდეგი სახე აქვს:

```
<HTML>
<STYLE>
```

```
#myimg {position:absolute; top:10; left:50; filter:progid:
    DXImageTransform . Microsoft . revealtrans (duration=3,
    transition=12)}
</STYLE>
<IMG ID = "myimg" onclick = "transform( )" SRC = "C:\Documents
    and Settings \ USER\My Documents\ My Pictures\Blue hills.jpg">
<SCRIPT>
function transform ( ) {
myimg . filters ["DXImageTransform.Microsoft.revealtrans"] . apply ( )
if  (document . all . myimg . src . indexOf ("C:\Documents and
    Settings\USER\ My Documents\My Pictures\Blue hills.jpg")! = -1)
document . all . myimg . src = ("C:\Documents and Settings \USER\My
    Documents\ My Pictures\Sunset.jpg")
else  document . all . myimg . src = ("C:\Documents and Settings \USER\
    My Documents\ My Pictures\Blue hills.jpg")
myimg . filters ["DXImageTransform.Microsoft.revealtrans"] . play ( )
}
</SCRIPT>
</HTML>
```

მობრუნება. სცენარში შესაძლებელია გამოყენებულ იქნეს **basicimage** ფილტრი. მას გააჩნია პარამეტრთა სიმრავლე, რომელთა საშუალებითაც შეიძლება გამოსახულების შემობრუნება 90^0-ის ჯერადი კუთხით, გამჭვირვალობის მინიჭება, სარკისებურად ასახვის შესაძლებლობა და სხვა. მათი გამოყენების წესი იგივეა, რაც **alpha** ფილტრის შემთხვევაში იყო. განვიხილოთ მობრუნების ეფექტის შექმნის შესაძლებლობა.

basicimage ფილტრის **rotation** პარამეტრი ღებულობს მთელრიცხვით მნიშვნელობას: **0** (მობრუნება არ არის), **1** (90^0), **2** (180^0), **3** (270^0). ქვემოთ მოყვანილ მაგალითში გამოსახულებაზე მაუსის დაწკაპუნებით ხდება მისი მობრუნება 90^0-იანი კუთხით:

```
<HTML>
<STYLE>
#myimg {filter:progid:DXImageTransform.Microsoft.basicimage}
</STYLE>
<IMG ID="myimg" SRC = "C:\Documents and Settings\USER\
    My Documents\My Pictures\Blue hills.jpg" onclick="rotor( )" >
<SCRIPT>
```

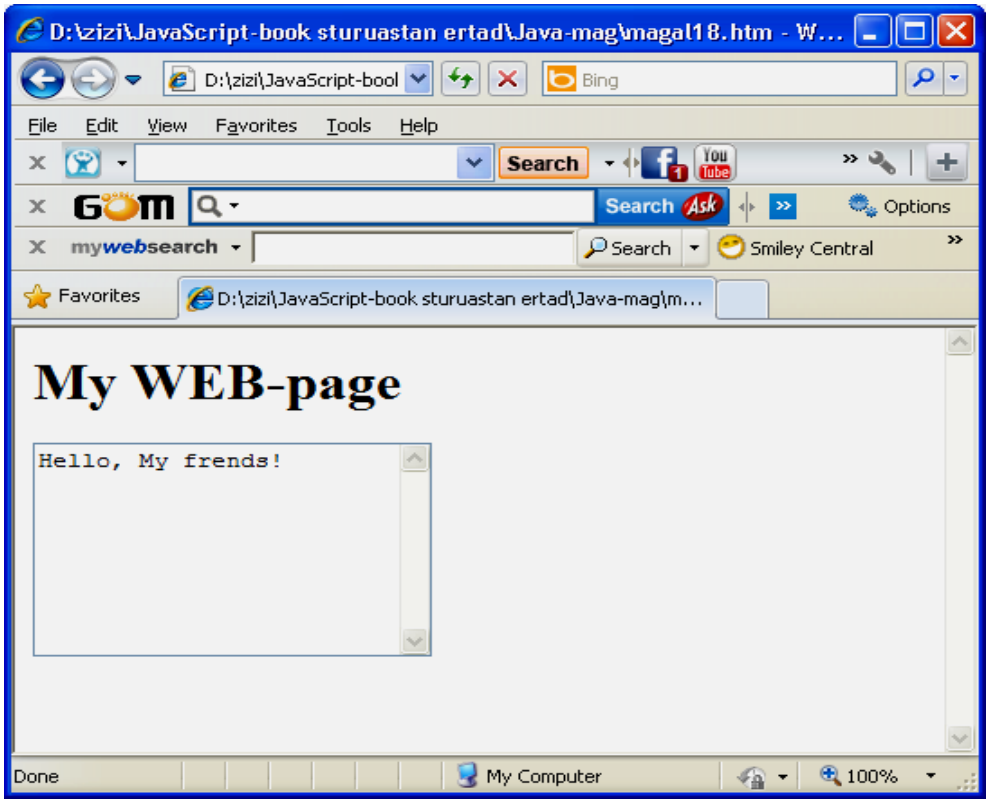
```
function rotor ( ) {
var r =
document . all . myimg . filters
     ["DXImageTransform.Microsoft.basicimage"] . rotation
if (r == 3)  r = 0
else       r++
document . all . myimg . filters
     ["DXImageTransform.Microsoft.basicimage"] . rotation = r
}
</SCRIPT>
</HTML>
```

ხილულ გრაფიკულ ელემენტზე ერთდროულად შეიძლება გამოყენებულ იქნეს რამდენიმე ფილტრი.

საბეჭდ მანქანაზე ბეჭდვის ეფექტი. ტექსტის ეტაპობრივად ნაწილ-ნაწილ გამოტანა (საბეჭდ მანქანაზე ბეჭდვის ეფექტი) შეიძლება განვახორციელოთ **setInterval ( )** მეთოდის გამოყენებით. ქვემოთ მოყვანილია **HTML**-დოკუმენტი, სათაურითა და ტექსტური არით, რომელსაც ტექსტი გამოჰყავს ტექსტურ არეში 0,1 წამის ინტერვალით. ფუნქცია **wrtext ( )** უბრალოდ ახდენს გამოსატანი ტექსტის ფორმირებას, ხოლო გამოტანა ხორციელდება ტექსტური არის **value** თვისებისთვის მნიშვნელობის მინიჭების გზით. **setInterval ( )** მეთოდი **wrtext ( )** ფუნქციას იძახებს მეორე პარამეტრში მითითებული დროის ინტერვალით. მაგალითში ეს ინტერვალი არის 100-ის ტოლი.

```
<HTML>
<H1> My WEB-page</H1>
<TEXTAREA ID = "mytext" ROWS = 8 COLS = 25>
     </TEXTAREA>
<SCRIPT>
var mystr = "Hello, My frends! "
var astr = mystr . split ("")
var typestr = ""
var i = 0
var xinterval = setInterval("wrtext ( )", 100)
function wrtext ( ) {
if (i < astr . length) {
```

```
        typestr+= astr [i]
        document . all . mytext . value = typestr
        i++
} else
        clearInterval
}
</SCRIPT>
</HTML>
```

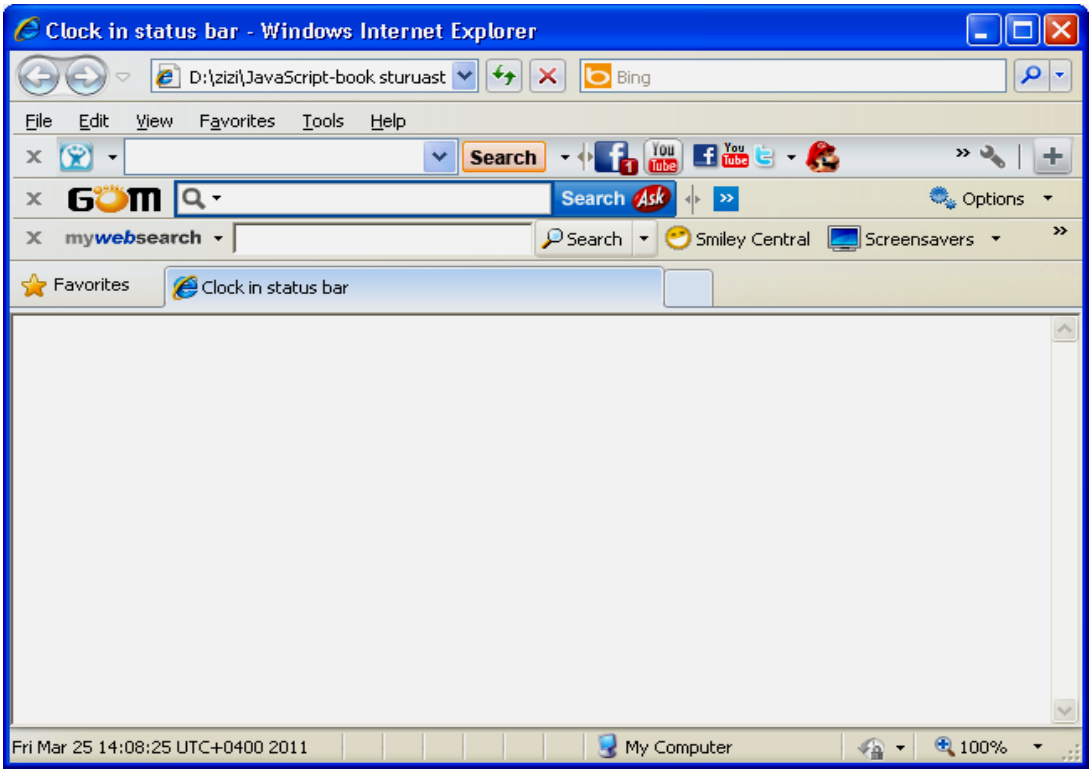


სხვადასხვა ტიპის საათის სცენარში განთავსება. მიმდინარე საათის ჩვენება შესაძლებელია მოთავსდეს ფანჯრის სტატუსის სტრიქონში ანდა form ველში. პირველი მაგალითი გვიჩვენებს საათის ჩვენებას დოკუმენტის ჩატვირთვის დროს. შექმნილი საათი, დოკუმენტის ჩატვირთვის შემდეგ განთავსდება ფანჯრის საინფორმაციო ზოლში (**Status Bar**):

```
<HTML>
<HEAD>
<TITLE>Clock in status bar</TITLE>
<script language="JavaScript">
function clock_status( )
```

```
{ window.setTimeout("clock_status( )",100); today=new Date();
      self.status=today.toString( );
}
</SCRIPT>
</HEAD>
<BODY background="ffffff" onLoad="clock_status( )"> </BODY>
</HTML>
```

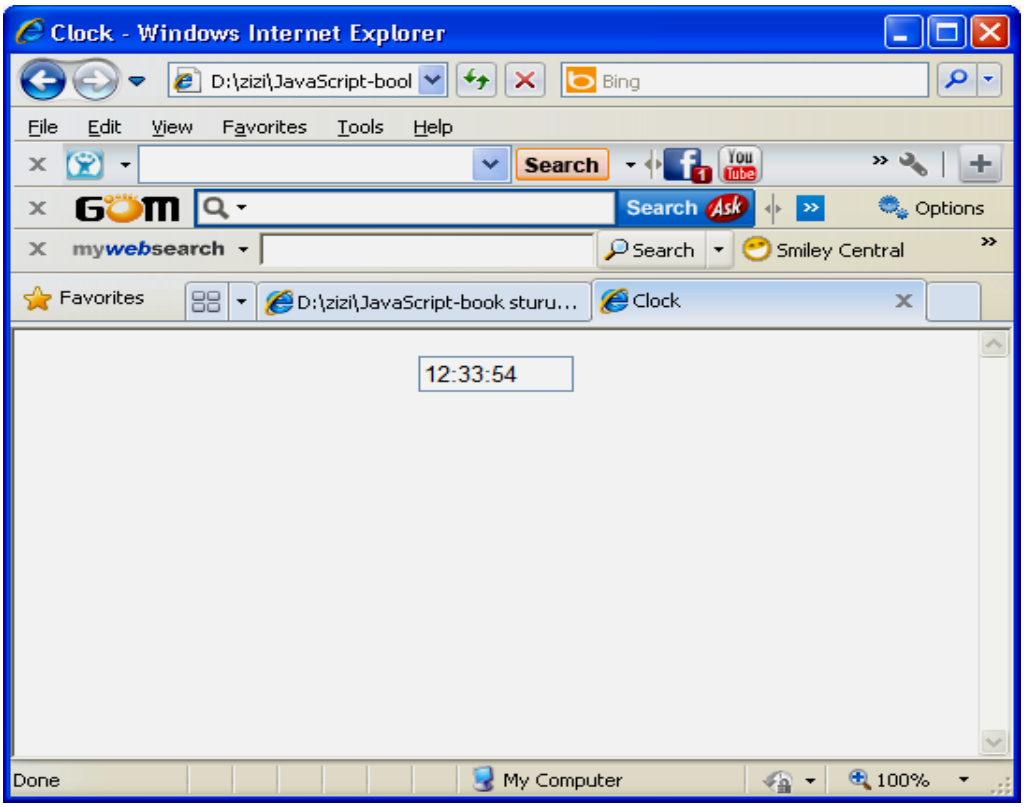
შენიშვნა. დააკვირდით სტატუსის სტრიქონს



მეორე მაგალითში საათი (შემოკლებული ვარიანტი) შექმნილია FORM ველში:

```
<HTML>
<HEAD>
<TITLE>Clock</TITLE>
<script language="JavaScript">
function clock_form( ) {
day=new Date( )
clock_f=day.getHours( )+":"+day.getMinutes(
      )+":"+day.getSeconds( )
document.form.f_clock.value=clock_f
id=setTimeout("clock_form()",100) }
```

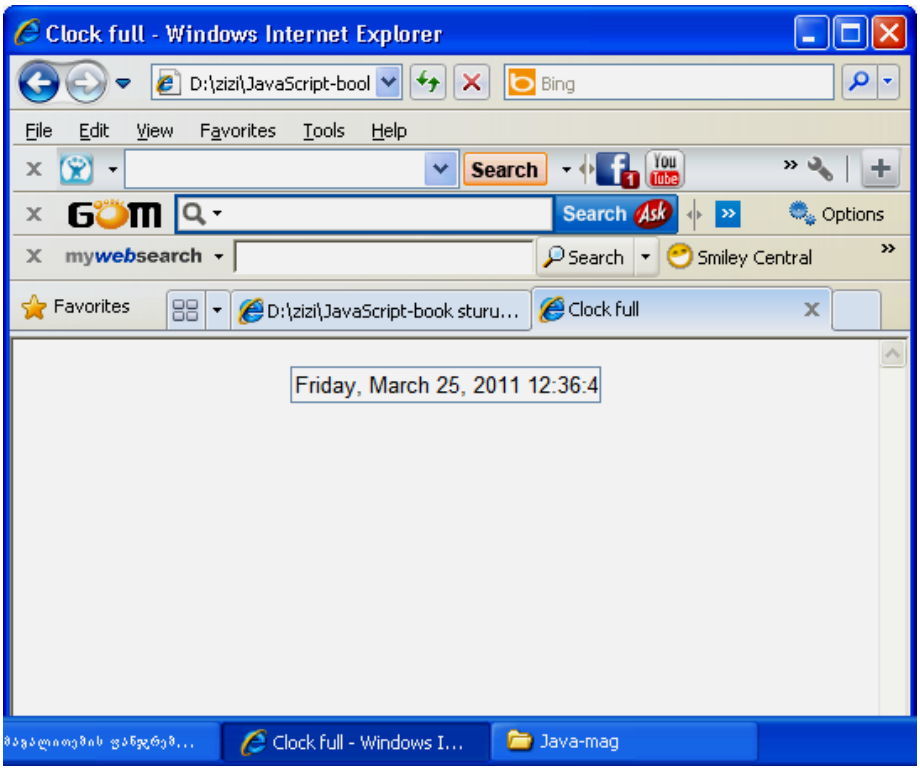
```
</SCRIPT>
</HEAD>
<BODY bgcolor="ffffff" onload="clock_form()">
<CENTER>
<FORM name="form" metod="get">
<INPUT name=f_clock maxlength=8 size=8>
</FORM>
</CENTER>
</BODY>
</HTML>
```



მესამე მაგალითი არის საათის ასახვის კიდევ ერთი ვარიანტი. ამასთან, ყურადღება უნდა მიექცეს იმას, რომ ფუნქციის გამოძახება ხდება დოკუმენტის ტანში და არა **<BODY>** ტეგში, როგორც ეს წინა მაგალითში იყო:

```
<HTML>
<HEAD>
<TITLE>Clock full</TITLE>
</HEAD>
<script language="JavaScript">
```

```
function fulltime( ) {
var time=new Date( );
document.clock.full.value=time.toLocaleString( );
setTimeout('fulltime()',500)
}
</SCRIPT>
<BODY bgcolor=ffffff text=ff0000>
<CENTER>
<FORM name=clock>
<INPUT type=text size=25 name=full>
</FORM>
<script language="JavaScript">
fulltime( );
</SCRIPT>
</CENTER>
</BODY>
</HTML>
```



იგივე ამოცანა შეიძლება შემდეგნაირად გადაწყდეს:

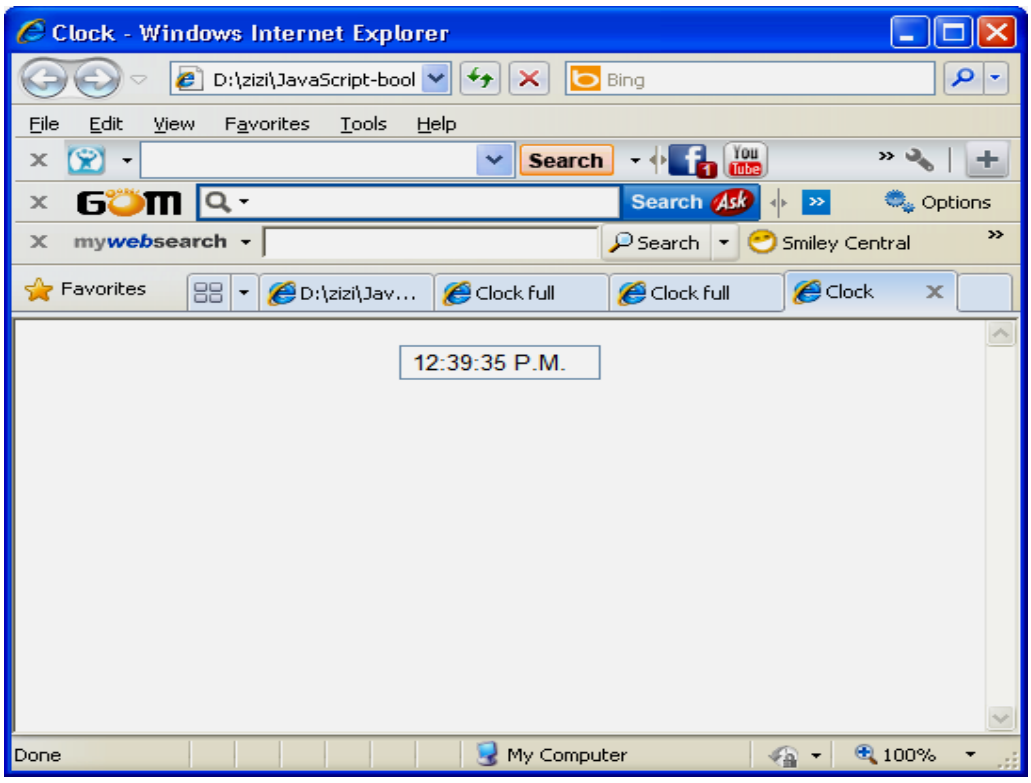
```
<SCRIPT language="JavaScript">
function fulltime( ) {
```

```
var time=new Date( );
document.clock.full.value=time.toLocaleString( );
setTimeout(" fulltime( )",500)
}
</SCRIPT>
<CENTER>
<FORM name=clock>
<INPUT type=text size=27 name=full>
</FORM>
<SCRIPT language="JavaScript">
fulltime( );
</SCRIPT>
</CENTER>
```

შემდეგ მაგალითში მოცემულია ველი მხოლოდ საათის ჩვენებით და დღის პირველი ან მეორე ნახევრის "**PM**"/"**AM**" მითითებით:

```
<HTML>
<HEAD>  <TITLE>Clock</TITLE>
<script language="JavaScript">
var timer=null;
var timerrun=false;
function stoptime( ) {
if(timerrun)
clearTimeout(timer);
timerrun=false;
}
function starttime( ) {
stoptime( );
showtime( );
}
function showtime( ) {
var day=new Date( );
var hours=day.getHours( );
var minutes=day.getMinutes( );
var seconds=day.getSeconds( );
var timevalue=" " + ((hours>12) ? hours-12 : hours)
timevalue += ((minutes<10) ? ":0" : ":") + minutes
timevalue += ((seconds<10) ? ":0" : ":") + seconds
```

```
timevalue +=(hours>=12) ? " P.M." : " A.M."
document.clock.next.value=timevalue;
timer=setTimeout('showtime( )',100);
timerrun=true;
}
</SCRIPT>
<BODY bgcolor=ffffff text=ff0000 onLoad="starttime()">
<CENTER>
<FORM name=clock>
<INPUT type=text name=next size=12 value=' '>
</CENTER>
</FORM>
</BODY>
</HTML>
```

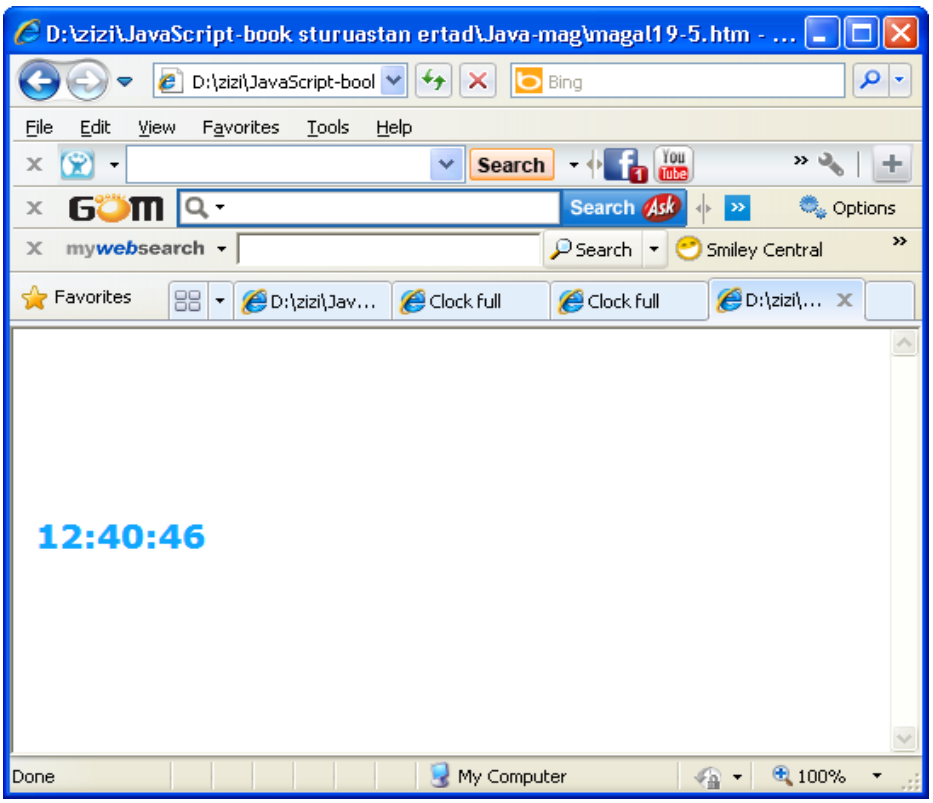


მაგალითი 5.

```
<HTML>
<HEAD>
<SCRIPT>
function showTime ( ) {
```

```
dayTwo = new Date ( );
hrNow = dayTwo . getHours ( );
mnNow = dayTwo . getMinutes ( );
scNow = dayTwo . getSeconds ( );
miNow = dayTwo . getTime ( );
if (hrNow == 0) {
hour = 0;
ap = " AM";
}  else if (hrNow <= 11)  {
ap = " AM";
hour = hrNow;
}  else if (hrNow == 12)  {
ap = " PM";
hour = 12;
}  else if (hrNow >= 13)  {
hour = (hrNow - 12);
ap = " PM";
}
if (hrNow >= 13)  {
hour = hrNow - 12;
}
if (mnNow <= 9)  {
min = "0" + mnNow;
}  else (min = mnNow)
if (scNow <= 9)  {
secs = "0" + scNow;
}  else  {
secs = scNow;
}
time = hour + ":" + min + ":" + secs ;
if  (document . all) {
document . all . clock . innerHTML = time;
document . all . clock1 . innerHTML = time;
}
if (document . layers)  {
document . clock . document . write ('<P style=\"font-size:10px;
font-weight:bold; font-family:Verdana; color:#a0a0a0;\">' + time +
'<\/P>');
```

```
document . clock1 . document . write ('<P style=\"font-size:10px; z-
index:-1; position:absolute; top:1; left:1; font-weight:bold; font-
family:Verdana; color:#11a6ff;\">' + time + '<\/P>');
document . clock . document . close ( );
document . clock1 . document . close ( );
}
setTimeout ('showTime()', 1000);
}
</SCRIPT>
<BODY onLoad="showTime ( )">
<DIV ID="clock" style="position: absolute; top: 115; left: 13; color:
#11a6ff; font-family: Verdana; font-size: 20px; font-weight: bold;
width: 110; height: 33">
</DIV>
<DIV ID="clock1" style="position: absolute; top: 115; left: 13;
color: #11a6ff; font-family: Verdana; font-size: 20px; font-weight:
bold; width: 110; height: 33">
</DIV>
</BODY>
</HTML>
```



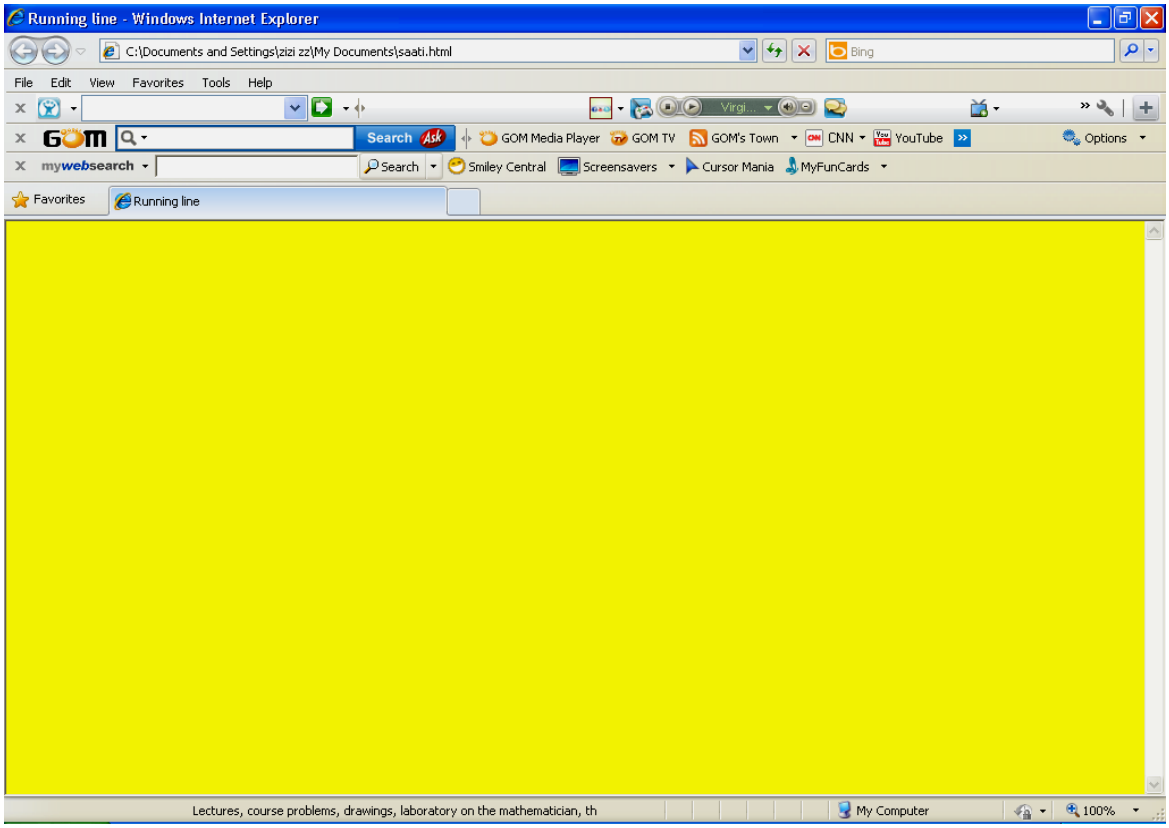
ეკრანზე გამონათდება ლურჯი ფერით ჩაწერილი დრო

მორბენალი სტრიქონი. მორბენალი სტრიქონის განთავსება სტატუსის ზოლში:

```
<HTML>
<HEAD>
<TITLE>Running line</TITLE>
<SCRIPT language="javascript">
function statusmessageobject(p,d) {
this.msg = message
this.out = " "
this.pos = position
this.delay = delay
this.i = 0
this.reset = clearmessage
}
function clearmessage() {
this.pos = position
}
var position = 100
var delay = 40
var message = "Lectures, course problems, drawings, laboratory on
      the mathematician, the physicist and" + " computer science"
var scroll = new statusmessageobject()
function scroller() {
for (scroll.i = 0;
scroll.i < scroll.pos;
scroll.i++) {
scroll.out += " "
}
if (scroll.pos >= 0)
scroll.out += scroll.msg
else
scroll.out = scroll.msg.substring(-scroll.pos,scroll.msg.length)
window.status = scroll.out
scroll.out = " "
scroll.pos--
if (scroll.pos < -(scroll.msg.length))
{ scroll.reset() }
```

```
setTimeout ('scroller()',scroll.delay) }
</SCRIPT>
</HEAD>
<BODY bgcolor="#FFFF00" onLoad="scroller()">
</BODY>
</HTML>
```

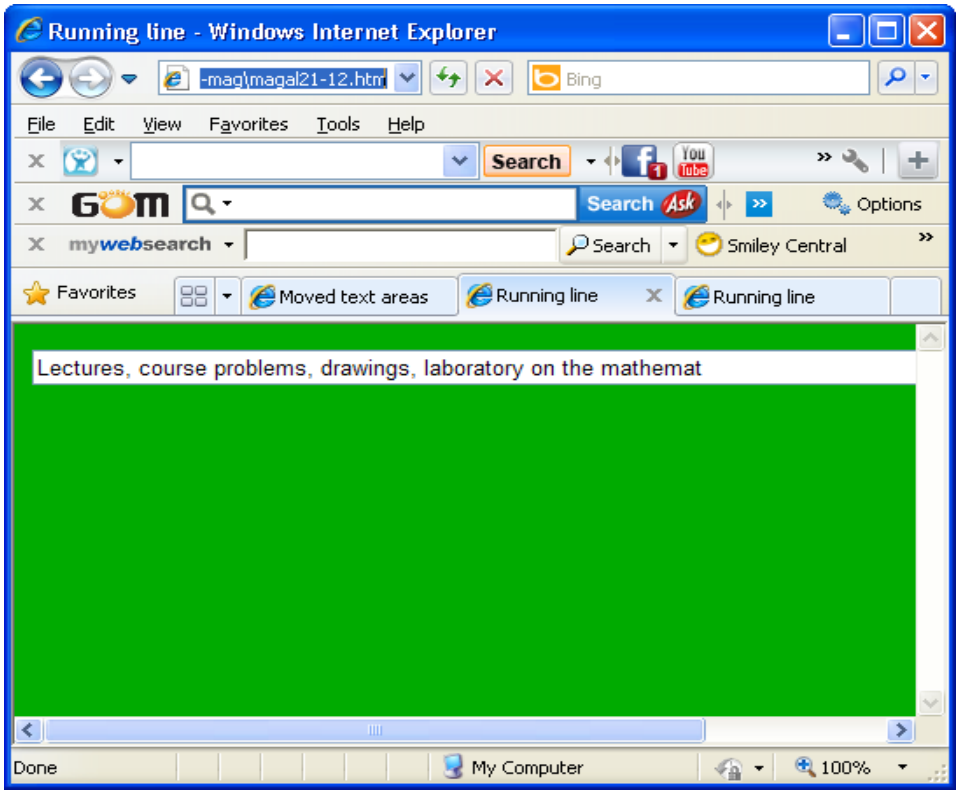
მოყვანილ მაგალითში ეკრანზე ამონათდება ყვითელი ფონის მქონე ფურცელი, სტატუსის ველში კი მორბენალი სტრიქონი, შესაბამისი ჩანაწერით.



მორბენალი სტრიქონის განთავსება **FORM** ველში:

```
<HTML>
<HEAD>
<TITLE>Running line</TITLE>
</HEAD>
<SCRIPT language="javascript">
var line = "Lectures, course problems, drawings, laboratory on the
      mathematician, the physicist and computer science"
var speed=100;
var i=0;
```

```
function m_line() {
if(i++<line.length) {
document.cit.forum.value=line.substring(0,i);
}
else{
document.cit.forum.value=" ";
i=0;
}
setTimeout('m_line()',speed);
}
</SCRIPT>
<BODY bgcolor=00ab00>
<CENTER>
<FORM name=cit>
<INPUT type=text size=100 name=forum>
</FORM>
</CENTER>
<SCRIPT language="JavaScript">
m_line ( );
</SCRIPT>
</BODY>
</HTML>
```



40. ელემენტთა მოძრაობა

ბრაუზერის ფანჯარაში დოკუმენტის ხილული ელემენტების გადაადგილება ხდება პოზიციის განმსაზღვრელი სტილის ცხრილის **top** და **left** პარამეტრების მნიშვნელობის შეცვლით. ამ პარამეტრების მითითება შეგვიძლია **STYLE** ატრიბუტში ან **<STYLE>** ტეგში იმ ტეგებისათვის, რომლებიც იძლევა ხილულ ელემენტებს (გამოსახულება, ტექსტი, მითითება, ღილაკი და სხვა). შემდეგ სცენარში უნდა განისაზღვროს ამ პარამეტრების შეცვლის ხერხები. შეიძლება ელემენტი ვაიძულოთ გადაადგილდეს მუდმივად, მოცემული დროის განმავლობაში ან ხდომილობის მიხედვით.

ჩვეულებრივ კოორდინატების გამოსათვლელად ციკლის ოპერატორი არ გამოიყენება, ვინაიდან, ვიდრე ციკლი სრულდება სცენარის სხვა გამოსახულება არ მუშაობს. გამოთვლითი პროცესის პარალელურად წარმართვისათვის გამოიყენება **setInterval ( )** და **setTimer ( )** მეთოდები.

განვიხილოთ მოცემული ტრაექტორიით მოძრაობა. სცენარის სქემას, რომელიც ახორციელებს ელემენტის უწყვეტ გადაადგილებას, აქვს შემდეგი სახე:

```
function init_move ( ) {
. . .
seInterval ("move ( ) ",  შეყოვნება)
}
function move ( ) {
. . .
}
init_move ( )
```

ამგვარად, ჩვენ ვქმნით ორ ფუნქციას, რომელთა სახელები შეიძლება იყოს ნებისმიერი. პირველი ფუნქცია იწვევს საწყისი მონაცემების მომზადებას და იძახებს **setInterval ( )** მეთოდს. მისი პირველი პარამეტრი არის ბრჭყალებში მოთავსებული მეორე ფუნქციის სახელი **move ( )**, რომელიც ცვლის ელემენტის კოორდინატებს. მეორე პარამეტრი არის შეყოვნების დრო მილიწამებში. ამის გამო, ელემენტის კოორდინატები თანდათან იცვლება და იქმნება მოძრაობის ეფექტი.

მოძრაობის სიჩქარე დამოკიდებულია კოორდინატების ნაზრდზე და შეყოვნების დროზე. მოძრაობის დასაწყებად საკმარისია პირველი ფუნქციის **init_move ( )**-ს გამოძახება.

წრფივი მოძრაობა. მაგალითის სახით განვიხილოთ გამოსახულების წრფივად გადაადგილების სცენარი.

```
<HTML>
<IMG ID="myimg" SRC = "C:\Documents and Settings\USER\
     My Documents\ My Pictures\Blue hills.jpg"
STYLE= "position:absolute; top:10; left:20">
<SCRIPT>
function init_move ( ) {
dx = 8
dy = 3
setInterval ("move ( )", 200)
move ( )
}
function move ( ) {
var y = parseInt(document.all.myimg.style.top)
var x = parseInt(document.all.myimg.style.left)
document.all.myimg.style.top = y + dy
document.all.myimg.style.left = x + dx
}
init_move ( )
</SCRIPT>
</HTML>
```

მოვახდინოთ ზემოთ განხილული კოდის გაუმჯობესება: გავაკეთოთ ისე, რომ გადასაადგილებელი ობიექტის იდენტიფიკატორი, აგრეთვე კოორდინატის ნაზრდი და შეყოვნების დრო მივაწოდოთ **init_move ( )** ფუნქციას პარამეტრის სახით. შედეგად მივიღებთ ნებისმიერი ელემენტის გადაადგილების უნივერსალურ ფუნქციას. შესაბამის კოდს ექნება შემდეგი სახე:

```
function init_move (xid, dx, dy) {
var prmstr = " ' " + xid + " '," + dx + "," + dy
prmstr = "move(" + prmstr + ")"
setInterval (prmstr, 100)
```

```
move ( )
}
function move (xid, dx, dy) {
y = parseInt(document.all[xid].style.top)
x = parseInt(document.all[xid].style.left)
document.all[xid].style.top = y + dy
document.all[xid].style.left = x + dx
}
init_move ("myimg", 10, 5)
```

მოძრაობის შეჩერება. ზემოთ განხილულ მაგალითში დოკუმენტის ელემენტის გადაადგილება მიმდინარეობს უწყვეტად, ვინაიდან ის მოძრაობს წრფეზე. ამიტომ უნდა მოხდეს **move ( )** ფუნქციის ისეთი გაუმჯობესება, რათა მან ელემენტის შეჩერება მოახდინოს ბრაუზერის ფანჯრის რომელიმე არეში. ზოგჯერ შეიძლება საჭირო გახდეს ელემენტმა გადაკვეთოს ბრაუზერის ფანჯრის საზღვარი და მეტი აღარ დაბრუნდეს. ამისათვის, გამოიყენება მეთოდი **clearInterval ( )**.

მოძრაობის დაწყებისა და გაჩერების სცენარი შეიძლება შემდეგნაირად გამოიყურებოდეს:

```
function init_move (xid, dx, dy) {
var prmstr = " ' " + xid + " '," + dx + "," + dy
prmstr = "move(" + prmstr + ")"
id_move = setInterval (prmstr, 100)
}
function move (xid, dx, dy) {
y = parseInt(document.all[xid].style.top)
x = parseInt(document.all[xid].style.left)
document.all[xid].style.top = y + dy
document.all[xid].style.left = x + dx
if (parseInt(document.all[xid].style.left) > 650)
    clearInterval(id_move)
}
init_move ("myimg", 10, 5)
```

მოძრაობა ელიფსზე. განვიხილოთ მოძრაობის ორგანიზაცია შეკრულ ტრაექტორიაზე. მაგალითად, მოძრაობა ელიფსზე მოიცემა რამდენიმე პარამეტრით, როგორიცაა დიდი და მცირე ნახევარღერძი,

ცენტრის მდებარეობა, მობრუნების კუთხე ჰორიზონტალის მიმართ, გადაადგილების კუთხური სიჩქარე და სხვა. კერძო შემთხვევაში, როდესაც ერთ-ერთი ნახევარღერძი ნულის ტოლია, ტრაექტორია გარდაიქმნება წრფედ. ამასთან, მოძრაობა ხან ერთი და ხან მეორე მიმართულებით მოხდება. ცხადია, ნახევარღერძების ტოლობის შემთხვევაში მივიღებთ მოძრაობას წრეზე.

ქვემოთ მოყვანილ მაგალითში განსაზღვრულია ორი ფუნქცია, რომელიც მოძრაობას იძლევა ელიფსზე:

```
function ellipse (xid, alpha, a, b, omega, x0, y0, ztime, dt) {
if (!alpha) alpha = 0
if (!a) a = 0
if (!b) b = 0
if (!omega) omega = 0
if (!x0) x0 = 0
if (!y0) y0 = 0
if (!ztime) ztime = 0
if (!dt) dt = 0
var t=-ztime
setInterval("move(' " + xid + " '," + alpha + "," + a + "," + b + "," +
    omega + "," + x0 + "," + y0 + "," + ztime + "," + dt + ")",
    ztime*1000)
}
function move (xid, alpha, a, b, omega, x0, y0, ztime, dt) {
t+=ztime
var x = a*Math.cos((omega*t + dt)*Math.PI/180)
var y = b*Math.sin((omega*t + dt)*Math.PI/180)
var as = Math.sin(alpha*Math.PI/180)
var ac = Math.cos(alpha*Math.PI/180)
document.all[xid].style.top = -x*as + y*ac + y0
document.all[xid].style.left = x*ac + y*as + x0
}
```

განვიხილოთ სცენარის შემცველი HTML-კოდი, რომელიც ჩატვირთავს ორ გამოსახულებას და ისინი ელიფსურ ტრაექტორიაზე იმოძრავებს:

```
<HTML>
<SCRIPT>
```

```
function ellipse (xid, alpha, a, b, omega, x0, y0, ztime, dt) {
. . .
}
function move (xid, alpha, a, b, omega, x0, y0, ztime, dt) {
. . .
}
var xstr='<IMG ID="myimg1" SRC = "C:\Documents and
    Settings\USER\ My Documents\My Pictures\Blue
    hills.jpg"><IMG ID="myimg2" SRC = "C:\Documents and
    Settings\USER\My Documents\My Pictures \Sunset.jpg">'
document.write(xstr)
ellipse ("myimg1", 60, 100, 30, 10, 170, 300, 0.2, 0)
ellipse ("myimg2", 80, 100, 130, 10, 140, 300, 0.2, 0)
</SCRIPT>
</HTML>
```

თუ **ellipse ( )** და **move ( )** ფუნქციების განმსაზღვრელ კოდს შევინახავთ ფაილში, მაგალითად, **ellipse.js**, მაშინ ზემოთ განხილული კოდი შეიძლება შემდეგნაირად ჩაიწეროს:

```
<HTML>
<SCRIPT SRC = "ellipse.js"></SCRIPT>
<SCRIPT>
var xstr='<IMG ID="myimg1" SRC = "C:\Documents and
    Settings\USER\ My Documents\My Pictures\Blue
    hills.jpg"><IMG ID="myimg2" SRC = "C:\Documents and
    Settings\USER\My Documents\My Pictures \Sunset.jpg">'
document.write(xstr)
ellipse ("myimg1", 60, 100, 30, 10, 170, 300, 0.2, 0)
ellipse ("myimg2", 80, 100, 130, 10, 140, 300, 0.2, 0)
</SCRIPT>
</HTML>
```

მოძრაობა ნებისმიერ მრუდზე. განვიხილოთ ხილული ელემენტის ნებისმიერ მრუდზე მოძრაობის მაგალითი, მოცემული ერთი ცვლადის გამოსახულებით. უფრო ზუსტად, გამოსახულება შეიცავს ერთ ცვლადს, მაგრამ ორ გამოსახულებას, რომელიც აღწერს ელემენტის ჰორიზონტალური და ვერტიკალური კოორდინატების ცვლილებას. **x**

ცვლადი შეიძლება განვიხილოთ როგორც მოძრაობის დამოუკიდებელი ცვლადი (მაგალითად, დრო).

ამგვარად, მოძრაობის გამომწვევ **curvemove ( )** ფუნქციას ექნება სამი სტრიქონული და ერთი რიცხვითი პარამეტრი. სტრიქონული პარამეტრები იქნება: ელემენტის **ID** იდენტიფიკატორის მნიშვნელობა, ჰორიზონტალური და ვერტიკალური კოორდინატების ცვლილების გამოსახულებები. რიცხვითი პარამეტრი კი იქნება ელემენტის კოორდინატთა გადაანგარიშების დრო.

```
function curvemove (xid, yexpr, xexpr, ztime) {
if (!xid) return null
if (!yexpr) yexpr = "y"
if (!xexpr) xexpr = "x"
if (!ztime) ztime = 100
x=0
setInterval("move(' " + xid + " ', ' " + yexpr + " ', ' " + xexpr " ')",
    ztime)
}
function move (xid, yexpr, xexpr) {
x ++
document.all[xid].style.top = eval(yexpr)
document.all[xid].style.left = eval(xexpr)
}
```

გადასაადგილებელი ელემენტის საწყისი პოზიციის განსაზღვრას არა აქვს არავითარი მნიშვნელობა, ვინაიდან **curvemove ( )** ფუნქციის გამოძახების შემთხვევაში ელემენტი განთავსდება წერტილში, რომლის კოორდინატები განისაზღვრება **xexpr** და **yexpr** გამოსახულების მნიშვნელობით **x=0**-სათვის. ქვემოთ მოყვანილია შესაბამისი მაგალითი:

```
<HTML>
<IMG ID="myimg" SRC = "C:\Documents and Settings\USER\My
    Documents\ My Pictures\Blue hills.jpg"
    style="position:absolute">
<SCRIPT>
function curvemove (xid, yexpr, xexpr, ztime) {
.  .  .
}
```

```
function move (xid, yexpr, xexpr) {
.  .  .
}
curvemove ("myimg", "100 + 50*Math.sin(0.3*x)", "50+x", 100)
</SCRIPT>
</HTML>
```

ამ მაგალითში გამოსახულება გადაადგილდება სინუსოიდის გასწვრივ, რომლის ამპლიტუდაა **50** პიქსელი და ჰორიზონტალური სიჩქარე **10** პიქსელი წამში. გრაფიკული ობიექტის საწყისი კოორდინატებია **100** და **50** პიქსელი ვერტიკალზე და ჰორიზონტალზე შესაბამისად.

41. ელემენტთა გადაადგილება მაუსით

განვიხილოთ ხილული ელემენტების მაუსის საშუალებით გადაადგილების ამოცანა. ასეთი ელემენტები შეიძლება იყოს გრაფიკული ობიექტები, ღილაკები, ტექსტური არეები, ცხრილები, მცურავი ჩარჩოები და სხვა.

გრაფიკული ობიექტების გადაადგილება. მაუსით გამოსახულების გადაადგილება შეიძლება განხორციელდეს სხვადასხვა გზით. განვიხილოთ ერთ-ერთი მათგანი: მომხმარებელი ცდილობს გამოსახულების გადაადგილებას მაუსის საშუალებით; შემდეგ მან უნდა აუშვას ხელი მაუსის ღილაკს და გადაიტანოს გამოსახულება საჭირო ადგილზე (ამასთან, მას შეუძლია ეჭიროს ხელი მაუსის ღილაკზე ან არა); საჭირო ადგილზე გაჩერების შემდეგ მომხმარებელი ხელს უშვებს მაუსის კლავიშს ან აწკაპუნებს მას რათა შეწყვიტოს გამოსახულების გადაადგილება. ქვემოთ მოყვანილია ამ ალგორითმის კოდი:

```
<HTML>
<HEAD><TITLE>The moved image</TITLE></HEAD>
<BODY id = "mybody">
<IMG ID="myimg" SRC = "C:\Documents and Settings\USER\
     My Documents\ My Pictures\Blue hills.jpg" ondragstart =
     "drag ( )" style="position:absolute; top:10; left:10">
```

```
</BODY>
<SCRIPT>
flag = false
var id_img =""
function drag ( ) {
flag = !flag
id_img = event.srcElement.id
}
    function mybody.onmousemove( ) {
    if (flag) {
    document.all[id_img].style.top = event.clientY
    document.all[id_img].style.left = event.clientX
    }
}
    function mybody.onmouseup ( ) {
    flag = false
    }
</SCRIPT>
</HTML>
```

აქ ფუნქცია **drag ( )**, რომელიც ამუშავებს **ondragstart** ხდომილობას (გადათრევის მცდელობა), დააყენებს **flag** ცვლად-ტრიგერს და არკვევს, თუ ვინ არის ამის ინიციატორი. **flag** ცვლადის მნიშვნელობა განსაზღვრავს, შეიძლება თუ არა ელემენტის გადაადგილება (თუ **flag=false**, მაშინ გადაადგილება არ შეიძლება). ამ მაგალითში გადაადგილების ინიციატორი არის მხოლოდ ერთი ელემენტი.

ახლა განვიხილოთ სცენარის კოდი, სადაც მოცემულია ორი გადასაადგილებელი ელემენტი:

```
<HTML>
<HEAD><TITLE>The moved image</TITLE></HEAD>
<BODY id = "mybody">
<IMG ID="myimg" SRC = "C:\Documents and Settings\USER\
     My Documents\ My Pictures\Blue hills.jpg" ondragstart =
     "drag ( )" style="position:absolute; top:10; left:10">
<IMG ID="my2" SRC = "C:\Documents and Settings\USER\
     My Documents\ My Pictures\Sunset.jpg" ondragstart =
     "drag ( )" style="position:absolute; top:10; left:10">
```

```
</BODY>
<SCRIPT>
flag = false
var id_img =""
    function drag ( ) {
    flag = !flag
    id_img = event.srcElement.id
    }
    function mybody.onmousemove( ) {
    if (flag) {
    document.all[id_img].style.top = event.clientY
    document.all[id_img].style.left = event.clientX
    }
}
    function mybody.onmouseup ( ) {
    flag = false
    }
</SCRIPT>
</HTML>
```

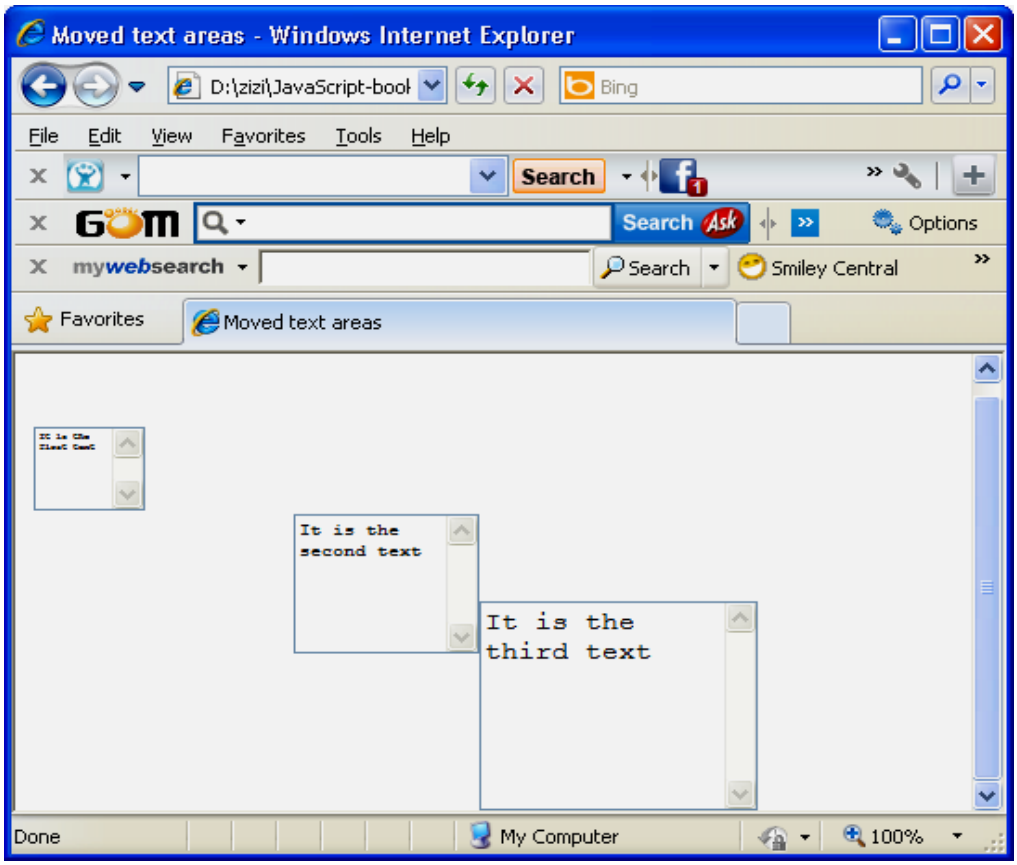
ტექსტური არის გადაადგილება. ქვემოთ მოყვანილია დოკუმენტის მაგალითი, რომელშიც შეიძლება ტექსტური არის გადაადგილება. ამასთან არის და ტექსტის შრიფტის ზომა განისაზღვრება ვერტიკალური კოორდინატის საშუალებით. ასე იქმნება პერსპექტივის ეფექტი (ახლოს და შორს). რაც ზემოთაა ტექსტური არე, მით შორსაა ის და პირიქით. თუ გვსურს ტექსტის მოახლოება საკმარისია მისი ქვემოთ გადანაცვლება. გამოსახულებისაგან განსხვავებით, ტექსტური არის გადასაადგილებლად მაუსით ორჯერ უნდა დავაწკაპუნოთ ამ არეზე.

```
<HTML>
<HEAD><TITLE>Moved text areas</TITLE></HEAD>
<BODY id = "mybody" background ="C:\Documents and Settings
    \USER\ My Documents\My Pictures\Blue hills.jpg">
<TEXTAREA ID = "t1" ondblclick = "drag ( )" STYLE =
    "position: absolute; top:50; left:10; font-size:large">It is the
    first text </TEXTAREA>
```

```
<TEXTAREA ID = "t2" ondblclick = "drag ( )" STYLE =
      "position: absolute; top:100; left:150;">It is the second text
      </TEXTAREA>
<TEXTAREA ID = "t3" ondblclick = "drag ( )" STYLE =
      "position: absolute; top:150; left:250;">It is the third text
      </TEXTAREA> </BODY>
<SCRIPT>
resizetext ( )
var flag = false
var id_img = ""
function drag ( ) {
flag = !flag
id_img = event . srcElement . id
}
function mybody . onmousemove( ) {
if (flag) {
      document . all[id_img] . style . top = event . clientY
      document . all[id_img] . style . left = event . clientX
      resizetext ( )
}
}
function mybody . onmouseup ( ) {
flag = false
}
function resizetext ( ) {
var y, size, idimg, idtext
for (i = 0; i < document . all . length; i++) {
      if (document . all[i] . tagName == 'TEXTAREA') {
         idtext = document . all[i] . id
         y = parseInt(document . all[idtext] . style . top)
         size = Math . min(y, 800)
         size = Math . max(size, 60)
         document . all[idtext] . style . width = size
         document . all[idtext] . style . height = 0.8*size
         document . all[idtext] . style . zIndex = y
         document . all[idtext] . style . fontSize = Math . max(2, y/10)
      }
}
```

```
}
</SCRIPT>
</HTML>
```

მივიღებთ:



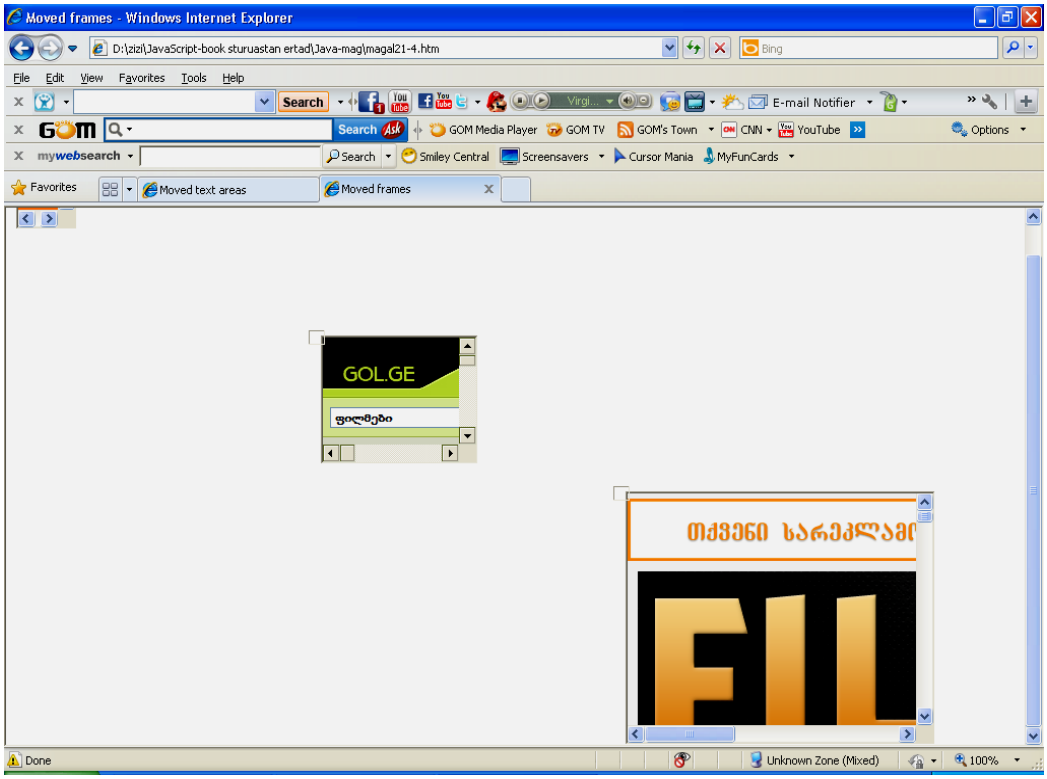
"მცურავი" ჩარჩოს გადაადგილება. თუ ზემოთ განხილული კოდის მოდიფიცირებას მოვახდენთ, მაშინ შეიძლება მივიღოთ დოკუმენტი მრავალი "მცურავი" ჩარჩოთი, რომლებშიც HTML-დოკუმენტებია ჩატვირთული. მაგრამ ამ შემთხვევაში წარმოიშვება მცირე პრობლემა: როგორ გადავცეთ "მცურავ" ჩარჩოს სიგნალი მისი გადაადგილების შესახებ, ვინაიდან მაუსის მასზე ზემოქმედებით ეს სიგნალი გადაეცემა არა ჩარჩოს, არამედ მასში გახსნილ დოკუმენტს. ქვემოთ მოყვანილ მაგალითში ყოველი ჩარჩოს წინ განთავსებულია მცირე ზომის გამოსახულება, რომელიც აღიქვამს ხდომილობას როგორც ნებართვას ჩარჩოს გადაადგილების შესახებ. ამგვარად, გამოსახულება მიიღებს ხდომილობას, რომლის დამამუშავებელი ბრძანება შეცვლის ჩარჩოსა და გამოსახულების პარამეტრებს.

```
<HTML>
<HEAD><TITLE>Moved frames</TITLE></HEAD>
```

```
<BODY id = "mybody" background ="C:\Documents and Settings
    \USER\ My Documents\My Pictures\Blue hills.jpg">
</BODY>
<SCRIPT>
var ahref = new Array ( )
ahref[0] = "http://www.internet.ge"
ahref[1] = "http://www.gol.ge"
ahref[2] = "http://www.file.ge"
var xstr = ""
for (i=0; i<ahref.length; i++) {
xstr+= '<IFRAME width=200 ID="fr' + i + ' " SRC = " '+ahref[i] + '
    " STYLE = "position:absolute; top:' + (5 + 150*i)+'; left:' +
    (10 + 300*i) + ' "></IFRAME>'
xstr+= '<IMG ondblclick="drag( )" width=16 height=14 SRC =
    "C:\Documents and Settings\USER\My Documents\My Pictures
    \ AG00090_.gif" id="im' + i + ' " STYLE =
    "position:absolute">'
}
document.write(xstr)
flag = false
id_img = ""
resizeframe ( )
function drag ( ) {
flag = !flag
id_img = event . srcElement . id
id_img = "fr" + id_img.substr(2)
}
function mybody . onmousemove( ) {
if (flag) {
    document . all[id_img] . style . top = event . clientY
    document . all[id_img] . style . left = event . clientX
    resizeframe ( )
}
}
function mybody . onmouseup ( ) {
flag = false
}
function resizeframe ( ) {
```

```
var y, size, idimg, idfr
for (i = 0; i < ahref . length; i++) {
      idfr = "fr" + i
         y = parseInt(document . all[idfr] . style . top)
         size = Math . min(y, 800)
         size = Math . max(60, size)
         document . all[idfr] . style . width = size
         document . all[idfr] . style . height = 0.8*size
         document . all[idfr] . style . zIndex = y
         idimg = "im" + i
         document . all[idimg] . style . top = y-5
         document . all[idimg] . style . left =
              parseInt(document . all[idfr] . style . left) - 12
         document . all[idimg] . style . zIndex =
              document . all[idfr] . style . zIndex
}
}
</SCRIPT>
</HTML>
```

შედეგი:



42. გეომეტრიული ფიგურების დაპროექტება

JavaScript-ში არ არის სპეციალური საშუალებები ნებისმიერი ტიპის ხაზის დასახაზად. თუ ჩვენ გვესაჭიროება ბრაუზერის ფანჯარაში გამოვსახოთ მართკუთხედი ან ჰორიზონტალური ხაზი, ამისათვის შეიძლება შესაბამისად გამოვიყენოთ <TABLE> და <HR> HTML-ტეგები. მაგრამ როგორ უნდა მოვიქცეთ თუ საჭიროა დახრილი წრფის, წრის ან განტოლებით მოცემული მრუდის დახაზვა?

ამ ამოცანის გადაწყვეტა მარტივად შეიძლება. ეკრანზე უნდა მოხდეს **1×1** პიქსელის ზომის გამოსახულების ფონისაგან განსხვავებული ფერით გამოტანა. შემდეგ უნდა მოხდეს ამ გამოსახულების რამდენიმეჯერ გამეორება კოორდინატების შესაბამისად, რომელთა მიწოდება ხდება **<IMG>** ტეგის **STYLE** ატრიბუტის **top** და **left** პარამეტრების მეშვეობით.

წრფე. უპირველეს ყოვლისა ვნახოთ როგორ უნდა დავსვათ წერტილი. მრავალი ასეთი წერტილისაგან შედგება ნებისმიერი წირი, რომელიც გვჭირდება ბრაუზერის ფანჯარაში გამოსასახად. ამისათვის შეიძლება გამოყენებულ იქნეს შემდეგი ტეგი:

```
<IMG SRC = "point.bmp" STYLE = "position:absolute; top:y;
    left:x">
```

სადაც **point.bmp** – ერთ პიქსელის შემცველი გრაფიკული ფაილის სახელია; **y**, **x** – პიქსელებში მოცემული რიცხვი, რომელიც უჩვენებს გრაფიკული ფაილის მდებარეობას. **1×1** პიქსელის ზომის წერტილის გამოსახულება შეიძლება შეიქმნას ნებისმიერი გრაფიკული რედაქტორის საშუალებით. ეკონომიურობის თვალსაზრისით უმჯობესია მისი შენახვა **bmp** ფორმატში.

ეკრანზე წერტილის გამოსახულების ზომების მისანიჭებლად უნდა გამოვიყენოთ ატრიბუტები **WIDTH** და **HEIGHT** (სიგანე და სიმაღლე):

```
<IMG SRC = "point.bmp" STYLE = "position:absolute; top:y;
    left:x" WIDTH = n HEIGHT =n>
```

ამ ატრიბუტების ერთნაირი მნიშვნელობა წერტილს მისცემს **n×n** პიქსელის ზომის კვადრატის ფორმას. ამასთან, წერტილი საწყისი ზომით **1×1** პიქსელი უბრალოდ გაიწელება. ამგვარად, შეგვიძლია ერთი

წერტილის ზომის ცვლილებით განვსაზღვროთ ხაზის სისქე. ჯერ განვიხილოთ წრფის დამხაზავი ფუნქციის საწყისი ვერსია, მოცემული საწყისი და საბოლოო **x1**, **y1**, **x2**, **y2** კოორდინატებით. ამას გარდა ფუნქციას უნდა მივაწოდოთ წრფის სისქის მაჩვენებელი **n** რიცხვითი პარამეტრი:

```
function line(x1, y1, x2, y2, n) {
var clinewidth = " WIDTH =" + n + " HEIGHT =" + n
var xstr = ""
var xstr0 = '<IMG SRC = "point.bmp" ' + clinewidth + 'STYLE =
     "position:absolute;'
var k = (y2 - y1)/(x2 - x1)
var x = x1
while (x <= x2) {
     xstr += xstr0 + 'top:' + (y1 + k*(x - x1)) + '; left:' + x + '">'
     x++
}
document.write(xstr)
}
```

მოცემულ ფუნქციაში ციკლის პარამეტრი არის ჰორიზონტალური კოორდინატა, რომლის მნიშვნელობა ენიჭება **left** პარამეტრს, ხოლო ვერტიკალური კოორდინატა (**top**) გამოითვლება დახრის კოეფიციენტის მიხედვით. ეს ფუნქცია უფრო კარგ შედეგს იძლევა, როდესაც წრფე გავლებულია მარცხნიდან მარჯვნივ და დახრის კუთხე ნაკლებია $\mathbf{45^0}$-ზე. დახრის უფრო მეტი კუთხის შემთხვევაში წრფე ხდება წყვეტილი. ამას გარდა, ეს ფუნქცია ხატავს მხოლოდ ისეთ წრფეებს, რომელთა საწყისი წერტილი მდებარეობს უფრო მარცხნივ და ზემოთ, ვიდრე წრფის ბოლო.

ახლა მოვახდინოთ ამ ფუნქციის მოდერნიზაცია:

```
function line(x1, y1, x2, y2, n) {
/*   x1, y1 – წრფის დასაწყისი
     x2, y2 –  წრფის დასასრული
     n –  წრფის სისქე  */
var clinewidth = ""
if (!n)
     clinewidth = " WIDTH =" + n + " HEIGHT =" + n
var xstr = ""
```

```
var xstr0 = '<IMG SRC = "point.bmp" ' + clinewidth + ' STYLE =
      "position:absolute;'
var x, k, direct
var vertical = Math.abs(y2-y1) > Math.abs(x2-x1)
if (vertical) {
      direct = (y2 > y1)
      x = y1
      k = (x2 - x1)/(y2 - y1)
} else {
      direct = (x2 > x1)
      if (direct) x = x1
      else x = x2
      k = (y2 - y1)/(x2 - x1)
}
while (true) {
      if (!vertical) {
      xstr += xstr0 + 'top:' + (y1 + k*(x - x1)) + '; left:' + x + '">'
      if (x == x2) break
      if (direct) x++
      else x--
   } else {
      xstr += xstr0 + 'left:' + (x1 + k*(x - y1)) + '; top:' + x + '">'
      if (x == y2) break
      if (direct) x++
      else x--
   }
}
document.write(xstr)
}
```

შემდგომში შეიძლება დაიხაზოს წყვეტილი ხაზი და აგრეთვე მივცეთ სხვადასხვა ფერი, რისთვისაც საჭიროა შეირჩეს შესაბამისი გრაფიკული ფაილი წერტილის გამოსახულებით. შეიძლება წინასწარ მომზადდეს სხვადასხვა ფერის წერტილების ფაილები. ხოლო წყვეტილი ხაზის გასავლებად საჭიროა პერიოდულად არ გამოვსახოთ მოცემული რაოდენობის წერტილები.

მრუდი. შევქმნათ პროგრამა, რომელიც დახაზავს ნებისმიერ წირს. მისი დაწერის დროს გამოვიყენებთ წრფის პროგრამის დაწერის დროს შეძენილ გამოცდილებას. დავუშვათ, რომ გამოსახულება, რომლითაც მოცემულია მრუდი შეიცავს **x** ცვლადს. ამ შემთხვევაში მრუდის წერტილის ჰორიზონტალური კოორდინატა გამოითვლება როგორც **x** ცვლადის მნიშვნელობა გარკვეულ საზღვრებში. ვერტიკალური კოორდინატის გამოსათვლელად გამოიყენება **eval ( )** ფუნქცია.

მრუდის დასახაზ ფუნქციას უნდა მიეწოდოს შემდეგი პარამეტრები: წერტილის გამოსახულების გრაფიკული ფაილის სახელი, მრუდის გამოსახულება, წირის საწყისი კოორდინატა, ხაზის წერტილების რაოდენობა, ხაზის სისქე და წყვეტის სიგრძე (თუ საჭიროა წყვეტილი ხაზის გავლება).

```
function curve(pict_file, yexpr, x0, y0, t, n, s) {
/*    pict_file – გრაფიკული ფაილის სახელი
      yexpr – x  ცვლადიანი გამოსახულება
      x0, y0 –  მრუდის დასაწყისის კოორდინატა
      t –  მრუდში წერტილების რაოდენობა (x  ცვლადის
      მნიშვნელობა)
      n –  წრფის სისქე
      s  –  წყვეტისა და პაუზის ინტერვალი  */
if (!yexpr) return null
if (!pict_file) pict_file = "C:\Documents and Settings\User\My
      Documents\ My Pictures\point.bmp"
if (!s) s = 0
if (!t) t = 0
var clinewidth = ""
if (!n)
      clinewidth = "WIDTH =" + n + " HEIGHT =" + n
var x
xstr0 = '<IMG SRC ="'+ pict_file + ' " ' + clinewidth + ' STYLE =
      "position:absolute; top: '
xstr = ""
var i = 0
draw = true
for (x = 0; x < t; x++) {
```

```
            if (draw)
                xstr += xstr0 + (y0 + eval(yexpr)) + '; left:' + (x0 + x) + ' ">'
            if (i > s&&s > 0) {
                draw = !draw
                i = 0
            }
            i++
        }
        document.write(xstr)
    }
```

ქვემოთ მოყვანილია ამ ფუნქციაზე მიმართვის რამდენიმე მაგალითი:

```
curve("","200-0.01*x*x",1,100,200,3)
curve("","100.0*Math.sin(6/250*(x))",30,120,300,12,0)
curve("","0.001*x*x*(x-75)",100,80,200,6,0)
```

ახლა მოვახდინოთ ამ ფუნქციის მოდიფიცირება ისე, რომ დაიხაზოს შეკრული წირი (მაგალითად, წრე, ელიფსი). ამისათვის, აუცილებელია მივაწოდოთ როგორც ვერტიკალური, ასევე ჰორიზონტალური კოორდინატის ცვლილების გამოსახულება. მათემატიკაში ამ მეთოდს ეწოდება პარამეტრული მეთოდი.

```
    function curve(pict_file, yexpr, xexpr, x0, y0, t, n, s) {
/*  pict_file – გრაფიკული ფაილის სახელი
    yexpr – x  ცვლადიანი გამოსახულება ვერტიკალური
    კოორდინატისათვის
    xexpr – x  ცვლადიანი გამოსახულება ჰორიზონტალური
    კოორდინატისათვის
    x0, y0 –  მრუდის დასაწყისის კოორდინატა
    t –  მრუდში წერტილების რაოდენობა (x  ცვლადის მნიშვნელობა)
    n –  წრფის სისქე
    s  –  წყვეტისა და პაუზის ინტერვალი  */
        if (!yexpr) return null
        if (!xexpr) xexpr = "x"
        if (!pict_file) pict_file = "C:\Documents and Settings\User\My
            Documents\ My Pictures\point.bmp"
        if (!s) s = 0
```

```
if (!t) t = 0
var clinewidth = ""
if (!n)
      clinewidth = "WIDTH =" + n + " HEIGHT =" + n
var x
xstr0 = '<IMG SRC =" '+ pict_file + ' " ' + clinewidth + ' STYLE =
      "position:absolute; top: '
xstr = ""
var i = 0
draw = true
for (x = 0; x < t; x++) {
      if (draw)
         xstr += xstr0 + (y0 + eval(yexpr)) + '; left:' + (x0 +
      eval(xexpr)) + ' ">'
      if (i > s&&s > 0) {
           draw = !draw
           i = 0
      }
      i++
   }
   document.write(xstr)
}
```

ქვემოთ მოყვანილია ე. წ. ლისაჟუს ფიგურის რამდენიმე მაგალითი:

```
curve("","80*Math.sin(6/25*x)", "60*Math.cos(6/25*x+1)",
      250,100,300,2,0)
curve("","60*Math.sin(6/25*x)", "60*Math.cos(6/25*x)",
      100,70,300,2,0)
curve("","80*Math.sin(6/25*x)", "80*Math.cos(6/50*x)",
      100,200,600,6,0)
curve("","80*Math.sin(6/250*x)", "80*Math.cos(6/75*x+1)",
      310,250,400,2,0)
```

ეს ფიგურები მიიღება იმ შემთხვევაში თუ კოორდინატების გამოსახულება წარმოადგენს სინუსს და კოსინუსს.

გამოსახულებებით მოცემული გრაფიკული დამოკიდებულება. პრინციპში ეს საკითხი ჩვენ უკვე ნაწილობრივ განვიხილეთ, რადგანაც

წინა თავში შევისწავლეთ ერთ ცვლადზე დამოკიდებული მრუდის აგება, ასევე წრფის აგება, რომელთა ცოდნა აუცილებელია კოორდინატთა სისტემის ასაგებად. ახლა საკმარისია მხოლოდ მათი გაერთიანება. მაგრამ ამ შემთხვევაში წინასწარ უნდა მოხდეს შემდეგი დამატებითი ამოცანების გადაწყვეტა: პირველ რიგში, უნდა განისაზღვროს როგორ მოხდება კოორდინატთა ღერძების გადაკვეთა და რომელი კვადრანტები უნდა გამოჩნდეს. მეორე, უნდა განისაზღვროს თითოეული ღერძის სიგრძე, რომელიც დამოკიდებული იქნება გამოსახულების მაქსიმალურ და მინიმალურ მნიშვნელობაზე. და ბოლოს, მოცემული უნდა იყოს ღერძებზე ციფრების დაწერის წესი. ამ ამოცანის გადაწყვეტის პროგრამის კოდი იქნება საკმაოდ მოცულობითი, ვიდრე ზემოთ განხილული კოდები.

მასივებით მოცემული გრაფიკული დამოკიდებულება. როგორც ცნობილია, ორ სიდიდეს შორის დამოკიდებულება ორი სვეტისაგან შემდგარი ცხრილის სახით შეიძლება იყოს მოცემული, სადაც ერთი სვეტი არგუმენტს შეესაბამება, ხოლო მეორე ამ წერტილში მის მნიშვნელობას. არგუმენტი გრაფიკზე გამოისახება ჰორიზონტალურ ღერძზე, ხოლო მისი მნიშვნელობა ვერტიკალურზე. თვით დამოკიდებულება ჩვეულებრივ შეიძლება წარმოვადგინოთ წერტილების სახით კოორდინატთა სიბრტყეზე, ან ამ წერტილებზე გამავალი ტეხილის სახით ან ვერტიკალური სვეტების სახით.

მონაცემების ცხრილი შეიძლება წარმოვადგინოთ ერთი ორგანზომილებიანი ან ორი ერთგანზომილებიანი მასივით. მაგალითად, წარმოვადგინოთ ორი ერთგანზომილებიანი მასივი, სადაც პირველ მასივში მოცემულია თარიღები, ხოლო მეორეში – შესაბამისად ჰაერის საშუალო დღიური ტემპერატურა.

ამ მრუდის ასაგებად საჭიროა მოვახდინოთ ზემოთ განხილული **line ( )** ფუნქციის ციკლში გამოძახება. ამასთან, წინასწარ უნდა მოხდეს პიქსელებში ჰორიზონტალური ღერძის მიმართ წერტილებს შორის მანძილის განსაზღვრა. წარმოიქმნება მასშტაბირების ამოცანა – მასივის ელემენტების მნიშვნელობების ეკრანული კოორდინატების პიქსელებში.

პირველ რიგში, უნდა მოვახდინოთ **line ( )** ფუნქციის მოდიფიკაცია: მასში სტრიქონი **document.write(xstr)** უნდა შეიცვალოს **return xstr** სტრიქონით.

ქვემოთ მოყვანილია ამ სცენარის მაგალითი:

```
<SCRIPT>
var aX = new Array ("One","Two","Three","Four","Five","Six")
var aY = new Array (1, 14, 10, 5, 8, 6)
var ky = 10, kx = 60
var x0 = 100, y0 = 200
var xstr = ""
for ( i = 0; i < aX.length – 1; i++ )  {
    x1 = x0 + kx * i
    y1 = y0 – ky * aY[i]
    x2 = x0 + kx * (i + 1)
    y2 = y0 – ky * aY[i + 1]
    xstr += line ("p_b.jpg", x1, y1, x2, y2, 4)
}
for (i = 0; i < 6; i++)  {
      /* ჭდეები ვერტიკალურ ღერძზე  */
xstr += "<b style = 'position:absolute; top:" + (y0 - i * 30 - 10) +";
      left: 75 '> " + i * 3 + "</b>"
  if (i > 0)  {
      /* ჰორიზონტალური ხაზი  */
     xstr += line ("point.bmp",x0,y0 – i*30, x0 + 5*kx, y0 – i*30, 1, 2)
      /* ვერტიკალური ხაზი  */
      xstr += line ("point.bmp", x0 + kx*i, y0, x0 + kx*i, 50, 1, 2)
  }
      /* ჭდეები ჰორიზონტალურ ღერძზე  */
xstr += "<b style = 'position:absolute; top:" + (y0 + 5) +"; left: " +
      (x0 + kx*i) +" '>" + aX[i] + "</b>"
}
      /* ვერტიკალური ღერძი  */
xstr += line ("point.bmp", x0, y0, x0, 50, 2)
      /* ჰორიზონტალური ღერძი  */
xstr += line ("point.bmp", x0, y0, x0 + 5*kx, y0, 2)
document . write (xstr)
```

</SCRIPT>

ამ სცენარში მასშტაბირება არ არის უნივერსალური, არამედ იგი შერჩეულია ხელით კონკრეტული საწყისი მონაცემების ანალიზის საფუძველზე. ეს ყოველივე შეიძლება განხორციელდეს ავტომატურად, რისთვისაც უნდა დაიწეროს უნივერსალური სცენარი.

დინამიკური წირები. ხშირად, სცენარების დაწერის დროს საჭირო ხდება ხაზების გადახატვა მოძრაობის ეფექტის შესაქმნელად, მონაცემთა დინამიკური განახლების გზით. ზემოთ განხილული **line ( )** და **curve ( )** კოდები სრულად ვერ ასახავს ამ პროცესებს, რისთვისაც საჭირო იქნება მათი მოდიფიცირება.

პირველ რიგში სტრიქონი **document.write(xstr)**, უნდა შეიცვალოს **return xstr** სტრიქონით. შედეგად ეს ფუნქციები არ დახაზავს ხაზებს, არამედ მოამზადებს მონაცემებს მათ დასახატად. ნებისმიერი ტიპის წირის დასახატად სცენარში უნდა ჩაიწეროს შემდეგი კოდი:

document . write (curve (პარამეტრები))

ან ამ ერთი სტრიქონის ნაცვლად ჩავწეროთ შემდეგი ორი:

var xstr = curve (პარამეტრები)
document . write (xstr)

ვინაიდან, მზად უნდა ვიყოთ დოკუმენტიდან ხაზის წასაშლელად, **line ( )** და **curve ( )** ფუნქციების მიერ დაბრუნებული ტეგების სტრიქონი ჩავსვათ რაიმე კონტეინერში, მაგალითად

<DIV ID = . . .>

ID ატრიბუტით, ხოლო შემდეგ ჩავწეროთ დოკუმენტში:

var cmycurve = "<DIV ID = 'mycurve '> " + curve (პარამეტრები)
+ "</DIV>" document . write (cmycurve)

იმისათვის, რომ წაიშალოს ხაზი დოკუმენტიდან, საკმარისია **<DIV>** კონტეინერული ტეგის შიგთავსი შეიცვალოს ცარიელი სტრიქონით:

document . all . mycurve . innerHTML = " "

აქ ვისარგებლეთ **innerHTML** თვისებით, რომელიც შეიცავს მითითებულ კონტეინერში მოთავსებულ მთელ **HTML**-კოდს. ამ

თვისებისათვის ახალი მნიშვნელობის მინიჭება მაშინვე გამოიწვევს დოკუმენტში შესაბამისი ელემენტის ცვლილებას. მოცემულ შემთხვევაში **<IMG . . .>** ტეგების მთელი მიმდევრობა იცვლება ცარიელი სტრიქონით, რის შედგადაც ხაზის გამოსახულება გაქრება. ამასთან, თვითონ ცარიელი **<DIV>** კონტეინერი დარჩება. აუცილებლობის შემთხვევაში მისი შევსება შესაძლებელია. ეს არის **HTML**-დოკუმენტების ელემენტების დინამიკური ცვლილების ტიპური მაგალითი.

იმისათვის, რომ წინა ხაზის ნაცვლად დაიხატოს ახალი ხაზი, საკმარისია სცენარში ჩაიწეროს შემდეგი კოდი:

document . all . mycurve . innerHTML = curve (ახალი პარამეტრები)

მაგალითის სახით მოვიყვანოთ შემდეგი სცენარი:

```
<SCRIPT>
<BUTTON onclick = "redraw ( )" > REDRAW</BUTTON>
<SCRIPT>
    var cmycurve = "<DIV ID = 'mycurve '> " + curve("",
        "80*Math.sin(6/25*x)" , "80*Math.cos(6/25*x) ", 100, 200,
        600, 6, 0) + "</DIV>"
    document . write (cmycurve)
    function redraw ( )  {
    document . all . mycurve . innerHTML = curve ("",
        "60*Math.sin(6/25*x)" , "60*Math.cos(6/25*x) ", 100, 150,
        300, 2, 0)
    }
    function curve (pict_file, yexpr, xexpr, x0, y0, t, n, s)  {
    if (!yexpr) return null
    if (!xexpr) xexpr = "x"
    if (!pict_file) pict_file = "point.bmp"
    if (!s) s = 0
    if (!t) t = 0
    var clinewidth = ""
    if (!n) clinewidth = 'WIDTH= ' + n + 'HEIGHT= ' + n
    var x
    xstr0 = '<IMG SRC=" + pict_file + ' " ' + clinewidth + ' STYLE
        = "position : absolute; top: '
    xstr = ""
    var i = 0, draw = true
```

```
    for (x = 0; x < t; x++)  {
    if (draw)
    xstr += xstr) + (y0 + eval (yexpr)) + '; left: ' + (x0 + eval (xexpr))
          + ' "> '
    if (i > s && s > 0)  {
         draw = !draw
         i = 0
    }
    i++
    }
    return xstr
     }
</SCRIPT>
</HTML>
```

მოცემული დოკუმენტის ჩატვირთვის შემთხვევაში ბრაუზერის ფანჯარაში გამოჩნდება ღილაკი და ლისაჟუს ფიგურა. ღილაკზე დაწკაპუნების შემდეგ ეს ფიგურა შეიცვლება წრით. ღილაკზე განმეორებითი დაწკაპუნება არ გამოიწვევს არავითარ ცვლილებას, ვინაიდან მოცემული ფიგურა იგივე ფიგურით იცვლება.

43. ფორმის მონაცემთა დამუშავება

HTML-დოკუმენტის ისეთი ელემენტები, როგორიცაა მონაცემთა შეტანის ველები, ტექსტური არეები, გადამრთველები და ალმები, ჩამოშლადი სიები და ღილაკები, შეიძლება გავაერთიანოთ ფორმაში. ფორმა <FORM> კონტეინერული ტეგის დახმარებით იქმნება, რომლის შიგნით ამ ფორმის ელემენტების ტეგებია განთავსებული. დოკუმენტის ობიექტურ მოდელში თითოეულ ფორმას შეესაბამება forms კოლექციაში შემავალი თავისი ობიექტი. ნებისმიერი ეს ელემენტი შეიძლება გამოყენებულ იქნეს ფორმის გარეშეც, თუმცა ის მხოლოდ უბრალოდ კონტეინერი კი არ არის, არამედ ამ ფორმის ელემენტებში არსებული ყველა მონაცემის სერვერზე გაგზავნის ორგანიზებისათვის განკუთვნილი კონტეინერი და ობიექტია.

როგორც ადრე, ასევე ახლაც მონაცემების სერვერზე გასაგზავნად სცენარი არ არის აუცილებელი. მონაცემების გასაგზავნად საკმარისია <FORM> ტეგში მივუთითოთ ACTION ატრიბუტი, ხოლო ფორმაში დავაყენოთ Submit ტიპის ღილაკი. ამ ღილაკზე დაწკაპუნება გამოიწვევს მონაცემთა გაგზავნის ინიცირებას. თუ არ არის მითითებული ACTION ატრიბუტი ან მისი მნიშვნელობა ცარიელია, ფორმის მონაცემები არ გაიგზავნება. ზოგადად, ეს შეიძლება იყოს ფაილის ან CGI-პროგრამის URL-მისამართი, რომელიც მიიღებს და დაამუშავებს გაგზავნილ მონაცემებს. მაგალითად,

ACTION = "http://www.myserver/cgi/myprogram.pl".

თუ ჩვენ გვსურს ელექტრონული ფოსტით ფორმის მონაცემების გაგზავნა, მაშინ **ACTION**-ის მნიშვნელობა უნდა ჩაიწეროს შემდეგი სტრიქონის სახით:

mailto : e-mail მისამართი

ასევე შეიძლება იყოს მითითებული შეტყობინების თემა:

mailto : e-mail მისამართი?subject = შეტყობინების თემა

<FORM> ტეგში ACTION ატრიბუტის გარდა საჭიროა მითითებული იყოს კიდევ ორი ატრიბუტი: **METHOD** და **ENCTYPE**. **METHOD** ატრიბუტს შეუძლია მიიღოს ორი მნიშვნელობა **POST** ან **GET**. მნიშვნელობის არჩევა მხოლოდ ფორმაზე აისახება, რომელშიც მონაცემები გადაიცემა. მომხმარებლისათვის უფრო მოსახერხებელია თუ **POST** მნიშვნელობას ავირჩევთ. **ENCTYPE** ატრიბუტს მივანიჭოთ მნიშვნელობა "**text/plain**". ამ შემთხვევაში გაგზავნილ შეტყობინებას ექნება

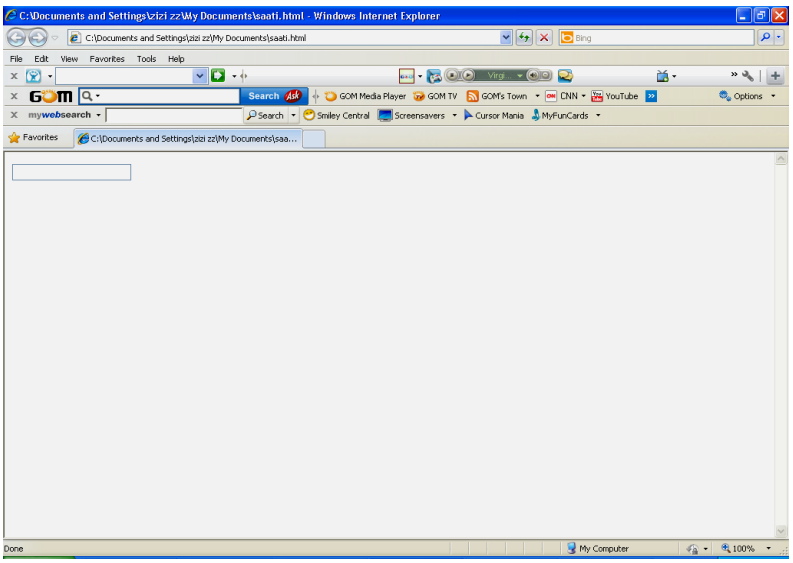
ელემენტის სახელი = მნიშვნელობა

წყვილის სახე. აქ **ელემენტის სახელი** ეს არის ელემენტის ტეგში **NAME** ატრიბუტის მნიშვნელობა, ხოლო **მნიშვნელობა** - იმავე ტეგში **VALUE** ატრიბუტის მნიშვნელობა. თუ არ მივუთითებთ **ENCTYPE** ატრიბუტს, მაშინ შეტყობინება იქნება წარმოდგენილი კოდირებული სახით.

ქვემოთ მოყვანილია **HTML**-დოკუმენტის მაგალითი ფორმით, რომელიც შეიცავს მონაცემთა შეტანის ველსა და **Submit** ტიპის ღილაკს:

<HTML>

```
<FORM METHOD = POST ACTION = "mailto:kapr@mail.ru"
      ENCTYPE = "text/plain">
<INPUT NAME = "Message" TYPE = "text" VALUE = "">
</FORM>
</HTML>
```

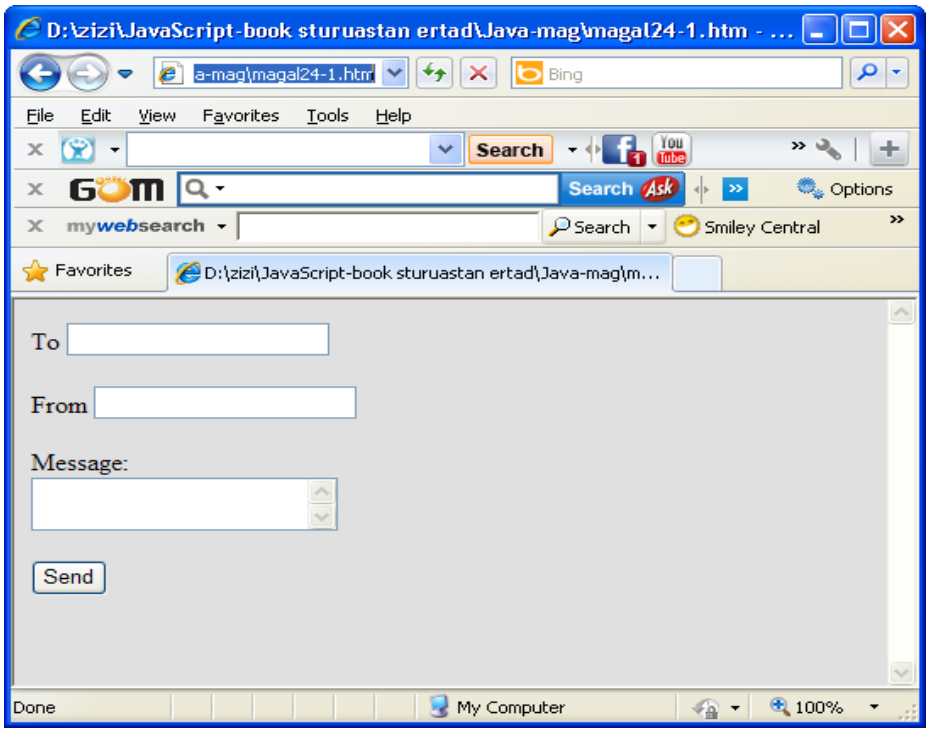


მონაცემთა გაგზავნა განხორციელდება **Submit** ტიპის ღილაკზე დაწკაპუნებით, რომელზეც ჩვენს მაგალითში იქნება წარწერა **Send**. მიმღების მისამართი მითითებულია როგორც **ACTION** ატრიბუტის მნიშვნელობა.

თუ მონაცემთა გაგზავნის წინ საჭიროა მათი შემოწმება ან რაიმე სხვა მოქმედებების ჩატარება, მაშინ უნდა დაიწეროს სცენარი. შემდეგ მაგალითში მოწმდება არის თუ არა ელექტრონული ფოსტის მისამართის ველში სიმბოლო "@" და ხომ არ არის ცარიელი თვით შეტყობინების ველი. ამ დროს არ მოხდება შეტყობინების გაგზავნა.

```
<HTML>
<FORM ID = "myform" METHOD = POST ACTION = ""
     ENCTYPE = "text/plain" style = "background:'e0e0e0' ">
To
<INPUT NAME = "email_to" TYPE = "text" VALUE = "">
<P>
From
<INPUT NAME = "email_from" TYPE = "text" VALUE = "">
<P>
Message: <BR>
```

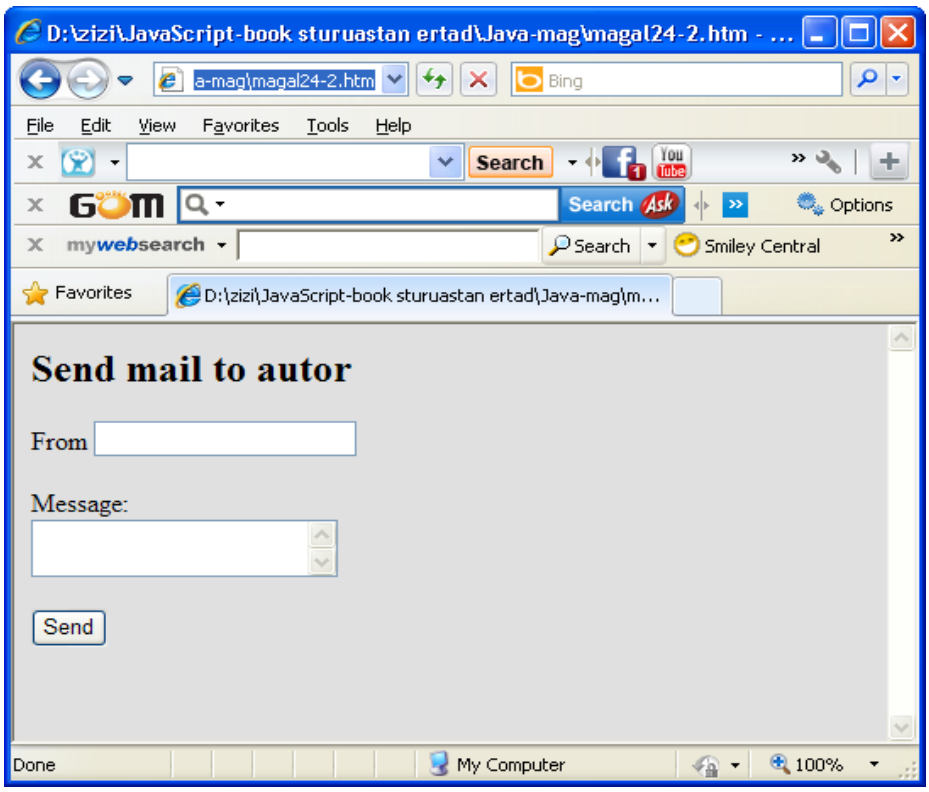
```
<TEXTAREA NAME = "Message" TYPE = "text" VALUE =
    ""></TEXTAREA>
<P>
<! Button-type Submit >
<INPUT NAME = "SUBMIT" TYPE = "submit" VALUE =
    "Send">
</FORM>
<SCRIPT>
function myform.onsubmit ( ) {
var noemail = myform . email_to . value . indexOf ('@') == - 1
var notext = !myform . Message . value
var xtext = "\nThe letter was not sent"
if (noemail || notext) {
   event.returnValue = false
   if (noemail)
      alert ("Invalid email recipient" + xtext)
   else
      alert ("No text messiges" + xtext)
} else
   myform . action = "mailto: " + myform . email_to . value
}
</SCRIPT>
</HTML>
```



ხშირად **WEB**-გვერდებზე განათავსებენ მიმართვას ან ღილაკს, რომლის საშუალებითაც გაიხსნება გვერდის ავტორისათვის ელექტრონული ფოსტით შეტყობინების გაგზავნის ფორმა. ამ შემთხვევაში საჭირო არ არის მიმღების საფოსტო მისამართის მითითება. თუ ჩვენ გვსურს ამ მისამართის გასაიდუმლოება, საჭიროა მივიღოთ ზოგიერთი ზომები. ყველაზე მარტივი მეთოდია, რომ შევინახოთ მისამართის ცალკეული კომპონენტები ცალკ-ცალკე ცვლადებში და შემდეგ კონკატენაციის საშუალებით მოვახდინოთ მათი გაერთიანება. შედეგად, სპეციალური პროგრამა-რობოტი **HTML**-დოკუმენტში ვერ იპოვის ელექტრონული ფოსტის მისამართის სტრუქტურის მქონე სტრიქონს. ქვემოთ მოვიყვანოთ ამ ტიპის მაგალითი:

```
<HTML>
<FORM ID = "myform" METHOD = POST ACTION = ""
    ENCTYPE = "text/plain" onsubmit = "return validator ( )"
    style = "background:'e0e0e0' ">
<H2> Send mail to autor</H2>
From
<INPUT NAME = "email_from" TYPE = "text" VALUE = "">
<P>
Message: <BR>
<TEXTAREA NAME = "Message" TYPE = "text" VALUE =
    ""></TEXTAREA>
<P>
<! Button-type Submit >
<INPUT NAME = "SUBMIT" TYPE = "submit" VALUE =
    "Send">
</FORM>
<SCRIPT>
var first = "xxx" , second = "mail"
function validator ( ) {
if (!myform.Message.value) {
   alert ("No text messiges\nThe letter was not sent")
   return false
} else
   myform . action = "mailto: " + first + "@" + second + ".ru"
   return true
```

```
}
</SCRIPT>
</HTML>
```



მაგალითში **validator** ფუნქცია აბრუნებს **true** ან **false** შედეგს, იმის მიხედვით შეგვიძლია გავაგზავნოთ თუ არა შეტყობინება.

44. მენიუ

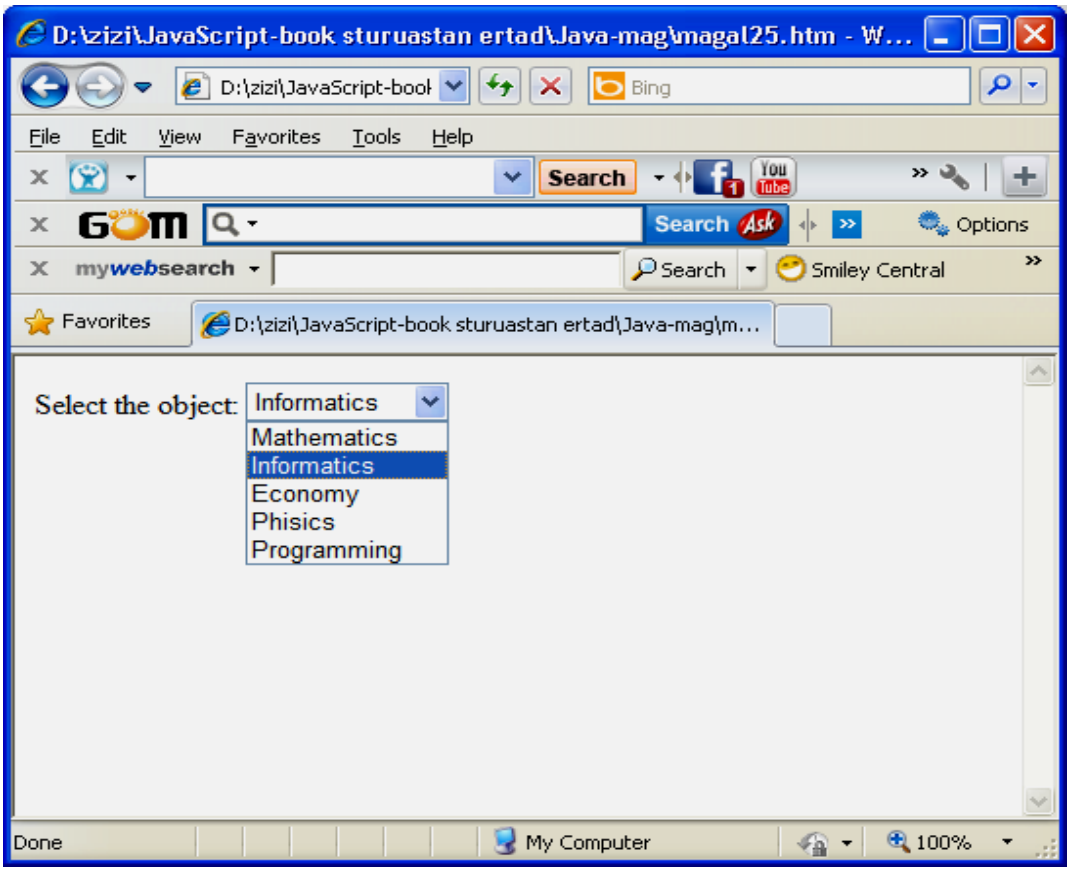
ჩამოშლადი სია

მარტივი მენიუ შეიძლება შეიქმნას **<SELECT>** და **<OPTION>** ტეგების დახმარებით. ჩვეულებრივ ასეთ კონსტრუქციებს უწოდებენ ჩამოშლად სიებს. ქვემოთ მოყვანილი არის ჩამოშლადი სიის გამოყენების მარტივი მაგალითი. ამ მაგალითში ჩამოშლადი სია მოცემულია კოდის სახით, ხოლო ამ სიიდან ამორჩეული ელემენტის დამუშავება ხდება სცენარით. მისი საშუალებით ხდება მხოლოდ სიიდან ამორჩეული ელემენტის ნომრის განსაზღვრა. მაგალითში ეს არის შესაბამისი შეტყობინების ფანჯრიის გამოტანა. მომხმარებლის მიერ სიის ელემენტზე ამორჩევა ხდება მაუსის მარცხენა ღილაკის დაწკაპუნებით.

ამასთან, შესაბამისი **<SELECT>** ტეგის დოკუმენტის ელემენტის ობიექტის **selectedIndex** თვისება მნიშვნელობად ენიჭება ამორჩეული ელემენტის ნომერი სიაში (ნუმერაცია0-დან იწყება). მომხმარებლის ამორჩევის დამუშავების ინიციაცია **onchange** ხდომილობით ხდება (ელემენტთა სიაში მოხდა ცვლილება). ამ ხდომილობის დამუშავება ხორციელდება **myselection ( )** ფუნქციით. ჩამოშლად სიაში ელემენტის საწყისი გამოყოფა და გამოჩენა **<OPTION>** ტეგის **SELECTED** ატრიბუტითაა მოცემული. ჩვენს მაგალითში გამოყოფილია ელემენტი „**Informatics**“.

```
<HTML>
Select the object:
<SELECT NAME = "TEST" onchange = "myselection ( )">
   <OPTION> Mathematics
   <OPTION SELECTED> Informatics
   <OPTION> Economy
   <OPTION> Phisics
   <OPTION> Programming
</SELECT>
<SCRIPT>
function myselection ( ) {
var testname, testnumber;
testnumber = document . all . TEST . selectedIndex
if (testnumber == 0)
   testname="Fundamentals of Mathematics"
else {
   if (testnumber == 1)
      testname="Basics algorithmization and Sciences"
   else {
      if (testnumber == 2)
                testname="Economy"
   else {
      if (testnumber == 3)
                testname="Elementary Physics"
   else {
      testname = "Programming language JavaScript"
     }
```

```
    }
   }
  }
alert ("You will pass: " + testname)
}
</SCRIPT>
</HTML>
```

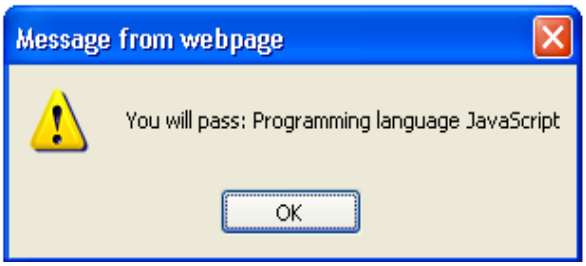


გაჩუმების პრინციპით ჩამოშლად სიაში ჩანს მხოლოდ ერთი ელემენტი. იმისათვის, რომ ჩამოშლადი სია იყოს ნაწილობრივ გახსნილი (ჩანდეს რამდენიმე ელემენტი), საჭიროა **<SELECT>** ტეგში მივუთითოთ ატრიბუტი **SIZE = გამოსაჩენი ელემენტების რაოდენობა**. მაგალითად:

```
<HTML>
Select the object:
<SCRIPT>
var N_sel = 1
var aoptions = new Array ( )
aoptions[0] = "Mathematics"
aoptions[1] = "Informatics"
aoptions[2] = "Economy"
```

```
aoptions[3] = "Phisics"
aoptions[4] = "Programming"
xstr = '<SELECT ID = "TEST" onchange = "myselection ( )">'
for (i = 0; i < aoptions . length; i++)  {
xprefix = (i == N_sel) ? 'SELECTED =' + N_sel: ''
xstr += '<OPTION ' + xprefix + '>' + aoptions[i]
}
xstr += '</SELECT>'
document . write (xstr)
function myselection ( )  {
var testname, testnumber;
testnumber = document . all . TEST . selectedIndex
if (testnumber == 0)
   testname="Fundamentals of Mathematics"
else  {
   if (testnumber == 1)
      testname="Basics algorithmization and Sciences"
   else  {
      if (testnumber == 2)
                testname="Economy"
   else  {
      if (testnumber == 3)
                testname="Elementary Physics"
   else  {
      testname = "Programming language JavaScript"
     }
     }
    }
   }
alert ("You will pass: " + testname)
}
</SCRIPT>
</HTML>
```

ჩამოშლილი მენიუდან რომელიმე პუნქტის შერჩევის შემდეგ ეკრანზე გამონათდება შეტყობინების ფანჯარა, შერჩეული თემის შესაბამისი ტექსტით. მაგალითად, პროგრამირების არჩევის შემდეგ გვექნება:



45. ნამდვილი მენიუ

მენიუ, რომლის შექმნასაც ჩვენ ახლა განვიხილავთ, შედგება მთავარი ჰორიზონტალური მენიუსა და რამდენიმე ვერტიკალური ქვემენიუსაგან. ქვემენიუ ჩამოიშლება მაუსის მაჩვენებლის მთავარი მენიუს ოფციაზე მიყვანის დროს. აუცილებელი არ არის, რომ მთავარი მენიუს ყველა ოფციას შეესაბამებოდეს ქვემენიუ: ზოგიერთი ოფცია შეიძლება იყოს ტერმინალური, ანუ არ შეიცავდეს ქვემენიუს. ოფციაზე მაუსის მარცხენა ღილაკით დაწკაპუნებით განხორციელდება სცენარით განსაზღვრული რაიმე ქმედება. ეს შეიძლება იყოს **WEB**-გვერდის **URL**-მისამართი ან **JavaScript**-ის კოდის შემცველი სტრიქონი.

ჩვენ ასეთი მენიუს სცენარის კოდი გავაკეთოთ უფრო უნივერსალური და გავაფორმოთ ორი ფაილის სახით, რომლის გაფართოებაა **.js**. პირველი ფაილი, **menu_prm.js**, შეიცავს მენიუს პარამეტრებს, რომლის დანართის შექმნის დროს მისი შინაარსი შეიძლება შევცვალოთ კონკრეტული ამოცანის მიხედვით. მაგალითად, ამ ფაილში განისაზღვრება ოფციების დასახელება და მათთვის კონკრეტული ქმედებები, ფერი და სხვა პარამეტრები. ამგვარად, ეს არის მენიუს აღწერის ცვლადი ნაწილი. მეორე ფაილი, **menu_bld.js**, შეიცავს მენიუს აგების მექანიზმისა და ფუნქციონირების აღწერას. მასში გამოიყენება პირველ ფაილში განსაზღვრული პარამეტრები. ეს არის მენიუს აღწერის მუდმივი ნაწილი. რა თქმა უნდა მომხმარებელს შეუძლია თავისი შეხედულების მიხედვით მოახდინოს ამ ფაილის შინაარსის კორექტირება.

იმისათვის, რომ შევქმნათ მენიუ საკმარისია **HTML**-დოკუმენტში უბრალოდ ჩავწეროთ შემდეგი სტრიქონები:

```
<SCRIPT SRC = "menu_prm.js"></SCRIPT>
<SCRIPT SRC = "menu_bld.js"></SCRIPT>
```

<SCRIPT> buildMenu ( ) </SCRIPT>

სადაც **buildMenu ( )** – **menu_bld.js** ფაილში განსაზღვრული ფუნქციაა. მას **menu_prm.js** ფაილში მოცემული პარამეტრების მიხედვით ეკრანზე გამოჰყავს მენიუ.

განვიხილოთ მაგალითი რომელშიც მთავარი (ჰორიზონტალური) მენიუ შეიცავს სამ ოფციას და მათ შორის პირველ ორს შეესაბამება ვერტიკალური ქვემენიუ. ბოლო ოფცია არის ტერმინალური. ოფციათა დასახელება შერჩეულია ისე, რომ ადვილად მისახვედრი იყოს რომელი რას მიეკუთვნება.

პირველ ფაილში ჩვენ ვაწვდით ფერებისა და შრიფტის პარამეტრებს, აგრეთვე ოფციის დასახელებისა და მოქმედებების შედგენილობას და კოორდინატებს. მენიუს ოფციათა დასახელება, მათი პოზიციონირება, სტატუსის ზოლის მოქმედებები და შედგენილობა მოცემული იქნება მასივის საშუალებით. ასე, რომ მენიუს სტრუქტურა მოცემული იქნება ორგანზომილებიანი მასივის სახით.

შენიშვნა. თუ მოქმედება არ არის **URL**-მისამართი, არამედ არის **JavaScript** ენაზე ჩაწერილი რაიმე კოდი, მაშინ იგი "**JavaScript**:" პრეფიქსით უნდა იწყებოდეს, რომელსაც მოსდევს წერტილ-მძიმით გამოყოფილი გამოსახულებანი.

მეორე ფაილში განსაზღვრულია ზოგიერთი ფუნქციები, რომლის დახმარებითაც მენიუ ჩამოიშლება და დაიწყებს ფუნქციონირებას. მენიუს ასახვა ხდება ცხრილების განმსაზღვრელი **HTML**-ტეგების გამოყენების საფუძველზე. **HTML**-კოდის ფრაგმენტები, რომლისაგანაც იკრიბება **HTML**-დოკუმენტის გენერირებული სტრიქონები, განსაზღვრულია მასივის ელემენტების სახით.

menu_prm.js ფაილის კოდი:

```
/* მენიუს პარამეტრები */
var clBorder = 'blue'          // ჩარჩოს ფერი
var clBgInact = '#83d6f5'      // ფონის ფერი
var clBgAct = '3d6f4fe'        // ოფციის ფონის განათების ფერი
var clFnInact = 'blue'         // ჩვეულებრივი წარწერის ფერი
```

```
var clFnAct = 'black'          // წარწერის ფერი ოფციის განათების
                                  დროს
var cFontSize = '18'           // შრიფტის ზომა
var cFontFamily = 'times'      // შრიფტის ტიპი
var closeTimeout = 500         // მენიუს შეყოვნების დრო, მლ.წმ
var selfPos = false            // მენიუს ოფციის პოზიციონირება
// თვით მენიუ
var menu = new Array ( )
menu [0] = new Array ( )
menu [0] [0] = new Array ("Menu 1", "", "Choose something from
    the submenu", 50, 10, 80, 50, 120, 120)
menu [0] [1] = new Array ("Submenu 1.1", "javascript:history.go(-
    1)", "Back")
menu [0] [2] = new Array ("Submenu 1.2",
    "http://www.yandex.ru", "Yandex")
menu [0] [3] = new Array ("Submenu 1.3", " javascript:alert('Hello!
    ')", "Welcome Window")
menu [1] = new Array ( )
menu [1] [0] = new Array ("Menu 2", "", "", 130, 10, 80, 130, 120,
    0)
menu [1] [1] = new Array ("Submenu 2.1", "", "")
menu [1] [2] = new Array ("Submenu 2.2", "", "")
menu [2] = new Array ( )
menu [2] [0] = new Array ("Menu 3", "http://www.Internet.ge",
    "INTERNET.GE", 210, 10, 80, 0, 0, 0)
```

menu_bld.js ფაილის კოდი:

```
var ie = document . all ? true ; false
var overBox = ' '
var timerID
var barHtml = new Array ( )
barHtml [0] = '<DIV ID = "divbarpos'
barHtml [1] = ' " STYLE = "position:absolute; left:'
barHtml [2] = 'px; top:'
barHtml [3] = 'px; " onmouseover = " openbox ( '
barHtml [4] = ') " onmouseout = " closebox ( '
barHtml [5] = ') " onclick = " clickbox ( '
```

```
barHtml [6] = ') "><TABLE CELLPADDING=0
    CELLSPACING=0><TR><TD BGCOLOR= " '
barHtml [7] = ' "><TABLE CELLPADDING= "0"
    CELLSPACING= "1" BORDER = "0"><TR><TD CLASS =
    "mnubarpos" ID = "mnubarpos'
barHtml [8] = ' " WIDTH = " '
barHtml [9] = ' " BGCOLOR = " '
barHtml [10] = ' " STYLE = "color:'
barHtml [11] = '; font-size: ' + cFontSize + '; font-family: ' +
    cFontFamily + '; ">'
barHtml [12] =
    '</TD></TR></TABLE></TD></<TR></TABLE></DIV>'
var boxHtml = new Array ( )
boxHtml [0] = '<DIV ID = "divbox'
boxHtml [1] = ' " STYLE = "position:absolute; visibility:hidden;
    left:'
boxHtml [2] = ' px; top:'
boxHtml [3] = ' px; font-family: ' + cFontFamily + '; " onmouseout =
    " closebox ( '
boxHtml [4] = ') "><TABLE CELLPADDING=0
    CELLSPACING=0><TR><TD BGCOLOR= " '
boxHtml [5] = ' "><TABLE CLASS = "mnubox" ID = "mnubox'
boxHtml [6] = ' " CELLPADDING= "0" CELLSPACING= "1"
    BORDER = "0" > '
boxHtml [7] = '<SPAN ID = "divboxpos '
boxHtml [8] = ' " onmouseover = "openboxpos ( '
boxHtml [9] = ') " onmouseout = "closeboxpos ( '
boxHtml [10] = ') " onclick = "clickboxpos ( '
boxHtml [11] = ') "><TR><TD CLASS = "mnuboxpos " ID =
    "mnuboxpos '
boxHtml [12] = ' " WIDTH = " '
boxHtml [13] = ' " BGCOLOR = " '
boxHtml [14] = ' " STYLE = "color: '
boxHtml [15] = '; font-size: ' + cFontSize + ' "> '
boxHtml [16] = '</TD></TR></SPAN>
boxHtml [17] = '</TABLE></TD></TR></TABLE></DIV>
function buildMenu ( ) {                    // მენიუს აგება
```

```
        if (ie)  {
              buildMenuBar ( );
              buildSubMenu ( );
              if (selfPos) PosMenu ( );
              ResizeSubMenu ( );
        }
}
function buildMenuBar ( )  {      // ჰორიზონტალური მენიუს აგება
        for (i = 0; i < menu . length; i++)  {
var s = barHtml [0] + i + barHtml [1] + menu [i] [0] [3] + barHtml
        [2] + menu [i] [0] [4] + barHtml [3] + i + barHtml [4] + i +
        barHtml [5] + i + barHtml [6] + clBorder + barHtml [7] + i +
        barHtml [8] + menu [i] [0] [5] + barHtml [9] + clBgInact +
        barHtml [10] + clFnInact + barHtml [11] + menu [i] [0] [0] +
        barHtml [12];
              document . writeln (s);
        }
}
function buildSubMenu ( )  {              // ქვემენიუს აგება
        for (i = 0; i < menu . length; i++)  {
              if (menu[i] . length > 1)  {
var s = boxHtml [0] + i + boxHtml [1] + menu [i] [0] [6] + boxHtml
        [2] + menu [i] [0] [7] + boxHtml [3] + i + boxHtml [4] +
        clBorder+ boxHtml [5] + i + boxHtml [6]
              for (j = 1; j < menu [i] . length; j++)  {
                    var s1 = i + ' , ' + j
                    var s2 = i + ' _ ' + j
s += boxHtml [7] + s2 + boxHtml [8] + s1 + boxHtml [9] + s1 +
        boxHtml [10] + s1 + boxHtml [11] + s2+ boxHtml [12] + menu
        [i] [0] [8] + boxHtml [13] + clBgInact + boxHtml [14] +
        clFnInact + boxHtml [15] + menu [i] [j] [0] + boxHtml [16]
              }
              s += boxHtml [17]
              document . writeln (s)
        }
    }
}
```

```
function PosMenu ( ) {
      for (i = 0; i < menu . length; i++) {
            barCurr = document . all ['divbarpos' + i ]
            if (i > 0) {
                barPrev = document . all ['divbarpos' + (i - 1) ]
                barCurr . style . left = barPrev . offsetLeft + barPrev .
      offsetWidth - 1
            }
            if (menu [i] . length > 1) {
               boxCurr = document . all ['divbox' + 1 ]
               boxCurr . style . pixelTop = barCurr . offsetTop +
                  barCurr . offsetHeight - 1
               boxCurr . style . pixelLeft = barCurr . offsetLeft
            }
      }
}
function ResizeSubMenu ( ) {
      for (i = 0; i < menu . length; i++) {
            if (menu [i] . length > 1 && menu [i] [0] [8] <= 0) {
                  el = document . all ['mnubox' + i]
                  w = document . all ['divbarpos' + i] . offsetWidth
                  if (el . offsetWidth < w) el . style . width = w
            }
      }
}
function openbox (x) {
      paintLayer ('mnubarpos' + x, active = true)
      showLayer ('divbox' + x)
      for (i = 0; i < menu . length; i++) if (i != x) closebox (i, 0)
      overBox = 'divbox' + x
      window . status = menu [x] [0] [2]
}
function closebox (x, timeout) {
      paintLayer ('mnubarpos' + x, active = false)
      clearTimeout (timerID)
      if (timeout ++ 0) hideLayer ('divbox' + x)
            else timerID = setTimeout ('hideLayer ("divbox' + x + ' ")',
      closeTimeout)
```

```
        overBox = ' '
        window . status = defaultStatus
}
function clickbox (x)  {
        clickMenu (menu [x] [0] [1] )
}
function openboxpos (x, y)  {
        paintLayer ('mnuboxpos' + x + '_' + y, active = true)
        overBox = 'divbox' + x
        window . status = menu [x] [y] [2]
}
function closeboxpos (x, y)  {
        window . status = defaultStatus
        paintLayer ('mnuboxpos' + x + '_' + y, active = false)
}
function clickboxpos (x, y)  {
        clickMenu (menu [x] [y] [1] )
}
function clickMenu (s)  {
        if (s . indexOf ('javascript:') == 0) eval (s);
                else if (s != ' ') window . location . href = s;
}
function paintLayer (layerID, active)  {
        if (layer = document . all [layerID])  {
                if (active)  { clBg = clBgAct; clFn = clFnAct }
                        else { clBg = clBgInact; clFn = clFnInact }
                layer . style . backgroundColor = clBg
                layer . style . color = clFn
        }
}
function showLayer (layerID)  {
        if (layer = document . all [layerID] )  layer . style . visibility =
        'visible'
}
function hideLayer (layerID)  {
        if (layer = document . all [layerID] && (overBox != layerID) )
                document . all [layerID] > style . visibility = 'hidden'
}
```

ყურადღება უნდა მიექცეს იმას, რომ მენიუსა და ქვემენიუების პოზიციონირება ხდება ერთმანეთისაგან დამოუკიდებლად. ეს საშუალებას იძლევა მენიუ ეკრანზე ვიზუალურად წარმოვადგინოთ სივრცეში გაბნეული ნაწილების სახით. ამგვარად, რამდენიმე მენიუს გასაკეთებლად აუცილებელი არ არის თითოეული მათგანისათვის შევქმნათ და ჩავტვირთოთ სპეციალური ფაილები აღწერებით.

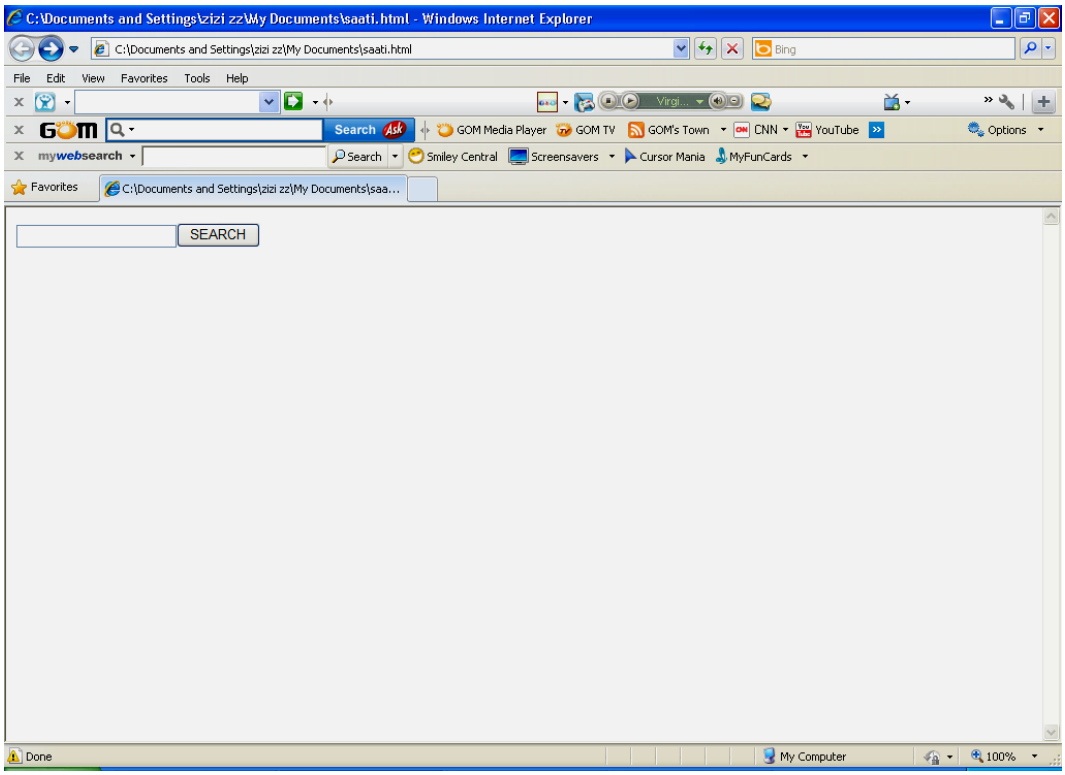
46. ძებნის ოპერაციები ტექსტურ არეში

Web-გვერდებზე განთავსებული დიდი მოცულობის ტექსტები, ჩვეულებრივ საძიებო სისტემითაა აღჭურვილი. გარეგნულად იგი გამოიყურება როგორც საძიებო ტექსტის შესატანი ველი და ღილაკი, რომელზეც მაუსის დაწკაპუნებით ჩართავს ძიების პროცედურას. ძებნის უმარტივესი ვარიანტის დროს ეს პროცედურა ფანჯარაში ტექსტს ისე გადაფურცლავს, რომ ნაპოვნი საძიებო ტექსტი გამოჩნდეს და იყოს მონიშნული. თუ ძებნა უშედეგოდ დამთავრდა, მაშინ ტექსტი ფანჯარაში არ იცვლის მდებარეობას და შესაძლებელია შესაბამისი შეტყობინებაც გამოჩნდეს.

ტექსტში ძებნის პროცესის გადასაწყვეტად გამოიყენება ობიექტი **TextRange**. ეს ობიექტი ხელმისაწვდომს ხდის ტექსტურ ინფორმაციას, რომლებიც განთავსებულია **<BODY>**, **<TEXTAREA>**, **<BUTTON>** და **<INPUT TYPE = "text">** ტეგებში მოთავსებულ ობიექტებში. ქვემოთ მოყვანილია ძებნის მაგალითი. იგი შეიცავს **myfind ( )** ფუნქციას, რომელიც ღილაკზე მაუსის დაწკაპუნებით იწყებს ძებნის ოპერაციას. ეს ფუნქცია **createTextRange ( )** მეთოდით ქმნის საძიებო ტექსტურ ველს. შემდეგ **findText ( )** მეთოდით ხდება ტექსტში ძებნა, და ბოლოს **scrollIntoView ( )** მეთოდით ფანჯარა ისე გადაიფურცლება, რომ ნაპოვნი ტექსტი გამოჩნდეს. **select ( )** მეთოდის დახმარებით მოხდება ნაპოვნი ტექსტის სხვა ფერით გამოყოფა.

```
<HTML>
<BODY>
<INPUT TYPE ="text" NAME ="WORD" VALUE ="" SIZE =20>
<BUTTON onclick = "mufind ( ) ">SEARCH</BUTTON>
```

```
<P>
<! Here is a text that search >
</BODY>
<SCRIPT>
function myfind ( ) {
obj = document . body . createTextRange ( )
obj . findText (WORD . value)
obj . scrollIntoView ( )
obj . select ( )
}
</SCRIPT>
</HTML>
```



თუ ძებნა უშედეგოდ დამთავრდა, მაშინ არაფერი არ მოხდება. მაგრამ თუ მომხმარებელმა საძიებო ტექსტის შეყვანის გარეშე ხელი დააჭირა ღილაკს, მაშინ წარმოიშვება შეცდომა. ეს რომ არ მოხდეს საჭიროა ნაწილობრივ შეიცვალოს პროგრამა შემდეგნაირად:

```
function myfind ( ) {
if (!WORD . value) return
obj = document . body . createTextRange ( )
obj . findText (WORD . value)
obj . scrollIntoView ( )
```

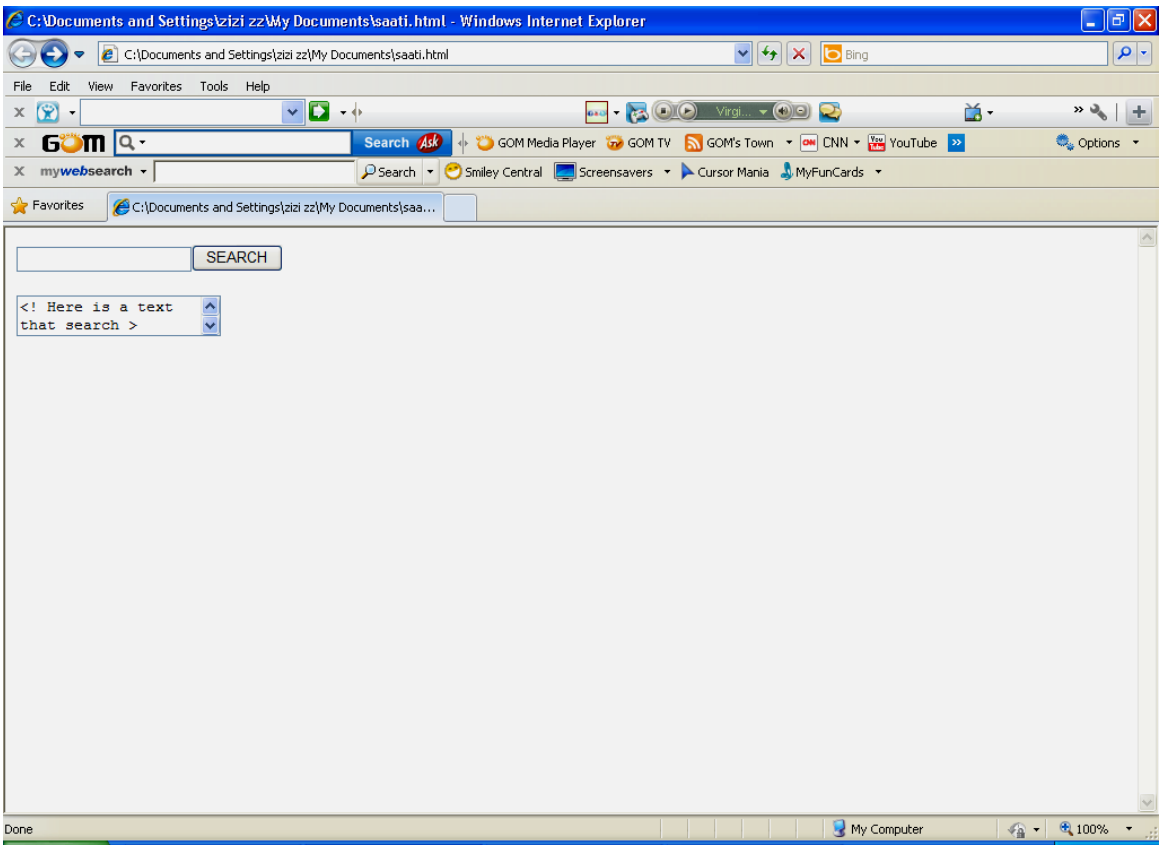
```
obj . select ( )
}
```

განვიხილოთ ვარიანტი, როდესაც ეკრანზე ტექსტთან ერთად გვაქვს მხოლოდ ღილაკი **SEARCH**. მასზე დაწკაპუნებით ეკრანზე გამოჩნდება ფორმა, სადაც მომხმარებელი ჩაწერს საძიებო ტექსტს. თუ ეს ველი ცარიელი არ იქნება, მაშინ განხორციელდება ძებნა, ხოლო ცარიელი ველის შემთხვევაში არაფერი არ მოხდება.

```
function myfind ( ) {
WORD = prompt ("Enter that finding: ", "")
if (!WORD) return
obj = document . body . createTextRange ( )
obj . findText (WORD)
obj . scrollIntoView ( )
obj . select ( )
}
```

ქვემოთ მოყვანილია ძებნის მაგალითი არა დოკუმენტში, არამედ **<TEXTAREA>** ტეგით მოცემულ ტექსტურ არეში.

```
<HTML>
<INPUT TYPE = "text" NAME = "WORD" VALUE = "" SIZE =
    20 >
<BUTTON onclick = "mufind ( ) ">SEARCH</BUTTON>
<P>
<TEXTAREA ID = "mytext">
<! Here is a text that search >
</TEXTAREA>
<SCRIPT>
function myfind ( ) {
if (!WORD . value) return
obj = document . all . mytext . createTextRange ( )
obj . findText (WORD . value)
obj . scrollIntoView ( )
obj . select ( )
}
</SCRIPT>
</HTML>
```



47. ცხრილები

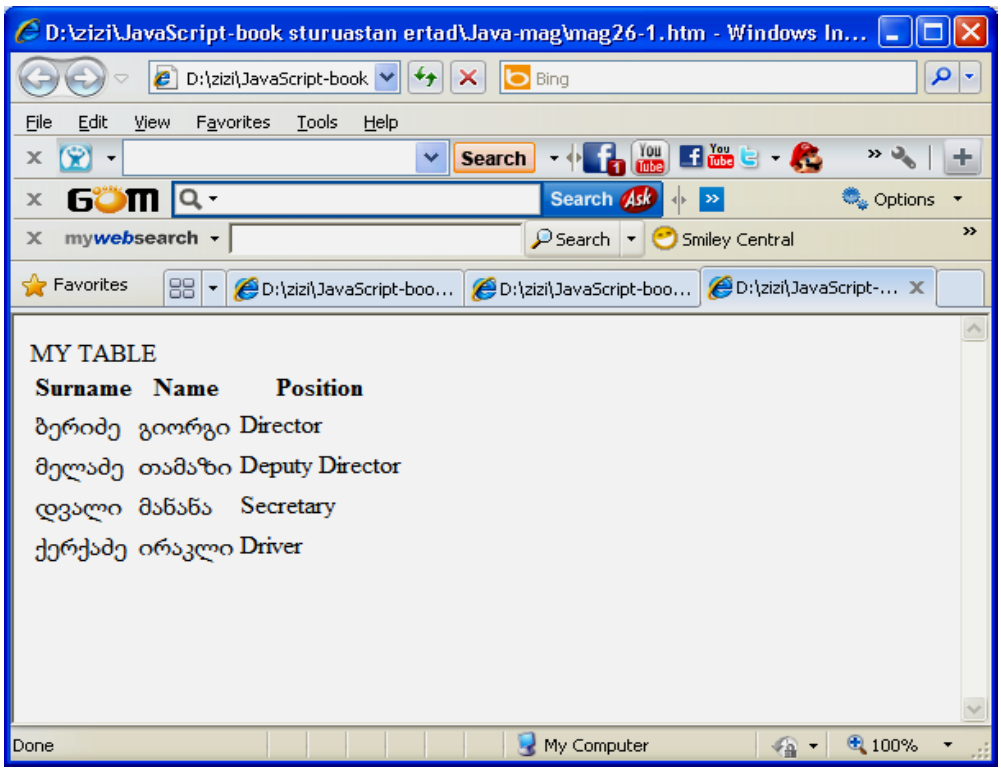
ცხრილები **Web**-დიზაინში ერთ-ერთ ყველაზე ხშირად გამოსაყენებელი ელემენტია. ეს განპირობებულია ცხრილების შესაქმნელი ტეგების, როგორიცაა **<TABLE>**, **<TR>**, **<TD>** და სხვა, გამოყენების სიმარტივითა და მოხერხებულობით. ჩვენ ფანჯრის მთელი არე უნდა დავყოთ მართკუთხა უჯრედებად ხილული და უხილავი საზღვრებით და მათში განვათავსოთ დოკუმენტის ელემენტები (გამოსახულებები, ტექსტები, მიმართვები, ღილაკები, სხვა ცხრილები და ა. შ.). ამგვარად, ცხრილები ასრულებს დოკუმენტის კარკასის მოვალეობას.

ცხრილის ელემენტებზე მიმართვა. ცხრილს, როგორც დოკუმენტის ობიექტს აქვს ორი კოლექცია, რომლის საშუალებითაც ხდება მიმართვა მის შეგთავსზე. მათ შორის პირველი არის სტრიქონების კოლექცია **rows**, ხოლო მეორე – უჯრების კოლექცია **Cells**. კოლექცია **Rows** შეიცავს ცხრილის ყველა სტრიქონს **<THEAD>** და **<TFOOT>** ტეგების შესაბამისი განყოფილებების ჩათვლით. კოლექცია **cells** შეიცავს **<TH>** და **<TD>** ტეგების მიერ შექმნილი ცხრილების ყველა ელემენტს. კოლექციის

ელემენტებზე მიმართვა ხორციელდება ან ინდექსის ან შესაბამის ტეგში **ID** ატრიბუტის მნიშვნელობის მიხედვით. ამგვარად, ცხრილის სტრიქონზე მიმართვისათვის შეიძლება გამოვიყენოთ <TR> ტეგში **ID** ატრიბუტის მნიშვნელობა, ხოლო უჯრედზე მიმართვისათვის <TH> ან <TD> ტეგებში **ID** ატრიბუტის მნიშვნელობა. ინდექსის (ნომერის) გამოყენების შემთხვევაში უნდა გვახსოვდეს, რომ ნუმერაცია იწყება 0-დან. ამასთან, ცხრილის უჯრედების ნუმერაცია ხდება მარცხნიდან მარჯვნივ და ზემოდან ქვემოთ.

განვიხილოთ მარტივი ცხრილის მაგალითი, რომელიც შეიცავს სამ სვეტსა და ოთხ სტრიქონს:

```
<HTML>
<TABLE ID = "mytab">
<THEAD>MY TABLE</THEAD>
<TH> Surname </TH><TH> Name</TH><TH> Position</TH>
<TR  ID = "r1">
<TD>ბერიძე</TD><TD>გიორგი</TD><TD>Director</TD>
</TR>
<TR  ID = "r2">
<TD>მელაძე</TD><TD>თამაზი</TD><TD>Deputy Director</TD>
</TR>
<TR  ID = "r3">
<TD>დვალი</TD><TD>მანანა</TD><TD>Secretary</TD>
</TR>
<TR  ID = "r4">
<TD>ქერქაძე</TD><TD>ირაკლი</TD><TD>Driver</TD>
</TR>
</TABLE>
</HTML>
```



ქვემოთ მოყვანილია ცხრილის ელემენტებზე მიმართვის რამდენიმე მაგალითი:

document . all . mytab . rows [0]
document . all . mytab . rows [1]
document . all . mytab . rows ["r1"]
document . all . mytab . cells [3]
document . all . mytab . rows [2] . cells [1]
document . all . mytab . rows ["r1"] . cells [2]

იმისათვის, რომ მივმართოთ ცხრილის სტრიქონს ან უჯრედს საჭიროა გავითვალისწინოთ ობიექტის იერარქია: ჯერ ჩვენ მივმართავთ დოკუმენტის ყველა ელემენტების **all** კოლექციას და მხოლოდ ამის შემდეგ – ცხრილის ელემენტებს. ყურადღება უნდა მიექცეს იმ გარემოებას, რომ ეს მიმართვები აბრუნებს არა ცხრილის ელემენტების შიგთავსს, არამედ მხოლოდ მიმართვას ელემენტებზე, როგორც ობიექტებზე.

ზოგჯერ საჭირო ხდება ცხრილის ელემენტების შეცვლა ახალი მნიშვნელობით ან გამოსახულებით, რაც შესაძლებელია შემდეგნაირად:

document . all . mytab . rows [ინდექსი1] . cells [ინდექსი 2] .
innerText = "ახალი მნიშვნელობა"

document . all . mytab . rows [ინდექსი1] . cells [ინდექსი 2] .
innerHTML = "<IMG SRC = 'pict.jpg'>"

სტრიქონებისა და უჯრედების კოლექციას, ისევე როგორც ნებისმიერ მასივს აქვს **length** თვისება, რომლის მნიშვნელობა კოლექციაში ელემენტების რაოდენობაა:

document . all . mytab . rows . length
document . all . mytab . cells . length
document . all . mytab . rows [2] . cells . length

ცხრილში სტრიქონის დამატება და წაშლა. ცხრილში ახალი სტრიქონის დამატება **insertRow ( )** მეთოდის დახმარებით ხდება. ეს მეთოდი მიმართვას აბრუნებს ახალ შექმნილ სტრიქონზე, რომელიც შემდეგ გამოიყენება უჯრედების ჩასასმელად. უჯრედები სტრიქონში ისმება **insertCell (უჯრედის ინდექსი)** მეთოდის საშუალებით. მოცემული მეთოდი აბრუნებს მიმართვას ახალ შექმნილ უჯრედზე, რომელიც შემდგომ გამოიყენება უჯრედის შიგთავსის მოსაცემად. უჯრედები სტრიქონში ისმება მიმდევრობით, გამოტოვების გარეშე დაწყებული ნულიდან. მაგალითად,

newrow = document . all . mytab . insertRow ( )
newcell = newrow . insertCell (0)
newcell . innerText = "Hello"
newcell = newrow . insertCell (1)
newcell . innerHTML = "<IMG SRC = 'pict.jpg'>"
newcell = newrow . insertCell (2)
newcell . innerHTML = "<B>guard</B>"

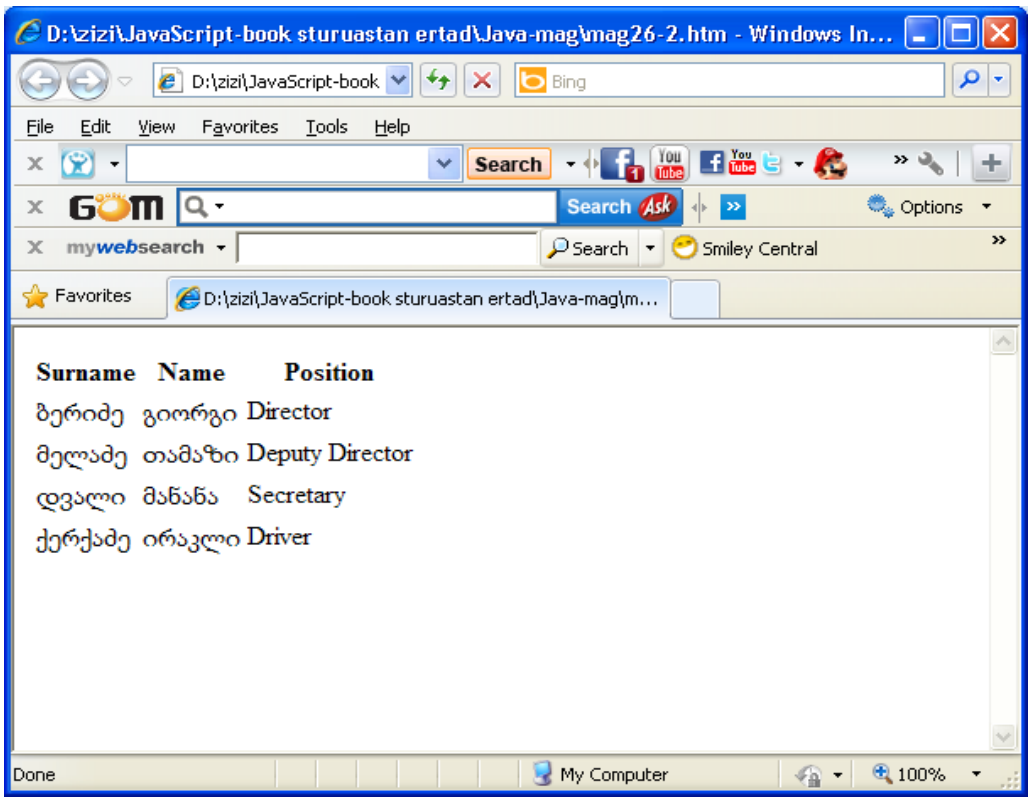
ცხრილის სტრიქონის წასაშლელად გამოიყენება **deleteRow (სტრიქონის ინდექსი)** მეთოდი. პარამეტრი მიუთითებს წასაშლელი სტრიქონის ნომერს. მაგალითად,

document . all . mytab . deleteRow (2)

ცხრილების გენერაცია სცენარის დახმარებით. **HTML** ტეგების საშუალებით ცხრილების შექმნაში ერთ-ერთ ყველაზე უხერხული მომენტი ამ ტეგების დიდი რაოდენობა არის. ამასთან დიდი ყურადღებაა საჭირო მათი სწორად განლაგებისა და კორექტირების დროს. საქმეს ძლიერ ამარტივებს უჯრების შიგთავსის შენახვა მასივების სახით და

ცხრილების გენერაცია სცენარის დახმარებით. ქვემოთ მოყვანილია ამის მაგალითი:

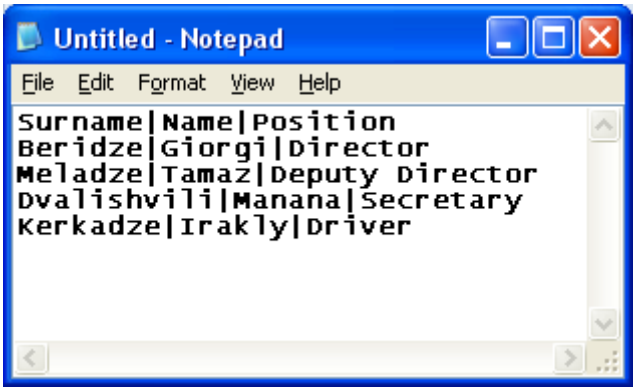
```
<HTML>
<SCRIPT>
ah = new Array ("Surname", "Name", "Position")
ad = new Array ( )
   ad[0] = new Array ("ბერიძე","გიორგი","Director")
   ad[1] = new Array ("მელაძე","თამაზი","Deputy Director")
   ad[2] = new Array ("დვალი","მანანა","Secretary")
   ad[3] = new Array ("ქერქაძე","ირაკლი","Driver")
strtab = "<TABLE>"
for (i = 0; i < ah . length; i++)  {
   strtab += "<TH>" + ah [i] + "</TH>"
}
for (i = 0; i < ad . length; i++)  {
   strtab += "<TR>"
   for (j = 0; j < ad [i] . length; j++)  {
         strtab += "<TD>" + ad [i] [j] + "</TD>"
   }
   strtab += "</TR>"
}
strtab += "</TABLE>"
document . write (strtab)
</SCRIPT>
</HTML>
```



მონაცემთა მარტივი ბაზები. ცხრილების ფორმირების ერთ-ერთი ეფექტური საშუალება ეს არის მართვის სპეციალური ელემენტი **ActiveX – Simple Tabular Data** (**STD** - მარტივი ცხრილური მონაცემები). ეს ელემენტი დოკუმენტში ჩაიწერება **<OBJECT>** ტეგის დახმარებით და საშუალებას იძლევა მარტივად ვმართოთ ჩვეულებრივ ტექსტურ ფაილში ჩაწერილი მონაცემები. **STD**-ს გამოყენებით შეგვიძლია შევცვალოთ, დავუმატოთ, წავშალოთ, მოვაწესრიგოთ, მოვძებნოთ და ამოვარჩიოთ (მოვახდინოთ ფილტრაცია) მონაცემები, მაგრამ მისი მთავარი ღირსება არის დიდი ცხრილების მარტივად შექმნა.

მონაცემები ამ დროს ინახება დისკზე ტექსტური ფაილების სახით. ამასთან, მათი ცხრილური სტრუქტურა დაცულია გამყოფი-სიმბოლოს საშუალებით. სტრიქონების გამყოფად ჩვეულებრივ გამოიყენება სტრიქონის გადაყვანის სიმბოლო (კლავიში **<Enter>**). ცალკეულ უჯრედებში მდგომი მონაცემების გამოსაყოფად გამოიყენება ნებისმიერი სიმბოლო, რომელიც სხვა მიზნებისათვის არ გამოიყენება ან არ გვხვდება მონაცემებში (მაგალითად, მძიმე ან ვერტიკალური ხაზი). ასეთი ფაილების შექმნა შესაძლებელია ნებისმიერი ტექსტური რედაქტორით (მაგალითად, **Notepad**).

ტექსტური ფაილის პირველ სტრიქონში სვეტების დასახელებებია მოცემული. მაგალითად,



STD - მონაცემთა მართვის ელემენტი **ActiveX**, ჩაშენებულია ბრაუზერში, ხოლო მისი დოკუმენტში ჩასასმელად საჭიროა შემდეგი ტეგების გამოყენება:

```
<OBJECT  ID = "mydbcontrol" CLASSID =
        "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">
    <PARAM NAME = "FieldDelim" VALUE = "|">
    <PARAM NAME = "DataURL" VALUE = "mydb.txt">
    <PARAM NAME = "UseHeader" VALUE = true>
    <PARAM NAME = "RowDelim" VALUE = "&# ASCII –
    კოდი;">
</OBJECT>
```

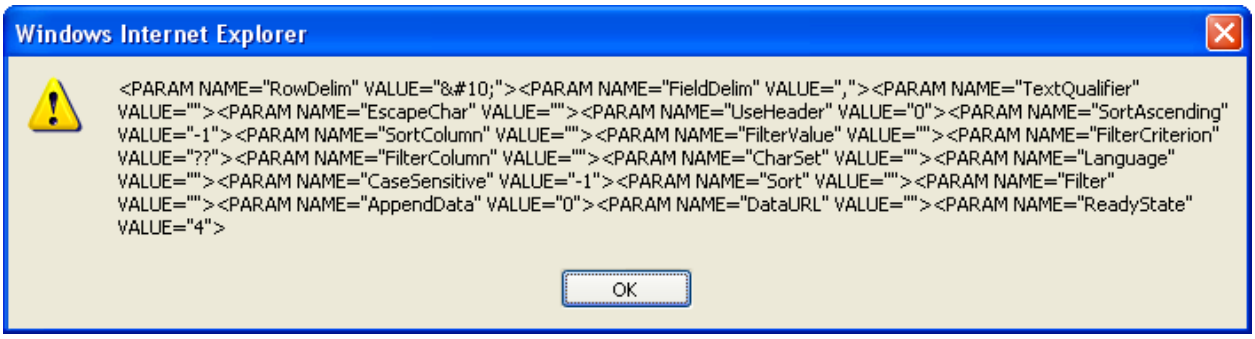
კონტეინერული ტეგი **<OBJECT>** შეიცავს მრავალ **<PARAM>** ტეგს, რომლის საშუალებითაც მოცემულია პარამეტრები. ქვემოთ მოცემულია ზოგიერთი მათგანის მნიშვნელობა:

- **FieldDelim** – ველების (უჯრედების) გამყოფი სიმბოლო;
- **DataUrl** – მონაცემთა ტექსტური ფაილის ადგილმდებარეობა (**URL**-მისამართი);
- **UseHeader** – განსაზღვრავს, შეიცავს თუ არა ტექსტური ფაილის პირველი სტრიქონი ველების დასახელებას;
- **RowDelim** – ტექსტურ ფაილში სტრიქონების გამყოფი სიმბოლო. გაჩუმების პრინციპით **VALUE** ატრიბუტის მნიშვნელობაა "**&#10**", ანუ სტრიქონის გადაყვანის სიმბოლოს კოდი.

გაჩუმების პრინციპით **STD**-ს მართვის ელემენტების ყველა პარამეტრების დათვალიერების მიზნით საკმარისია ბრაუზერში ჩაიტვირთოს შემდეგი **HTML**-დოკუმენტი:

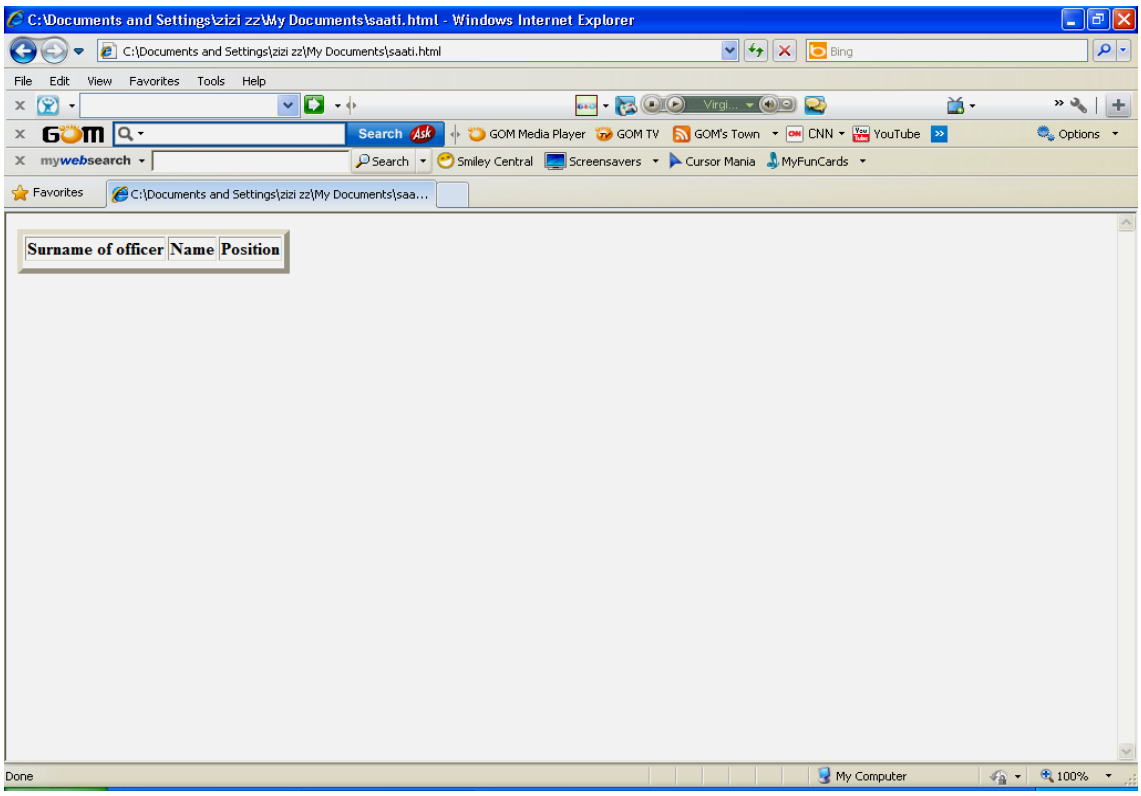
```
<HTML>
<OBJECT  ID = "mydbcontrol" CLASSID =
      "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">
</OBJECT>
<SCRIPT>
alert (mydbcontrol . innerHTML)
</SCRIPT>
</HTML>
```

ეკრანზე გამოვა შემდეგი სახის დიალოგური ფანჯარა:



იმისათვის, რომ ბრაუზერის ფანჯარაში მონაცემები **mydb.txt** ტექსტური ფაილიდან აისახოს ცხრილში საჭიროა იმავე დოკუმენტში ჩაიწეროს შემდეგი კოდი:

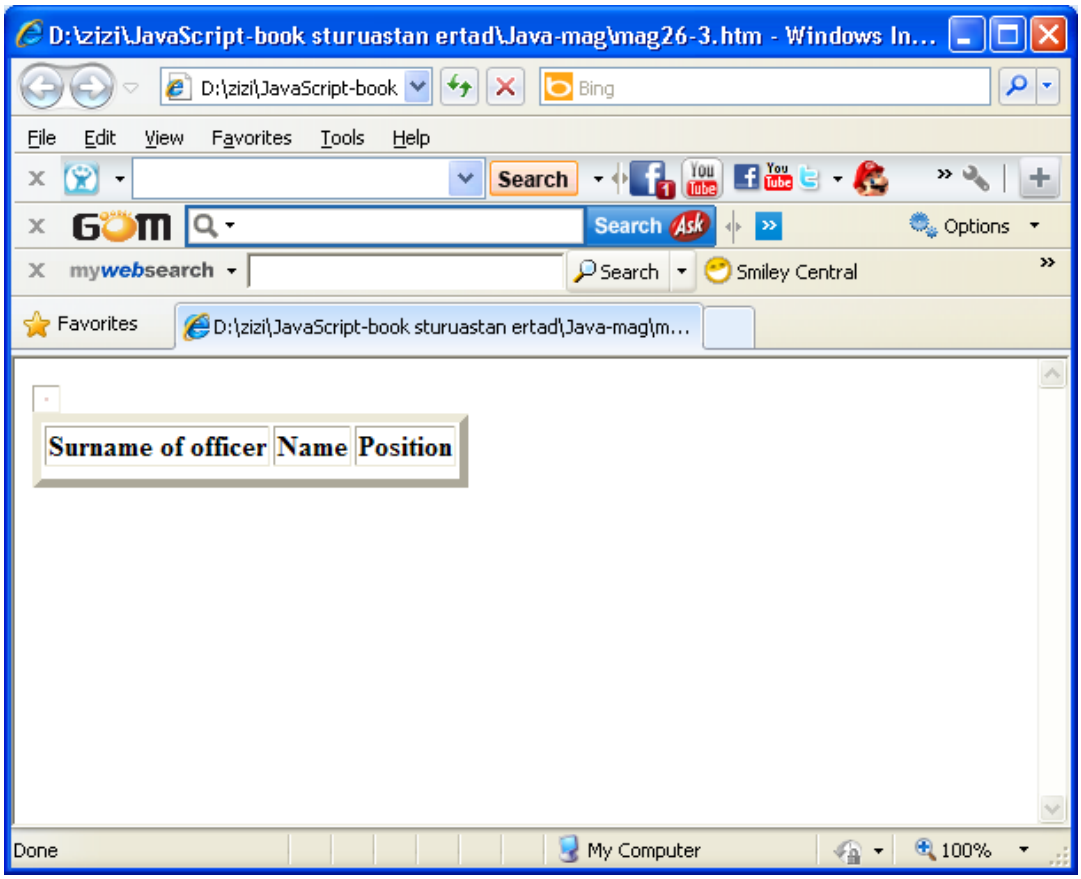
```
<TABLE DATASRC = #mydbcontrol BORDER =5>
<THEAD>
<TH>Surname of
      officer</TH><TH>Name</TH><TH>Position</TH>
</THEAD>
<TR>
<TD><SPAN DATAFLD = "Surname"></SPAN></TD>
<TD><SPAN DATAFLD = "Name"></SPAN></TD>
<TD><SPAN DATAFLD = "Position"></SPAN></TD>
</TR>
</TABLE>
```



მონაცემთა ტექსტურ ფაილში ტექსტური ინფორმაციის გარდა შეიძლება იყოს **HTML**-კოდიც. იმისათვის, რომ ეს კოდები გამოისახოს ცხრილში არა მარტო როგორც ტექსტი, ამისათვის საჭიროა **<TD>** ტეგში დაემატოს ატრიბუტი **DATAFORMATAS = "html"**. ამასთან, თუ ცხრილის უჯრედი არ შეიცავს ტეგებს, მაშინ მასში უბრალო ტექსტი აისახება, წინააღმდეგ შემთხვევაში კი **HTML**-კოდის შესრულების შედეგი. ეს საშუალებას მოგვცემს ცხრილის უჯრედში ჩავსვათ გამოსახულება, ჰიპერმიმართვა, ღილაკი და სხვა ელემენტი. ქვემოთ მოყვანილია ამის მაგალითი:

```
<HTML>
<OBJECT  ID = "mydbcontrol" CLASSID =
          "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">
    <PARAM NAME = "FieldDelim" VALUE = "|">
    <PARAM NAME = "DataURL" VALUE = " mydb.txt">
    <PARAM NAME = "UseHeader" VALUE = true>
</OBJECT>
<TABLE DATASRC = #mydbcontrol BORDER = 5>
<THEAD>
<TH>Surname of
    officer</TH><TH>Name</TH><TH>Images</TH>
```

```
</THEAD>
<TR>
<TD><SPAN DATAFLD = "Surname"></SPAN></TD>
<TD><SPAN DATAFLD = "Name"></SPAN></TD>
<TD><SPAN DATAFLD = "Portrait"
DATAFORMATAS = "html"></SPAN> </TD>
</TR>
</TABLE>
</HTML>
```



STD-ს მართვის ელემენტს საშუალება აქვს გამოიყენოს სხვადასხვა სახის პარამეტრები. კერძოდ, არის ფილტრისა (ამორჩევის) და დალაგების (მოწესრიგების) მოსამართი პარამეტრები.

ფილტრის მოსამართად გათვალისწინებულია შემდეგი სამი პარამეტრი:

```
<PARAM NAME = "FilterColumn" VALUE = "ველის სახელი">
<PARAM NAME = "FilterCriterion" VALUE = "შედარების
    ოპერატორი">
<PARAM NAME = "FilterValue" VALUE = "ნიმუში">
```

შედარების ოპერატორად გამოიყენება: =, ==, !=, >, <, >= და <= ოპერატორები.

მაგალითად:

<PARAM NAME = "FilterColumn" VALUE = "Surname">
<PARAM NAME = "FilterCriterion" VALUE = "==">
<PARAM NAME = "FilterValue" VALUE = "Beridze">

იმისათვის, რომ ფილტრი გამოვრთოთ და მთელი ცხრილი მისაწვდომი გავხადოთ საჭიროა პირობა გამოვიყენოთ, რომელსაც აკმაყოფილებს ცხრილის ყველა სტრიქონი. მაგალითად:

<PARAM NAME = "FilterCriterion" VALUE = "=">
<PARAM NAME = "FilterValue" VALUE = "">

გაჩუმების პრინციპით ფილტრი რეგისტრზეც რეაგირებს, რომელშიც მონაცემებია აკრებილი. ჩვენ შეგვიძლია რეგისტრის მართვაც. ამისათვის შემდეგი პარამეტრი გამოიყენება:

<PARAM NAME = "CaseSensitive" VALUE = "მნიშვნელობა">

VALUE ატრიბუტს შეუძლია მიიღოს მნიშვნელობები **0/false** (არ არის დამოკიდებული) ან **1/true** (დამოკიდებულია).

მონაცემთა დასალაგებლად (მოწესრიგება)შემდეგი პარამეტრი გამოიყენება:

<PARAM NAME = "SortColumn" VALUE = "ველის სახელი">

გაჩუმების პრინციპით დალაგება ხდება სიმბოლოთა კოდის ზრდადობის მიხედვით, მაგრამ ჩვენ შეგვიძლია იგი შემდეგი პარამეტრით შევცვალოთ:

<PARAM NAME = "SortAscending" VALUE = 0>

საწყის მდგომარეობას დავუბრუნდებით როცა **VALUE = 1**.

ცხრილის სტრიქონებს შორის გადაადგილების საშუალებას გვაძლევს **recordset** ობიექტი. მისი სინტაქსია:

ობიექტის id . recordset . მეთოდი

მონაცემთა ბაზის ჩანაწერებს შორის გადაადგილების მეთოდებია:

- **movePrevios ( )** – წინა ჩანაწერზე გადასვლა;
- **moveNext ( )** – მომდევნო ჩანაწერზე გადასვლა;
- **moveFirst ( )** – პირველ ჩანაწერზე გადასვლა;

- **moveLast ( )** – ბოლო ჩანაწერზე გადასვლა;

მომდევნო **moveNext ( )** და **movePrevios ( )** წინა ჩანაწერზე გადასვლამდე უნდა შემოწმდეს **eof** (მონაცემთა ფაილის დასასრული) და **bof** (მონაცემთა ფაილის დასაწყისი) თვისებათა მნიშვნელობა. ვინაიდან, თუ მიმდინარე სტრიქონი არის ბოლო, მაშინ არ შეიძლება **moveNext ( )** მეთოდის გამოყენება და ანალოგიურად, თუ მიმდინარე სტრიქონი არის პირველი, მაშინ არ შეიძლება **movePrevios ( )** მეთოდის გამოყენება.

ქვემოთ მოყვანილია **HTML**-კოდის მაგალითი, სადაც ეკრანზე გამოდის მონაცემთა ბაზის ერთი ჩანაწერი ნავიგაციის ღილაკებით:

```
<HTML>
<HEADER><TITLE> Example forms of
      STD</TITLE></HEADER>
<OBJECT  ID = "mydbcontrol" CLASSID =
                  "CLSID:333C7BC4-460F-11D0-BC04-
                  0080C7055A83">
      <PARAM NAME = "FieldDelim" VALUE = "|">
      <PARAM NAME = "DataURL" VALUE = " mydb.txt">
      <PARAM NAME = "UseHeader" VALUE = true>
</OBJECT>
<!  Field Data>
<TABLE WIDTH = 75%>
<TR><TH>Surname</TH>
<TD>
< INPUT TYPE = "text" DATASRC = #mydbcontrol DATAFLD =
      "Surname">
</TD></TR>
<TR><TH>Name</TH>
<TD>< INPUT TYPE = "text" DATASRC = #mydbcontrol
      DATAFLD = "Name">
</TD></TR>
<TR><TH>Portrait</TH>
<TD><SPAN  DATASRC = #mydbcontrol  DATAFLD = "Portrait"
            DATAFORMATAS = "html"></SPAN>
 </TD></TR>
</TABLE>
<P>
```

```
<! Buttons displaced by recordings >
<INPUT  NAME = "cmdFirst" TYPE = "BUTTON" VALUE =
        "<<" onclick = "First ( ) ">
<INPUT  NAME = "cmdPrevious" TYPE = "BUTTON" VALUE =
        "<" onclick = "Previous ( ) ">
<INPUT  NAME = "cmdNext" TYPE = "BUTTON" VALUE = ">"
        onclick = "Next ( ) ">
<INPUT  NAME = "cmdLast" TYPE = "BUTTON" VALUE =
        ">>" onclick = "Last ( ) ">
<SCRIPT>
function First ( )  {
mydbcontrol . recordset . moveFirst ( )
}
function Previous ( )  {
if (!mydbcontrol . recordset . bof)
mydbcontrol . recordset . movePrevious ( )
}
function Next ( )  {
if (!mydbcontrol . recordset . eof)
mydbcontrol . recordset . moveNext ( )
}
function Last ( )  {
mydbcontrol . recordset . moveLast ( )
}
</SCRIPT>
</HTML>
```

48. ცხრილის მონაცემთა დალაგება (მოწესრიგება)

განვიხილოთ ცხრილის სტრიქონების დალაგება ამა თუ იმ სვეტის მნიშვნელობის მიხედვით. ჩვენს მაგალითში მონაცემთა მოწესრიგების მიზნით საკმარისია მაუსით დავაწკაპუნოთ ცხრილის საჭირო სვეტის სათაურზე.

```
<HTML>
<OBJECT  ID = "mydbcontrol" CLASSID =
        "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">
```

```
        <PARAM NAME = "FieldDelim" VALUE = "|">
        <PARAM NAME = "DataURL" VALUE = " mydb.txt">
        <PARAM NAME = "UseHeader" VALUE = true>
        <PARAM NAME = "SortColumn" VALUE = "Surname">
        <PARAM NAME = "SortAscending" VALUE = 1>
</OBJECT>
<TABLE  DATASRC = #mydbcontrol  BORDER = 5>
<THEAD >
<TH  onclick = "sort ('Surname')">Surname of officer</TH>
<TH  onclick = "sort ('Name')">Name</TH>
<TH>Portrait</TH>
</THEAD >
<TR>
<TD><SPAN DATAFLD = "Surname"></SPAN></TD>
<TD><SPAN DATAFLD = "Name"></SPAN></TD>
<TD><SPAN  DATAFLD = "Portrait"
      DATAFORMATAS = "html"> </SPAN>
</TD></TR>
</TABLE>
<SCRIPT>
var x = document . all . mydbcontrol . innerHTML
var obj = '<OBJECT  ID = "mydbcontrol" CLASSID =
      "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">' + x
function  sort (field)  {
var  y = document . all . mydbcontrol
y . outerHTML = obj + '<PARAM  NAME = "SortColumn"
      VALUE = " ' + field + ' "></OBJECT>
}
</SCRIPT>
</HTML>
```

მოცემულ მაგალითში მონაცემების დალაგება ტექსტური ველების მიხედვით ზრდადობით ხდება. იგივე მაგალითი შეიძლება ისე გადავაკეთოთ, რომ მონაცემების დალაგება განხორციელდეს საწინააღმდეგო მიმდევრობით, რისთვისაც სცენარში უნდა ჩავწეროთ **<PARAM NAME = "SortAscending" VALUE = 0>** პარამეტრი.

49. ცხრილის მონაცემთა ფილტრაცია

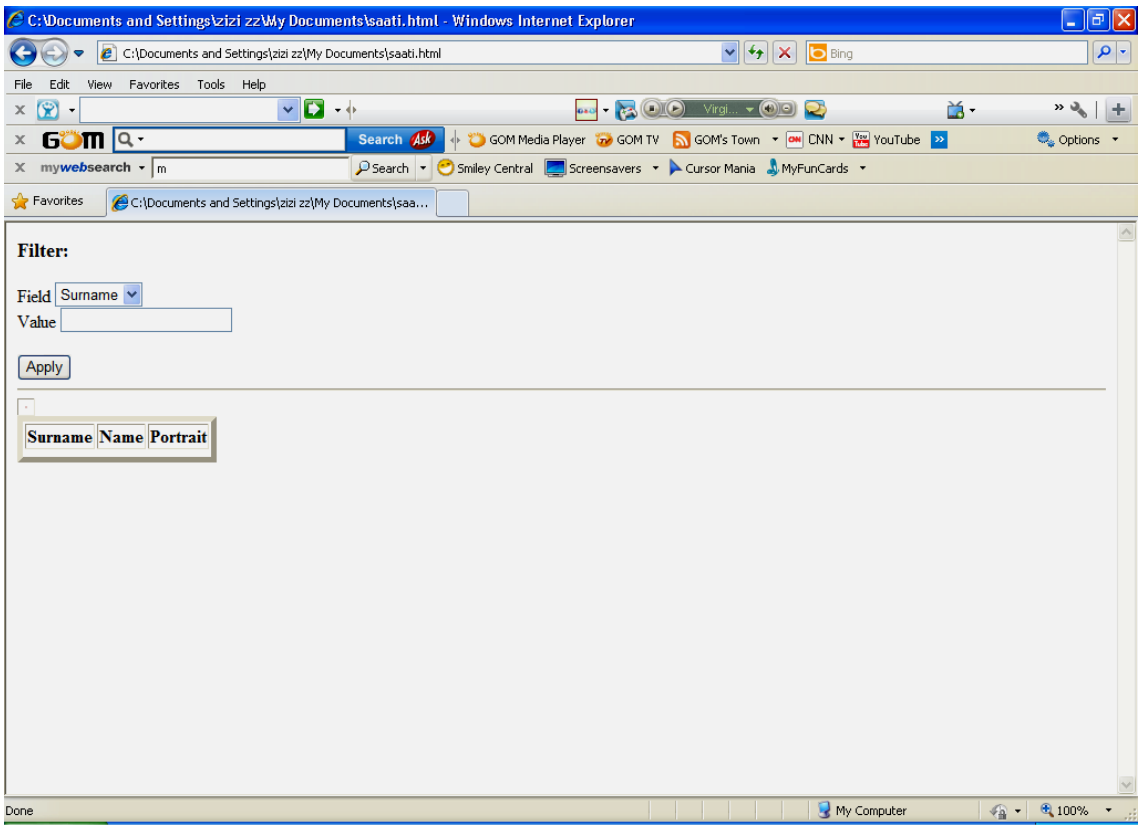
მოცემული კრიტერიუმით ცხრილის მონაცემების ფილტრაციისათვის მოვიყვანოთ მაგალითი, სადაც ბრაუზერის ფანჯარაში გამოტანილია ცხრილი მონაცემებით და ელემენტებით, რომლის საშუალებითაც ფილტრის მარტივი პირობის ფორმირებისათვის პროგრამას შეიძლება მივაწოდოთ ველი (სვეტი) და მნიშვნელობა, შემდეგი სახით:

ველის სახელი = მნიშვნელობა

ველის სახელი ამოირჩევა ჩამოშლადი სიიდან, ხოლო მნიშვნელობა კლავიატურის დახმარებით შეგვყავს. ფილტრის დასაყენებლად მაუსით უნდა დავაწკაპუნოთ **Apply** ღილაკზე. თუ მნიშვნელობის შეტანის ველი ცარიელია, მაშინ ამ ღილაკზე დაწკაპუნება გამოიწვევს მთელი ცხრილის გამოტანას. აქვე უნდა აღინიშნოს, რომ ჩვენს მაგალითში შედეგი არ არის დამოკიდებული რეგისტრზე.

```
<HTML>
<H3>Filter:</H3>
Field
<! A drop-down list the field names >
<SELECT  NAME = "FLD">
<OPTION  value = "Surname">Surname
<OPTION  value = "Name">Name
</SELECT>
<BR>
Value
<! The input field values to filter button >
<INPUT  NAME = "INP" VALUE = " "  TYPE = "text">
<P>
<BUTTON  onclick = "filter ( ) "> Apply</BUTTON>
<HR>
<! Control STD >
<OBJECT  ID = "mydbcontrol" CLASSID =
          "CLSID:333C7BC4 460F 11D0 BC04 0080C7055A83">
     <PARAM NAME = "FieldDelim" VALUE = "|">
     <PARAM NAME = "DataURL" VALUE = "mydb.txt">
     <PARAM NAME = "UseHeader" VALUE = true>
```

```
</OBJECT>
<! Table to display the data >
<TABLE  DATASRC = #mydbcontrol  BORDER = 5>
<THEAD >
<TH>Surname</TH>
<TH>Name</TH>
<TH>Portrait</TH>
</THEAD >
<TR>
<TD><SPAN DATAFLD = "Surname"></SPAN></TD>
<TD><SPAN DATAFLD = "Name"></SPAN></TD>
<TD><SPAN  DATAFLD = "Portrait"
          DATAFORMATAS = "html"></SPAN>
</TD></TR>
</TABLE>
<SCRIPT>
var obj = "<OBJECT  ID = 'mydbcontrol'  "
obj += "CLASSID = 'CLSID:333C7BC4-460F-11D0-BC04-
           0080C7055A83'>"
obj += "<PARAM NAME = 'FieldDelim' VALUE = ' | '>"
obj += "<PARAM NAME = 'DataURL' VALUE = 'mydb.txt'>"
obj += "<PARAM NAME = 'UseHeader' VALUE = true>"
function  filter ( )  {
var  cpar = " "
if (INP . value)  {
        cpar = "<param name = 'FilterColumn'  value = ' " +
              FLD . value + " ' >"
        cpar += "<param name = 'FilterValue'  value = ' " +
              INP . value + " ' >"
        cpar += "<param name = 'FilterCriterion'  value = '=' >"
        cpar += "<PARAM  NAME = 'CaseSensitive'  VALUE =
              false >"
        }
document . all . mydbcontrol . outerHTML = obj + cpar +
         "</OBJECT>"
}
</SCRIPT>
</HTML>
```



50. ძებნა საიტზე

საძიებო სისტემის საფუძველია მონაცემთა ბაზა, რომელიც შეიცავს საძიებო სახეს (გამხსნელი სიტყვა) და **Web**-გვერდზე მიმართვებს შორის დამოკიდებულებას. ეს საძიებო მონაცემთა ბაზა ჩვეულებრივ ტექსტურ ფაილში იქმნება, რომელსაც ჩვენს მაგალითში **search.txt** ჰქვია. მასში არის მხოლოდ ორი სვეტი **p1** და **p2**. პირველი სვეტი შეიცავს გამხსნელი სიტყვებს, ხოლო მეორე – იმ **HTML**-დოკუმენტებზე მიმართვებს, რომლებშიც არის შესაბამისი გამხსნელი სიტყვები. ტექსტურ ფაილში ველების გამყოფად გამოიყენება ვერტიკალური ხაზი. ამგვარად, მონაცემთა ბაზის სტრუქტურას აქვს შემდეგი სახე:

p1|p2

გამხსნელი სიტყვა|მიმართვა

უპირველეს ყოვლისა, კარგად უნდა იყოს გააზრებული გამხსნელი სიტყვების სიტემა, რადგან ძებნა იყოს ეფექტური. შემდეგ უნდა დამუშავდეს საძიებო სისტემის მუშაობის ალგორითმი. კერძოდ, უნდა

გადაწყდეს ძებნა დამოკიდებული იქნება თუ არა შეტანილი საძიებო სახის რეგისტრზე, ძებნის შედეგად გამოიტანოს თუ არა პირველივე ნაპოვნი დოკუმენტი, თუ გამოიტანოს ყველა ნაპოვნ დოკუმენტზე მიმართვა და ა. შ.

```
<HTML>
<! The control for the table of links >
<OBJECT  ID = "fnd1" CLASSID =
        "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">
    <PARAM NAME = "FieldDelim" VALUE = "|">
    <PARAM NAME = "DataURL" VALUE = "search.txt">
    <PARAM NAME = "UseHeader" VALUE = true>
    <PARAM NAME = "CaseSensitive" VALUE = false>
</OBJECT>
<! The control for the keyword table >
<OBJECT  ID = "fnd2" CLASSID =
        "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">
    <PARAM NAME = "FieldDelim" VALUE = "|">
    <PARAM NAME = "DataURL" VALUE = " search.txt">
    <PARAM NAME = "UseHeader" VALUE = true>
    <PARAM NAME = "CaseSensitive" VALUE = false>
    <PARAM NAME = "CharSet" VALUE = "windows-1251">
</OBJECT>
<! The interface of the search engine >
<! Field input image >
<INPUT  TYPE ="text" Name ="WORD" VALUE ="" SIZE =20>
<! Button start searching >
<BUTTON  onclick = "findkeyword ( ) ">Search</BUTTON><BR>
<! It will appear in the search results >
<B  ID = "rezult"></B>
<! Table of keywords >
<TABLE  STYLE = "visibility : hidden">
<TR>
<TD><INPUT  NAME = "f1" DATASRC = #fnd2  DATAFLD =
    "p1"> </TD>
</TR>
</TABLE>
<! Table of links >
```

```
<TABLE  ID = "mytab" DATASRC = #fnd1  WIDTH = 350
      ALIGN = LEFT  STYLE = "visibility : hidden">
<TR>
<TD><SPAN  DATAFLD = "p2" DATAFORMATAS = "html">
      </SPAN></TD>
</TR>
</TABLE>
<SCRIPT>
var obj = "<OBJECT  ID = 'fnd1'  "
obj += "CLASSID = 'CLSID:333C7BC4-460F-11D0-BC04-
            0080C7055A83'>"
obj += "<PARAM NAME = 'FieldDelim' VALUE = ' | '>"
obj += "<PARAM NAME = 'DataURL' VALUE = 'search.txt'>"
obj += "<PARAM NAME = 'UseHeader' VALUE = true>"
obj += "<PARAM NAME = "CaseSensitive" VALUE = false>"
function  findkeyword ( )  {
if (!WORD . value) return
var  xfltr = "" . y . cpar
fnd2 . recordset . moveFirst ( )
while (!fnd2 . recordset . eof)  {
      y = f1 . value . toLowerCase ( )
      if (y . indexOf (WORD . value . toLowerCase ( ) ) >= 0)  {
            xfltr = f1 . value
                  break
      }
      fnd2 . recordset . moveNext ( )
}
if (!xfltr)
            document . all . rezult . innerText = "Nothing found"
      else
            document . all . rezult . innerText = ""
cpar = obj + "<param name = 'FilterColumn'  value = 'p1' >"
cpar += "<param name = 'FilterCriterion'  value = '=' >"
cpar += "<param name = 'FilterValue'  value = ' " + xfltr + " ' >
          </OBJECT>"
document . all . fnd1 . outerHTML = cpar
mytab . style . visibility = "visible"
}
```

```
</SCRIPT>
</HTML>
```

განხილულ მაგალითში ძებნა დამოკიდებული არ არის საძიებო სახის რეგისტრზე, სისტემა სტრიქონ-სტრიქონ ათვალიერებს მონაცემთა ბაზის პირველი ველის მნიშვნელობას, ამოწმებს შეიცავს თუ არა პირველი ველის მნიშვნელობა მომხმარებლის მიერ შეტანილ საძიებო სახეს. თუ შეიცავს, მაშინ ძებნა წყდება, ხოლო წინააღმდეგ შემთხვევაში ძებნა გრძელდება. ძებნის შედეგად დოკუმენტზე გამოდის მიმართვა. მოცემულ მიმართვაზე მაუსით დაწკაპუნება ამ დოკუმენტს ბრაუზერის ფანჯარაში გამოიტანს. თუ ძებნა უშედეგოდ დასრულდა, მაშინ შესაბამის შეტყობინებას გამოიტანს.

ძებნის ინტერფეისი მარტივია: საძიებო სახის შესატანი ველი და ძებნის პროცედურის დაწყების ღილაკი. შედეგს გამოიტანს შეტანის ველის ქვემოთ.

ჩვენ განვიხილეთ საძიებო სისტემის ყველაზე მარტივი ვარიანტი. მისი სრულყოფა შეიძლება რამდენიმე მიმართულებით განხორციელ-დეს. ქვემოთ მოვიყვანოთ მაგალითი, სადაც საძიებო სისტემა დაათვალიერებს მონაცემთა ბაზის ყველა ჩანაწერს და შედეგად გამოიტანს საძიებო სახის შესაბამის ყველა მიმართვას. ამ შემთხვევაში მონაცემთა ფილტრაცია ამოვარდება და კოდი უფრო გამარტივდება.

```
<HTML>
<! Control >
<OBJECT  ID = "fnd1" CLASSID =
          "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">
     <PARAM NAME = "FieldDelim" VALUE = "|">
     <PARAM NAME = "DataURL" VALUE = "search.txt">
     <PARAM NAME = "UseHeader" VALUE = true>
     <PARAM NAME = "CaseSensitive" VALUE = false>
</OBJECT>
<! The interface of the search engine >
<! Field input image >
<INPUT  TYPE ="text" Name ="WORD" VALUE ="" SIZE =20>
<! Button start searching >
<BUTTON  onclick = "findkeyword ( ) ">Search</BUTTON><BR>
<! It will appear in the search results >
```

```
<B  ID = "rezult"></B>
<! Table searchable database>
<TABLE  STYLE = "visibility : hidden">
<TR>
<TD><INPUT  NAME = "f1" DATASRC = #fnd1  DATAFLD =
     "p1"> </TD>
<TD><INPUT  NAME = "f2" DATASRC = #fnd1  DATAFLD =
     "p2" DATAFORMATAS = "html"></TD>
</TR>
</TABLE>
<SCRIPT>
function  findkeyword ( )  {
if (!WORD . value) return
var  xrezult = "" . y
fnd1 . recordset . moveFirst ( )
while (!fnd1 . recordset . eof)  {
     y = f1 . value . toLowerCase ( )
     if (y . indexOf (WORD . value . toLowerCase ( ) ) >= 0)  {
          xrezult += f2 . value + "<BR>"
     }
     fnd1 . recordset . moveNext ( )
}
if (!xrezult)
          document . all . rezult . innerText = "Nothing found"
     else
          document . all . rezult . innerText = ""
}
</SCRIPT>
</HTML>
```

51. HTML-დოკუმენტის ჩასმა ცხრილში

დოკუმენტი გარე **HTML**-ფაილიდან შეიძლება ჩავსვათ ცხრილში. ეს შესაძლებელია იმიტომ, რომ **HTML**-ფაილი ჩვეულებრივ ტექსტური ფაილია. ვინაიდან ამ ფაილის პირველ სტრიქონში ჩაწერილი **<HTML>**

ტეგი, ამიტომ ჩვენ იგი შეგვიძლია მონაცემთა ბაზის ველის სახელის სახით გამოვიყენოთ. ამგვარად, ჩასასმელ **HTML**-ფაილს განვიხილავთ როგორც ერთ ველიან (სვეტი) მონაცემთა ბაზას. ამ ფაილის მიმართ გვექნება მხოლოდ ერთი ძირითადი მოთხოვნა: ნებისმიერი კონტეინერული ტეგი მთლიანად უნდა განთავსდეს ერთ სტრიქონში. დასაწყისში განვიხილოთ ცხრილში ერთი **HTML**-დოკუმენტის ჩასმის მაგალითი:

```
<HTML>
<! Control >
<OBJECT  ID = "mydbcontrol" CLASSID =
        "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">
    <PARAM NAME = "FieldDelim" VALUE = "|">
    <PARAM NAME = "DataURL" VALUE = "mydocum.htm">
    <PARAM NAME = "UseHeader" VALUE = true>
</OBJECT>
<TABLE  DATASRC =#mydbcontrol  WIDTH = 350>
<TR>
<TD><SPAN  DATAFLD = "<HTML>" DATAFORMATAS =
    "html"> </SPAN>
</TD></TR>
</TABLE>
</HTML>
```

აქ იგულისხმება, რომ დოკუმენტში **<HTML>** ტეგი მთავრული ასოებითაა ჩაწერილი. თუ ის პატარა ასოებითაა ჩაწერილი, მაშინ იგი უნდა აისახოს ატრიბუტ **DATAFLD = "<html>"** მნიშვნელობაში.

ახლა განვიხილოთ ცხრილში რამდენიმე **HTML**-დოკუმენტის ჩასმის მაგალითი:

```
<SCRIPT>
var  aurl = new Array ( )
aurl [0] = "html1.htm"
aurl [1] = "html2.htm"
var  tabwidth = 800;
var  xobject = new Array ( )
xobject [0] = "<OBJECT  ID = 'idobj"
xobject [1] = " 'CLASSID =
        "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">'
```

```
xobject [1] += '<PARAM NAME = "FieldDelim" VALUE = "|">'
xobject [1] += '<PARAM NAME = "UseHeader" VALUE = true>'
xobject [1] += '<PARAM NAME = "CharSet" VALUE = "windows-
    1251">'
xobject [1] += '<PARAM NAME = "DataURL" VALUE = " '
xobject [2] += ' "></OBJECT>'
var  xtab = new Array ( )
xtab [0] = '<TABLE  WIDTH =' + tabwidth + ' DATASRC = #'
xtab [1] = '><TR><TD><SPAN  DATAFLD = "<HTML>"
    DATAFORMATAS = "html">'
xtab [1] += '></SPAN></TD></TR></TABLE>'
var  sobj = ""
for (i = 0; i < aurl . length; i ++)  {
sobj += xobject [0] + i + xobject [1] + aurl [i] + xobject [2] + xtab [0]
    + 'idobj' + i + xtab [1]
}
document . write (sobj)
</SCRIPT>
```

საცდელად ავიღოთ შემდეგი ორი მარტივი **HTML**-დოკუმენტი:

html1.htm ფაილი:

```
<HTML>
<H3>Document 1</H3>
<IMAGE  SRC = "pict.jpg">This is - just a picture
<A  HREF = "db0.htm">Description of databases in text files</A>
<BUTTON  onclick = "alert ('Hello!!! ') ">Click</BUTTON>
    This is a – button
</HTML>
```

და **html2.htm** ფაილი:

```
<HTML>
<H3>Document 2</H3>
<I>This - the contents of a document from the second file. </I>
It just the text
</HTML>
```

ჩვენ შეგვიძლია ძირითად დოკუმენტში განვათავსოთ მართვის ელემენტები (მიმართვები, ღილაკები), რომელთა საშუალებითაც ერთი და იგივე ცხრილში სხვადასხვა **HTML**-დოკუმენტები ჩაიტვირთება.

ამისათვის, აუცილებელია შესაბამისად შეიცვალოს **DataUrl** პარამეტრის მნიშვნელობა. ზემოთ განხილული მაგალითი წარმოადგენს **HTML**-დოკუმენტის ჩასმის კიდევ ერთ საშუალებას (”მცურავი” ჩარჩოს **<IFRAME>** გარდა) კლიენტის მხრიდან ანუ მომხმარებლის ბრაუზერით, და არა სერვერით.

52. ცხრილის მონაცემების დამუშავება

ხშირად მონაცემთა ბაზებში ინახება პირველადი მონაცემები, ხოლო ეკრანზე აისახება მათი დამუშავების შედეგები. ჩვეულებრივ ხდება რიცხვითი მონაცემების დამუშავება. მაგალითად, საჭიროა ცხრილის სვეტების ჯამის გამოთვლა ან უნდა ვიპოვოთ მაქსიმალური ან მინიმალური მნიშვნელობა. არსებობს უფრო რთული ამოცანებიც. ნებისმიერ შემთხვევაში უფრო მოსახერხებელია ეს მონაცემები ცხრილიდან გადავიტანოთ მასივში.

როდესაც მონაცემთა ცხრილი მთლიანად შექმნილია **HTML**-ის ტეგების დახმარებით (**STD** მართვის ელემენტის გამოყენების გარეშე), მაშინ წაკითხვის ამოცანა მარტივია. ქვემოთ მოყვანილია ამ ამოცანის გადაწყვეტის სცენარი, სადაც **<TABLE>** ელემენტს აქვს **ID = "mytab"** და საჭიროა მასივში წავიკითხოთ იმ სვეტების მნიშვნელობა, რომელთა ინდექსია **n**:

```
myarray = new Array ( )
for (i = 0; i < document . all . mytab. rows . length; i ++)  {
myarray [i] = parseFloat (document . all . mytab . rows [i] . cells [n])
}
```

აქ რიცხვით ტიპად გარდაქმნისათვის გამოყენებულია ფუნქცია **parseFloat**, რათა შენარჩუნებულ იქნეს რიცხვის წილადი ნაწილი.

განვიხილოთ ცხრილის სვეტის მნიშვნელობების წაკითხვის ამოცანა **STD** მართვის ელემენტის გამოყენებით. როგორც ცნობილია ასეთი ცხრილის მონაცემთა წყაროს წარმოადგენს ტექსტური ფაილი:

```
<HTML>
<! Control >
<OBJECT  ID = "mydatacontrol" CLASSID =
          "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">
     <PARAM NAME = "FieldDelim" VALUE = "|">
     <PARAM NAME = "DataURL" VALUE = "mydata.txt">
     <PARAM NAME = "UseHeader" VALUE = true>
</OBJECT>
<! Table>
<TABLE  ID = "mytab">
<TR>
<TD><INPUT  NAME = "zp" DATASRC = #mydatacontrol
     DATAFLD = "Salary" onchange = "zpchange ( ) ">
</TD>
<! There may be a definition the other columns of the table >
</TR>
</TABLE>
<P>
The amount of payment <B  ID ="sum"></B>
<SCRIPT>
var  azp = new Array ( )
zpchange ( )
function  zpchange ( )  {
var  S = 0, i = 0
mydatacontrol . recordset . moveFirst ( )
while (!mydatacontrol . recordset . eof)  {
azp [i] = parseFloat (document . all mytab . zp . value)
S += azp [i]
mydatacontrol . recordset . moveNext ( )
i ++
}
document . all . sum . innerText = S
}
</SCRIPT>
</HTML>
```

აქ იგულისხმება, რომ მონაცემთა წყარო განთავსებულია **mydata.txt** ფაილში და ამ ფაილში არსებობს ველი ხელფასი (**Salary**), ხოლო ველები ერთმანეთისაგან გამოყოფილია ვერტიკალური ხაზით.

53. Web-გვერდების დაცვა პაროლის საშუალებით

თუ ჩვენ გვინდა საიტი ან მისი ცალკეული გვერდი დავიცვათ პაროლის საშუალებით, მაშინ მთავარია უზრუნველვყოთ პაროლისა და აგრეთვე, დაცული გვერდების მისამართების შენახვა. ნათელია, რომ ისინი ცხადი სახით არ უნდა ფიგურირებდეს **HTML**-დოკუმენტებსა და სცენარებში. ამ ამოცანის საკმაოდ მარტივი და საიმედო გადაწყვეტა იმაში მდგომარეობს, რომ პაროლი დაემთხვეს სერვერზე განთავსებულ რომელიმე ფაილის სახელს (გაფართოების გარეშე). წინააღმდეგ შემთხვევაში მომხმარებელს ფაილზე მიმართვაზე უარი უნდა ეთქვას. ფაილის არსებობის შემოწმება ხდება **FileExists ( )** მეთოდით.

პაროლით დაცვის ამოცანის გადაწყვეტის მეორე ვარიანტი ისევ ეფუძნება პაროლისა და ფაილის სახელის დამთხვევას. მაგრამ ამ ვარიანტში არა ფაილების სისტემის ობიექტი გამოიყენება, არამედ ტექსტური მონაცემთა ბაზის მართვის **ActiveX** ელემენტი. ფაილის სახელი, რომელიც ემთხვევა პაროლს არის ტექსტური ფაილი და შეიცავს ორ სვეტს (ველი) და ორ სტრიქონს (ჩანაწერი). პირველ მონაცემთა ბაზის ველის იდენტიფიკატორები. მეორე სტრიქონში ასევე ორი სიტყვა – ამ ტექსტური ფაილის სახელი გაფართოების გარეშე და **HTML**-ფაილის სახელი ან გადასასვლელი გვერდის **URL**-მისამართი.

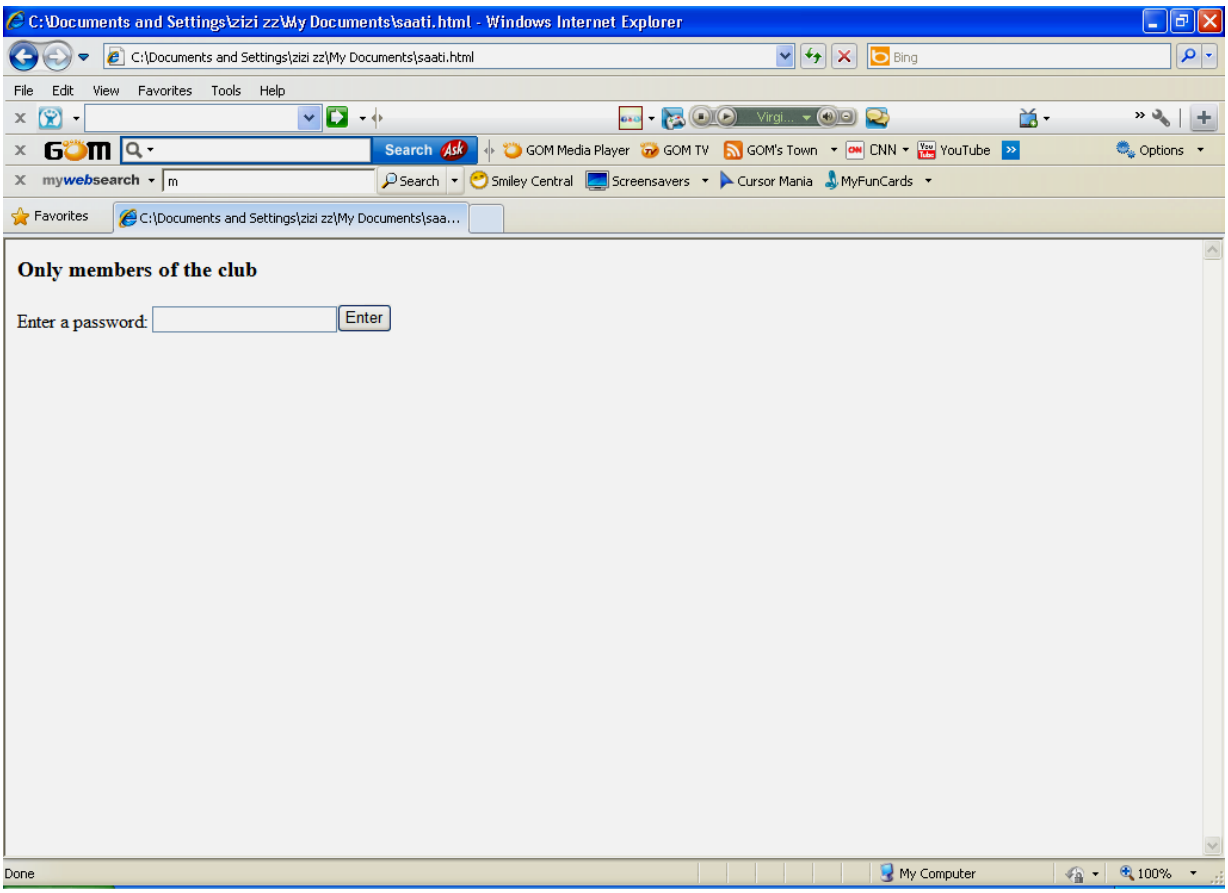
მაგალითად,

password|myspecurl
myspecfile|http://www.myweb.ge/mypage.htm

ქვემოთ მოყვანილია მაგალითი, სადაც **HTML**-დოკუმენტში განთავსებულია შეტანის ტექსტური ველი და ღილაკი:

```
<HTML>
<H3>Only members of the club</H3>
Enter a password:
<INPUT  ID = "pw1"  TYPE = "text"  VALUE = "">
```

```
<BUTTON  ID ="pwenter">Enter</BUTTON>
<B  ID = "dbelem"></B>
<SCRIPT>
function  pwenter . onclick ( )  {
if (!document . all . pw1 . value)
      return
dbstr = '<OBJECT  ID = "mydbcontrol" ' + 'CLASSID =
            "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">'
            + '<PARAM NAME = "DataURL" VALUE = " ' +
            document . all .pw1 . value + ' .txt"> ' + '<PARAM
            NAME = "FieldDelim" VALUE = "|">' +
       '<PARAM NAME = "UseHeader" VALUE =
       true></OBJECT>' +
        '<TABLE  STYLE = "visibility : hidden"><TR><TD>' +
        '<INPUT  ID = "pw2" TYPE = "text" DATASRC =
      #mydbcontrol" ' +
        ' DATAFLD = "password"></TD><TD>' +
        '<INPUT  ID = "xurl" TYPE = "text" DATASRC =
      #mydbcontrol" ' +
        ' DATAFLD = "myspecurl"></TD></TR></TABLE>'
document . all . dbelem . innerHTML = dbstr
setTimeout ("validation ( ) ", 1000)
}
function validation ( )  {
if (document . all . pw1 . value == document . all . pw2 . value)  {
      document . all . pw1 . value = " "
      window . location . href = document . all . xurl . value
}  else
      alert ("Password is not correct!")
}
</SCRIPT>
</HTML>
```



სურვილის შემთხვევაში შესაძლებელია ამ სცენარის სრულყოფა. თუ მომხმარებლის მიერ შეყვანილი პაროლი სწორი აღმოჩნდა, მაშინ მისი შენახვა **cookie**-ფაილში შეიძლება, იმისათვის რომ მომხმარებლის მიერ ამ გვერდის შემდგომში მონახულების დროს ხელახლა არ დასჭირდეს პაროლის შეყვანა.

ქვემოთ ამის მაგალითია მოყვანილი:

```
<HTML>
<H3>Only members of the club</H3>
<a  href = "#"  ID = "myref"  onclick = "pwvalid ( )">CLICK</a>
<B  ID = "dbelem"></B>
<SCRIPT>
var  pw
function  pwvalid ( )  {
pw = readCookie ("myspecrecord")
if  (!pw)
        pw = pwinput ( )
else
        pwenter ( )
```

```
}
function  pwinput ( )  {
var  inpstr = 'Enter a password:<INPUT  ID = "pw1"  TYPE =
      "text"  VALUE = "">'  +
      '<BUTTON  onclick = "pw = document . all . pw1 . value;
      pwenter ( )">Enter </BUTTON>'
function  pwenter ( )  {
if (!pw)
      return
dbstr = '<OBJECT  ID = "mydbcontrol" ' + 'CLASSID =
                  "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A8
                  3">' + '<PARAM NAME = "DataURL" VALUE =
                  " ' + pw + ' .txt"> ' +
       '<PARAM NAME = "FieldDelim" VALUE = "|">' +
       '<PARAM NAME = "UseHeader" VALUE =
       true></OBJECT>' +
        '<TABLE  STYLE = "visibility : hidden"><TR><TD>' +
        '<INPUT  ID = "pw2" TYPE = "text" DATASRC =
      #mydbcontrol" ' +
        ' DATAFLD = "password"></TD><TD>' +
        '<INPUT  ID = "xurl" TYPE = "text" DATASRC =
      #mydbcontrol" ' +
        ' DATAFLD = "myspecurl"></TD></TR></TABLE>'
document . all . dbelem . innerHTML = dbstr
setTimeout ("validation ( ) ", 1000)
}
function validation ( )  {
if (pw == document . all . pw2 . value)  {
      window . location . href = document . all . xurl . value
      var  d = new Date ( )
      var  d2 = d . getTime ( ) + (365 * 24 * 60 * 60 * 1000)
      d . setTime (d2)
      writeCookie ("myspecrecord", pw, d)
}  else
      alert ("Password is not correct!")
}
function  readCookie (name)  {
var  xname + name + "="
```

```
var  xlen = xname . length
var  clen = document . cookie . length
var  i = 0
while (i < clen)  {
var  j = i + xlen
if  (document . cookie . substring (i, j) == xname)
      return  getCookieVal (j)
i = document . cookie . indexOf (" ", 1) + 1
if (i == 0)  break
}
return  null
}
function  getCookieVal (n)  {
var  endstr = document . cookie . indexOf (";", n)
if (endstr == -1)
      endstr = document . cookie . length
return  unescape (document . cookie . substring (n, endstr))
}
function  writeCookie (name, value, expires, path, domain, secure)  {
      document . cookie = name + "=" + escape (value) +
      ((expires) ? "; expires =" + expires . toGMTString ( ) ; "") +
      ((path) ? "; path =" + path ; "") +
      ((domain) ? "; domain =" + domain ; "") +
      ((secure) ? "; secure" ; "")
}
</SCRIPT>
</HTML>
```

ამ მაგალითში პაროლის შენახვის ვადა არის 1 წელი.

ს ა რ ჩ ე ვ ი

რედაქტორი ნ. დოლიძე

გადაეცა წარმოებას 01.09.2011. ხელმოწერილია დასაბეჭდად 11.10.2011. ქაღალდის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 15,5. ტირაჟი 100 ეგზ.

საგამომცემლო სახლი "ტექნიკური უნივერსიტეტი",
თბილისი, კოსტავას 77