გ. ღვინეფაძე

Javascript

და

მისი შესაძლებლობების განმავითარებელი

თანამედროვე ტექნოლოგიები

"ტექნიკური უნივერსიტეტი"

საქართველოს ტექნიკური უნივერსიტეტი

გ. ღვინეფაძე

Javascript

და

მისი შესაძლებლობების განმავითარებელი თანამედროვე ტექნოლოგიები

დამტკიცებულია სახელმძღვანელოდ
სტუ-ს სარედაქციო-საგამომცემლო
საბჭოს მიერ, ოქმი N2, 28.10.2015

თბილისი
2015

უაკ 681.3.06

წიგნში განხილულია WEB-დოკუმენტების შესაქმნელად განკუთვნილი სცენარების სპეციალიზებული ენა Javascript და ამ ენის შესაძლებლობების განმავითარებელი, ბოლო წლებში შემუშავებული ტექნოლოგიები: Ajax, jQuery და Json.

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

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

© გამომცემლობა “ტექნიკური უნივერსიტეტი”, 2015

ISBN

http://www.gtu.ge/publishinghouse/

ყველა უფლება დაცულია. ამ წიგნის ნებისმიერი ნაწილის (ტექსტი, ფოტო, ილუსტრაცია თუ სხვა) გამოყენება არც ერთი ფორმითა და საშუალებით (ელექტრონული თუ მექანიკური) არ შეიძლება გამომცემლის წერილობითი ნებართვის გარეშე.

საავტორო უფლებების დარღვევა ისჯება კანონით.

JavaScript

შესავალი

WWW (მსოფლიო აბლაბუდა) **Web**-დოკუმენტების შესაქმნელად თავდაპირველად მხოლოდ **HTML** ენის შესაძლებლობებს იყენებდა. ამ მარტივი ენის დესკრიპტორებით (ელემენტებით) ხდება დოკუმენტის ფორმატირება, რაც ბროუზერს საშუალებას აძლევს **Web**-ფურცელი ავტომატურად ასახოს ეკრანზე. მაგრამ ეკრანზე დინამიზმის შესატანად (სურათების ცვლა, ინტერაქტიური ელემენტების გამოყენება), ასევე, ფორმების შევსების კონტროლისა და კომპიუტერთან დიალოგის ორგანიზების მიზნით, საჭირო გახდა უფრო რთული, ე. წ. **სცენარების** მომზადების ენების გამოყენება. ისინი, ჩვენთვის ნაცნობი დაპროგრამების ენებისაგან განსხვავებით, არ საჭიროებენ პროგრამების კომპილაციას – თანმიმდევრულად სრულდება პროგრამის, რომელსაც აქ სცენარს (სკრიპტსაც) უწოდებენ, თითოეული სტრიქონი, ანუ ხდება მისი **ინტერპრეტირება.**

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

სცენარების მომზადების ენებს შორის დღეს მსოფლიოში ყველაზე პოპულარულია კომპანია **Netscape Communication Corporation-**ის მიერ შექმნილი **JavaScript** ენა. მასზე შექმნილი სცენარები გასაგებია ყველა პოპულარული ბროუზერისთვის **(Internet Explorer, Firefox, Netscape, Safari, Opera, Camino** და სხვ.), მაშინ, როდესაც, მაგალითად, ასევე სკრიპტული **VBScript** ენა მხოლოდ **Internet Explorer-**ზეა ორიენტირებული.

თუ როგორი მნიშვნელოვანი (და საინტერესო) დავალებების შესრულება შეუძლია ამ ენას, წარმოდგენა შეიძლება შეგვექმნას კომპანია **Google**-ის ისეთ

ნამუშევრებთან გაცნობისას, როგორიცაა, მაგალითად, **Google Maps** ანდა **GMail** სამსახური.

JavaScript ენის კონსტრუქციებს გარეგნულად ბევრი აქვს საერთო **Java** ენის შესაბამის კონსტრუქციებთან, მაგრამ ეს უკანასკნელი არის უფრო მძლავრი, მაშასადამე, უფრო რთული ენაც.

Java-ზეც შეიძლება **Web**-ფურცლებზე გამოსაყვანი ფრაგმენტებისათვის სპეციალური პროგრამების – ე. წ. *აპლეტების* დაწერა, მაგრამ ბროუზერის მიერ მათი უშუალოდ გამოყენება ვერ ხერხდება – აპლეტები ჯერ საკლასიფიკაციო ფაილებში უნდა იქნეს კომპილირებული. არის სხვა სირთულეებიც, რის გამოც, თუ განსაკუთრებული დანიშნულების ამოცანების გადაწყვეტა არ გვიწევს, **Web**-ფურცლების შექმნისას უპირატესობას ვანიჭებთ **JavaScript** ენას.

ყველა თანამედროვე ბროუზერს შეუძლია **JavaScript-**ზე დაწერილი სცენარების შესრულება, მაგრამ მუშაობის დაწყებამდე (ან შეფერხების შემთხვევაში) უნდა გადავამოწმოთ, ჩართულია კი ეს შესაძლებლობა?

ამ მიზნით, მაგალითად, **Opera** ბროუზერისათვის უნდა მივაკითხოთ შემდეგ ჩანართს:

Opera È **Settings** È **Preferences** È **Advaced** და მოვნიშნოთ **Enable JavaScript** უბანი.

ჩვენი პირველი სცენარები

Notepad ტექსტურ რედაქტორში ავკრიფოთ პროგრამული კოდი, რომელშიც ჯერ გამოყენებული იქნება მხოლოდ **HTML** ენის შესაძლებლობები (აქვე შევნიშნოთ, რომ სცენარების შესაქმნელად და გასამართავად სავსებით საკმარისია ჩვენს განკარგულებაში იყოს უმარტივესი ტექსტური რედაქტორი **Notepad,** თუმცა სპეციალიზებული რედაქტორები მეტ სერვისს გვთავაზობენ):

```
<html>
  <head>
    <title>FIRST  PAGE</title>
    <style>
       h3, p  {font-family: LitNusx}
    </style>
  </head>
  <body>
    <h3>კეთილი იყოს თქვენი მობრძანება ინტერნეტის სამყაროში!</h3>
    <p> მაშ ასე, ორშაბათიდან ვიწყებთ ახალი საქართველოს შენებას!
    </p>
   </body>
</html>
```

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

```
<html>
  <head>
    <title> Next  Page </title>
    <style>
       h3, p  {font-family: LitNusx}
    </style>
  </head>
  <body>
     <p>ვაგრძელებთ საქმიანობას! ვისახავთ ახალ მიზნებს.
         ოპერატიულად გატყობინებთ ჩვენი დოკუმენტის ბოლო
        ცვლილების თარიღს.
     </p>
```

```
<script language= "JavaScript">
    document.write(document.lastModified);
</script>
</body>
</html>
```

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

ვხედავთ, რომ ამ მაგალითში სცენარი მოთავსებულია **Web-**დოკუმენტის პროგრამის სხეულში. მაგრამ, საერთოდ, შესაძლებელია, იგი სათაურის უბანშიც განვალაგოთ. ასეთ შემთხვევაში სცენარი, როგორც წესი, სხვა სცენარების მიერ <body> უბნიდან მიერ გამოიძახება, როგორც *ფუნქცია.*

ზოგადად, **JavaScript** ენაზე დაწერილი სცენარის შემცველი **Web-**დოკუმენტი შემდეგი სტრუქტურისაა:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title > Web-ფურცლის სათაური </title>
<meta name="Generator" content="EditPlus">
<meta name="Author" content="ავტორის სახელი">
<script type="text/javascript">
აქ განთავსდება სცენარი
</script>
</head>
<body>
აქ განთავსდება Web-ფურცლის შინაარსობრივი ნაწილი
</body>
</html>
```

კოდის აკრეფისას ამჯობინებენ მის იერარქიულ სტრუქტურას გრაფიკულადაც შესაბამისი სახე მიეცეს (სპეციალიზებული რედაქტორები ამ საქმეს თვითონ უძღვებიან). მაგალითად, ჯობია, ზემოთ მოყვანილი კოდი ამგვარად წარმოვადგინოთ:

```
<html>
 <head>
   <title > Web-ფურცლის სათაური </title>
   <meta name="Generator" content="EditPlus">
   <meta name="Author" content="ავტორის სახელი">
   <script type="text/javascript">
     აქ განთავსდება სცენარი
   </script>
 </head>
 <body>
     აქ განთავსდება Web-ფურცლის  შინაარსობრივი ნაწილი
 </body>
</html>
```

თუ სცენარი მოცულობით დიდია და/ან მრავალ ფაილში გამოიყენება, მას, როგორც წესი, განათავსებენ ცალკე ფაილში, რომელსაც უკეთდება გაფართოება **js,** ხოლო **HTML**-ფაილის (ფაილების) **script**-ელემენტში კი მიეთითება მისაერთებელი ფაილის **URL**, მაგალითად, ასე:

```
<script type="text/javascript" src="scripts/JavaScriptFile.js">
</script>
```

აქვე შევნიშნოთ, რომ მოცემული მიდგომა ანალოგიურია **HTML-**ფაილთან მისგან ცალკე მდგომი **css-**ფაილის მიერთების წესის.

შემდეგ, სცენარი შეიძლება განვათავსოთ **HTML-**დესკრიპტორშიც ე. წ. **ხდომილობის დამმუშავებელი კონსტრუქციის** სახით (იხ. ქვემოთ). აქ მხოლოდ

შევნიშნავთ, რომ ამ შემთხვევაში ხდომილობის დამმუშავებლისათვის <**script**> </**script**> ფრჩხილების გამოყენება საჭირო აღარ არის.

აღვნიშნოთ, რომ სცენარში ხშირად უჩვენებენ **JavaScript** ენის ვერსიასაც. ეს კეთდება იმ მიზნით, რომ ძველმა, მაშასადამე, ნაკლები შესაძლებლობების მქონე ბროუზერებმა უნაყოფოდ არ სცადოს მათთვის გაუგებარი სცენარების შესრულება, რაც ზოგჯერ ბროუზერის "ჩამოკიდებასაც" კი იწვევს (თუმცა ბოლო ხანებში ასეთ ბროუზერებს ცოტა ვინმე თუ იყენებს).

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

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

```
<html>
 <head>
  <title>ჩემი ასაკი წამებში</title>
  <style> h2, p {font-family: LitNusx}
  </style>
 </head>
 <body>
  <h2>წამია კაცის ცხოვრება, მხოლოდ საკითხავია რამდენი?</h2>
  <hr>
  <script language="JavaScript">
    now=new Date();
    DT=new Date("jan 01 1981 00:00:00");  // უჩვენეთ დაბადების თარიღი
    seconds=(now - DT) / 1000;
    document.write("<P>ჩემი დაბადებიდან გავიდა " + seconds  + " წამი");
  </script>
 </body>
</html>
```

დოკუმენტი ჩავტვირთოთ ბროუზერში.

აქვე გავაკეთოთ ზოგადი ხასიათის შენიშვნა – სცენარის დაწერისას განსაკუთრებული ყურადღება უნდა მიექცეს შემდეგ გარემოებას:

JavaScript ენა ცვლადებსა და სხვა კონსტრუქციებში ერთმანეთისგან განასხვავებს დიდსა და პატარა ასოებს. სწორედ ამ ფაქტორის გაუთვალისწინებლობა ხდება ხშირ შემთხვევაში შეცდომის მიზეზი.

მიღებული შედეგის დამრგვალებისთვის შეიძლება გამოვიყენოთ **Math**.**round** *სპეციალური ფუნქცია.* იგი **document**.**write** ოპერატორის წინ უნდა ჩავსვათ:

seconds=**Math.round**(**seconds**); *//გამოიყენეთ ზემოთ მოყვანილ სცენარში!*

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

Math.**round** ფუნქციის მუშაობის პრინციპი გასაგები ხდება შემდეგი მაგალითებიდან:

1. x=Math.round(20.49) – გვიბრუნებს 20-ს;
2. x=Math.round(20.5) – გვიბრუნებს 21-ს;
3. x=Math.round(-20.5) – გვიბრუნებს -20-ს;
4. x=Math.round(-20.51) – გვიბრუნებს -21-ს;

გარდა **Math**.**round**-ისა, **JavaScript** ენა იყენებს რიცხვის დამრგვალების სხვა ფუნქციებსაც (მათ მეთოდების სახელითაც მოიხსენიებენ). ესენია:

- **Math**.**ceil** – გვიბრუნებს იმ უმცირეს მთელ რიცხვს, რომელიც არგუმენტს აღემატება ან მისი ტოლია.
- **Math.floor** – გვიბრუნებს იმ უდიდეს მთელ რიცხვს, რომელიც არგუმენტზე მცირე ან მისი ტოლია.
- **toPrecision** ფუნქცია. მას სინტაქსურად ასე გამოსახავენ: **numObj.toPrecision ([precision])**

ამ ფუნქციისათვის precision არგუმენტის მითითება (იგი 1 – 21 დიაპაზონის ფარგლებს არ უნდა გასცდეს) სავალდებულო არ არის. აღნიშნული არგუმენტი უჩვენებს ყ ვ ე ლ ა ციფრის რაოდენობას (“ზედმეტები” მოიკვეთება).

მოვიტანოთ ამ ფუნქციის გამოყენების მაგალითები:

Number("60").toPrecision(3) // 60.0

Number("60.1234").toPrecision(3) // 60.1

კრიტიკულ შემთხვევებში ხდება შედეგის ექსპონენციალურ ფორმაში გადაყვანა:

("60.1234").toPrecision(1) // 6e+1

- **toFixed** ფუნქცია. მას სინტაქსურად ასეთი სახით გამოსახავენ: **numObj.toFixed([fractionDigits])**

ფუნქციისათვის **fractionDigits** არგუმენტის მითითება (იგი 0 – 20 დიაპაზონის ფარგლებს არ უნდა გასცდეს) სავალდებულო არ არის. აღნიშნული არგუმენტი უჩვენებს ციფრების რაოდენობას ათობითი წერტილის შემდეგ. დუმილით ამ არგუმენტის მნიშვნელობა ნულის ტოლია). საჭიროების შემთხვევაში რიცხვი დამრგვალდება ან მთელ სიგრძემდე შეივსება ნულებით.

ფუნქციის არსში უკეთ გასარკვევად განვიხილოთ მაგალითები:

var n = 12345.6789;

n.toFixed(); // 12346: ხდება დამრგვალება, წილადური ნაწილის სიგრძე ნულია;

n.toFixed(1); // 12345.7: ხდება დამრგვალება

n.toFixed(6); // 12345.678900: ხდება ნულებით შევსება

(1.23e+20).toFixed(2); // 123000000000000000000.00

(1.23e-10).toFixed(2) // 0.00

დავალება: *კოდის ტექსტურ რედაქტორში (მაგალითად, Notepad-ში) აკრეფისას ეკრანზე ქართული ტექსტის გამოსატანად გამოიყენეთ რამდენიმე ვარიანტი: ზემოთ ნაჩვენები და TaskBar-ზე ka ოფციის არჩევით.*

ამჯერად, მიზნად დავისახოთ, რომ ზემოთ მოტანილმა სცენარმა დრო გამოთვალოს წუთებშიც და ეს შედეგიც გამოტანილი იქნეს ეკრანზე:

document.write ოპერატორის შემდეგ სცენარში ვამატებთ ასეთ ფრაგმენტს:

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

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

მივცეთ Web-ფურცელს უფრო მიმზიდველი სახე!

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

```
<html>
 <head>
  <title>შევქმნათ მორბენალი სტრიქონი!</title>
  <style>
   h2, p  {font-family: LitNusx}
  </style>
  <script language="JavaScript">
    var msg= "Hello,  Baby!";
    var spacer= "  ...                    ...  " ;
    var pos=0;
```

```
function ScrollMessage() {
  window.status=
  msg.substring(pos, msg.length) + spacer + msg.substring(0,pos);
  pos++;
    if (pos > msg.length) pos=0;
  window.setTimeout("ScrollMessage()", 200);
  }
  ScrollMessage();
 </script>
</head>
<body>
 <center><h2>მორბენალი სტრიქონის მაგალითი</h2></center>
 <p> შეხედეთ სტატუსის სტრიქონს! </p>
</body>
</html>
```

ამ კოდში გასარკვევად დაგვჭირდება **JavaScript**-ის რიგი შესაძლებლობე-ბის შესწავლა, მაგრამ მანამდე შევნიშნოთ, რომ მონიტორზე საჭირო ადგილას (და არა მარტო **Status Bar**-ში) მოძრავი სტრიქონის გამოტანა შესაძლებელია **HTML** ენის კუთვნილი **marquee** ელემენტით, ამასთან, პარამეტრების მეშვეობით აირჩევა მოძრაობის სახეები.

დავალება: *მოიძიეთ* ***marquee*** *ელემენტი ინტერნეტში და გაეცანით მის შესაძლებლობებს. იხ., მაგალითად, საიტი*

http://www.quackit.com/html/codes/html_marquee_code.cfm).

შევისწავლოთ **JavaScript** ენის ის შესაძლებლობები, რომელთა დახმარებითაც მიიღწევა ზემოთ მოყვანილ სცენარში დასახული მიზანი.

ზემოთ მოყვანილი ფაილი გავუშვათ შესრულებაზე, დავაფიქსიროთ, რა ნაკლოვანებებით ხასიათდება იგი და შევეცადოთ მათ აღმოფხვრას.

ქვემოთ გადმოცემული მასალის ათვისების შემდეგ კი დავწეროთ სცენარები სტრიქონის მოძრაობის სხვადასხვა ვარიანტებისათვის.

დავალებები:

1. *მოძრავი სტრიქონი მიემართება საპირისპირო მიმართულებით;*
2. *მოძრავი სტრიქონი სამჯერ შედის "ვვირაბში" და ჩერდება;*
3. *სტრიქონი მოძრაობს ქანქარისებურად;*
4. *სტრიქონი სამჯერ შედის "გვირაბში" ჯერ მარცხნიდან მარჯვნივ, შემდეგ იცვლის მიმართულებას, პროცესს იმეორებს ასევე სამჯერ და ჩერდება;*
5. *სტრიქონი პერიოდულად ნებართვას იღებს მომხმარებლისაგან მოძრაობის გაგრძელებაზე, მიმართულების შეცვლაზე, სისწრაფის გაზრდა-შემცირებაზე.*

ფუნქციები და ობიექტები

პირველ ყოვლისა, გავეცნოთ **ფუნქციის** ცნებას:

ფუნქცია წარმოადგენს JavaScript-ის ოპერატორების ჯგუფს, რომელიც გამოძახების შემთხვევაში სრულდება, როგორც ერთი მთლიანობა.

სცენარის ხშირად გამოსაძახებელი ერთი და იმავე ფრაგმენტის ფუნქციის სახით გაფორმება მნიშვნელოვნად ამარტივებს ამ სცენარის სტრუქტურას.

ფუნქციები არსებობს **ჩაშენებული** და **მომხმარებლის** მიერ შექმნილი.

მოვიყვანოთ მომხმარებლის მიერ ისეთი მარტივი ფუნქციის ფორმირების მაგალითი, რომლის სხეულში გამოიძახება **Javscript** ენაში ჩაშენებული **alert**() ფუნქცია:

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

ამ მაგალითში **function** საკვანძო სიტყვის გამოყენებით განვსაზღვრეთ **Greet**() ფუნქცია, რომლის გამოძახებისას *(იხ. ქვემოთ)* შესრულდება ფიგურულ ფრჩხილებში განთავსებული ოპერატორების თანმიმდევრობა. განსახილველ შემთხვევაში ეს გახლავთ ერთადერთი **alert**() ოპერატორი, რომელიც თვითონვე წარმოადგენს **ჩაშენებულ ფუნქციას.** მისი დანიშნულებაა ეკრანზე რაიმე შეტყობინების გამოტანა.

ფუნქციისათვის უფრო მეტი მოქნილობის მისაცემად უმეტეს შემთხვევაში იყენებენ პარამეტრებს *(არგუმენტებს).* მათი ჩამონათვალი მიეთითება მრგვალ ფრჩხილებში. შევნიშნოთ, რომ ეს ფრჩხილები გამოიყენება მაშინაც, როცა ფუნქცია პარამეტრების გადაცემას არ საჭიროებს.

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

გამოძახებისას ფუნქციაში შემავალი ოპერატორებით სრულდება შესაბამისი მოქმედებები.

მაგალითად, ზემოთ მოყვანილი კოდის ფრაგმენტი შეიძლებოდა ასე გაგვერთულებინა:

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

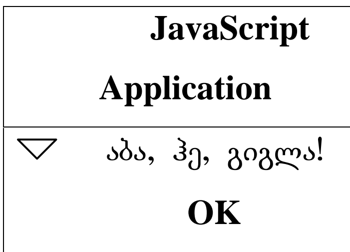
ცხადია, ფუნქციის გამოძახების მომენტში **who** ცვლადისათვის განსაზღვრული უნდა იყოს რაიმე მნიშვნელობა.

ფუნქციას ტრადიციულად <**head**> უბანში განსაზღვრავენ, ხოლო გამოძახებას ახდენენ კოდის სხეულიდან – მიუთითებენ ფუნქციის სახელს და, საჭიროებისას, განსაზღვრავენ პარამეტრების მნიშვნელობას. მაგალითად:

```
<html>
 <head>
   <title>ფუნქციების  გამოყენება</title>
```

```
    <style>  h2, p  {font-family: litnusx}  </style>
    <script language="javascript">
      function Greet(who) {
      alert("აბა,  ჰე, " + who  );
      }
     </script>
  </head>
  <body>
   <h2>ვისწავლოთ  ფუნქციებთან  მუშაობა!</h2>
   <script language="JavaScript">
     Greet("გიგლა!");
     Greet("გაგა!");
   </script>
 </body>
</html>
```

Greet ("გიგლა!") ოპერატორით ფუნქციის პირველი გამოძახებისას ეკრანზე გამოდის შემდეგი შემცველობის ფანჯარა:



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

შემდეგ, ფუნქციას არა მარტო შეტყობინების გამოტანა შეუძლია – მისი მეშვეობით ხშირად გამოთვლიან რაიმე მნიშვნელობას. ამ მიზნით, ფუნქციას შესაძლოა რამდენიმე პარამეტრი გადაეცეს, მაგრამ სცენარისთვის უკან დასაბრუნებელი მნიშვნელობა კი მხოლოდ ერთადერთი შეიძლება იყოს და ეს

მნიშვნელობა **return** ოპერატორისა და მასში შესაბამისი ცვლადის მითითებით თვით ამ ფუნქციის სახელთან დაკავშირდეს.

განვიხილოთ მაგალითი:

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

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

ცხადია, ფუნქციის გამოძახების მომენტში **a, b, c, d** ცვლადების მნიშვნელობა განსაზღვრული უნდა იყოს.

Average ფუნქციის მნიშვნელობას კი შემდგომ გამოვიყენებთ სხვა ოპერატორებში. ეს შეიძლება პირდაპირი გზითაც მოხდეს, მაგალითად, ამგვარად:

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

დასაშვებია ფუნქციის გამოძახება გამოსახულების შემადგენელი ნაწილის სახითაც:

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

Javascript-ზე დაწერილ სცენარში ფუნქციის გამოძახება შეიძლება მოხდეს არ მხოლოდ ზემოთ ნაჩვენები მარტივი გზით, არამედ მომხმარებლის მიერ პროგრამასთან დიალოგში ფაქტობრივად ნებისმიერი ხდომილობის ინიცირებისას, მაგალითად, რაიმე ობიექტზე დაწკაპუნების მომენტში.

განვიხილოთ მაგალითები:

ა) დავწეროთ სცენარი, რომელშიც ფუნქცია დაითვლის, თუ რამდენჯერ დავაწკაპუნეთ **web**-ფურცელზე:

```
<html>
<head>
 <title>ფუნქციის გამოძახება ხდომილობით</title>
</head>
<body>
  თქვენ დოკუმენტზე დააწკაპუნეთ <input id="clicked" size="3"  value="0">-ჯერ.
 <script type="text/javascript">
   var clickCount = 0;
  function documentClick(){
     document.getElementById('clicked').value = ++clickCount;
   }
   document.onclick = documentClick;
 </script>
<body>
<html>
```

ბ) ფუნქცია დაითვლის, თუ რამდენჯერ დავაწკაპუნეთ **input** უბანზე:

```
<html>
<head>
  <title> ფუნქციის გამოძახების კიდევ ერთი ხერხი (ხდომილობით) </title>
</head>
<body>
თქვენ input-ზე დააწკაპუნეთ
<input id="clicked" size="3" onclick= documentClick();  value="0">-ჯერ.
 <script type="text/javascript">
     var clickCount = 0;
   function documentClick(){
     document.getElementById('clicked').value = ++clickCount;
```

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

ბოლო მაგალითში ტექსტური ველის (ზოგადად, ობიექტის), რომელზეც ქმედება უნდა განხორციელდეს, მისათითებლად მივმართავთ მისი იდენტიფიკატორით მონიშვნის ხერხს (ტექსტური ველისათვის გამოყენებულია id ატრიბუტი მისთვის "clicked" მინიჭებული მნიშვნელობით). აღსანიშნავია, რომ ამ წესით შესაძლებელია მოხდეს web-ფურცელზე არსებული ნებისმიერი ობიექტის სახელდება, ოღონდ თითოეულისათვის მნიშვნელობა უნდა იყოს უნიკალური, თუნდაც ისინი ერთ ტიპს განეკუთვნებოდეს.

შემდეგ, შევნიშნოთ, რომ ზოგჯერ სავალდებულო არაა, ფუნქციას ყველა არგუმენტი გადაეცეს. ასეთ შემთხვევაში არგუმენტების ჩამონათვალში ყველა არასავალდებულოს განათავსებენ სავალდებულოთა შემდგომ.

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

ვიცით, რომ ცვლადი შეიცავს მხოლოდ ერთ მნიშვნელობას *(რიცხვითს, ტექსტურს, თარიღის ტიპის და სხვ.),* მაშინ, როცა **ობიექტი წარმოადგენს რაიმე სახელთან დაკავშირებული ცვლადების კრებულს.** ამავე დროს, ცხადია, ცვლადებს საკუთარი სახელიც აქვთ.

თითოეულს ამ ცვლადთაგანს უწოდებენ **თვისებას**.

შევნიშნავთ, რომ დასაშვებია, თვისებები სხვადასხვა ტიპის იყოს.

ობიექტს, მაგალითად, შეიძლება წარმოადგენდეს *მომხმარებელი*, ხოლო მისი თვისებები იყოს:

სახელი, გვარი, მისამართი, ტელეფონის ნომერი და სხვ.

რომელიმე კონკრეტული ობიექტისათვის, მაგალითად, **Bob**-ისთვის, დამახასიათებელ ცალკეულ თვისებას ასე შეიძლება მივმართოთ:

Bob. address ან **Bob. phone**

თვისება შესაძლოა იერარქიული სტრუქტურის მქონეც იყოს, ანუ ფაქტობრივად, თვითონ წარმოადგენდეს ობიექტს. ასეთ შემთხვევასთან გვაქვს საქმე, მაგალითად, მასივის კონკრეტული ელემენტის სიგრძის ჩვენებისას:

names[7]. length

თვისებების გარდა, ობიექტები, როგორც წესი, შეიცავენ **მეთოდებსაც**.

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

ვნახოთ, თუ როგორ ხდება ამ კონცეფციის მიხედვით ცვლადის გამოცხადება-აღქმა ობიექტად, შემდეგ მისთვის (სხვადასხვა გზით) თვისებების დანიშვნა და ამ თვისებებისათვის მნიშვნელობების მიცემა. გარდა ამისა, დასაშვებია ობიექტთან რაიმე ფუნქციის (მეთოდის) მიბმაც მისთვის თვისების დანიშვნის მსგავსი ხერხით:

<u>I გზა</u>

```
var myObj = new Object;
myObj.a = 5;
myObj['b'] = 10;
myObj.c = 20;
myObj.getTotal=function(){
 alert(this.a+this.b+this.c);
};
```

<u>II გზა</u>

```
var myObj = {a:5, b:10, c:20,
 getTotal:function() { alert(this.a+this.b+this.c); }};
```

თვისებებისა და მეთოდისადმი მიმართვა მარტივი წესით ხდება. მაგალითად:

myObj.a ანდა ასე – myObj['a'];

myObj.getTotal();

განსაკუთრებული ხაზგასმის ღირსია შემდეგი გარემოება:

getTotal() ფუნქცია myObj ცვლადში (ობიექტში) განთავსებულ, მაგრამ ამ ფუნქციის გარეთ არსებულ ცვლადებს (თვისებებს) this პრეფიქსის მეშვეობით მიმართავს.

(ცხადია, ეს ხერხი ზოგადია და იგი მოქმედებს ობიექტად გამოცხადებული ნებისმიერი ცვლადისა და მისი შემდგენელი ელემენტებისათვის!).

განვიხილოთ მაგალითი.

დავუშვათ, ვქმნით ობიექტ myAnimal-ს:

```
var myAnimal = {
  name:  'felix',
  species:  'cat',
  talk:   function(){ alert('Meow!'); },
  callOver: function(){ alert(this.name+' ignores you'); },
  pet:   function(){ alert('Purr!'); }
}
```

ვთქვათ, მიზნად ვისახავთ კომპიუტერში შევიტანოთ ინფორმაცია სხვა კატის ანკეტური მონაცემების შესახებაც. ამასთან, განსხვავება არის მხოლოდ ცხოველის სახელში (მას ჰქვია არა 'felix', არამედ, დავუშვათ, 'Sam' ან 'Patty').

განმეორებითი მონაცემების ხელახლა შეტანას შესაძლებელია თავი ამგვარი ხერხით ავარიდოთ:

```
function Cat(name) {
 this.name     = name;
 this.species  = 'Cat';
 this.talk     = function() { alert('Meow!'); }
 this.callOver = function() { alert(this.name+' ignores you'); },
 this.pet      = function() { alert('Purr!'); }
}
```

```
var felix = new Cat('Felix');
var sam   = new Cat('Sam');
var patty = new Cat('Patty');
felix.pet();                 // გამოჰყავს 'Purr!'
sam.callOver();          // გამოჰყავს 'Sam ignores you'.
alert(patty.species);   // გამოჰყავს 'Cat'
```

შემდეგ, **JavaScript**-ში საქმე გვაქვს 3 ტიპის ობიექტებთან:

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

სამივე ტიპის ობიექტებს უფრო დაწვრილებით შემდგომ გავეცნობით, მაგრამ მანამდე აღვნიშნავთ, რომ სტანდარტული ობიექტ-ორიენტირებული პარადიგმით გათვალისწინებული მიდგომის ნაცვლად, რომელიც გულისხმობს Class სტრუქტურაზე ორიენტირებას, JavaScript იყენებს რამდენადმე განსხვავებულ გადაწყვეტილებას. კერძოდ, აქ თითოეულ ფუნქციას შეუძლია კლასის როლის შესრულება და ის ქმედებები, რომლებიც კლასიკურ ობიექტ-ორიენტირებულ ენებში Class სტრუქტურაზე დაყრდნობით ხორციელდება, აქ რეალიზდება *ჩადგმული ფუნქციების* მეშვეობით. ასეთ მიდგომას ეწოდა პროტოტიპინგი (Prototyping).

საინტერესოა, რომ განხილული, ერთი შეხედვით საკმაოდ უჩვეულო კონცეფცია, სპეციალისტების აზრით, ხშირ შემთხვევაში სხვა არსებულ მეთოდებთან შედარებით გაცილებით უფრო მოქნილი და პროგრამის-ტისთვის მეტი შესაძლებლობების მიმწოდებელიც აღმოჩნდა.

ხდომილობები და მათი დამუშავება

როგორც ზემოთ უკვე აღვნიშნეთ, გარდა <**script**> უბანში მოთავსებისა, **HTML** კოდში შესაძლებელია სცენარი ხდომილობათა დამმუშავებლის როლ-შიც მოგვევლინოს.

ხდომილობათა მაგალითებია:

- თაგვის მაჩვენებლის მოთავსება გრაფიკული თუ ტექსტური სახის კავშირზე;
- თაგვის ღილაკზე ხელის დაჭერა ან დაწკაპუნება**;**
- **Web**-ფურცლის ეკრანზე გამოყვანის დამთავრება და სხვ.

ამრიგად, ხდომილობის დამმუშავებელი სცენარის გამოძახების ინიციატორი შეიძლება იყოს როგორც მომხმარებელი, ისე კომპიუტერიც.

გრაფიკულ კავშირზე თაგვის მაჩვენებლის მოთავსება **JavaScript**-ის მიერ აღიქვება, როგორც **MouseOver** ხდომილობა. მსგავსი სახე აქვს სხვა ხდომილობებსაც.

დავუშვათ, სწორედ ასეთ ხდომილობას გვაქვს საქმე ეკრანზე გამოტანილი **button.gif** ნახატისათვის და მოითხოვება, რომ ხდომილობის დამმუშავებლის როლში გამოძახებული იქნეს სცენარი, ვთქვათ, **Highlight**() ფუნქციის სახით.

JavaScript-ში ამ მოთხოვნის რეალიზებას უზრუნველყოფს შემდეგი კონს-ტრუქცია:

```
<img src="button.gif"  onMouseOver="Highlight()">
```

ხაზს ვუსვამთ ზუსტად ასეთი სინტაქსის გამოყენების აუცილებლობას.

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

შენიშვნა: აქვე შევეხოთ ერთ საკითხსაც. ბროუზერების ძველი ვერსიებისთვის **JavaScript** ენა გაუგებარია და რომ არ მოხდეს მათი

მუშაობის შეფერხება (ან ეკრანზე **JavaScript** ენის კოდის მექანიკურად გამოტანა), იყენებენ **HTML** ენის კომენტარებს:

```
<!-- კოდი -->
```

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

ქვემოთ მოყვანილია მაგალითი, თუ როგორ შეიძლება ‘‘დავუმალოთ’’ კოდი ძველ ბროუზერს:

```
<script language= "JavaScript ">
<!--
 document.write(“თქვენს ბროუზერს შეუძლია JavaScript-თან მუშაობა”);
// -->
</script>
```

<u>კომენტარები</u>

JavaScript-ის კოდში კომენტარების ტრივიალური დანიშნულებით გამოსაყენებლად მიმართავენ შემდეგ კონსტრუქციებს (შევნიშნავთ, რომ ქვემოთ ნაჩვენები კომენტარები გამოიყენება **Java** ენაშიც):

```
// ეს კომენტარია
a=a+1;      // ეს კომენტარია
/* ეს კი გახლავთ
სამ სტრიქონზე
განთავსებული კომენტარი */
```

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

JavaScript-ზე დაპროგრამების ძირითადი საშუალებები

ც ვ ლ ა დ ე ბ ი

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

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

ცვლადების სწორად დაწერილი სახელების მაგალითებია:

total_number_of_fish

TotalNum Totalnum temp5

_var94

გლობალური და ლოკალური ცვლადები

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

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

მაშასადამე, **გლობალური ცვლადები უნდა გამოვაცხადოთ მთავარ სცენარში.**

ამ მიზნით, შეიძლება გამოვიყენოთ **var** საკვანძო სიტყვა ან მხოლოდ მინიჭების ოპერატორი:

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

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

გლობალური ცვლადების მთავარი სცენარის თავშივე გამოცხადება დაუწერელი წესია, თუმცა დასაშვებია, ასეთი ცვლადები ნებისმიერ ადგილას გამოცხადდეს. აღვნიშნოთ, რომ თუ ცვლადს ვიყენებთ ‘‘ოფიციალურად’’ გამოცხადებამდე *(ან ღია სახით მნიშვნელობის მინიჭების გარეშე)*, მას განესაზღვრება ნულოვანი მნიშვნელობა.

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

ქვემოთ მოყვანილ ლისტინგში **name1** და **name2** გლობალური ცვლადები სცენარის სათავეშივე განისაზღვრება (და მოცემულ შემთხვევაში მნიშვნელობებიც ეძლევა), **who** ლოკალური ცვლადი კი იქმნება **Greet**() ფუნქციაში:

```
<html>
<head>
 <title>ფუნქციების გამოყენების სხვა მაგალითი. გლობალური და
        ლოკალური ცვლადები
 </title>
 <style>  h2, p  {font-family: LitNusx} </style>
 <script language= "JavaScript">
   var name1="გიგი";
   var name2="გაგა";
   function Greet(who) {
```

```
    alert("დღეს პროგრამაშია " + who );
    var name2="გურამი";
   }
 </script>
</head>
<body>
 <h2>ვისწავლოთ ფუნქციებთან მუშაობა!</h2>
 <p>შეტყობინება გამოდის ორჯერ</p>
 <script language= "JavaScript">
  Greet(name1);
  Greet(name2);
 </script>
</body>
</html>
```

მივაქციოთ ყურადღება:

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

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

JavaScript-ში ცვლადებისთვის მნიშვნელობების მისანიჭებლად, გარდა “=” ოპერატორისა, დასაშვებია სიმბოლოთა შემდეგი კომბინაციების გამოყენებაც:

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

lines –= 1; **lines** – –;

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

++ და –– ოპერატორები შეიძლება ცვლადის სახელის წინაც დავსვათ.

აღვნიშნოთ, რომ სიტუაციიდან გამომდინარე, მათ შეიძლება რამდენადმე განსხვავებული როლიც დაეკისროთ.

დავუშვათ, **lines** ცვლადის მნიშვნელობაა 40.

alert (**lines**++) და **alert** (++**lines**) გამოსახულების შესრულება სხვადასხვა შედეგებს მოგვცემს:

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

II შემთხვევაში კი ჯერ ცვლადი 41-ის ტოლი გახდება და ამის შემდეგ სწორედ ეს შედეგი აისახება ეკრანზე.

მონაცემთა ტიპები JavaScript-ში

JavaScript-ში ცვლადს ტიპი განესაზღვრება არა გამოცხადების, არამედ მისთვის მნიშვნელობის მიცემის მომენტში, ამ მნიშვნელობის ტიპის მიხედვით.

ეს ხერხი, რომელიც დინამიკური ტიპიზაციის სახელით მოიხსენიება, ფართოდ გამოიყენება დაპროგრამებისა და სპეციფიკაციების თანამედროვე ენებში. JavaScript-ის გარდა, ესენია:

Smalltalk, Python, Objective-C, Ruby, PHP, Perl, Lisp, xBase, Erlang.

დინამიკური ტიპიზაცია საშუალებას იძლევა ერთსა და იმავე ცვლადს პროგრამის სხვადასხვა ნაწილებში სხვადასხვა ტიპი განვუსაზღვროთ (მეტიც, JavaScript-ში დასაშვებია, ცვლადი var დარეზერვირებული სიტყვით "ოფიციალურად" არც გამოვაცხადოთ).

ამ მიდგომას გააჩნია როგორც დადებითი, ისე უარყოფითი მხარეები.

დადებითია ის, რომ პროგრამა (სცენარი) უფრო კომპაქტური ხდება, ჩქარდება კომპილატორის მუშაობა, **eval()** ფუნქციით შესაძლებელი ხდება ნებისმიერი გამოსახულების მნიშვნელობის გამოთვლა, ადვილდება მონაცემთა ბაზებთან და ვებ-სამსახურებთან მუშაობა – ისინი ინფორმაციას გამოყენებით

პროგრამებს სწორედ ასეთი, დინამიკურად ტიპიზებული სახით უგზავნიან, რაც განსაკუთრებით ღირებულია ცვლადი სიგრძის მქონე მონაცემებთან მუშაობისას.

მიდგომის ნაკლი კი ის არის, რომ კომპილატორი ვეღარ ახერხებს შეცდომის აღმოჩენას პროგრამისტის მიერ ცვლადის დაწერილობაში ან მისთვის შეუძლებელი ოპერაციის მაშინვე, საწყის ეტაპზევე ფიქსირებას (სტატიკური ტიპიზაციის მიდგომისაგან განსხვავებით), რის გამოც ამგვარი შეცდომების აღმოჩენა ხდება მხოლოდ პროგრამის შესრულებისას. ნაკლი გახლავთ ისიც, რომ ობიექტ-ორიენტირებულ ენებზე დაწერილი პროგრამებში ცვლადებისათვის მნიშვნელობის განსაზღვრისას ვერ ხერხდება ამ მნიშვნე-ლობის ავტომატურად შევსების სერვისით სარგებლობაც.

ვნახოთ, თუ როგორ ხდება **JavaScript-**ში ცვლადებისათვის ტიპის განსაზღვრა:

- **რიცხვითი ტიპი.** განისაზღვრება ცვლადისათვის შესაბამისი მნიშვნელობების მინიჭებისას: **total=**31 ან **total=**3.91;
- **სტრიქონული ტიპი.** თუ იმავე ცვლადს მივანიჭებთ ტექსტურ მნიშვნელობას: **total=**"**tree**", **total=**'**tree**' (ორივე ხერხი დასაშვებია!) ან **amount=**"**12345**"**,** მაშინ ცვლადის ტიპი სტრიქონულად გარდაიქმნება. თუ სტრიქონული ტიპის ცვლადის მნიშვნელობა თავად შეიცავს ორმაგ ბრჭყალს, შეცდომის თავიდან ასაცილებლად იგი უნდა შეიცვალოს ცალმაგით, მაგალითად, **dasaxeleba**="ჟურნალი 'ფენომენი' და სხვ.". დასაშვებია შებრუნებული ვარიანტიც: **dasaxeleba** = 'ჟურნალი "ფენომენი" და სხვ. ';
- **ბულის ანუ ლოგიკური**. იღებს **true** და **false** მნიშვნელობებს. როგორც წესი, ასეთი შედეგები მიიღება ორი გამოსახულების შედარების ან ლოგიკური ოპერაციების ჩატარების შედეგად;

- **<u>ნულოვანი</u>**. ეს ტიპი ენიჭება გამოუცხადებელ ცვლადს, რომლის მნიშვნელობა განისაზღვრება **null** სიტყვით. ამ მნიშვნელობას იღებს გამოუცხადებელი **fig** ცვლადი შემდეგ ოპერატორში:

  document.write(fig);

  *(იგულისხმება, რომ **fig** ცვლადი ამ ოპერატორამდე გამოცხადებული არ იყო).*

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

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

ოპერატორი **document.write** (*"ჯამის სიდიდეა "* + **total**) ეკრანზე გამო-იყვანს შეტყობინებას:

ჯამის სიდიდეა 40

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

აქვე უნდა აღვნიშნოთ, რომ ცვლადისათვის ტიპის გარდაქმნა ყოველთვის როდი ხერხდება. მაგალითად, თუ **total** ცვლადი სტრიქონული ტიპისაა, მაშინ **average = total / 3** ოპერატორი ვერ შესრულდება. ასეთ შემთხვევებში მიმართა-ვენ ცვლადისათვის ტიპის გარდაქმნის შემდეგ ფუნქციებს:

- **parseInt() –** ტექსტური ტიპი გადაჰყავს მთელრიცხოვან ტიპში;
- **parseFloat()** – ტექსტური ტიპი გადაჰყავს მცურავწერტილიან რიცხვით ტიპში.

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

stringvar = "**30** დათვი";

numvar = parseInt (stringvar);

ოპერატორების შესრულების შედეგად **numvar** მიიღებს 30-ის ტოლ მნიშ-ვნელობას.

დავალება: *შესრულებაზე გაუშვით შემდეგი სცენარი და გაანალიზეთ ეკრანზე გამოტანილი შედეგები:*

```
<html>
 <head>
    <title> ტექსტური ტიპის ცვლადის გარდაქმნა </title>
    <style>  h2, p  {font-family: LitNusx} </style>
 </head>
 <body>
   <center>
    <h3>ტექსტური ტიპის ცვლადის გარდაქმნა მთელ და მცურავწერტილიან
        რიცხვითი ტიპის ცვლადებად
    </h3>
   </center>
 <script language="JavaScript">
   stringvar = "30 დათვი";
   numvar = parseInt(stringvar);
  document.write(numvar);
   stringvar = "30.5 დათვი";
   numvar = parseFloat(stringvar);
  document.write("<br>" + numvar);
   stringvar = "30.5 დათვი";
   numvar = parseInt(stringvar);   // !!!
  document.write("<br>" + numvar);
 </script>
```

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

ცვლადებისათვის მომხმარებლების მიერ მნიშვნელობების მინიჭება

აღნიშნული მიზნით გამოიყენება **prompt** ფუნქცია. მომხმარებლის მიერ შეტანილი ინფორმაცია, ჩვეულებრივ, მიენიჭება რაიმე ცვლადს. შესაძლებელია ინფორმაციის შემტან ოპერატორს შევთავაზოთ ამ ცვლადისათვის ყველაზე ხშირად გამოყენებული მნიშვნელობაც.

ჯერ ვაჩვენოთ ამ ფუნქციის გამოყენება შემდეგი მარტივი კოდის მაგალითზე:

```
<html>
 <head>
  <title>მომხმარებელთან დიალოგის მაგალითი</title>
  <style>
   h2, h3, p  {font-family: LitNusx}
  </style>
 </head>
 <body>
  <h2>ვისწავლოთ ფურცლის აწყობა!</h2>
  <p>მიღებულია ინფორმაცია: </p>
  <script language="JavaScript">
   saxeli=prompt("შეიტანეთ თქვენი სახელი");
   gvari= prompt("შეიტანეთ თქვენი გვარი");
  furclis_satauri= prompt("შეიტანეთ ფურცლის სათაური");
   document.write("<H2>" + furclis_satauri + "</h2>");
   document.write("<H3>" + saxeli +  " " + gvari + "</h3>");
```

```
 </script>
 <p>ფურცელი იმყოფება აწყობის სტადიაში.</p>
</body>
</html>
```

საერთოდ კი, **prompt** ფუნქციის სინტაქსი შემდეგი სახისაა (მისი დემონსტრირება ხდება კონკრეტულ მაგალითზე):

```
var person = prompt("შეიტანეთ თქვენი გვარი და სახელი", "დავით ბერიძე");
```

აი, კიდევ ამ ფუნქციით სარგებლობის ერთი მაგალითი:

```
<!DOCTYPE html>
<html>
<body>
  <p>დააწკაპუნეთ ღილაკზე!</p>
  <button onclick="myFunction()">Try it</button>
  <p id="demo"></p>
  <script>
 function myFunction() {
    var person = prompt("შეიტანეთ თქვენი სახელი და გვარი",
                            "რობინზონ კრუზო");
    if (person != null) {
      document.getElementById("demo").innerHTML =
      "დიდად პატივცემულო " + person + "! როგორ ბრძანდებით?";
    }
  }
 </script>
</body>
</html>
```

დავალება:

გაანალიზეთ document.getElementById("demo").innerHTML ოპერატორის დანიშნულება და მისი ქმედებით მიღებული შედეგი!

შენიშვნა: prompt *ფუნქციით სარგებლობისას გამორიცხული არ არის, მომხმარებელმა დიალოგის რეჟიმში პარამეტრს მიანიჭოს დაუშვებელი მნიშვნელობა, ვთქვათ, ივი გასცდეს შესატანი რიცხვისათვის წინასწარ დადგენილი დიაპაზონის ფარგლებს, ან რიცხვის ნაცვლად კლავიატურაზე შემთხვევით რაიმე ტექსტი აკრიფოს. ცხადია, სცენარში ყოველი ასეთი შემთხვევისათვის გათვალისწინებული უნდა იქნეს სათანადო რეაგირება - უნდა შემოწმდეს, აკმაყოფილებს თუ არა შეტანილი მონაცემი მისდამი წაყენებულ მოთხოვნებს, რაც ხდება პირობითი ოპერატორების მეშვეობით (იხ. ქვემოთ) და მიღებული იქნეს შესაბამისი გადაწყვეტილება.*

ვნახოთ, როგორ შეიძლება მოხდეს შესატანი ცვლადის მნიშვნელობის სისწორეზე შემოწმება:

```
<script language= "JavaScript">
    var apples = 5;
  alert('იყიდება  5 ' + apples + ' ვაშლი!');
    var eat = prompt('რამდენი მოგართვათ?', '1');
    var eaten  = parseInt(eat);
  if(isNaN(eaten)){
  alert(გაითვალისწინეთ, სულ რამდენი ვაშლია მარაგში!');
 }
 else if(eaten > apples){
  alert('ხომ გაცნობეთ, რომ გვაქვს მხოლოდ ' + apples + ' ვაშლი. თქვენ
        კი გსურთ მიირთვათ ' + eaten + ' ცალი!');
 }
 else if(eaten < 0){
```

```
  alert('უარყოფითი რაოდენობის ვაშლების შეჭმა? ცოტა გაუგებარია
        თქვენი სურვილი! ');
  }
 else {
  apples -= eaten;
  alert(დაგვრჩა ' + apples + ' ვაშლი!');
  }
</script>
```

სტრიქონული მონაცემები (ტექსტი)

უფრო დაწვრილებით შევისწავლოთ სტრიქონული მონაცემები ანუ ტექსტი.

როცა ცვლადს ვანიჭებთ ამა თუ იმ ტექსტურ მნიშვნელობას, **JavaScript** ქმნის ე. წ. **String** ობიექტს. აღვნიშნოთ, რომ ასეთი ობიექტის შექმნა უშუა-ლოდაც შეიძლება.

ვაჩვენოთ ორივე გზა ქვემოთ მოყვანილი ოპერატორების მაგალითზე:

test = "ეს ტესტია";

test = **new String** ("ეს ტესტია");

შედეგად შეიქმნება ტოლფასი სტრიქონული ობიექტები.

ობიექტში, გარდა მნიშვნელობისა, ინახება ინფორმაცია სტრიქონის სიგრძის შესახებაც **length** თვისების მეშვეობით. ვაჩვენოთ ამ თვისების გამოყენების მაგალითი:

test = "ეს ტესტია. ";

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

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

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

ToUpperCase() – ტექსტი გადაჰყავს ასომთავრულ რეგისტრში;

ToLower Case() – ტექსტი გადაჰყავს სტრიქონულ რეგისტრში.

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

გავეცნოთ სხვა მეთოდების დანიშნულებასაც:

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

ვთქვათ, გვაქვს შემდეგი ცვლადი:

alpha = "**ABCDEFGHIJKLMNOPQRSTUVWXYZ**";

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

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

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

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

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

დავალება: *დაწყვეთ სცენარი, რომლითაც შეამოწმებთ, რამდენად აითვისეთ ზემოთ მოცემული მასალა და თქვენი ვარიანტი შეადარეთ ქვემოთ მოყვანილ კოდს:*

```
<html>
 <head>
  <title> გავეცნოთ substring მეთოდს! </title>
  <style>  h2, p  {font-family: LitNusx} </style>
 </head>
 <body>
  <center>
```

```
  <h2>გავეცნოთ <font face="arial">substring</font> მეთოდს!</h2>
 </center>
 <script language="JavaScript">
     alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
   beta = alpha.substring (0, 4)   // დაგვიბრუნებს ABCD მნიშვნელობას,
   document.write("<H3>" + beta + "</h3>");
   beta = alpha.substring (10, 12)  // დაგვიბრუნებს KL-ს,
   document.write("<H3>" + beta + "</h3>");
   beta = alpha.substring (6, 7)   // დაგვიბრუნებს G-ს,
   document.write("<H3>" + beta + "</h3>");
   beta = alpha.substring (0, 26)  // დაგვიბრუნებს მთელ ალფაბეტს,
   document.write("<H3>" + beta + "</h3>");
   beta = alpha.substring (6, 6)   // გვიბრუნებს ცარიელ სტრიქონს.
   document.write("<H3>" + beta + "</h3>");
 </script>
</body>
</html>
```

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

loc=temp.indexOf("მთა", 19);

აქვე მივუთითოთ, რომ ძებნისას გაითვალისწინება სიმბოლოების რეგისტრიც.

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

<u>მასივები</u>

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

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

ელემენტების **ინდექსირება აქაც ნოლიდან იწყება,** ამიტომ ბოლო ელემენტის ინდექსი იქნება 29. ამის შემდეგ შესაძლებელია მნიშვნელობები განესაზღვროს მასივის ცალკეულ ელემენტებს, მაგალითად:

score[0] = 5;

score[25] = 9;

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

ახლა კი გავეცნოთ სტრიქონული ცვლადის ნაწილებად დაყოფის **split**() მეთოდს, რომელიც საშუალებას იძლევა სტრიქონი დავყოთ რაიმე სიმბოლოს მიხედვით და განცალკევებული ნაწილების მნიშვნელობა მივანიჭოთ მასივის ელემენტებს.

ვთქვათ, მოცემულია შემდეგი სტრიქონული ცვლადი:

test="მაკრატელი";

parts=**test**.**split**("ა"); ოპერატორების შესრულების შედეგად შეიქმნება სამელემენტიანი მასივი:

parts[0]= "მ";

parts[1]= "კრ";

parts[2]= "ტელი";

და, პირიქით, **join**() მეთოდი მასივის ელემენტებს ერთ სტრიქონში გააერ-თიანებს:

Fullname = **parts.join**("ა");

თუ გამაერთიანებელი სიმბოლოს მითითება საჭირო არ არის, მაშინ აუცი-ლებელია, პარამეტრად ვუჩვენოთ *მძიმე.*

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

names[0]= "**Fred**";

names[1]= "**George**";

names[2]= "**Alex**";

შემდეგი ოპერატორი შეგვიქმნის ახალ, ალფაბეტის მიხედვით მოწესრიგე-ბულ მასივს:

sortednames=names.sort();

დავუბრუნდეთ ზემოთ მოყვანილ *მორბენალი სტრიქონის პროგრამას.*

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

spacer=". . .   . . . ";

გვჭირდება კიდევ ერთი, რიცხვითი ტიპის ცვლადი **pos**, რომელიც განსაზღვრავს იმ სიმბოლოს ნომერს, რომლის წინაც ხდება სტრიქონის "გაჭრა". მისი საწყისი მნიშვნელობა არის ნული.

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

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

ამავე პროგრამაში გამოიყენება კიდევ ერთი, **window.setTimeout**() მეთოდი **window**-ობიექტის განახლების დროის განსაზღვრისათვის.

მეთოდის სინტაქსი შემდეგი სახისაა:

timeout_id = window.setTimeout(func|code, delay);

მის პირველ არგუმენტად მიეთითება რაიმე ფუნქცია ან შესრულებადი კოდის სტრიქონი, ხოლო მეორე არგუმენტით განისაზღვრება ფანჯრის შემცველობის განახლების დრო მილიწამებში.

ქვემოთ, აღნიშნული მეთოდის გამოძახებისას ა) სცენარში პირველ არგუმენტად გამოყენებულია კოდის შემცველი სტრიქონი, ხოლო ბ)-ში - ფუნქცია:

ა) პირველი არგუმენტია კოდის შემცველი სტრიქონი

```
<html>
 <head>
   <title>setTimeout  მეთოდი</title>
   <script language="JavaScript">
       setTimeout('alert("გავიდა  ერთი  წამი")', 1000);
       window. setTimeout('alert("გავიდა  ერთი  წამი")', 1000);
       setTimeout('alert("მ ო რ ჩ ა  კ ი ნ ო !")', 1000);
   </script>
 </head>
 <body>
  <center>
    <h2>გავეცნოთ  setTimeout მეთოდს  უფრო  დაწვრილებით!</h2>
    <h2>setTimeout  მეთოდის პირველი არგუმენტია კოდის შემცველი
        სტრიქონი, მეორე – დაყოვნების დრო მილიწამებში.
    </h2
   </center>
 </body>
```

```
</html>
```

ბ) პირველი არგუმენტია ფუნქცია

```
<html>
 <head>
   <title>setTimeout  მეთოდი</title>
   <script language="JavaScript">
       function second_passed() {
         alert("გავიდა  ერთი  წამი ");
   }
     setTimeout(second_passed, 1000);  // აქ  გამოიძახება  ფუნქცია!
     setTimeout(second_passed, 1000);  // აქ  გამოიძახება  ფუნქცია!
     setTimeout(second_passed, 1000);  // აქ  გამოიძახება  ფუნქცია!
   </script>
 </head>
 <body>
   <center>
     <h2>გავეცნოთ  setTimeout მეთოდს  უფრო  დაწვრილებით!</h2>
     <h2>setTimeout  მეთოდის  პირველი  არგუმენტია  ფუნქცია,  მეორე
            –  დაყოვნების  დრო  მილიწამებში.
     </h2
   </center>
 </body>
</html>
```

აღვნიშნოთ, რომ სცენარში არცთუ იშვიათად საჭირო ხდება **setTimeout** მეთოდის ფუნქციონირების შეწყვეტა. ამ მიზნით, გამოიყენება მეთოდი **clearTimeout(timeout_id),** რომლისთვისაც აუცილებელი ხდება არგუმენტად

setTimeout გასაუქმებელი მეთოდის იდენტიფიკატორის მითითება და ამ მოთხოვნიდან გამომდინარე, **setTimeout** მეთოდის სახელდებაც.

მოგვყავს **clearTimeout(timeout_id)** მეთოდის გამოყენების მაგალითი:

```
<html>
 <head>
  <title>გავეცნოთ clearTimeout მეთოდსაც!</title>
 </head>
 <body>
   <center><h3>გავეცნოთ clearTimeout მეთოდსაც!</h3></center>
   <input type="button" onclick="on()" value="ტაიმ-აუტის გაშვება"/>
   <input type="button" onclick="off()" value="ათვლის შეჩერება"/>
  <script>
    function go() { alert("გავიდა 2 წამი, შეგიძლიათ ხელახლა გაუშვათ ტაიმ-აუტი ან გაუქმოთ ათვლის რეჟიმი") }
       function on() { timeoutId = setTimeout(go, 2000);
       }
       function off() {
          clearTimeout(timeoutId);
       }
  </script>
 </body>
</html>
```

<u>პირობითი ოპერატორები</u>

პირობით ოპერატორებს (ისევე, როგორც ქვემოთ განხილულ ციკლებს) ჩვენ უკვე გავეცანით ადრე შესწავლილ ენებში (მეტიც, გამოვიყენეთ კიდეც ზემოთ მოყვანილ სცენარებში). ვხედავთ, რომ ამ ენათაგან თითოეულში მათი სინტაქსი გარკვეული თავისებურებებით ხასიათდება, გამონაკლისი არც

Javascript ენა გახლავთ, ამის გამო მათ ამ პარაგრაფში უფრო დაწვრილებით შევისწავლით:

if ოპერატორი

გამოყენების მაგალითები:

```
if (a == 1) window.alert ("მოიძებნა ერთი ერთეული! ");
```

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

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

== *ტოლია* (და არა =)

!= *განსხვავდება*

< <= > >=

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

|| – ლოგიკური “ან” ოპერატორი,

&& – ლოგიკური “და” ოპერატორი,

! – უარყოფის ოპერატორი.

მოვიყვანოთ გამოყენების მაგალითები:

```
if ( phone == " " || email == "  ") window.alert ("შეცდომაა!");
if ( phone == " " && email == " " ) window.alert ("შეცდომაა!");
```

JavaScript იყენებს **else** ოპერატორსაც:

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

```
}
else {
 alert (" a ცვლადის არასწორი მნიშვნელობაა " + a );
}
```

პროგრამირების თანამედროვე ენებში დასაშვებია ვისარგებლოთ ასეთი კონსტრუქციითაც:

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

იგი ტოლფასია შემდეგი ფრაგმენტის:

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

შემდეგ, **if** ოპერატორების წყებას ხშირად ცვლიან **switch** ოპერატორით. მოვიყვანოთ მაგალითები:

```
<html>
<body>
 <p id="demo"></p>
 <script>
  var day;
  switch (new Date().getDay()) {
   case 0:  day = "კვირა";
      break;
   case 1: day = "ორშაბათი";
      break;
   case 2: day = "სამშაბათი";
      break;
   case 3: day = "ოთხშაბათი";
      break;
```

```
  case 4:  day = "ხუთშაბათი";
     break;
  case 5:  day = "პარასკევი";
     break;
  case  6: day = "შაბათი";
     break;
 }
  document.getElementById("demo").innerHTML = "დღეს არის " + day;
</script>
</body>
</html>
```

switch ოპერატორი ხშირად შეიცავს **default** ოფციასაც:

```
<html>
<body>
 <p>default ოფცია</p>
<script language= "JavaScript">
  var button="0";
switch (button) {
 case "მომდევნო  გვერდი":
 window.location="next.html";
 break;
 case "წინა  გვერდი":
 window.location="prev.html";
 break;
 case "საწყისი  გვერდი":
 window.location="home.html";
 break;
 case "უკან  დაბრუნება":
```

```
  window.location="back.html";
  break;
  default: // მოცემულ სცენარში იმუშავებს ეს ვარიანტი
  window.location="06.html";
 }
 </script>
</body>
</html>
 </script>
```

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

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

button = window.prompt (’’საით გავსწიოთ?**’’);**

მოვიყვანოთ ისეთი სცენარის მაგალითი, რომელშიც გამოყენებული იქნება **window.prompt** და **switch** ოპერატორები:

```
<html>
<head>
<title>მონაცემების შემოწმება</title>
<style>
  h2, p  {font-family: LitNusx}
</style>
</head>
<body>
```

```
<h2>მივცეთ მასებს საკუთარი არჩევანის გაკეთების უფლება!</h2>
<hr>
<script language ="JavaScript">
  where =  window.prompt ("დღეს რომელ საიტს ვესტუმროთ? ");
switch (where) {
    case "posta" :
window.location = "http://www.gmail.com";
break;
    case "Microsoft" :
window.location = "http://www.microsoft.com";
break;
    default :
window.location = "http://www.gtu.ge";
}
</script>
</body>
</html>
```

<u>ციკლები</u>

JavaScript-ენა რამდენიმე სახის ციკლს იყენებს:

- ***<u>ციკლი for</u>***

მოვიყვანოთ **for** ციკლის გამოყენების მაგალითი:

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

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

- <u>*ციკლი* ***while***</u>

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

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

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

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

- <u>*ციკლი* ***do … while***</u>

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

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

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

- <u>*ციკლი* ***for … in***</u>

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

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

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

უსასრულო ციკლი. ციკლის შეწყვეტა-გაგრძელება

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

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

```
while ( true) {
 n ++;
 if ( value[n] == 1 ) break;
}
```

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

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

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

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

```
<html>
<head>
  <title>ციკლის გამოყენების მაგალითი</title>
  <style>
    h2, p  {font-family: LitNusx}
  </style>
</head>
<body>
  <h2> ციკლის გამოყენების მაგალითი </h2>
 <p> შეიტანეთ რამდენიმე სახელი. ისინი ეკრანზე დანომრილი სიის სახით
  აისახებიან.
 </p>
 <hr>
<script  language="JavaScript">
  names =  new Array();
  i = 0;
  do {
  next =  window.prompt ("შეიტანეთ შემდეგი სახელი");
  if ( next > " " ) names[i] = next;
  i = i + 1;
       }
  while ( next > " " );
```

```
document.write ("<h2>" + "შეტანილია  " +  (names.length) + "  სახელი"  +
                "</h2>");
document.write ("<ol>" );
for (i in  names ) {
document.write ("<li>" +  names[i] + "<br>" );
                }
document.write ("</ol>" );
</script>
</body>
</html>
```

შენიშვნა: განსაკუთრებული ყურადღება მივაქციოთ სცენარში HTML-ოპერატორების აწყობის წესს.

ობიექტები

ჩვენ უკვე შეგვექმნა გარკვეული წარმოდგენა ობიექტებზე, მათ თვისებებსა და მეთოდებზე. ვიცით, რომ **JavaScript**-ში საქმე გვაქვს 3 სახის ობიექტებთან. ამჯერად უფრო დაწვრილებით განვიხილოთ ეს თემა.

ობიექტების შექმნა

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

myname = new String("ესეც ასე!"**);**

new გასაღებური სიტყვა **JavaScript-**ს აცნობებს, რომ საჭიროა **String** ობიექტის ახალი ეგზემპლარის შექმნა. იგი იქნება სტრიქონული ცვლადი

myname სახელით და "ესეც ასე!" მნიშვნელობით. აღვნიშნავთ, რომ ეს მიდგომა მუშაობს სხვა ჩაშენებული ობიექტებისთვისაც, გამონაკლისს წარმოადგენს მხოლოდ **Math** ჩაშენებული ობიექტი *(იხ. ქვემოთ).*

ამგვარივე წესით ახალი ეგზემპლარები შეიძლება შევქმნათ არა მარტო **String, Date, Array** და სხვა ჩაშენებული, არამედ ე.წ. **მომხმარებელთა ობიექტებისთვისაც** (ამ სახის ობიექტების შექმნა-მოდიფიცირების საკითხსაც ქვემოთ განვიხილავთ).

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

თვისება გახლავთ ობიექტში შენახული, რაიმე მნიშვნელობის მქონე ცვლადი.

ჩვენ უკვე ცნობილია, თუ როგორ შეიძლება მივმართოთ ამა თუ იმ ობიექტის თვისებას, მაგალითად, მასივის სიგრძეს:

names.length (**names** მასივის სახელია).

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

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

მეთოდი გახლავთ ობიექტის თვისებების გადამამუშავებელი ფუნქცია, წარმოდგენილი ოპერატორის ან ოპერატორების ერთობლიობის სახით, რომელთა შესრულების შედეგი ამ ობიექტშივე შეინახება ერთ-ერთი თვისების რანგში.

მეთოდის გამოძახება ხდება ფუნქციის გამოძახების ანალოგიურად, მაგალითად, ამგვარად:

value.toUpperCase();

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

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

```
finish = Math.round(num);
```

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

JavaScript-ის სცენარების დაწერისას კოდის კომპაქტურად ჩასაწერად ხშირად იყენებენ გასაღებურ **with** სიტყვას, რომლის შემდეგ მრგვალ ფრჩხილებში მითითებული ობიექტი მომდევნო, ფიგურულ ფრჩხილებში მოთავსებული ოპერატორების ჯგუფის მიერ პრეფიქსად გამოიყენება იქ არსებული თვისებებისა და მეთოდებისათვის.

მოგვყავს მაგალითი:

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

length თვისებისა და **toUpperCase()** მეთოდისათვის ასეთი პრეფიქსის როლს ასრულებს **lastname** ობიექტი.

Math ობიექტი

Math ობიექტის თვისებები მათემატიკური კონსტანტებია, ხოლო მეთოდები – მათემატიკური ფუნქციები.

ზემოთ აღვნიშნეთ, რომ **Math** ჩაშენებული ობიექტისთვის საჭირო არ არის ახალი ეგზემპლარების შექმნა.

Math ობიექტის შესაძლებლობების ილუსტირება მოვახდინოთ შემდეგ მარტივ მაგალითზე, გამოთვალოთ სიდიდე:

x= Math.sqrt(Math.PI*12345);

აქ **Math.PI** თვისებაში შენახულია π რიცხვის მნიშვნელობა, ხოლო **Math.sqrt** მეთოდით ხდება ფრჩხილებშიდა გამოსახულებიდან კვადრატული ფესვის ამოღება.

Math ობიექტთან დაკავშირებულ რამდენიმე ხშირად გამოყენებულ ფუნქციას (მეთოდს) ჩვენ უკვე გავეცანით. გავიხსენოთ ისინი და გავეცნოთ ზოგ სხვა ფუნქციასაც:

Math.ceil() – რიცხვი მრგვალდება მეტობით;

Math.floor() – რიცხვი მრგვალდება ნაკლებობით;

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

დავალება: *ინტერნეტში მოიძიეთ უფრო დაწვრილებითი ინფორმაცია* ***Math*** *ობიექტის თვისებებისა და მეთოდების შესახებ.*

რომელიმე ათობით ნიშნამდე *(მაგალითად, მეასედამდე)* რიცხვის დასამრ-გვალებლად შეიძლება ასე მოვიქცეთ:

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

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

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

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

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

ვაჩვენოთ ამ ფუნქციების გამოყენების მაგალითი – დავწეროთ შემთხვევითი რიცხვების გენერატორის პროდუქტის "შემთხვევითობაზე" შემოწმების კოდი:

```
<html>
 <head>
  <title>შემთხვევითი რიცხვების გენერატორი</title>
  <style>
      h2, p  {font-family: LitNusx}
  </style>
 </head>
 <body>
   <h2> გენერატორის შემოწმება </h2>
   <p> რამდენად შემთხვევითია შემთხვევითი რიცხვების გენერატორით
   მიღებული რიცხვები? გამოვთვალოთ 100 000 ასეთი რიცხვის
   საშუალო მნიშვნელობა.
   <hr>
   <script language="JavaScript">
     total = 0;
    for (i = 0;  i < 100000; i++) {
       num = Math.random();
      total += num;
     if (( i/10000-Math.round(i/10000)) ==0)
        window.status = "გენერირებულია " + i + " რიცხვი" ;
      }
```

```
    average = total/100000;
    average = Math.round (average*1000) / 1000;
    document.write (<H2> "შემთხვევითი რიცხვების საშუალო
    არითმეტიკული ტოლია:  "  +  average  + "</h2>");
    window.status = "გენერირებულია " + i + " რიცხვი" ;
  </script>
 </body>
</html>
```

დავალება: *რამდენჯერმე შეცვალეთ ზემოთ მოტანილ კოდში გენერირებული რიცხვების რაოდენობის განმსაზღვრელი პარამეტრი და სცენარის გაშვების შემდეგ გამოიტანეთ დასკვნები.*

Date ობიექტი

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

birthday = new Date();

birthday = new Date("Jan 20 2002 11:00:0"**);**

birthday = new Date(5, 26, 2002**);**

birthday = new Date(5, 26, 2002, 11, 0, 0**);**

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

ა) **setDate() –** განსაზღვრავს დღის აღმნიშვნელ რიცხვს;

setMonth() – განსაზღვრავს თვეს;

setYear() – განსაზღვრავს წელიწადს;

setTime() – განსაზღვრავს პერიოდს მილიწამებში 1970 წლის 1 იანვრიდან;

setHours(), setMinutes და **setSeconds** დროის შესაბამისი სიდიდეების განსაზღვრისთვის გამოიყენება.

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

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

holiday. setYear(2002);

holiday. getYear();

Date ობიექტის მნიშვნელობის ჩაწერის ფორმატის შესაცვლელად გამოიყენება შემდეგი ორი მეთოდი:

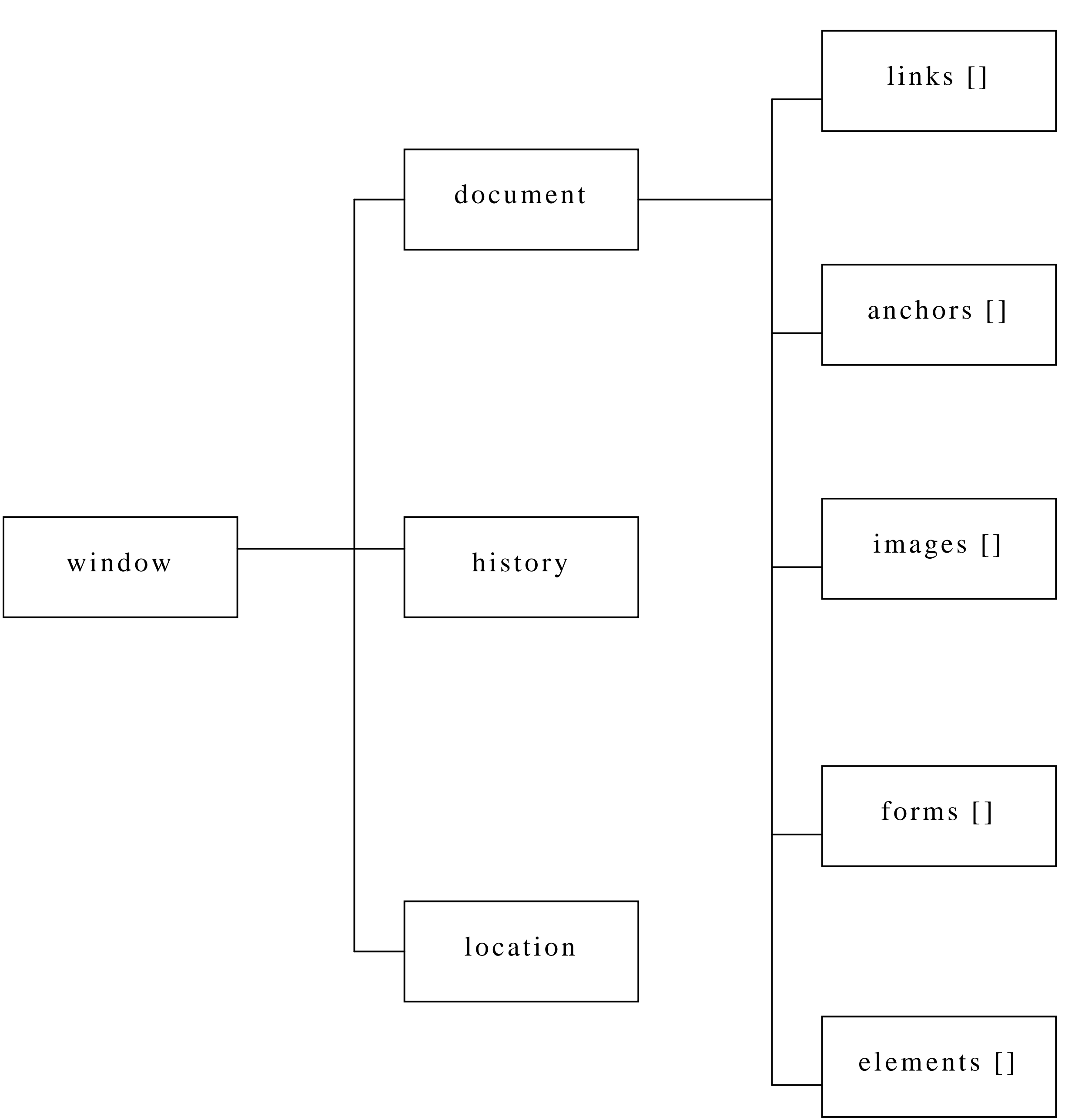
- **Date.parse()** – ეს მეთოდი **Date** ტიპის ობიექტის მნიშვნელობის ტექსტური სახით ჩაწერას ცვლის მილიწამების რიცხვით, რომლებიც ათვლა ხდება 1/1/1970 თარიღიდან;
- **Date.UTC()** – ახორციელებს უკუგარდაქმნას.

მომდევნო თავში შევისწავლით ბროუზერის ობიექტებს და მათთან მუშაობის საშუალებებს.

აღსანიშნავია, რომ ყველა ეს ობიექტი **web**-ტექნოლოგიების დარგში პოლიტიკის განმსაზღვრელი სპეციალისტების მიერ წარმოდგენილი იქნა ერთიანი, ე. წ. ბროუზერის ობიექტების მოდელის სახით, შესაბამისად, თავდაპირველად გავეცნობით ამ მოდელს, მის სტრუქტურას და შემდეგ გადავალთ მასთან მუშაობის საშუალებების შესწავლაზე.

ბროუზერის ობიექტების მოდელი

ბროუზერის ობიექტების მართვა **JavaScript**-ის დიდი ღირსება გახლავთ. ამ ობიექტებს წარმოადგენენ შემდეგი იერარქიული სტრუქტურის სახით:



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

შევნიშნავთ, რომ ზემოთ მოტანილ ნახაზზე ასახულია ბროუზერის მხოლოდ უმნიშვნელოვანესი ობიექტები.

იერარქიული სტრუქტურის სათავეში იმყოფება **window**-ობიექტი. ჩვენ უკვე ვიცნობთ მის ზოგიერთ თვისებასა და მეთოდს. ესენია:

window.status თვისება და **window.alert(), window.confirm(), window.prompt()** მეთოდები.

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

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

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

გავეცნოთ დოკუმენტის სხვა მნიშვნელოვან თვისებებსაც:

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

- **title** – შეიცავს მიმდინარე **Web-**ფურცლის სათაურს **<title> </title>** დესკრიპტორული წყვილის შიგნით;
- **referrer** – გვიჩვენებს წინა ფურცლის მისამართს, რომელზეც, როგორც წესი, ფიგურირებს მიმდინარე ფურცელზე დაყრდნობა *(ჰიპერკავშირი);*
- **lastModified** – შეიცავს **Web-**ფურცლის ბოლო კორექტირების თარიღს. ეს მონაცემი ინახება სერვერზე და ბროუზერს გადაეგზავნება **Web-**ფურცელთან ერთად *(აქვე შევნიშნოთ, რომ ზოგი სერვერი ამ სერვისს არ უზრუნველყოფს).*

შემდეგ, ერთ-ერთ პროგრამაში ჩვენ გავეცანით, თუ როგორ შეიძლება გამოვიტანოთ მონიტორზე **Web-**ფურცლის ბოლო კორექტირების თარიღი.

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

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

დაყრდნობები, ანკერები

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

ანკერი **HTML-**დოკუმენტის იმ ადგილს მონიშნავს, რომელზეც შესაძლებელია გადავიდეთ დაყრდნობის (**link**-ის) მეშვეობით.

მოვიყვანოთ ანკერ-ობიექტის მაგალითი:

<A NAME="top">

ამ ანკერზე გადასვლა კი განხორციელდება შემდეგი დაყრდნობა-ობიექტის მეშვეობით:

<A HREF="#top">

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

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

შესაბამისი თვისებით შეიძლება მივიღოთ ინფორმაცია ამ მასივების სიგრძის შესახებ:

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

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

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

link1 = links[0].href

location ობიექტი

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

window.location.href = "http:/www.gtu.ge";

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

მართალია, **location.href** თვისება იმავე **URL**-მისამართს შეიცავს, რომელიც წარმოადგენს **document.URL** თვისების მნიშვნელობას, მაგრამ, რადგანაც უკანასკნელის შეცვლა დაუშვებელია, ახალი **Web-**ფურცლის გამოსაძახებლად მიმართავენ მხოლოდ **location-**ობიექტს.

location.reload მეთოდით დოკუმენტი ბროუზერში ხელახლა ჩაიტვირთება.

history ობიექტი

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

history-ობიექტი 4 თვისებით ხასიათდება:

history.length – ერთი სეანსის მანძილზე ჩათვალიერებული ფურცლების რიცხვი;

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

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

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

რაც შეეხება მისამართების მასივში ნებისმიერი სასურველი ფურცლის მისამართის არჩევასა და გადასვლის განხორციელებას, ეს მიიღწევა **history.go(n)** და **history.go(-n)** მეთოდების გამოყენებით. მაგალითად, შესაძლებელია ეკრანზე გამოვიყვანოთ ნახატები – ერთ მათგანზე ასახული იქნება მარჯვნივ, ხოლო მეორეზე - მარცხნივ მიმართული ისრები.

დავწეროთ კოდი, რომლითაც ამ ისრებზე დაწკაპუნებისას **history.go(+1)** და **history.go(-1)** მეთოდებით მიიღწევა **Forward** და **Back** ღილაკების გამოყენების ეფექტი:

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

მივაქციოთ ყურადღება – დაყრდნობებში გამოყენებულია **javascript:URL** და **history.go** მეთოდების კომბინაცია. ამრიგად, შეიძლება **JavaScript-**ის

ოპერატორებს **<script> <script>** წყვილის გარეშეც მივმართოთ. ნახატების ასახვა კი ხდება **HTML** ენის საშუალებებით.

მომხმარებლის ობიექტები

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

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

საბოლოოდ, აი, როგორ გამოიყურება **Card**-ობიექტების კონსტრუქტორი:

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

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

აქვე აღვნიშნოთ, რომ **Card** გახლავთ ობიექტის ზოგადი სახელი. რაც შეეხება ახალ ობიექტებს *(ანუ ობიექტის ეგზემპლარებს, რომელთაც სხვა, ასევე ობიექტზე ორიენტირებულ ენებში კლასის ეგზემპლარი ეწოდება)*, თითოეულს ექნება ნებისმიერი ინდივიდუალური სახელი.

ობიექტის ეგზემპლარი ასე შეიძლება შევქმნათ:

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

დასაშვებია ობიექტის თვისებების მნიშვნელობების მოგვიანებით განსაზღვრაც:

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

ახლა კი ვაჩვენოთ, როგორ ხდება ობიექტებში მეთოდის დამატება.

(შევნიშნოთ, რომ ობიექტებში აუცილებელია თუნდაც ერთი მეთოდის არსებობა).

მეთოდი გახლავთ ფუნქცია, რომელიც გარკვეული წესით დაამუშავებს ობიექტის თვისებებს.

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

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

საინტერესოა, რომ **PrintCard** ფუნქცია არ საჭიროებს პარამეტრების ჩვენებას. იგი, როგორც ქვემოთ ვნახავთ, **Card**-ობიექტის კონსტრუქტორის მიერ გამოიძახება, როგორც მეთოდი, და სწორედ ამ ობიექტის თვისებებს იყენებს პარამეტრებად.

ფუნქციის შექმნის შემდეგ აუცილებელია ინფორმაციის მოთავსება **Card**-ობიექტის განსაზღვრებაში *(ანუ **Card**-ფუნქციაში)*:

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

ვხედავთ, რომ მეთოდიც ისევე გამოცხადდა, როგორც თვისება. ოღონდ იგი ეყრდნობა შესაბამის ფუნქციას *(მოცემულ შემთხვევაში **PrintCard**-ს).*

ავამოქმედოთ შექმნილი მეთოდი **holmes**-ობიექტის ეგზემპლარისათვის. ოპერატორს ექნება სახე:

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

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

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

ჩაშენებული ობიექტების გაწყობა

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

დავუშვათ, გვსურს **String** ჩაშენებულ ობიექტში ჩავამატოთ **heading** მეთოდი, რომელიც ამა თუ იმ სტრიქონს ეკრანზე სასურველი დონის სათაურის რანგში გამოიტანს, მაგალითად, შეგვეძლოს კოდში შემდეგი ბრძანების გამოყენება:

document. write ("ეს ტესტია!"**.heading(1));**

რიცხვი „**1**“ აქ **heading** მეთოდისთვის *(ფუნქციისთვის)* პარამეტრია. ცხადია, შეიძლებოდა მის მაგივრად აგვერჩია „**2**“, „**3**“ და ა.შ. სიდიდეები.

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

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

ადვილი შესამჩნევია, რომ **addhead** ფუნქციაში იქმნება **HTML**-ოპერატორი.

წინა შემთხვევისაგან განსხვავებით, **String** ობიექტში **heading** მეთოდის ჩამატებას ჩვენ არაპირდაპირი გზით ვახორციელებთ – ჩაშენებული ობიექტების მოდიფიცირება ხდება შემდეგი სპეციალური ოპერატორით:

String.prototype.heading = addhead;

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

```
<html>
 <head> <title> მეთოდის  ჩამატება </title>
  <style>
    p  {font-family: LitNusx}
   </style>
 </head>
 <body>
  <script language= "JavaScript ">
   function addhead (level) {
     text = this.toString ( );
     return ("<H"+level+">"+text+"</h"+level+">");
    }
 String.prototype.heading = addhead;
    document.write ("<P>ეს  ტესტია!".heading(1));
  </script>
 </body>
</html>
```

კიდევ ერთხელ გადავავლოთ თვალი შექმნილ კოდს:

დასაწყისში ვქმნით **addhead()** ფუნქციას, რომელსაც შემდეგ **prototype** საკვანძო სიტყვით გავამწესებთ **String** ობიექტის მეთოდად, **heading** სახელით. დასასრულ, ვახდენთ ამ მეთოდის შესაძლებლობების დემონსტრირებას "ეს ტესტია!" სტრიქონის მაგალითზე.

მოვიყვანოთ მწყობრში შესწავლილი მასალა – მიზნად დავისახოთ **Card**-ობიექტის რამდენიმე ეგზემპლარის შექმნა და ეკრანზე გამოყვანა:

```
<html>
 <head> <title> პირადი  ბარათები </title>
  <style>
   h2, p  {font-family: LitNusx}
  </style>
  <script language="JavaScript">
     function PrintCard()  {
     line1="სახელი:    " + this.name + "<BR>\n";
     line2="მისამართი:    " + this.address + "<BR>\n";
     line3="ტელ.(სამსახ.):  "+ this.workphone + "<BR>\n";
     line4="ტელ.(სახლ.):  " + this.homephone + "<BR>\n";
       document.write ("<P>" + line1 +  line2  +  line3  +  line4);
     }
     function  Card (name, address,  work,  home) {
this.name = name;
this.address = address;
this.workphone = work;
this.homephone = home;
     this.PrintCard=PrintCard;
     }
</script>
</head>
<body>
<h2> პირადი  ბარათები  </h2>
  <p>  აქედან  იწყება  სცენარი. </ p>
  <hr>
<script language= "JavaScript ">
//  ობიექტების  შექმნა
```

```
 gigi = new Card ("გიგი გურული", "კოსტავას 75", "37-37-37", "39-11-12");
 lia= new Card ("ლია ბერიძე", "რუსთაველის 25", "93-23-34", "36-45-55");
  tea= new Card ("თეა გიგაური", "წერეთლის 47", "34-34-34", "95-23-89");
// ობიექტების ასახვა
   gigi.PrintCard();
   lia.PrintCard();
   tea.PrintCard();
</script>
 <p> სცენარის დასასრული</ p>
 </body>
</html>
```

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

აღსანიშნავია, რომ ზემოგანხილული წესით შეიძლება ობიექტისთვის შევქ-მნათ შვილობილი ობიექტები. ჯერ ვქმნით კონსტრუქტორის ფუნქციას ახალი ობიექტისთვის და შემდეგ მშობლად გათვალისწინებულ ობიექტში ვამატებთ ახალ თვისებას. მაგალითად, თუ შევქმენით **Nicknames** ობიექტი თანამშრო-მელთა ფსევდონიმების დასაფიქსირებლად და გვსურს იგი **Card**-ობიექტთან მიმართებაში შვილობილად ვაქციოთ, ამ **Card**-ობიექტს კონსტრუქტორში უნდა დავუმატოთ შემდეგი ოპერატორი:

this.nick = new Nicknames ( );

კვლავ ხდომილობების შესახებ. თაგვთან დაკავშირებული სხვა ხდომილობების დამმუშავებლნი

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

ხდომილობათა დამმუშავებელი ეწოდება სცენარს, რომელსაც შეუძლია ხდომილობების დაფიქსირება და განსაზღვრული მოქმედებების შესრულება.

ხდომილობის მაგალითად შეიძლება მოვიყვანოთ თაგვის მაჩვენებლის მოთავსება **Web**-ფურცლის რომელიმე ობიექტზე – **MouseOver.** მოითხოვება, რომ ამ ხდომილობის დამმუშავებლის სახელი იყოს **onMouseOver.**

ანალოგიური წესით იქმნება სხვა ხდომილობების და მათი დამმუშავებლების სახელწოდებანიც.

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

```
<a  href="http://posta.ge/"
  onMouseOver="window.alert ('ვესტუმროთ ფოსტას?');">
  დააწკაპუნეთ აქ
</a>
```

როცა ხდომილობის დამუშავება რამდენიმე ოპერაციის შესრულებას მოითხოვს, დასაშვებია, ხდომილობების დამმუშავებელში ისინი ერთმანეთისაგან წერტილ-მძიმეებით განვაცალკეოთ. მაგრამ, საერთოდ, უმჯობესია ვისარგებლოთ ფუნქციით, რომელიც განლაგდება <**head**> უბანში და, ვთქვათ, ასეთი გზით გამოიძახება:

```
<a href="# bottom"  onMouseOver="Dolt ();">
```

თაგვის მიმთითებელი აქ მოათავსეთ!

</**a**>

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

მოვიყვანოთ მაგალითი:

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

თაგვთან დაკავშირებულ ხდომილობებს ხშირად იყენებენ გამოსახულებებთან დაკავშირებული ქმედებების მაუწყებელი მეტი ვიზუალური ეფექტისათვის, მაგალითად:

```
<img src="button_off.gif"
  onmouseover="this.src='button_over.gif';"
  onmousedown="this.src='button_down.gif';"
  onmouseout ="this.src='button_off.gif';"
  onmouseup  ="this.src='button_over.gif';"
>
```

კოდში **button_off.gif** გამოსახულება რეაგირებს 4 ხდომილებაზე:

onmouseover, onmousedown, onmouseout და **onmouseup**-ზე.

ვხედავთ, რომ მდგომარეობის ცვლილებების დასაფიქსირებლად დამატებით გათვალისწინებულია კიდევ სამი გამოსახულება.

დავალება: *ინტერნეტში მოიძიეთ უფრო დაწვრილებითი ინფორმაცია ხდომილობების და მათი დამმუშავებლების შესახებ.*

event ობიექტი

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

ჩამოვთვალოთ **event**-ობიექტის თვისებები:

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

თაგვთან დაკავშირებული ხდომილობების დამმუშავებლნი

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

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

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

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

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

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

<**A href="http://gtu.ge" onClick="return** (**window.confirm**
('დარწმუნებული ხართ?')); "> *დააწკაპუნეთ აქ*
</**a**>

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

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

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

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

onLoad ხდომილობის დამმუშავებელი

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

```
<BODY onLoad="alert ('ჩატვირთვა დამთავრდა'); ">
```

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

დაყრდნობის აღწერის დამატება

ამა თუ იმ დაყრდნობაზე თაგვის მიმთითებლის მოთავსებისას ხშირად მი-მართავენ იმ მოსალოდნელი ქმედების აღწერას, რომელიც შედეგად მოჰყვება დაწკაპუნებას. მაგალითად, შეიძლება სტატუსის სტრიქონში იმ საიტის შესახებ გამოვიყვანოთ მეტ-ნაკლებად ვრცელი ინფორმაცია, რომელზეც ვაპირებთ გა-დასვლას. ამ შემთხვევაში ვიყენებთ **onMouseOver** და **onMouseOut** ხდომილობების დამმუშავებლებს. უკანასკნელი გვაწვდის სტატუსის სტრიქონში

ძველი ინფორმაციის აღდგენის საშუალებას იმ მომენტისათვის, როცა თაგვის მიმთითებელი სხვა ზონაში გადაინაცვლებს. როგორც წესი, ეს გახლავთ მიმდინარე **web**-ფურცლის **URL** მისამართი.

სტატუსის სტრიქონში დაყრდნობის აღმწერი ტექსტის გამოყვანა-ამოგდებისათვის უმჯობესია ფუნქციების გამოყენება.

ქვემომოყვანილ მაგალითში ამ მიზნების განხორციელებას ემსახურება **describe(text)** და **clearstatus()** ფუნქციები:

```
<html>
<head>
  <title>ჰიპერკავშირების აღწერა</title>
  <script language = "JavaScript">
  function describe (text) {
    window.status = text;
    return true;
  }
  function clearstatus () {
    window.status = " ";
  }
  </script>
  <style>
   h2, p, ul {font-family: LitNusx}
  </style>
 </head>
 <body>
   <h2> ჰიპერკავშირების აღწერა </h2>
   <p> თაგვის მაჩვენებელი მოათავსეთ ჰიპერკავშირებზე მათი აღწერების
   გამოსაყვანად!
```

```
</p>
<ul>
<li><a href= "order.html"
      onMouserOver = "describe('შეუკვეთეთ  საქონელი');  return true;"
      onMouseOut = "clearstatus()";>
  საქონლის  შეკვეთა
       </a>
</li>
<li><a href = "email.html"
      onMouserOver = "describe('წერილის  გაგზავნა');  return true;"
      onMouseOut = "clearstatus()";>
      ელექტრონული  ფოსტა
      </a>
 </li>
 <li><a href="adm.html"
       onMouserOver="describe('წინადადებები'); return true;"
       onMouseOut = "clearstatus()";>
       ადმინისტრაციისადმი  მიმართვა
      </a>
  </li>
</ul>
</body>
</html>
```

describe ფუნქციაში **return true** ოპერატორი საშუალებას იძლევა, სტა-ტუსის სტრიქონში არ მოხდეს **URL** მისამართის ნაცვლად **text** შეტყობინების გამოყვანა.

დასასრულ, განვიხილოთ რამდენიმე ამოცანა, რომლებშიც გამოყენებულია **Javascript** ენის ზემოთ აღწერილი საშუალებები. აქვე შევნიშნოთ, რომ ესა თუ ის ამოცანა, როგორც წესი, შესაძლებელია რამდენიმე გზით გადაწყდეს.

ცხადია, საერთოდ, უნდა ვეცადოთ, რომ შევარჩიოთ კოდის, სცენარის ყველაზე ეფექტიანი ვარიანტი. მაგალითად, ქვემოთ მოყვანილი "თაგვების დოღისათვის" შესაძლებელი იყო კოდი სხვადასხვაგვარად დაწერილიყო. დინამიზმის უზრუნველყოფისათვის ჩვენ უპირატესობა მივეცით ნახატი-ობიექტების შრეებზე განლაგების და ამ შრეების ურთიერთის მიმართ დაძვრის ხერხს.

კოდის დაწერისას ამავე დროს მეტად სასურველია, იგი იყოს ადვილად მოდიფიცირებადიც უკვე გადაწყვეტილი ამოცანისადმი მომავალში წაყენებული ახალი მოთხოვნების რეალიზების გასაადვილებლად. აღვნიშნავთ, რომ ხშირ შემთხვევაში ამ მოთხოვნის დაკმაყოფილების უზრუნველყოფა ხდება ჩვენ მიერ ზემოთ განხილული ობიექტ-ორიენტირებულ პარადიგმაზე დაყრდნობით. ქვემოთ სწორედ ეს მიდგომაა გამოყენებული ცოდნის გამოკითხვა-შეფასების ამოცანისათვის სცენარის დაწერისას.

ამოცანა - "თაგვების დოღი"

სანამ "თაგვების დოღისათვის" დაწერილ კოდს გავეცნობოდეთ, გავიხსენოთ თუ როგორ ხდება **div** ელემენტის შრედ **(layer)** ქცევა:

```
<html>
<head>
  <title> შრეები (ფენები)</title>
  <meta http-equiv="Content-Type" content="text/html;
        charset=windows-1252">
  <style>
     p,h1,div {font-family: AcadNusx}
  </style>
</head>
<body>
```

```
  <center><h1> შრეები (ფენები)</h1></center>
  <div id="layer1"  style="position:static; background-
   color:lightgreen">
   <p>ეს არის სტატიკური ფენა</p>
  </div>
 <p>ეს არის სტატიკურ ფენაში კიდევ ერთი აბზაცი</p>
  <div id="layer2" STYLE="position:absolute;
   left:400; top:200; width:100; height:200; background-
    color:yellow"> ეს მომდევნო ფენაა
  </div>
  <div id="layer3" style="position:absolute; left:50;
   top:200; visibility:hidden"> ეს  ფენა არ აისახება
  </div>
  <div id="layer4" STYLE="position:absolute; left:450;
   top:300; width:200; height:100; background-color:red">
    ეს კი ბოლო ფენაა
  </div>
 </body>
</html>
```

გადავდივართ ამოცანაზე – "თაგვების დოღი":

სტარტზე დგას 3 თაგვი. შემთხვევითი რიცხვების გენერატორი დროის ფიქსირებული მონაკვეთის გავლის მომენტებში მათთვის განსაზღვრავს წინ გადაადგილების ბიჯს გარკვეულ დიპაზონში. ფინიშზე პირველად მისული თაგვი გამარჯვებულია. პროგრამა თითოეული თაგვისათვის აითვლის სერიაში გამარჯვებათა რაოდენობას.

დავალება: *მოიფიქრეთ ამ სცენარის გაუმჯობესების ვარიანტები, მაგალითად, შესაძლებელია, დოღს მიეცეს ტოტალიზატორის სახე. მასში მონაწილეებს ეძლევათ ერთნაირი რაოდენობის თანხა (ქულები), რომელიც შეეძლებათ ნაწილ-ნაწილ გახარჯონ შეჯიბრებათა სერიის თითოეულ ეტაპზე ჰანდიკაპის ბიჯის საყიდლად.*

```
<html>
<head>
   <title>თაგვების დოღი</title>
   <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
   <style>p, h1, a {font-family: AcadNusx} </style>
<script>
    var ekr_sigane = 795;  // დოღის დისტანცია პიქსელებში
    var _pos1 = -95;  // საწყისი პოზიცია I   თაგვისათვის
    var _pos2 = -95;  // საწყისი პოზიცია II  თაგვისათვის
    var _pos3 = -95;  // საწყისი პოზიცია III თაგვისათვის
    var pos1;   // მიმდინარე პოზიცია  I  თაგვისათვის
    var pos2;   // მიმდინარე პოზიცია  II  თაგვისათვის
    var pos3;   // მიმდინარე პოზიცია  III თაგვისათვის
     var speed1;  // ბიჯი  I  თაგვისათვის
     var speed2;  // ბიჯი  II  თაგვისათვის
     var speed3;  // ბიჯი  III   თაგვისათვის
    var k1 = 0;  // მოგებათა რიცხვი  I  თაგვისათვის
    var k2 = 0;  // მოგებათა რიცხვი  II  თაგვისათვის
    var k3 = 0;  // მოგებათა რიცხვი  III  თაგვისათვის
  pos1 = _pos1;
  pos2 = _pos2;
  pos3 = _pos3;
function next() {
  speed1=Math.floor(Math.random()*7);
  speed2=Math.floor(Math.random()*7);
  speed3=Math.floor(Math.random()*7);
     pos1 +=  speed1;
     pos2 +=  speed2;
     pos3 +=  speed3;
     if ( pos1>ekr_sigane ) { k1++;  alert ( "გაიმარჯვა წითელმა! ამ სერიაში მისი 
მოგებათა რიცხვია  "  +  k1 + "." ); };
     if ( pos2>ekr_sigane ) { k2++;  alert ( "გაიმარჯვა ლურჯამ! ამ სერიაში მისი 
მოგებათა რიცხვია  " +  k2 + "." ); };
     if ( pos3>ekr_sigane ) { k3++;  alert ( "გაიმარჯვა მწვანემ! ამ სერიაში მისი 
მოგებათა რიცხვია  " +  k3 + "." ); };
```

```
 if (pos1>ekr_sigane  || pos2>ekr_sigane  ||  pos3>ekr_sigane)
   { pos1 = _pos1; pos2 = _pos2; pos3 =_pos3; }             // სტარტზე დაბრუნება
 if (document.layers) {
           document.layers[0].left=pos1;
           document.layers[1].left=pos2;
           document.layers[2].left=pos3;
                      }
 else {
    mouse1.style.left=pos1;
    mouse2.style.left=pos2;
    mouse3.style.left=pos3;
       }
             window.setTimeout("next();",10);
}
</script>
</head>
<body  onLoad="next();">
  <h1 align=center>თაგვების დოღი</h1>
  <p><p><p><p><p>
<div  id="mouse1"   style="position:absolute;  left:0;top:150;  wight:100;  hight:100;
visibility:show; background-color: red"   > <img src="mouse5.bmp"> </div>
<div  id="mouse2"   style  ="position:absolute;  left:0;top:250;  wight:100;  hight:100;
visibility:show; background-color: blue"  > <img src="mouse5.bmp"> </div>
<div  id="mouse3"   style="position:absolute;  left:0;top:350;  wight:100;  hight:100;
visibility:show; background-color: green"> <img src="mouse5.bmp"> </div>
  <p><p><p><p><p>
<center>
   <a href="#" onClick="window.close()"> <img src="mouse5.bmp"  width=55>
   </a>
</center>
<center>
  <a href="#" onClick="window.close()">  შევწყვიტოთ!  </a>
</center>
</body>
</html>
```

ამოცანა - ‘‘ფორმებთან მუშაობა’’

გავეცნოთ **Javascript-**ის საშუალებების გამოყენებით ჩვენთვის უკვე ნაცნობ **HTML-**ფორმებთან მუშაობის ზოგიერთ ხერხს. თავდაპირველად დავწეროთ ერთი ასეთი ფორმისათვის კოდი:

```
<form name="tutform" onsubmit="return noform();" class="codesnip" style="background-color:#FFF;z-index:10;">
<center>
 <table width="50%">
  <tr>
   <td>სახელი:</td>
   <td><input name="firstname"></td>
   <td>აირჩიეთ საყვარელი ფერი:</td>
   <td rowspan="3" valign="top">
    <input type="radio" name="color" value="blue">ლურჯი<br />
    <input type="radio" name="color" value="red">წითელი<br />
    <input type="radio" name="color" value="green">მწვანე
   </td>
   <td rowspan="3" valign="top">
    <input type="radio" name="color" value="yellow">ყვითელი<br />
    <input type="radio" name="color" value="black">შავი<br />
    <input type="radio" name="color" value="other">სხვა
   </td>
  </tr>
  <tr>
   <td></td>
   <td><input name="lastname"></td>
  </tr>
  <tr>
   <td>E_mail:</td>
   <td><input name="email"></td>
  </tr>
  <tr>
```

```
    <td colspan="2"><input type="submit" value="ფორმის გაგზავნა">
    <input type="reset" value="ფორმის გასუფთავება"></td>
    <td colspan="3" align="right">
      <button id="lockbutton" onclick="swapLock();  return false;">
       ფორმის ადგილზე დაფიქსირება
      </button></td>
   </tr>
  </table>
 </center>
</form>
```

ეკრანზე ფორმა ასე სახით აისახება:

სახელი:
E_mail:
ფორმის გაგზავნა ფორმის გასუფთავება
აირჩიეთ საყვარელი ფერი:
ლურჯი ყვითელი
წითელი შავი
მწვანე სხვა
ფორმის ადგილზე დაფიქსირება

Javascript-იდან ფორმისადმი მიმართვისათვის ვქმნით ამ ენის ობიექტს, რომელშიც მიეთითება ფორმის სახელი, მაგალითად:

document.forms.tutform

ფორმის ელემენტისადმი (შეტანის ველი, ასარჩევი ელემენტი, ალამი და სხვ.) მიმართვა კი, მაგალითად, ასე შეიძლება მოხდეს:

document.forms.tutform.elements.firstname.value

შემდეგ, როდესაც შესაბამის ღილაკზე დაწკაპუნებით ვაპირებთ ფორმა სერვერზე გადავაგზავნოთ, ბროუზერი ამოწმებს, შეიცავს თუ არა **form** ელემენტის საწყისი ტეგი **onsubmit** ატრიბუტს და თუ ეს ასეა, იგი ასრულებს ამ ატრიბუტთან მიბმულ კოდს, რომელიც, მაგალითად, ამგვარი სახის შეიძლება იყოს (ხდება გასაგზავნი ინფორმაციის სისწორის შემოწმება ადგილზევე, სერვერზე გადაგზავნამდე):

```
<FORM ONSUBMIT="return validateForm();">
   <!-- შემმოწმებელი კოდი -->
</FORM>
```

ვაჩვენოთ, თუ როგორ შეიძლება შემოწმდეს, შეავსო თუ არა მომხმარებელმა სახელის შესატანად განკუთვნილი ველი:

```
function validateForm(){
  var form_object = document.forms.tutform;
    if(form_object.elements.firstname.value == ''){
    alert("თქვენ უნდა შეიტანოთ საკუთარი სახელი!");
    return false;
  } else if(form_object.elements.lastname.value == ''){
    alert(თქვენ უნდა შეიტანოთ საკუთარი გვარი!');
    return false;
  }
  return true;
}
```

თუ ფუნქცია return ოპერატორით გვიბრუნებს true მნიშვნელობას, ფორმა გადაეგზავნება ადრესატს, ხოლო false მნიშვნელობის დაბრუნებისას ეს ოპერაცია აღარ ტარდება და აუცილებელი ხდება alert-ის მეშვეობით მომხმარებელს შევატყობინოთ, თუ რა არის შეფერხების მიზეზი.

შემდეგ, ვხედავთ, რომ რადიო-ღილაკებიდან ერთ-ერთის მონიშვნით აირჩევა საყვარელი ფერი. როგორც წესი, ამ ღილაკებს ერთიდაიმავე სახელს არქმევენ, რაც აადვილებს ციკლის მეშვეობით მათ ჩათვალიერება-შემოწმებას, მაგალითად, როცა გვსურს შევიტყოთ, მოვნიშნეთ თუ არა ერთ-ერთი რადიო-ღილაკთაგანი:

```
<input type="radio" name="color" value="blue">ლურჯი
<input type="radio" name="color" value="red">წითელი
```

```
<input type="radio" name="color" value="green">მწვანე

function validateForm(){
  var radios = document.forms.tutform.elements.color;
  for(var i=0; i<radios.length; i++){
    if(radios[i].checked) return true;
  }
  alert("მოითხოვება აირჩიოთ ერთ-ერთი ფერი! ");
  return false;
}
```

ამოცანა - "ცოდნის გამოკითხვა-შეფასება"

(HTML საგანში მიღებული ცოდნის რამდენიმე კითხვაზე გაცემული პასუხების სისწორის შემოწმების მაგალითზე)

დავალება: *მოითხოვება, დაწეროთ პროგრამა, რომელიც:*

1. *რაიმე საგნობრივი სფეროდან მომხმარებელს დაუსვამს გარკვეული რაოდენობის შეკითხვას და მისცემს მას საშუალებას, შემოთავაზებული პასუხებიდან აირჩიოს, მისი აზრით, სწორი. მცდარი არჩევანისას რესპოდენტს უნდა ეცნობოს ამის თაობაზე მაშინვე, ხოლო სეანსის დამთავრების შემდეგ კი გამოუვიდეს შეტყობინება, რამდენ შეკითხვაზე გასცა სწორი პასუხი.*
2. *ასევე მოითხოვება, ყოველი სეანსისათვის შეიცვალოს როგორც კითხვების, ისე თითოეულ კითხვაზე სავარაუდო პასუხების თანმიმდევრობა შემთხვევითი რიცხვების გენერატორის დახმარებით.*

დავალების შესრულების შემდეგ გაეცანით ქვემოთ მოყვანილ კოდს, შეადარეთ იგი თქვენს ნამუშევარს და სიი სახით დააფიქსირეთ ორივე პროგრამის ღირსება-ნაკლოვანებები:

```
// ------  < begin -  მოსამზადებელი სამუშაოები ------------------------------
var i=1;  //  მასივის ელემენტების ინდექსირებისათვის დამხმარე ცვლადი
var n=4;       //  მასივის სიგრძე
var q=0;       //  სწორი პასუხების რაოდენობა
var pasuxi=" ";      // "მუშა" პასუხი
document.all.item("Ans_ID").innerHTML =
შეკითხვა #" + i + " (" + n + "-დან)"+"<FONT face=Arial><br>SOS !!!</font>";

  // ------  < begin - ობიექტის გამოცხადება თვისებებით:
  შეკითხვა, ტყუილ-მართალი პასუხები,სწორი პასუხის ნომერი -----------
function Card ( shek,pas1,pas2,pas3,pas4,sworipas )
{
 this.shek=shek;
 this.pas1=pas1;
 this.pas2=pas2;
 this.pas3=pas3;
 this.pas4=pas4;
 this.sworipas=sworipas;
}
 quest=new Array();
 quest[1]=new Card(" *რა არის ჰიპერტექსტი?",
 "<FONT face=Arial>text</font><FONT><font face=AcadNusx>-faili</font>",
"ჩვეულებრივ ტექსტზე უფრო მეტი ინფორმაციის შემცველი ტექსტი ",
"ჩვეულებრივ ტექსტზე უფრო მეტი ფუნქციების მქონე ტექსტი",
"ჩვეულებრივ ტექსტზე უფრო მეტად ინფორმატიულ-ფუნქციონალური
ტექსტი", 4);
 quest[2]=new Card(" რამდენ ტეგს შეიცავს ელემენტი?",
"ორს",
```

```
"ერთს",
"ერთს ან ორს",
"ორს ან სამს",   3);
 quest[3]=new Card(" * რა ფუნქცია აქვს <font face=Arial>HR</font>-ელემენტს?",
"ტექსტი გადაჰყავს მომდევნო სტრიქონზე",
"იწყებს ახალ აბზაცს",
"ავლებს ხაზს",
"გადავყავართ სხვა საიტზე",  3);
 quest[4]=new Card("* რამდენ როლს ასრულებს კოდში <font face=Arial>A</font>-ელემენტი?",
"ერთადერთს",
"ორს",
"სამს",
"ხან ერთს, ხან ორს",  4);
 // ------   end > - ობიექტის გამოცხადება თვისებებით: შეკითხვა, ტყუილ-მართალი პასუხები,სწორი პასუხის ნომერი -----------
quest[0]=new Card(" ",   " ", " ", " ", " ",   0);       // “მუშა” მასივის ელემენტი
// ------ end > -  მოსამზადებელი სამუშაოები ------------------------------------

// ------ < begin - კითხვების დასტის აჭრა ----------------------------------------
var gr = 0; var gr1 = 0;
  for (m1=0; m1<10; m1++)
{  gr = Math.round(Math.random()*3)+1;
   gr1= Math.round(Math.random()*3)+1;
   if (gr != gr1)
  { quest[0] = quest[gr1];   quest[gr1] = quest[gr];  quest[gr] = quest[0]; }
}
```

```
// ------ end > - კითხვების დასტის აჭრა -----------------------------------

// ------ < begin - კითხვის და პასუხების გამოტანა -------------------------
        document.all.item("Sek").innerHTML =quest[1].shek;
        document.all.item("pas1").innerHTML ='1.–'+quest[1].pas1;
        document.all.item("pas2").innerHTML ='2.–'+quest[1].pas2;
        document.all.item("pas3").innerHTML='3._'+quest[1].pas3;
        document.all.item("pas4").innerHTML='4._'+quest[1].pas4;
// ------  end > - კითხვის და პასუხების გამოტანა

        function m(a)
        {
        // ------ <  begin - პასუხის სისწორის შემოწმება --------------------
        if (a==quest[i].sworipas)
          q++;
        else
          alert („არა, სწორი პასუხია“ "+quest[i].sworipas);
        //  ------ end > - პასუხის სისწორის შემოწმება ----------------------

        // ------  < begin - მომდევნო შეკითხვის მომზადება ---------------
        i++;
        if (i < 5)
        { gr = Math.round(Math.random()*3)+1;
        //  ------ < begin - პასუხების დასტის აჭრა –1 --------
        if (gr>1)
        {    pasuxi=quest[i].pas1;
         quest[i].pas1=quest[i].pas2;
        quest[i].pas2=quest[i].pas3;
```

```
quest[i].pas3=quest[i].pas4;
quest[i].pas4=pasuxi;
--quest[i].sworipas;
if (quest[i].sworipas == 0)
quest[i].sworipas=n;
}
// ------ end > - პასუხების დასტის აჭრა-1 --------

// ------ < begin - პასუხების დასტის აჭრა-2 --------
if (gr<2)
{   pasuxi=quest[i].pas4;
quest[i].pas4=quest[i].pas3;
quest[i].pas3=quest[i].pas2;
quest[i].pas2=quest[i].pas1;
quest[i].pas1=pasuxi;
++quest[i].sworipas;
if (quest[i].sworipas == n+1)
 quest[i].sworipas=1;
}
// ------ end - პასუხების დასტის აჭრა-2 --------
 document.all.item("Ans_ID").innerHTML =
 "შეკითხვა #" + i + "  (" + n + -დან)";
// ------  end >  - მომდევნო შეკითხვის მომზადება ----------------

// ----- < begin - მომდევნო კითხვის და პასუხების გამოტანა -
document.all.item("Sek").innerHTML =quest[i].shek;
document.all.item("pas1").innerHTML ='1._'+quest[i].pas1;
document.all.item("pas2").innerHTML ='2._'+quest[i].pas2;
```

```
document.all.item("pas3").innerHTML ='3._'+quest[i].pas3;
document.all.item("pas4").innerHTML ='4._'+quest[i].pas4;
// ----- end > - მომდევნო კითხვის და პასუხების გამოტანა -
  }
else
{
i=1;

//  ------ < begin – შემაჯამებელი ინფორმაციის გამოტანა ---
 abc="<CENTER><font face=Arial>END </font>
 <font face=AcadNusx>სეანსის დასასრული </font></center>";
 document.write(abc);
 alert ( n + "  შეკითხვიდან თქვენ გაეცით "+q+" სწორი პასუხი");
// ------  end > - შემაჯამებელი ინფორმაციის გამოტანა ---------

 parent.close();
 document.close();
 document.open();
 document.write("<H3><center><font face=AcadNusx>
 მენიუში აირჩიეთ სასურველი პუნქტი! </font></center></h3>");
 document.close();
}
```

დავალება: *მოიფიქრეთ ამ პროგრამის გაუმჯობესებული ვარიანტები, მაგ.:*

ა) გაზარდეთ შეკითხვების რიცხვი, სცენარში დაუმატეთ მათგან მხოლოდ ნაწილის შემთხვევითობის წესით ამორჩევის შესაძლებლობა.

ბ) მოახდინეთ კითხვების რანჟირება სირთულის მიხედვით. დაყავით კითხვები რამდენიმე კატეგორიად ისე, რომ არჩევანი უნდა შეიცავდეს ჯამურად ერთნაირი სირთულის კითხვებს.

JSON ფორმატი, toJSON მეთოდი

JSON (https://ru.wikipedia.org/wiki/JSON) ტექსტური ფორმატია მონაცემების გასაცვლელად. იგი დაფუძნებულია JavaScript ენაზე და როგორც წესი, სწორედ ამ ენის სცენარებში გამოიყენება. მისი ავტორია დუგლას კროკფორდი.

JSON ფორმატით შესაძლებელი ხდება ობიექტები სტრიქონის სახით წარმოვადგინოთ, რაც მეტად აადვილებს სერვერებიდან კლიენტებისათვის ამ ობიექტებში არსებული მონაცემების გადაგზავნას. მონაცემების როლში კი გვევლინება:

- JavaScript-ობიექტები `{ ... };`
- მასივები `[ ... ];`
- სტრიქონები (ორმაგ ბრჭყალებში მოქცეული), რიცხვები, ლოგიკური `true`/`false` და `null` მნიშვნელობები.

რაც შეეხება JavaScript ენაში არსებულ, JSON-თან მუშაობისათვის განკუთვნილ უმთავრეს მეთოდებს, ესენია:

`JSON.parse` - JSON ფორმატის სტრიქონიდან აღადგენს ობიექტს;

`JSON.stringify` - JavaScript-დან მონაცემების ქსელში გადასაცემად ობიექტს გარდაქმნის JSON ფორმატის სტრიქონად.

`JSON.parse` მეთოდი

მოვიყვანოთ JSON ფორმატის სტრიქონის JavaScript ენის ობიექტად, მასივად და მნიშვნელობად გარდაქმნის მაგალითები:

მაგალითი:

```
var numbers = "[0, 1, 2, 3]";
```

```
numbers = JSON.parse(numbers);

alert( numbers[1] ); // 1
```

მაგალითი:

```
var user = '{ "name": "John", "age": 35, "isAdmin": false, "friends": [0,1,2,3] }';

user = JSON.parse(user);

alert( user.friends[1] ); // 1
```

აღსანიშნავია, რომ ობიექტები და მასივები შესაძლებელია იყოს უფრო რთული სტრუქტურისაც – თავის მხრივ შეიცავდნენ სხვა ობიექტებს და მასივებს, მხოლოდ, ცხადია, ეს უკანასკნელებიც უნდა შეესაბამებოდნენ `JSON` ფორმატს.

ადვილი შესამჩნევია, რომ `JSON` ფორმატში შესრულებული ობიექტები ძალიან ჰგავანან სტანდარტულ `Javascript` ობიექტებს. მათ შორის განსხვავება ისაა, რომ სტრიქონების მიმართ წაყენებული მოთხოვნები გამკაცრებულია – ისინი მ ხ ო ლ ო დ ორმაგ ბრჭყალებში უნდა იყვნენ მოქცეულნი (და ეს ეხება არა მარტო სტრიქონული ტიპის ელემენტის მნიშვნელობას, არამედ, საერთოდ, ობიექტის შემადგენელი ნებისმიერი დასახელებასაც).

დავალება 1:

იპოვეთ ქვემოთ მოყვანილი კოდის ფრაგმენტში შეცდომები (პასუხი იხ. თავის ბოლოში):

```
{ name: "John",
  "surname": 'Johnson',
  "age": 35
```

```
  "isAdmin": false
}
```

შენიშვნა: *JSON ფორმატში კომენტარების გამოყენება დაუშვებელია, რადგანაც იგი მხოლოდ მონაცემების გადაცემისათვის გამოიყენება. თუმცა აქვე ისიც უნდა აღინიშნოს, რომ არასტანდარტულ, გაფართოებული შესაძლებლობების მქონე JSON5 ფორმატში დასაშვებია როგორც Javascript ენაში არსებული კომენტარების გამოყენება, ისე – ელემენტების სახელების ჩაწერა ბრჭყალებში მათი მოქცევის გარეშეც.*

`JSON.parse` მეთოდი მონაცემების ანალიზისათვის უფრო რთულ ალგორითმებსაც იყენებს. დავუშვათ, სერვერიდან გადმოგვეცა `event` ხდომილობის შესატყვისი ობიექტი თავისი მონაცემებით, რომელიც უნდა აღდგეს, ანუ გარდაიქმნას JavaScript-ობიექტად:

```
var str = '{"title":"კონფერენცია","date":"2015-11-30T12:00:00.000Z"}';
```

თავდაპირველად მივმართოთ ნაცად ხერხს:

```
var str = '{"title":"კონფერენცია","date":"2015-11-30T12:00:00.000Z"}';

var event = JSON.parse(str);

alert( event.date.getDate() ); // ამ სტრიქონში მოხდება
                                // შეცდომა!
```

შეცდომას განაპირობებს ის გარემოება, რომ `event` ხდომილობის (აქ ობიექტის) `.date` თვისების მნიშვნელობას წარმოადგენს თარიღი!

ამრიგად, საჭიროა მეთოდს ეცნობოს, რომ ამ შემთხვევაში სტრიქონი უნდა გარდაიქმნას თარიღის ტიპის მონაცემად - “მაცნეს” როლი ეკისრება **JSON.parse(str,reviver)** მეთოდის მეორე, არასავალდებულო **reviver** პარამეტრს, რომელიც, თავის მხრივ, წარმოადგენს ასეთ ფუნქციას:

function(key, value).

JSON.parse სტრიქონიდან ობიექტის წაკითხვისას **reviver** პარამეტრის არსებობის შემთხვევაში ამ ობიექტის JavaScript-ობიექტად გარდასაქმნელად ანალიზდება წყვილები: გასაღები-მნიშვნელობა და გვიბრუნდება ან გარდაქმნილი მნიშვნელობა ან undefined (როცა თვისების გამოტოვებაა საჭირო).

განსახილველი შემთხვევისათვის ვქმნით წესს, რომლის მეშვე-ობითაც date გასაღებური სიტყვა აღიქვება მხოლოდ თარიღად (UTC ფორმატში):

```
var str = '{"title":"კონფერენცია","date":"2015-11-30T12:00:00.000Z"}';

var event = JSON.parse(str, function(key, value) {
  if (key == 'date') return new Date(value);
  return value;
});
alert( event.date.getDate() ); // და ყველაფერი წესრიგშია!
```

აღვნიშნოთ, რომ ეს წესი ვრცელდება ჩადგმულ ობიექტებზეც:

```
var schedule = '{ \
  "events": [ \
```

```
    {"title":"კონფერენცია","date":"2015-11-30T12:00:00.000Z"}, \
    {"title":"დაბადების თარიღი","date":"2015-04-18T12:00:00.000Z"} \
  ]\
}';

schedule = JSON.parse(schedule, function(key, value) {
  if (key == 'date') return new Date(value);
  return value;
});

alert( schedule.events[1].date.getDate() );
```

JSON.stringify მეთოდი, სერიალიზაცია

`JSON.stringify(value, replacer, space)` მეთოდი მნიშვნელობას გარდაქმნის JSON ფორმატის სტრიქონად. ამ ქმედებას უწოდებენ სერიალიზაციას.

მაგალითი:

```
var event = {
  title: "კონფერენცია",
  date: "დღეს"
};

var str = JSON.stringify(event);
alert( str ); // {"title":"კონფერენცია","date":"დღეს"}
// უკუგარდაქმნა
event = JSON.parse(str);
```

ობიექტის სერიალიზაციისას გამოიძახება მისი **toJSON** მეთოდი. თუ ასეთი მეთოდი ობიექტის არსენალში არ არსებობს, უჩვენებენ მის თვისებებს (ფუნქციების ჩვენება დაუშვებელია):

```
var room = {
  number: 23,
  occupy: function() {
    alert( this.number );
  }
};
event = {
  title: "კონფერენცია",
  date: new Date(Date.UTC(2015, 0, 1)),
  room: room
};

 alert( JSON.stringify(event) );
```

```
/*
  {
    "title":"კონფერენცია",
    "date":"2015-01-01T00:00:00.000Z",  // (1)
    "room": {"number":23}               // (2)
  }
*/
```

მივაქციოთ ყურადღება:

1. თარიღი გარდაიქმნა სტრიქონად. საქმე ისაა, რომ თარიღი ჩაშენებული (სტანდარტული) ობიექტია. მასში

ფიგურირებს toJSON მეთოდი, რომლის შესრულებითაც მიიღება სტრიქონი UTC ზონაში.

2. რაც შეეხება room ობიექტს, მას მას არ გააჩნია მეთოდი. შედეგად მისი სერიალიზაცია ხდება თვისებების ჩამოთვლით, მაგრამ თუ ამ ობიექტში ჩავამატებდით მეთოდს, ცხადია, შედეგიც სხვაგვარი იქნებოდა:

```
var room = {
  number: 23,
  toJSON: function() {
      return this.number;
    }
};
alert( JSON.stringify(room) ); // 23
```

თვისებების გამორიცხვა

JSON ფორმატში გადავიყვანოთ ობიექტი, რომელიც DOM სტრუქტურაშია იდენტიფიცირებული:

```
var user = {
  name: "დავითი",
  age: 25,
  window: window
};
alert( JSON.stringify(user) ); // შეცდომაა!
// TypeError: Converting circular structure to JSON
(текст из Chrome)
```

შეცდომა მოხდა შემდეგი მიზეზის გამო:

`window` არის რთული სტრუქტურის მქონე გლობალური ობიექტი, რომლის გარდაქმნაც ასე მარტივად ვერ ხერხდება (და ეს არც არის საჭირო). გამოსავალი მოიძებნება თვისებების იმ მასივის ჩვენებით, რომლებიც ექვემდებარება სერიალიზაციას. ასე, მაგალითად:

```
var user = {
  name: "დავითი",
  age: 25,
  window: window
};
alert( JSON.stringify(user, ["name", "age"]) );
// {"name":"დავითი","age":25}
```

უფრო რთული სიტუაციების შემთხვევაში მეორე პარამეტრის როლში შეიძლება გამოვიყენოთ `function(key, value)` ფუნქცია. იგი გვიბრუნებს სერიალიზებულ `value` ან `undefined` მნიშვნელობას (ამ უკანასკნელს მაშინ, როდესაც საჭირო არ არის თვისება ფიგურირებდეს შედეგში):

```
var user = {
  name: "დავითი",
  age: 25,
  window: window
};
var str = JSON.stringify(user, function(key, value) {
  if (key == 'window') return undefined;
  return value;
});
```

```
alert(str); // {"name":"დავითი","age":25}
```

ზემოთ მოყვანილ მაგალითში function(key, value) ფუნქცია ახდენს window თვისების იგნორირებას, ხოლო სხვა თვისებები-სათვის კი ყოველგვარი დამუშავების გარეშე უბრალოდ უბრუნებს სტანდარტულ ალგორითმს მათ მნიშვნელობას.

აქვე უნდა აღინიშნოს, რომ თუ ობიექტი რთული სახისაა – იგი თვითონ შეიცავს ჩადგმულ ობიექტებს, მასივებს და ა.შ., ფუნქციით მოხდება მათი დამუშავება რეკურსიული წესით.

გავხადოთ დაფორმატება უფრო მიმზიდველი სახის!

ზემოთ განხილულ მეთოდში შესაძლებელია გამოყენებული იქნეს მესამე space პარამეტრიც, მონიტორზე ინფორმაციის უფრო სრულყოფილად ასახვის მიზნით:

```
JSON.stringify(value, replacer, space)
```

თუ space პარამეტრი რიცხვის სახით იქნება მოცემული, მაშინ JSON ფორმატში ელემენტის ობიექტში ჩადგმის დონე მიეთითება შესაბამისი რაოდენობის ხარვეზების (შუალედების) მეშვეობით და ეკრანზე მისი გამოტანისას უკეთ გვექმნება წარმოდგენა ობიექტის იერარქიულ სტრუქტურაზე. ხოლო თუ space პარამეტრი სტრიქონული მნიშვნელობისაა, – შესაბამის ადგილზე ჩაიდგმება სწორედ ეს სტრიქონი.

მოგვყავს მაგალითი:

```
var user = {
  name: "დავითი",
  age: 25,
  roles: {
```

```
    isAdmin: false,
    isEditor: true
  }
};
var str = JSON.stringify(user, "", 4);
alert( str );
/* Результат -- красиво сериализованный объект:
{
    "name": "დავითი",
    "age": 25,
    "roles": {
        "isAdmin": false,
        "isEditor": true
    }
}
*/
```

დავალება 1-ზე პასუხი:

ქვემოთ მოყვანილი კოდის ფრაგმენტი შეიცავს შემდეგ შეცდომებს:

```
{  name: "John", // გასაღებური სიტყვა name მოთავსებული
                 //  უნდა იყოს ბრჭყალებში (ორმაგში)!
  "surname": 'Johnson', // დაიშვება მხოლოდ ორმაგი
ბრჭყალები!
  "age": 35
  "isAdmin": false
}
```

დავალება 2 - ობიექტის გარდაქმნა

ქვემოთ მოყვანილი `leader` *ობიექტი გადაიყვანეთ JSON ფორმატში:*

```
var leader = {
  name: "John",
  age: 35
};
```

შემდეგ კი JSON ფორმატში მიღებული სტრიქონი ისევ წარმოადგინეთ თავდაპირველი ობიექტის სახით.

დავალება 3 – ურთიერთზე დაყრდნობილი ობიექტების გარდაქმნა

ქვემოთ მოყვანილი `team` *ობიექტი გადაიყვანეთ JSON ფორმატში:*

```
var leader = {
  name: "Johnson"
};

var soldier = {
  name: "John"
};

// ეს ობიექტები ერთმანეთს ეყრდნობა!
leader.soldier = soldier;
soldier.leader = leader;

var team = [leader, soldier];
```

AJAX ტექნოლოგია

AJAX ტექნოლოგია წარმოადგენს შემდგომ ნაბიჯს კლიენტის მხარეზე მოქმედი დინამიკური WEB-გამოყენებების შექმნის სფეროში, რომელთა დამუშავება, დაყენება და შესრულება ხორციელდება უფრო სწრაფად და უკეთესად მის გარეშე შექმნილებთან შედარებით. ამასთან, უმჯობესდება მომხმარებელთან ინტერაქტიურობის ხარისხიც (ტრადიციული HTML-ფორმებისაგან განსხვავებით).

AJAX ტექნოლოგია ემყარება JavaScript ენისა და HTTP მოთხოვნების სიმბიოზს, მხოლოდ JavaScript-ის სცენარების შესრულება ხდება ასინქრონულად, ფონურ რეჟიმში, ამასთან, სერვერიდან ინფორმაციის გადმოსაგზავნად მეტწილად იყენებენ XML ენის შესაძლებლობებს (საერთოდ კი, დასაშვებია ამ მიზნით ნებისმიერი სხვა ფორმატის, მათ შორის ტექსტის, გამოყენებაც).

ზემოთ აღნიშნული თავისებურებებიდან გამომდინარე, AJAX ტექნოლოგიის არსის განსამარტავად ამგვარ "ფორმულასაც" კი იყენებენ:

AJAX = ასინქრონული JavaScript + XML

შემდეგ, AJAX ტექნოლოგიის თავისებურებას წარმოადგენს ის გარემოებაც, რომ აქ JavaScript ენას მნიშვნელოვანი როლი ეკისრება ბროუზერსა და სერვერს შორის ურთიერთობის უზრუნველყოფაში. AJAX –ის მიერ JavaScript გამოიყენება მონაცემების გასაცვლელად ბროუზერსა და Web-სერვერს შორის. სწორედ ამ პროცესში იკვეთება სცენარების ასინქრონულად შესრულების დადებითი მხარე:

HTTP მოთხოვნებზე დაყრდნობით, ფონურ რეჟიმში ბროუზერსა და Web-სერვერს შორის შედარებით მცირე მოცულობის ინფორმაციის გაცვლა მიმდინარეობს WEB-ფურცლის გადატვირთვის გარეშე, რაც მნიშვნელოვნად ამაღლებს პროცესის სისწრაფეს.

ხაზგასასმელია ის გარემოებაც, რომ აღნიშნული ტექნოლოგიის გამოყენებაზე მხოლოდ ბროუზერია "პასუხისმგებელი", ანუ WEB-სერვერის მხარეს არ მოითხოვება რაიმე დამატებითი ქმედებების

ჩატარება. ამასთან, ბროუზერების (მხედველობაში გვაქვს თანამედროვე ბროუზერები) პროგრამული უზრუნველყოფისათვისაც საჭირო არ არის ახალი კომპონენტების შემუშავება, რადგანაც AJAX იყენებს მხოლოდ ღია სტანდარტებს შემდეგი ტექნოლოგიებისა:

JavaScript,

XML,

HTML,

CSS.

ზემოთ ჩამოთვლილი სტანდარტები უცხო ელემენტებს არ წარმოადგენს ყველა ძირითადი თანამედროვე ბროუზერისათვის და კომპიუტერული პლატფორმისათვის.

Web-ტექნოლოგიებმა უკვე დაამტკიცეს, რომ მათი გამოყენება შესაძლებელია არა მხოლოდ ინტერნეტისათვის, არამედ – ტრადიციული, სამაგიდო კომპიუტერებისათვის განკუთვნილი პროგრამული სისტემების შემუშავების დროსაც. თუმცა ასეთ შემთხვევაში, როგორც წესი, მოითხოვება "სუფთა" Web-ტექნოლოგიები გამდიდრდეს დამატებითი შესაძლებლობებითაც. სწორედ ამ მიზნის მიღწევას ემსახურება AJAX-ტექნოლოგია.

AJAX-ის გამოყენების მაგალითი

ქვემოთ ვაჩვენოთ AJAX-ის დახმარებით შექმნილი სცენარის მაგალითი (მოითხოვება HTML-ფორმის ტექსტურ ველში გვარის აკრეფვის დაწყებისთანავე გამოყენებამ შემოგვთავაზოს შესაძლო ვარიანტები):

```
<form>
  გვარი:
  <input type="text" id="txt1"
   onkeyup="showHint(this.value)">
</form>
   <p>ეს გვარი? <span id="txtHint"></span></p>
```

ჩანს, რომ "txt1"-თი იდენტიფიცირებულ ტექსტურ ველში პირველი ასოს შეტანისთანავე, ასევე ყოველი მომდევნოს აკრეფვისას (კლავიშის აშვების მომენტში) ხდება showHint() ფუნქციის გამოძახება, რომელიც მიმართავს სერვერს და იქიდან მიღებული მონაცემებით (ამ შემთხვევაში გვარით) განსაზღვრავს span ელემენტის მნიშვნელობას.

აღვნიშნოთ, რომ JavaScript-ენაზე დაწერილი **showHint()** ფუნქცია უნდა განთავსდეს WEB-ფურცლის HTML-კოდის HEAD უბანში:

```
function showHint(str)
{
if (str.length==0)
{
document.getElementById("txtHint").innerHTML=""
return
}
xmlHttp=GetXmlHttpObject()
if (xmlHttp==null)
{
alert ("მოცემულ ბროუზერს არ შეუძლია HTTP-მოთხოვნების
      შესრულება")
return
}
var url="gethint.asp"
url=url+"?q="+str
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
}
```

განვიხილოთ, თუ რა ოპერაციებს ახორციელებს ზემოთ მოყვანილი showHint() ფუნქცია:

- უწინარეს ყოვლისა, ხდება შემდეგი პირობის შემოწმება (str.length > 0) – შეტანილია თუ არა ტექსტურ ველში რაიმე ინფორმაცია;
- თუ ეს პირობა შესრულებულია, ფუნქცია ცდილობს, შექმნას xmlHttp სახელწოდების მქონე, სერვერთან xml ენაზე ურთიერთობისათვის განკუთვნილი ობიექტის ეგზემპლარი (თუკი ბროუზერი ამის საშუალებას იძლევა);
- თუ პასუხი უარყოფითია, დისპლეიზე ვკითხულობთ შესაბამის შეტყობინებას, დადებითი პასუხის შემთხვევაში კი განისაზღვრება სერვერზე გასაგზავნი ფაილის მისამართი და სახელი (URL);
- აღნიშნულ URL-ს ემატება q პარამეტრი, რომლის მნიშვნელობაც განისაზღვრება ტექსტური ველის ახალი შემცველობით;
- იმ მიზნით, რომ შეტყობინების მიღებისას სერვერმა გადაგზავნილი ფაილის ძებნა არ განახორციელოს კეშ-მეხსიერებაში, URL-ს დაემატება შემთხვევითი რიცხვების გენერატორის მიერ გენერირებული რაიმე რიცხვი;
- xmlHttp ობიექტის ეგზემპლარის რაიმე ცვლილებისას ხდება stateChanged ფუნქციის გამოძახება (იხ. ქვემოთ);
- xmlHttp მზადდება სერვერზე გადაგზავნისათვის (მას გაღებისას გადაეცემა ზემოთ ფორმირებული URL პარამეტრი);
- სერვერზე იგზავნება შესაბამისი შეტყობინება.

აქვე შევნიშნოთ, რომ თუ Enter-ზე ხელის დაჭერისას შეტანის ველი ცარიელი აღმოჩნდება, ფუნქცია მასში განათავსებს txtHint ტექსტს.

როგორც ზემოთ აღვნიშნეთ, stateChanged ფუნქციის შესრულებაზე გაშვება ხდება xmlHttp ობიექტის ეგზემპლარის მდგომარეობის ყოველი ცვლილებისას. ამასთან, იმ შემთხვევაში, როცა ეს მდგომარეობა

მოინიშნება კოდით "4" ან "complete"-თი, txtHint ველში ჩაიწერება სერვერიდან გადმოგზავნილი პასუხი.

განვიხილოთ stateChanged() ფუნქციის დანიშნულება. მისი კოდია:

```
function stateChanged()
{
 if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
 {
 document.getElementById("txtHint").innerHTML=xmlHttp.responseText
 }
}
```

ხაზი უნდა გაესვას შემდეგ გარემოებას:

AJAX-გამოყენებები სრულდება მხოლოდ ისეთ ბროუზერებში, რომლებშიც ჩადებულია XML-ტექნოლოგიის სრული მხარდაჭერის შესაძლებლობა.

ზემოთ მოყვანილ მაგალითში ვხედავთ, რომ ხდება GetXmlHttpObject ფუნქციის გამოძახება და მოცემული ობიექტის შესატყვისი ეგზემპლარის შექმნა. ფუნქციას აქვს შემდეგი სახე:

```
function GetXmlHttpObject(handler)
{
 var objXMLHttp=null
 if (window.XMLHttpRequest)
  {
   objXMLHttp=new XMLHttpRequest()
  }
else if (window.ActiveXObject)
  {
   objXMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
  }
 return objXMLHttp
}
```

მოვიყვანოთ მწყობრში განხილული მასალა და ქვემოთ წარმოვადგინოთ ამ მაგალითის მთლიანი კოდი.

საწყისი HTML ფაილი შეიცავს მარტივ HTML ფორმას და დაყრდნობას js ფაილზე:

```
<html>
<head>
<script src="clienthint.js"></script>
</head>
<body>
<form>
სახელი:
<input type="text" id="txt1"
onkeyup="showHint(this.value)">
</form>
<p>ამ სახელს ირჩევთ: <span id="txtHint"></span></p>
</body>
</html>
```

ზედა ფაილიდან გამოძახებული clienthint.js ფაილის კოდია:

```
var xmlHttp
function showHint(str)
{
if (str.length==0)
{
document.getElementById("txtHint").innerHTML=""
return
}
xmlHttp=GetXmlHttpObject()
if (xmlHttp==null)
{
alert ("HTTP მოთხოვნების შესრულებას ეს ბროუზერი ვერ
ახერხებს")
```

```
return
}
var url="gethint.asp"
url=url+"?q="+str
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
}
function stateChanged()
{
if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
{
document.getElementById("txtHint").innerHTML=xmlHttp.responseText
}
}
function GetXmlHttpObject()
{
var objXMLHttp=null
if (window.XMLHttpRequest)
{
objXMLHttp=new XMLHttpRequest()
}
else if (window.ActiveXObject)
{
objXMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
}
return objXMLHttp
}
```

AJAX-ის სერვერული ფურცლები ASP-ისა და PHP-ისთვის

ამთავითვე შევნიშნოთ, AJAX-ის გამოყენებისათვის სპეციალური სერვერის არსებობა საჭირო არ გახლავთ. AJAX-ტექნოლოგიაზე ორიენტირებული WEB-ფურცლების დამუშავება შეუძლია ინტერნეტში მომუშავე ნებისმიერ სერვერს, მაგალითად, IIS-ს. ზემო მაგალითში ამ სერვერის მეშვეობით ხდება JavaScript-სცენარის მიერ გამოძახებული მარტივი სახის მქონე gethint.asp დასახელების ფაილის კოდის დამუშავება და კლიენტის მოთხოვნის შესრულება – მასივიდან ამორჩეული შესაბამისი სახელების მისთვის დაბრუნება. gethint.asp ფაილისათვის კოდი დაწერილია VBScript-ზე და აქვს სახე:

```
<%
dim a(30)
'შევავსოთ სახელების მასივი:
a(1)="Anna"
a(2)="Brittany"
a(3)="Cinderella"
a(4)="Diana"
a(5)="Eva"
a(6)="Fiona"
a(7)="Gunda"
a(8)="Hege"
a(9)="Inga"
a(10)="Johanna"
a(11)="Kitty"
a(12)="Linda"
a(13)="Nina"
a(14)="Ophelia"
a(15)="Petunia"
a(16)="Amanda"
a(17)="Raquel"
```

```
a(18)="Cindy"
a(19)="Doris"
a(20)="Eve"
a(21)="Evita"
a(22)="Sunniva"
a(23)="Tove"
a(24)="Unni"
a(25)="Violet"
a(26)="Liza"
a(27)="Elizabeth"
a(28)="Ellen"
a(29)="Wenche"
a(30)="Vicky"
'URL-იდან გამოვაცალკევოთ q პარამეტრი:
q=ucase(request.querystring("q"))
'თუ მასივის სიგრძე q>0,  ჩავათვალიეროთ რეკომენდაციები:
if len(q)>0 then
  hint=""
  for i=1 to 30
    if q=ucase(mid(a(i),1,len(q))) then
      if hint="" then
        hint=a(i)
      else
        hint=hint & " , " & a(i)
      end if
    end if
  next
end if
'გამოვიტანოთ ჩათვალიერების შედეგი:
if hint="" then
  response.write("ძიება უშედეგოდ დამთავრდა")
```

```
else
 response.write(hint)
end if
%>
```

ამჯერად კი განვიხილოთ იმავე ამოცანის გადაწყვეტის მაგალითი PHP-ტექნოლოგიაზე დაყრდნობით.

პირველ ყოვლისა, აღვნიშნოთ, რომ აუცილებელია "clienthint.js" ფაილში URL ცვლადის მნიშვნელობა "gethint.asp" შევცვალოთ "gethint.php"-ით.

კოდს ექნება შემდეგი სახე:

```
<?php
// შევავსოთ სახელების მასივი:
$a[]="Anna";
$a[]="Brittany";
$a[]="Cinderella";
$a[]="Diana";
$a[]="Eva";
$a[]="Fiona";
$a[]="Gunda";
$a[]="Hege";
$a[]="Inga";
$a[]="Johanna";
$a[]="Kitty";
$a[]="Linda";
$a[]="Nina";
$a[]="Ophelia";
$a[]="Petunia";
$a[]="Amanda";
$a[]="Raquel";
$a[]="Cindy";
$a[]="Doris";
```

```
$a[]="Eve";
$a[]="Evita";
$a[]="Sunniva";
$a[]="Tove";
$a[]="Unni";
$a[]="Violet";
$a[]="Liza";
$a[]="Elizabeth";
$a[]="Ellen";
$a[]="Wenche";
$a[]="Vicky";
// URL-იდან გამოვაცალკევოთ q პარამეტრი:
$q=$_GET["q"];
// თუ მასივის სიგრძე q>0, ჩავათვალიეროთ რეკომენდაციები:
if (strlen($q) > 0)
{
 $hint="";
 for($i=0; $i<count($a); $i++)
 {
 if (strtolower($q)==strtolower(substr($a[$i],0,strlen($q))))
  {
  if ($hint=="")
   {
   $hint=$a[$i];
   }
  else
   {
   $hint=$hint." , ".$a[$i];
   }
  }
 }
```

```
}
if ($hint == "")
{
$response="ძიება უშედეგოდ დამთავრდა";
}
else
{
$response=$hint;  // შესაბამისი მნიშვნელობების მინიჭება
}
// პასუხის გამოტანა
echo $response;
?>
```

AJAX-ის მონაცემთა ბაზასთან დაკავშირების მაგალითი

დავუშვათ მოითხოვება, WEB-ფურცელზე გამოტანილი იქნეს მონაცემთა ბაზაში განთავსებული კლიენტების სია, ხოლო რომელიმე მათგანის არჩევისას – ამ კლიენტის შესახებ ბაზაში არსებული ინფორმაცია.

ამოცანის შესატყვისი, ქვემოთ მოყვანილი HTML-კოდი შეიცავს მარტივ HTML-ფორმას და JavaScript-ის სცენარზე დაყრდნობას:

```
<html>
<head>
 <script src="selectcustomer.js"></script>
</head>
<body>
 <form>
 აირჩიეთ შემკვეთი:
 <select name="customers" onchange="showCustomer(this.value)">
  <option value="ALFKI">Alfreds Futterkiste
  <option value="NORTS ">North/South
```

```
   <option value="WOLZA">Wolski Zajazd
  </select>
 </form>
 <p>
  <div id="txtHint"><b>აქ გამოვა ინფორმაცია შემკვეთის შესახებ.</b></div>
 </p>
</body>
</html>
```

ჩანს, რომ “customers” ჩამოშლად სიაში მონაცემების (შემკვეთის) ყოველი არჩევისას, ანუ "onchange" ხდომილებისას, გამოიძახება "showCustomer()" ფუნქცია, რომლის შესრულების შედეგადაც "txtHint" სახელის მქონე div ელემენტი შეივსება WEB-სერვერიდან გადმოგზავნილი ინფორმაციით.

რაც შეეხება JavaScript-ის სცენარს, იგი ინახება selectcustomer.js ფაილში და ასეთი შემცველობისაა:

```
var xmlHttp
function showCustomer(str)
{
 xmlHttp=GetXmlHttpObject()
 if (xmlHttp==null)
 {
  alert ("HTTP მოთხოვნების შესრულებას ეს ბროუზერი ვერ ახერხებს")
  return
 }
var url="getcustomer.asp"
url=url+"?q="+str
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
```

```
}
function stateChanged()
{
 if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
 {
 document.getElementById("txtHint").innerHTML=xmlHttp.responseText
 }
}
function GetXmlHttpObject()
{
 var objXMLHttp=null
 if (window.XMLHttpRequest)
 {
 objXMLHttp=new XMLHttpRequest()
 }
 else if (window.ActiveXObject)
 {
 objXMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
 }
return objXMLHttp
}
```

ზემოთ მოყვანილი კოდიდან ჩანს, რომ JavaScript-ის სცენარი, თავის მხრივ, იძახებს სერვერზე განთავსებულ getcustomer.asp ფაილს. ეს asp-ფაილი მუშავდება ინტერნეტის საინფორმაციო სერვერის (IIS) მიერ. ქვემოთ მოყვანილ მაგალითში იგი შეიცავს VBScript ენაზე დაწერილ კოდს. აქვე უნდა შევნიშნოთ, რომ შესაძლებელია ამ კოდის გადაწერა PHP და ნებისმიერ სხვა სერვერულ ენაზეც.

მოცემულ კოდში SQL ბრძანებების მეშვეობით ხდება მონაცემთა ბაზიდან ინფორმაციის ამოკრეფვა და მათი გამოყვანა ეკრანზე HTML ცხრილის სახით:

```
sql="SELECT * FROM CUSTOMERS WHERE CUSTOMERID="
sql=sql & request.querystring("q")

set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open(Server.Mappath("/db/northwind.mdb"))
set rs = Server.CreateObject("ADODB.recordset")
rs.Open sql, conn

response.write("<table>")
do until rs.EOF
for each x in rs.Fields
response.write("<tr><td><b>" & x.name & "</b></td>")
response.write("<td>" & x.value & "</td></tr>")
next
rs.MoveNext
loop

response.write("</table>")
```

AJAX-ის მეშვეობით XML ფაილიდან მონაცემების ამორჩევის მაგალითი

ქვემოთ მოყვანილ მაგალითში WEB-ფურცელზე გამოდის მონაცემთა ბაზაში განთავსებული მუსიკოს-შემსრულებელთა სია, ამასთან, რომელიმე მათგანის არჩევისას – ეკრანზე გამოიტანება მისი ნაწარმოებების შემცველი კომპაქტ-დისკოს შესახებ ინფორმაციაც:

```
<html>
<head>
  <script src="selectcd.js"></script>
</head>
<body>
<form>
აირჩიეთ კომპაქტ-დისკო:
<select name="cds" onchange="showCD(this.value)">
  <option value="Bob Dylan">Bob Dylan</option>
  <option value="Bonnie Tyler">Bonnie Tyler</option>
  <option value="Dolly Parton">Dolly Parton</option>
</select>
</form>
<p>
<div id="txtHint"><b>აქ გამოდის ინფორმაცია კომპაქტ-დისკოს
შესახებ.</b></div>
</p>
</body>
</html>
```

"cds" სიაში ელემენტის არჩევისას ანუ "onchange" ხდომილობისას გამოიძახება "showCD" ფუნქცია. ამ ფუნქციის კოდი მოთავსებულია "selectcd.js" ფაილში:

```
var xmlHttp

function showCD(str)
{
xmlHttp=GetXmlHttpObject()
if (xmlHttp==null)
{
alert ("HTTP მოთხოვნების შესრულებას ბროუზერი ვერ ახერხებს")
return
}
var url="getcd.asp"
url=url+"?q="+str
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
}

function stateChanged()
{
if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
{
document.getElementById("txtHint").innerHTML=xmlHttp.responseText
}
}

function GetXmlHttpObject()
{
var objXMLHttp=null
if (window.XMLHttpRequest)
{
objXMLHttp=new XMLHttpRequest()
}
```

```
else if (window.ActiveXObject)
{
objXMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
}
return objXMLHttp
}
```

ზემოთ მოყვანილი კოდიდან ჩანს, რომ JavaScript-ის სცენარი, თავის მხრივ, იძახებს სერვერზე განთავსებულ getcd.asp ფაილს. ეს asp-ფაილი მუშავდება ინტერნეტის საინფორმაციო სერვერის (IIS) მიერ. ქვემოთ მოყვანილ მაგალითში იგი შეიცავს VBScript ენაზე დაწერილ კოდს. შევნიშნოთ, რომ შესაძლებელია ამ კოდის გადაწერა PHP და ნებისმიერ სხვა სერვერულ ენაზეც.

მოცემული კოდით ხდება XML ფაილის დამუშავება და შედეგების HTML კოდის სახით ეკრანზე გამოტანა:

```
q=request.querystring("q")

set xmlDoc=Server.CreateObject("Microsoft.XMLDOM")
xmlDoc.async="false"
xmlDoc.load(Server.MapPath("cd_catalog.xml"))

set nodes=xmlDoc.selectNodes("CATALOG/CD[ARTIST='" & q & "']")

for each x in nodes
for each y in x.childnodes
response.write("<b>" & y.nodename & ":</b> ")
response.write(y.text)
response.write("<br />")
next
next
```

AJAX-ისთვის XMLHttpRequest ობიექტის გამოყენება

AJAX ტექნოლოგიით სარგებლობის საქმეში უმნიშვნელოვანეს როლს ასრულებს JavaScript ენაში არსებული XMLHttpRequest ობიექტი, მით უფრო, როცა საუბარია WEB 2.0 ტექნოლოგიაზე.

გავეცნოთ ამ ობიექტს, მის თვისებებსა და მეთოდებს.

ამთავითვე შევნიშნოთ, რომ სხვადასხვა ბროუზერები XMLHttpRequest ობიექტის შესაქმნელად განსხვავებულ გზებს მიმართავენ:

კერძოდ, Internet Explorer ბროუზერი მიზნის მისაღწევად იყენებს ActiveXObject საშუალებას, დანარჩენები ბროუზერები კი სარგებლობენ JavaScript ენის არსენალში მყოფი XMLHttpRequest ობიექტით.

შესაბამისად, ობიექტის შექმნისას საჭირო ხდება კოდში ამ პრობლემის გათვალისწინება-გადაჭრა, რაც ამგვარად ხდება:

```
var XMLHttp=null

if (window.XMLHttpRequest)
 {
 XMLHttp=new XMLHttpRequest()
 }
else if (window.ActiveXObject)
 {
 XMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
 }
```

ამრიგად, ამ ობიექტით სარგებლობის მიზნით თავდაპირველად იქმნება XMLHttp ცვლადი, რომელსაც ენიჭება null მნიშვნელობა. შემდეგ მოწმდება, შესაძლებელია თუ არა ბროუზერის მოცემული ვერსიისათვის window.XMLHttpRequest ობიექტით სარგებლობა (ასეთი რამ დასაშვებია ისეთი თანამედროვე ბროუზერებისათვის, როგორებიცაა: Firefox, Mozilla და

Opera). თუ ეს ასეა, მაშინ იქმნება ამ ობიექტის ეგზემპლარი. წინააღმდეგ შემთხვევაში მოწმდება ბროზერისათვის window.ActiveXObject ობიექტის გამოყენების შესაძლებლობა. პირობის დაკმაყოფილებისას (რაც ხდება Internet Explorer 5.5 და უფრო მაღალი დონის ვერსიებისათვის) ფორმირდება შესაბამისი ობიექტის ეგზემპლარი:

XMLHttp=new ActiveXObject().

მოვიყვანოთ სხვა მაგალითიც, რომელშიც გამოყენებული იქნება XMLHttpRequest-ობიექტის უფრო ახალი, სწრაფქმედი ვერსია. თუმცა თუ ბროუზერს ამ სიახლით ("Msxml2.XMLHTTP" ობიექტით) სარგებლობა არ შეუძლია, მაშინ მიმართვა ხდება Microsoft.XMLHTTP ობიექტისადმი:

```
var XMLHttp=null
try
{
XMLHttp=new ActiveXObject("Msxml2.XMLHTTP")
}
catch(e)
 {
 try
 {
 XMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
 }
}
if (XMLHttp==null)
{
XMLHttp=new XMLHttpRequest()
}
```

ზემოთ მოყვანილ მაგალითშიც თავდაპირველად ფორმირდება null მნიშვნელობის მქონე XMLHttp ცვლადი. შემდეგ ხორციელდება Internet

Explorer 6-სა და მომდევნო ვერსიებისათვის ობიექტის ეგზემპლარის შექმნის მცდელობა:

XMLHttp=new ActiveXObject("Msxml2.XMLHTTP")

თუ ეს ქმედება შეცდომას იწვევს, მაშინ გამოიყენება ადრე განხილული, ძველი მიდგომა, გათვალისწინებული Internet Explorer 5.5 ბროუზერისათვის:

XMLHttp=new ActiveXObject("Microsoft.XMLHTTP")

შემდეგ, თუ XMLHttp-ის მნიშვნელობა კვლავ null გახლავთ, პროგრამა ცდილობს ობიექტი სტანდარტად ქცეული გზით შექმნას:

XMLHttp=new XMLHttpRequest()

მოკლედ XMLHttpRequest-ის სხვა მეთოდების შესახებაც:

open() მეთოდით ფორმირდება Web-ისადმი მოთხოვნა,

send() მეთოდით ხდება სერვერისათვის მოთხოვნის გაგზავნა,

abort() მეთოდით კი ამ მოთხოვნის გაუქმება.

<u>XMLHttpRequest ობიექტის თვისებები:</u>

XMLHttpRequest ობიექტის თვისებებიდან გამოვარჩევთ ორ მათგანს:

პირველი გახლავთ **readyState თვისება**. იგი განსაზღვრავს XMLHttpRequest ობიექტის მიმდინარე მდგომარეობას.

ქვემოთ მოყვანილია readyState თვისების შესაძლო მნიშვნელობები:

მდგომარეობა	აღწერა
0	მოთხოვნა არ არის ინიციალიზებული
1	მოთხოვნა ფორმირებულია
2	მოთხოვნა გაიგზავნა
3	მოთხოვნა მუშავდება
4	მოთხოვნა შესრულდა

- readyState=0 – ცვლადს აღნიშნული მნიშვნელობა მიენიჭება XMLHttpRequest ობიექტის შექმნის შემდეგ. ამ მნიშვნელობას იგი ინარჩუნებს open() მეთოდის გამოძახებამდე;
- readyState=1 – ამ მნიშვნელობას ცვლადი ღებულობს open() მეთოდის გამოძახების შემდეგ;
- readyState=2 – მნიშვნელობის მინიჭება ხდება send() მეთოდის გამოძახების შემდეგ;
- readyState=3 – ბროუზერი სერვერს დაუკავშირდა, მაგრამ ჯერ სერვერიდან პასუხის მიღების პროცესი არ დასრულებულა;
- readyState=4 – დასრულდა სერვერიდან პასუხის მიღება.

სხვადასხვა ბროუზერები სხვადასხვაგვარად რეაგირებენ აღწერილ სიტუაციებზე. მაგალითად, ზოგი მათგანი არ იტყობინება “0” და “1” მდგომარეობების შესახებ.

აქვე უნდა აღინიშნოს, რომ AJAX ინტერესდება მხოლოდ ბოლო მდგომარეობით, ანუ სიტუაციით, როდესაც სერვერიდან მონაცემების გადმოგზავნა დამთავრდა და უკვე შესაძლებელი არის მათი დანიშნულებისამებრ გამოყენება.

რაც შეეხება XMLHttpRequest ობიექტის სხვა, responseText თვისებას, მისი დანიშნულება არის სერვერიდან გადმოგზავნილი ტექსტის შენახვა.

jQuery

შესავალი

jQuery წარმოადგენს მიერთებად ბიბლიოთეკას JavaScript-ისათვის. იგი მნიშვნელოვნად აადვილებს JavaScript-სა და HTML-ს შორის კავშირს. jQuery-ს ასევე მოიხსენიებენ როგორც JavaScript ენის ბაზაზე შექმნილ ფრეიმვორკსაც *(იხ. დანართი №1).*

jQuery-ის შექმნის საჭიროება განაპირობა შემდეგმა გარემოებებმა:

- ცნობილია, რომ სხვადასხვა ბროუზერები JavaScript-ზე დაწერილ სცენარებს არცთუ იშვიათად სხვადასხვაგვარად ასრულებენ, მაგალითად, განსხვავებულია დოკუმენტის ობიექტურ მოდელთან (DOM) მათი მუშაობის წესები, რაც, ცხადია გარკვეულ პრობლემებს წარმოშობს.
- თვით JavaScript ენაც მთელი რიგი ხარვეზებით ხასიათდება. მაგალითად, იგი ობიექტთან (ტეგთან) მისადგომად იყენებს ამორჩევის მხოლოდ ორ საშუალებას:
    - ტეგის დასახელებას;
    - ტეგის იდენტიფიკატორს.

  ამ დროს გამოუყენებელი რჩება დღეისათვის ძალიან ფართოდ გავრცელებული **სტილების კასკადური ცხრილების** (CSS) მეშვეობით დაკვალიფიცირებული *ტეგების კლასებისადმი* მიდგომის შესაძლებლობა. jQuery კი საშუალებას იძლევა CSS სელექტორების დახმარებით მარტივად იქნეს ამორჩეული როგორც ცალკეული ტეგები, ასევე მათი კლასები. ამასთან, დასაშვებია ამ პროცესში სხვადასხვა კრიტერიუმების გამოყენებაც.
- წლების განმავლობაში JavaScript-ზე დაწერილი სცენარების გაანალიზებიდან ნათელი გახდა, რომ საკმაოდ ხშირად საქმე გვაქვს სტანდარტული ამოცანების გადაწყვეტასთან, რომლებისთვისაც აზრი აქვს ცალკე მოდულების შექმნას. მაგალითად, jQuery-

ში გათვალისწინებული ავტომატური ციკლების მეშვეობით ადვილად ხდება მასივებში ელემენტების ჩათვალიერება.

სწორედ, ზემოთ ჩამოთვლილი პრობლემების მოხსნისათვის არის განკუთვნილი jQuery. მისი მეშვეობით მარტივდება შეღწევა როგორც DOM-ის ნებისმიერ ელემენტთან, ასევე – ამ ელემენტების ატრიბუტებსა და შემცველობაზე მანიპულაციების ჩატარებაც. ამასთან, აღსანიშნავია jQuery-ის სხვა ღირსებანიც:

- jQuery-ის შესაძლებლობებით უფასოდ შეიძლება ვისარგებლოთ. მის ჩამოსატვირთად უნდა მივმართოთ jquery.com საიტს.
- ფაილის ზომა მცირეა – იგი რამდენიმე ათეულ კილობაიტს არ აღემატება (დღეისათვის შეკუმშული სახით 30 კბ-ზე ცოტა მეტია).
- ინტერნეტში შესაძლებელია მოვიძიოთ jQuery-ის შესაძლებლობებით სარგებლობის ძალიან ბევრი მაგალითი.
- jQuery ბიბლიოთეკას მოიცავს მრავალ პლაგინს.
- იგი, ფაქტობრივად, მუდამ ახლდება, ანუ ვითარდება.
- jQuery-ის გამოყენებისათვის საჭირო არ გახლავთ რაიმე განსაკუთრებული მომზადება, ახლის დაუფლება – საკმარისია ვერკვეოდეთ CSS-სა და Java Script-ის შესაძლებლობებში.

დაბოლოს, აღვნიშნავთ, რომ jQuery უზრუნველყოფს ფრიად მოხერხებულ API ინტერფეისსაც *(იხ. დანართი №2)* Ajax-თან სამუშაოდ.

შენიშვნა: Ajax (Asynchronous Javascript and XML - ასინქრონული Javascript და XML) არის Web-გამოყენების აგებისადმი ისეთი მიდგომა, რომლის დროსაც WEB-ფურცელზე მონაცემების განახლების დროს ფურცლის მთლიანი გადატვირთვა არ ხდება, რაც მნიშვნელოვნად ამაღლებს ფაილთან მუშაობის სისწრაფესა და უფრო მოხერხებულს ქმნის მას.

jQuery არცთუ დიდი ხანია, რაც ინტერნეტ-სამყაროს მოევლინა. ამ ბიბლიოთეკის შემქმნელმა ჯონ რეზიგმა jQuery საზოგადოებას წარუდგინა 2006 წელს ნიუ-იორკში გამართულ კონფერენციაზე. ამავე

წლის დასაწყისშივე იქცა jQuery ბიბლიოთეკა Internet Explorer-ის შემადგენელ ნაწილად.

jQuery-ის შექმნის ძირითადი მიზანი გახლდათ, გაადვილებულიყო JavaScript-ზე დაწერილი სცენარების იმ ფრაგმენტთა კოდირება, რომლებიც მრავალჯერად გამოყენებას პოულობენ ინტერნეტში განთავსებისათვის განკუთვნილ ფაილებში. ამასთან, რეზიგი შეეცადა, თავის JavaScript-გამოყენებებში მაქსიმალურად მოეხსნა კროს-ბროუზერული მოხმარების პრობლემებიც.

მაინც, რას წარმოადგენენ აღნიშნული ბიბლიოთეკის კომპონენტები და რა შესაძლებლობებს გვთავაზობენ ისინი?

ესენი გახლავთ JavaScript-პლაგინები და Ajax-დამატებანი, რომლებიც უზრუნველყოფენ ხდომილობების დამუშავებას, ვიზუალურ ეფექტებს, ასევე, გადაადგილებებს DOM-იერარქიულ სტრუქტურაში XPath-ის იდეოლოგიაზე დაყრდნობით და სხვ.

შემდეგ, ისევე, როგორც CSS მიდგომა იღებს თავის თავზე დაფორმატების პრობლემების გადაწყვეტას და გამოაცალკევებს შესაბამის საშუალებებს HTML-ის სტრუქტურისაგან, ამგვარადვე იქცევა jQuery-იც ზემოთ აღნიშნული შესაძლებლობების რეალიზებისას. მაგალითად, ღილაკზე თაგვით დაწკაპუნებისას აღარ არის საჭირო პირდაპირ კოდში მოხდეს შესაბამისი ხდომილობის დამმუშავებლის ჩვენება. ტრადიციული ხერხისაგან განსხვავებით, ამჯერად, მართვა გადაეცემა JQuery-ს, რომელიც ჯერ მოახდენს ღილაკის იდენტიფიცირებას, შემდეგ კი განახორციელებს ამ ხდომილობისათვის (მოცემულ შემთხვევაში ღილაკზე თაგვით დაწკაპუნებისთვის) განკუთვნილ ქმედებებს.

მანიპულაციათა ობიექტების სტრუქტურისა და აღწერილის მსგავსი ქცევების ამდაგვარ განცალკევებას არამომაბეზრებელი JavaScript-ის პრინციპის სახელით მოიხსენიებენ.

ორიგინალური სახის გახლავთ jQuery ბიბლიოთეკის ორგანიზების კონცეფცია. ეს ბიბლიოთეკა წარმოგვიდგება კომპაქტური უნივერსალური ბირთვისა და პლაგინების ერთობლიობად. შედეგად, საჭირო აღარ არის მასში შემავალი ფუნქციების თეორიულად ყველა შესაძლო შემთხვევაზე გათვლა-ორიენტირება, რაც ძალიან გაზრდიდა კოდის (ამასთან, დიდწილად გამოუყენებადის) მოცულობას. შედეგად, შესაძლებელი ხდება რესურსისათვის „მოვიმარაგოთ“ სწორედ ის არსენალი (JavaScript-ფუნქციონალურობა), რომელიც, სავარაუდოდ, სავსებით საკმარისი იქნება კლიენტის (ამ შემთხვევაში დამპროექტებლის) წინაშე მდგომი ამოცანების გადასაწყვეტად.

HTML-ფაილთან jQuery ბიბლიოთეკის მისაერთებლად ვიყენებთ ასეთ მიდგომას:

```
<head>
  <script type="text/javascript" src="js/jquery.js">
</head>
```

მაშასადამე, jQuery ბიბლიოთეკის შემცველ ფაილს განვათავსებთ ჩვენი HTML-ფაილის მეზობლად შექმნილ js სახელის მქონე საქაღალდეში. რაც შეეხება თვითონ jQuery ბიბლიოთეკის შემცველ ფაილს, უმჯობესია იგი Google-დან გადმოვწეროთ. ასეთ შემთხვევაში საკმაოდ სწრაფად მოვიპოვებთ gzip ფორმატში მჭიდროდ დაარქივებულ, კომპაქტური ზომის მქონე სასურველი რესურსის (პლაგინის) უახლეს ვერსიას.

Google-ში შექმნილია სპეციალური საცავი jQuery-ის მინიმიზებული ვერსიებისათვის. საიტში ასეთი რესურსის გადმოსაწერად ვიყენებთ მიმართვას:

```
<script  type="text/javascript" src= "http://ajax.googleapis.com/
      ajax/libs/jquery/1.7.0/jquery.min.js">
</script>
```

ჩამოვთვალოთ ის უპირატესობანი, რომელთაც იძლევა jQuery-ის Google-დან ჩამოტვირთვის ხერხი:

- აღნიშნულ საცავს იყენებს მრავალი მსხვილი პროექტი, რომელთაც ერთდროულად მიმართავს მილიონობით მომხმარებელი (მაგალითად, twitter.com სისტემის). შესაბამისად, ძალიან დიდია ალბათობა იმისა, რომ ჩვენი პროექტის მიერ მოთხოვნილი პლაგინი (*იხ. დანართი №3*) უკვე იმყოფებოდეს კლიენტებისათვის ინფორმაციის დროებითი შენახვისათვის განკუთვნილ საცავში - კეშში. ასეთ შემთხვევაში მისი ჩამოტვირთვა, ფაქტობრივად, მომენტალურად მოხდება. ასეც რომ არ იყოს, პლაგინის გადმოწერა შესაძლებელი იქნება რომელიმე უახლოესი პროქსი სერვერიდან (*იხ. დანართი №3*), რაც, ცხადია, მაინც უფრო სწრაფად განხორციელდება, ვიდრე იგივე ქმედება დაშორებული სერვერიდან.
- თუ მოითხოვება jQuery-ის სულ მთლად ახალი ვერსიის მიერთება, რის გამოც პლაგინი ვერც ერთ კეშში ვერ მოიძიება, მაშინ მისი გადმოწერა მოხდება უშუალოდ Google-ის რომელიმე სერვერიდან (ამ მძლავრ სისტემას საკუთარი სერვერების საკმაოდ ფართო ქსელი გააჩნია.)
- მხედველობაში მისაღებია ის გარემოებაც, რომ ფაილის gzip შემჭიდროვება Google-ის სერვერებზე ორჯერადად ხორციელდება და შესაძლებელი ხდება, მაგალითად, უკვე 76 კილობაიტამდე შეკუმშული jquery 1.4.3 ფაილის ზომა განმეორებადი შეკუმშვით 26 კილობაიტამდე იქნეს დაყვანილი.

ბიბლიოთეკის მიერთების შემდეგ ჩვენს განკარგულებაშია jquery() ფუნქცია, რომლის გამოძახებითაც შესაძლებელია:

- ავირჩიოთ საჭირო ელემენტები;
- დავაკავშიროთ მათთან ხდომილებები;
- განვახორციელოთ მათზე სხვადასხვა ქმედებები.

აღვნიშნავთ, რომ jquery()-ის ნაცვლად დასაშვებია გამოყენებული იქნეს ფუნქციისადმი მიმართვის შემოკლებული ვარიანტიც: $(). მაგრამ გამორიცხული არაა, რომ თუ სხვა ფრეიმვორკებიც გამოიყენება, მათაც ფუნქციებისადმი მიმართვის ეს, შემოკლებული წესი გამოიყენონ. ასეთ შემთხვევებში ცხადია, აჯობებს მივმართოთ ფუნქციის გამოძახების jquery() ვარიანტს.

1. ელემენტების ამორჩევები

1.1. ელემენტების ამორჩევის სამი ძირითადი მეთოდი

როგორც უკვე აღვნიშნეთ, ელემენტების ამოსარჩევად jQuery იყენებს CSS-ით მოწოდებულ შესაძლებლობებს.

ნებისმიერი CSS ფაილის გაღებისას ჩვენ დავინახავთ ე.წ. **სელექტორებს** – ფიგურულ ფრჩხილებში განთავსებულ, სტილების აღმწერ ინფორმაციას და ამ ფრჩხილების წინ რაიმე ელემენტის, კლასის ან იდენტიფიკატორის დასახელებას. ამასთან, თუ თვით ამ დასახელებას წინ წერტილი უძღვის, საქმე გვაქვს კლასთან, გისოსის (#) შემთხვევაში – იდენტიფიკატორთან, ხოლო იმ შემთხვევაში, თუ დასახელების წინ არანაირი სიმბოლო არ ფიგურირებს, მაშინ – ტეგთან.

აქვე აღვნიშნოთ, რომ კასკადურ ცხრილებში ეს მონიშვნები (არჩევანი) გამიზნულია ცხრილებშივე განსაზღვრული სტილებით HTML ფაილში ელემენტების დაფორმატებისათვის, ხოლო jQuery ამავე ამორჩევებს ისევ ელემენტებისადმი მისადგომად იყენებს, ოღონდ მათი სხვადასხვა წესით დასამუშავებლად.

ამრიგად, HTML კოდში ელემენტების ამორჩევას jQuery-იც იმავე წესებით ახორციელებს, რომლებიც გამოიყენება ბროუზერის მიერ CSS ცხრილებიდან ამოღებული ინფორმაციის დამუშავებისათვის ცალკეული ელემენტების თუ კლასებისათვის სტილების განსაზღვრისას. ამასთან, დასაშვებია გამოყენებული იქნეს ელემენტებთან მიდგომის სამივე ზემოთ დასახელებული წესი.

განვიხილოთ თითოეული მათგანი:

ა) ელემენტის ამორჩევა დასახელების მიხედვით

მოვიყვანოთ მაგალითები. ტეგის ამორჩევას შესაძლებელია ჰქონდეს ასეთი სახე:

`$('p');` – *აირჩევა ყველა აბზაცი*

აქვე შევნიშნოთ, რომ იმავე მიზნის მისაღწევად Jacascript-ში შესაძლებელია შემდეგი ნოტაციის გამოყენებაც:

```
document.getElementsByTagName("p");
```

ადვილი შესამჩნევია, რომ პირველი ვარიანტი გაცილებით უფრო კომპაქტურია.

ბ) ელემენტის ამორჩევა იდენტიფიკატორის მიხედვით

ელემენტების ამორჩევა იდენტიფიკატორის მიხედვით jQuery-შიც ხდება CSS-სათვის მიღებული წესებით. აქაც მოსაძებნი ელემენტების იდენტიფიკატორის დასახელების წინ ფიგურირებს # სიმბოლო. მაგალითად, jQuery-ში ფორმირებული ნოტაცია:

```
$('#element');
```

უზრუნველყოფს HTML კოდში ყველა იმ ელემენტის ამორჩევას, რომლებისთვისაც ატრიბუტ id-ის მნიშვნელობა id="element".

გ) ელემენტების ამორჩევა კლასის მიხედვით

ამ შემთხვევაშიც ადგილი აქვს CSS-ისათვის მიღებულ წესებთან მსგავსებას. მაგალითად:

```
$('.greened');
```

ოპერატორით მოხდება ყველა იმ ელემენტის ამორჩევა, რომლებისთვისაც განსაზღვრულია ატრიბუტ კლასის შემდეგი მნიშვნელობა: class="greened".

ამრიგად, jQuery იყენებს ელემენტების ამორჩევის 3 შემდეგ ძირითად წესს:

```
$('p');          // ტეგის დასახელების მიხედვით
```

```
$('#element');    // იდენტიფიკატორის დასახელების მიხედვით
$('.greened');    // კლასის დასახელების მიხედვით
```

1.2. ელემენტების ამორჩევის უფრო რთული მეთოდები

ა) ჩადგმული ტეგები

დავუშვათ, მოითხოვება, ამორჩეული იქნეს ისეთი აბზაცები, რომელთა შიგნით ფიგურირებს strong ტეგი. ასეთი შემთხვევისათვის გამოვიყენებთ შემდეგი სახის ოპერატორს:

```
$('p strong');
```

შევნიშნოთ, რომ ამგვარივე წესი გამოიყენებოდა CSS ცხრილებისთვისაც.

ბ) მომდევნო ტეგი

მომდევნო ტეგის ამოსარჩევად შემოთავაზებული არის შემდეგი სახის კონსტრუქცია:

```
$('div + img');
```

გ) შვილობილი ტეგი

შვილობილი ტეგის ამოსარჩევად კი შესაძლებელია ასეთი მიდგომის გამოყენება:

```
$('div > img');
```

1.3. ჩადგმული ელემენტები

ა) ელემენტის (ელემენტების) ამორჩევა მასში ჩადგმული ელემენტის (ატრიბუტის) ზუსტი მნიშვნელობის მიხედვით

ძალიან ხშირად საჭირო არის ისეთი ელემენტების ამორჩევა, რომელთა შიგნითაც ფიგურირებს მოცემული მნიშვნელობის მქონე ესა თუ ის ელემენტი (ატრიბუტი).

მოვიყვანოთ ასეთი მოთხოვნის რეალიზების მაგალითი ელემენტის საწყის ტეგში არსებული ატრიბუტის ზუსტი მნიშვნელობისათვის:

$('img[alt=vardi]');

ამ ოპერატორის შესრულების შედეგად ამოირჩევა ყველა ის ნახატი-ელემენტი, რომლებისთვისაც alt ატრიბუტის მნიშვნელობა არის vardi.

ბ) ელემენტის (ელემენტების) ამორჩევა მისი ატრიბუტის მნიშვნელობის დასაწყისის მიხედვით

ელემენტები ამოირჩევა ატრიბუტის მნიშვნელობის დასაწყისის მიხედვით. მაგალითად, ოპერატორი:

$('img[src^=photo]');

მოახდენს ყველა იმ ნახატი-ელემენტის ამორჩევას, რომლებშიც ჩატვირთული გამოსახულების დასახელება (ან მისამართი) იწყება სიტყვა photo-თი, მაგალითად, - “photo007.jpg”.

გ) ელემენტის (ელემენტების) ამორჩევა მისი ატრიბუტის მნიშვნელობის დაბოლოების მიხედვით

ელემენტები ამოირჩევა წყაროს დასახელების დაბოლოების მიხედვით. მაგალითად, ოპერატორი:

$('img[src$=001.jpg]');

მოახდენს ყველა იმ ნახატი-ელემენტის ამორჩევას, რომელთა წყაროს დასახელება მთავრდება სიტყვა 001.jpg-თი, მაგალითად, - “photo001.jpg”.

დ) ამორჩევა მნიშვნელობაში შესვლის მიხედვით

ელემენტები ამოირჩევა შემდეგი კრიტერიუმის მიხედვით – კრიტერიუმში შედის თუ არა მოყვანილი სიტყვა. მაგალითად, ოპერატორი

$('img[src*=001]');

მოახდენს ყველა იმ ნახატის ამორჩევას, რომელთა წყაროს დასახელებაში (მისამართში) ფიგურირებს სტრიქონი 001.

1.4. ამორჩევის შედეგების ფილტრირება

ა) ფილტრაცია ჩამონათვალში ნომრის ლუწობა-კენტობაზე შემოწმების მიხედვით

მოგვყავს მაგალითი კოდში შეტანილი ნახატების "პირველ-მეორეზე გათვლისა":

```
$('img:even');   // ლუწი ელემენტების ამორჩევა
$('img:odd');    // კენტი ელემენტების ამორჩევა
```

ბ) ყველანი, გარდა ერთადერთი "განკიცხული" კლასისა

```
$('img:not(.green img)');
```

ოპერატორით ამოირჩევა ყველა ელემენტი (მოცემულ შემთხვევაში ნახატები), გარდა იმ ნახატების, რომლებიც მიეკუთვნებიან green კლასს.

გ) ტეგის შემცველობის მიხედვით

ქვემოთ მოყვანილი ოპერატორით შეირჩევა ყველა ის ელემენტი-ნახატი, რომელთათვისაც გათვალისწინებულია alt ატრიბუტი:

```
$('img:has(alt)');
```

დ) ამორჩევა ტექსტური ფრაგმენტის მიხედვით

აბზაცი ამოირჩევა, თუ იგი შეიცავს განსაზღვრულ ტექსტურ ფრაგმენტს. მაგალითად:

```
$('p:contains(რაიმე ტექსტი)');
```

ე) პირველი/ბოლო ელემენტის ამორჩევა

კოდში არსებული მოცემული სახის ელემენტებიდან აირჩევა პირველი (ან ბოლო) ელემენტი:

```
$('img:first');   // პირველი
$('img:last');    // ბოლო
```

ე) ხილული/დამალული ელემენტების ამორჩევა

კოდში არსებული მოცემული სახის ელემენტებიდან აირჩევა დამალული (ან ხილული) ელემენტები:

```
$('img:hidden');     //  დამალული
$('img:visibility'); //  ხილული
```

2. მოქმედებები ელემენტებზე

2.1. text()

ამ მეთოდის გამოყენებით შესაძლებელია ტექსტის შემცველ ამა თუ იმ ელემენტისაგან (მაგალითად, ეს შეიძლება იყოს P აბზაცი ან რომელიმე H სათაური) ტექსტის მიღება/შეცვლა.

მოვიყვანოთ მაგალითები:

```
$(document).ready ( function() {
                    $('p').text();
                    }
          );
```

აბზაციდან ამოვიღეთ ტექსტი. როგორც წესი, მას რომელიმე ცვლადში იმახსოვრებენ:

```
$(document).ready(function(){
  var textp = $('p').text();
 });
```

გამოვიყვანოთ ეს ტექსტი ეკრანზე:

```
$(document).ready(function(){
  var textp = $('p').text();
  alert(textp);
});
```

არსებული ტესტის შეცვლა კი ამგვარად ხორციელდება:

```
$(document).ready(function(){
```

```
  var textp = $('p').text('არსებულის შემცვლელი ტექსტი');
});
```

2.2. hide(), show()

ზოგჯერ მოითხოვება დამალული იქნეს ესა თუ ის ელემენტი. ამ მიზანს ემსახურება hide() მეთოდი, რომელსაც შეიძლება გადაეცეს შემდეგი 2 პარამეტრი:

- დრო გაქრობამდე (მილიწამებში);
- ფუნქციის დასახელება, რომელიც უნდა შესრულდეს მოცემული ელემენტის გაქრობის შემდეგ.

მოვიყვანოთ მაგალითი:

```
$(document).ready(function(){
  $('#example_id').hide(2000);
});
```

ამ კოდის შესრულების შედეგად 'example_id' იდენტიფიკატორით მონიშნული ელემენტი 2 წამის შემდეგ ეკრანიდან გაქრება.

დამალული ელემენტის კვლავ დისპლეიზე გამოსაყვანად კი ვიყენებთ ანალოგიური პარამეტრების მქონე show() მეთოდს:

```
$(document).ready(function(){
  $('#example_id').hide(2000);
  $('#example_id').show(2000);
});
```

2.3. ჯაჭვური ფუნქციები

ჯაჭვური ფუნქციების დახმარებით ზემოთ აღნიშნული ქმედებები შესაძლებელია ერთ სტრიქონში მოვაქციოთ:

```
$(document).ready(function(){
  $('#example_id').hide(2000).show(2000);
});
```

აღსანიშნავია, რომ ამ ხერხს უფრო ხშირად მიმართავენ, ვიდრე ზემოთ მოყვანილს.

2.4. ავტომატური ციკლები

მაგრამ, როგორ მოვიქცეთ ისეთ შემთხვევებში, როდესაც ერთდროულად არის საჭირო რამდენიმე ელემენტის დამალვა-გამოყვანა?

jQuery დასახული მიზნის რეალიზაციას უზრუნველყოფს ციკლების გამოყენების გარეშე – საჭიროა მხოლოდ ასეთი ელემენტების ამორჩევა განვახორციელოთ იდენტიფიკატორის დახმარებით და შევასრულოთ მათზე შესაბამისი მოქმედებანი.

2.5. ელემენტების სიმაღლე-სიგანის განსაზღვრა

ზოგჯერ მოითხოვება მიღებული იქნეს ინფორმაცია ელემენტის ზომების შესახებ. აღნიშნული ქმედება შემდეგნაირად განხორციელდება:

```
$(document).ready(function(){
  var wExample = $('#example_id').width();
  var hExample = $('#example_id').height();
});
```

რაც შეეხება ელემენტების სიგანე-სიმაღლის შეცვლას, შესაბამისი ფუნქციების პარამეტრებს ვანიჭებთ სასურველ მნიშვნელობებს:

```
$(document).ready(function(){
  $('#example_id').width(200);
  $('#example_id').height(300);
});
```

დაბოლოს, აქაც შესაძლებელია აღნიშნული ქმედებების ერთ ჯაჭვში მოქცევა:

```
$(document).ready(function(){
  $('#example_id').width(200).height(300);
});
```

2.6. ელემენტებისათვის HTML კოდის განსაზღვრა

ვნახეთ, რომ text() ფუნქციის მეშვეობით შესაძლებელია არჩეული ელემენტიდან ტექსტის მიღება და შეცვლაც. მაგრამ თუ მოითხოვება HTML კოდის ამოღება-შეცვლა, ამ მიზნით გამოყენებული უნდა იქნეს ასეთი მიდგომა (მაგალითად, აბზაცისათვის):

```
<p><strong>თქვენს წინაშეა სქელშრიფტიანი აბზაცი</strong></p>
```

text() ფუნქციის გამოყენებით ჩვენს განკარგულებაში გადმოეცემოდა მხოლოდ ტექსტი “თქვენს წინაშეა სქელშრიფტიანი აბზაცი”, მაშინ როდესაც HTML() ფუნქციის დახმარებით:

```
$(document).ready(function(){
  $('p').html();
});
```

ამორჩეული იქნება შემდეგი კოდის ფრაგმენტი:

```
“<p><strong>თქვენს წინაშეა სქელშრიფტიანი აბზაცი</strong></p>”
```

რაც შეეხება ამორჩეულ კოდში ცვლილებების შეტანას, HTML() ფუნქციაც ამ დავალებას text()-ის ანალოგიურად ახორციელებს.

2.7. ელემენტების მდოვრე გაქრობა-გამოჩენა

ზემოთ განხილული hide() და show() ფუნქციები ელემენტების გაქრობა-გამოყვანას ახდენდნენ ყოველგვარი ვიზუალური ეფექტების გარეშე, განსხვავებით fadeOut() და fadeIn() ფუნქციებისა. ამ უკანასკნელთ გადაეცემათ 2 პარამეტრი:

- მდოვრედ გაქრობა-გამოყვანის დროის მონაკვეთი,
- ფუნქცია, რომელიც უნდა შესრულდეს ამის შემდეგ.

მოვიყვანოთ მაგალითი:

```
$(document).ready(function(){
  $('img').fadeOut(1000).fadeIn(1000);
});
```

გამოიყენება ასევე fade() ფუნქციაც მესამე, დამატებითი პარამეტრით, რომლის დანიშნულება გახლავთ გაქრობის ხარისხის – გამჭვირვალობის დონის – განსაზღვრა (ვარირებს 0 – 1 დიაპაზონში):

```
$(document).ready(function(){
  $('img').fadeTo(1000,0.3).fadeTo(1000,1);
});
```

slideUp() და slideDown() ფუნქციებით კი შესაძლებელია გაქრობა-გამოყვანისას პროცესებისათვის მიმართულების ჩვენებაც – გაქრობა ქვემოდან ზემოთკენ, ხოლო გამოყვანა, ცხადია, - საპირისპირო მიმართულებით:

```
$(document).ready(function(){
  $('img').slideUp(1000).slideDown(1000);
});
```

2.8. ელემენტების ატრიბუტებთან მუშაობა

ვთქვათ, ეკრანზე ფიგურირებს რაიმე გამოსახულება:

```
<img src="http://მისამართი" height="100" width="200" alt="naxati_1.bmp">
```

და მოითხოვება გამოსახულების ატრიბუტებთან შეღწევა, მათი მნიშვნელობების შეცვლა, შესაძლოა ატრიბუტის ამოგდებაც კი.

ამ მიზნების მისაღწევად შეიძლება გამოყენებული იქნეს attr და attr() ფუნქციები:

```
$(document).ready(function(){
 var imgAdress = $('img').attr('src'); // ცვლადს გადაეცემა ნახატის მისამართი,
 var imgHeight = $('img').attr('height'); // აქ კი - ნახატის სიგანე.
 $('img').attr('width', '400'); // width ატრიბუტი მიიღებს მნიშვნელობა 400-ს
 $('img').removeAttr('alt'); // alt ატრიბუტი ამოვარდება
});
```

2.9. მოცემულ კლასში ელემენტის დამატება/ამორიცხვა

დავუშვათ მოცემული საიტისათვის CSS-ში განსაზღვრულია ასეთი კლასიც:

```
.tagText
{
 font-family: arial;
 margin-right: 20pt;
 color:#ffffff
}
```

jQuery-ში გათვალისწინებულია addClass() და removeClass() ფუნქციები საჭირო კლასში ამა თუ იმ ელემენტის გაწევრება-ამორიცხვისათვის.

დავუშვათ ჩვენს განკარგულებაშია შემდეგი აბზაცი:

```
<p id="main">მოგესალმებით ერთი რიგითი აბზაცი!</p>
```

მოვიყვანოთ ამ აბზაცის კლასში ჩარიცხვა-ამორიცხვის მაგალითები:

```
$(document).ready(function(){
  $('#main').addClass('tagText');
});

$(document).ready(function(){
  $('#main').removeClass('tagText');
});
```

2.10. CSS-თან მუშაობა

საინტერესოა, რომ jQuery-ის მეშვეობით შესაძლებელი ხდება, კორექტივები შევიტანოთ უშუალოდ CSS ცხრილებშიც.

მოვიყვანოთ შესაბამისი მაგალითები ზემოთ განხილული CSS ცხრილისათვის:

```
.tagText
```

```
{
 font-family: arial;
 margin-right: 20pt;
 color: #ffffff
}
```

ქვემოთ მოყვანილი წესით, მაგალითად, შესაძლებელია გავიგოთ, რომელი შრიფტი არის დანიშნული CSS ცხრილის მიერ main სახელით იდენტიფიცირებული ელემენტისათვის:

```
$(document).ready(function(){
  var textFont = $('#main').css('font-family');
});
```

ვხედავთ, რომ ამ მიზნის მიღწევაში გვეხმარება css() ფუნქცია – textFont ცვლადი მიიღებს “arial” მნიშვნელობას.

იმ მიზნის მისაღწევად, რომ ელემენტის ამა თუ იმ ატრიბუტს შევუცვლოთ მნიშვნელობა, მაგალითად, ფერი, საჭიროა, CSS ფუნქციის პარამეტრებად ვუჩვენოთ ატრიბუტის დასახელება და მძიმით გამოყოფილი მისი მნიშვნელობა:

```
$(document).ready(function(){
  $('#main').css('color', '#ff00ff');
});
```

ახლა კი მოვიყვანოთ მაგალითი რამდენიმე ატრიბუტისათვის ჯაჭვური წესით მნიშვნელობების შეცვლისა:

```
$(document).ready(function(){
  $('#main').css('color', '#ff00ff').css('font-family', 'verdana' );
});
```

იგივე ქმედებანი შესაძლებელი იყო მოგვეხდინა ისეთი ნოტაციების მეშვეობითაც, რომლებიც დაქვემდებარებულია CSS-ისთვის დამახასიათებელ სინტაქსს:

```
$(document).ready(function(){
  $('#main').css({
    'color' : '#ff00ff',
    'font-family' : 'verdana'
  });
});
```

იმ შემთხვევაში, როდესაც გვსურს, ესა თუ ის ქმედება დროში გაწელილად შესრულდეს, აქაც შესაძლებელია მივმართოთ animate() ფუნქციას:

```
$(document).ready(function(){
  $('#main').animate({
    'marginRight':'10px'
  },5000);
});
```

ზემოთ მოყვანილი კოდის შესრულების შედეგად აბზაცის დაცილება დოკუმენტის მარჯვენა კიდიდან 5 წამის განმავლობაში 10 პიქსელამდე დაიყვანება.

2.11. კონტენტის დამატება

დავუშვათ საიტის სტრუქტურაში, DOM ხეზე განთავსებულია რაიმე სურათი:

```
...
<img id="simple" src="http://მისამართი">
...
```

ამ სურათის წინ რაიმე კონტენტის დამატება შემდეგნაირად შესაძლებელია მოვახდინოთ:

```
$(document).ready(function(){
  $('#simple').before('<p>მე ვარ ნახატის წინ before () ფუნქციით
  დამატებული აბზაცი');
});
```

ხოლო მის უკან კი – ამგვარი წესით:

```
$(document).ready(function(){
  $('#simple').after('<p>მე ვარ ნახატის შემდეგ after () ფუნქციით დამატებული აბზაცი');
});
```

2.12. ელემენტების ციკლში გადარჩევა

ჯერ მოვიყვანოთ კოდის შესაბამისი ფრაგმენტი:

```
// სიის პუნქტების შემცველობა ციკლში გამოვიყვანოთ მანამ,
// სანამ არ შეგვხვდება 'stop' კლასის წევრი <li> პუნქტი.
$('li').each(function(i,elem) {
  if ($(this).is(".stop")) {
    alert("ციკლი  შეწყდა სიის " + i + "-ურ პუნქტზე.");
    return false;
   } else {
      alert(i + ': ' + $(elem).text());
     });
```

2.13. ამონაკრებში ელემენტების რიცხვის განსაზღვრა

ამ მიზნის შესრულებას ემსახურება size() ფუნქცია:

```
$(document).ready(function(){
  $('img').size();
});
```

2.14. კონკრეტულ ელემენტთან შეღწევა (შემოტანა)

კონკრეტულ ელემენტთან შეღწევა ხორციელდება get() ფუნქციის მეშვეობით (აღსანიშნავია, რომ ეს ფუნქცია გვიბრუნებს არა jQuery, არამედ javascript ტიპის ობიექტს!):

2.15. ელემენტის კლონირება

დავუშვათ გვესაჭიროება DOM სტრუქტურაში არსებული რაიმე ელემენტის, მაგალითად, ნახატის, ეკრანის ამა თუ იმ ადგილას გამოყვანა. clone() ფუნქციით ვახდენთ მის კლონირებას და ვიმახსოვრებთ ცვლადში (შემდეგ კი კლონირებულ ელემენტს დანიშნულებისამებრ ვიყენებთ - before() ან after() ფუნქციით ვსვამთ საჭირო ადგილას):

```
$(document).ready(function(){
        var Image = $('img').clone();
    });
```

2.16. განსხვავებული ტიპის მქონე ელემენტების ამორჩევა

არცთუ იშვიათად მოითხოვება, არჩეული იქნეს განსხვავებული ტიპის ელემენტები, რათა შემდგომ მათზე ჩატარდეს ესა თუ ის მოქმედება (მაგალითად, დავმალოთ ისინი). მოვიყვანოთ ასეთი ამოცანის გადაწყვეტის მაგალითი:

```
$(document).ready(function(){
  var s = $('img, p').size();      // ყველა ნახატის და აბზაცის არჩევა
  s.hide();                         // მათი დამალვა
});
```

3. ხდომილობებზე ელემენტების რეაქცია

3.1. თაგუნასთან დაკავშირებული ხდომილობები

განვიხილოთ თაგუნასთან დაკავშირებული ხდომილობები. ისინი ამა თუ იმ ელემენტს მიაბამენ რაიმე დამმუშავებელს და მეტწილად მაშინვე შეჰყავთ ძალაში შესაბამისი ხდომილობა.

ა) .mouseover()

მოვიყვანოთ გამოყენების მაგალითები:

```
// foo-თი  იდენტიფიცირებულ  ელემენტზე ვაყენებთ .mouseover()
// ხდომილობის დამმუშავებელს
$('#foo').mouseover(function(){
 alert ('თქვენ კურსორი განათავსეთ foo ელემენტის ზონაში.');
});

// foo ელემენტისათვის mouseover  ხდომილობის გამოძახება
$('#foo').mouseover();

// აქ block  კლასის ელემენტებისათვის ვაყენებთ კიდევ ერთ
// ხდომილობის დამმუშავებელს, რომელსაც გადავცემთ მონაცემებს.

$('.block').mouseover({a:12, b:"abc"}, function(eventObject){
 var externalData = "a=" + eventObject.data.a + ", b=" +
 eventObject.data.b;

alert ('block კლასის ელემენტზე გამოჩნდა კურსორი. '+
     'ამ ხდომილობის დამმუშავებელს გადაეცა მონაცემები: ' +
     externalData );
 });
```

ბ) .mouseout()

გამოყენების მაგალითები:

```
// foo-თი  იდენტიფიცირებულ  ელემენტზე ვაყენებთ .mouseout()
// ხდომილობის დამმუშავებელს
$('#foo').mouseout(function(){
 alert(' თქვენ კურსორი გაიყვანეთ foo ელემენტის ზონიდან.');
});

 // foo ელემენტისათვის mouseout ხდომილობის გამოძახება
$('#foo').mouseout();
```

```
// აქ block კლასის ელემენტებისათვის ვაყენებთ კიდევ ერთ
// ხდომილობის დამმუშავებელს, რომელსაც გადავცემთ მონაცემებს.
$('.block').mouseout({a:12, b:"abc"}, function(eventObject){
 var externalData = "a=" + eventObject.data.a + ", b=" + eventObject.data.b;
 alert(" კურსორი გავიდა block კლასის ელემენტის ზონიდან. '+
      'ამ ხდომილობის დამმუშავებელს გადაეცა მონაცემები:' +
externalData );
});
```

გ) .click()

გამოყენების მაგალითები:

```
// foo-თი  იდენტიფიცირებულ  ელემენტზე ვაყენებთ .click()
// ხდომილობის დამმუშავებელს
$('#foo').click(function(){
 alert ('თქვენ დააჭირეთ foo ელემენტზე.);
});

// foo ელემენტისათვის click ხდომილობის გამოძახება
$('#foo').click();

// აქ block კლასის ელემენტებისათვის ვაყენებთ კიდევ ერთ
// ხდომილობის დამმუშავებელს, რომელსაც გადავცემთ მონაცემებს.
$('.block').mouseover({a:12, b:"abc"}, function(eventObject){
 var externalData = "a=" + eventObject.data.a + ", b=" +
 eventObject.data.b;
 alert (' თქვენ დააჭირეთ block კლასის ელემენტზე. '+
      'ამ ხდომილობის დამმუშავებელს გადაეცა მონაცემები:' +
      externalData );
 });
```

დ) .dblclick()

გამოყენების მაგალითები:

```
// foo-თი  იდენტიფიცირებულ  ელემენტზე ვაყენებთ .dblclick()
// ხდომილობის დამმუშავებელს
$('#foo').dblclick(function(){
  alert ('თქვენ 2-ჯერ დააწკაპუნეთ foo ელემენტზე.');
});

// foo ელემენტისათვის dblclick ხდომილობის გამოძახება
$('#foo').dblclick();

// აქ block კლასის ელემენტებისათვის ვაყენებთ კიდევ ერთ
// ხდომილობის დამმუშავებელს, რომელსაც გადავცემთ მონაცემებს.
$('.block'). dblclick ({a:12, b:"abc"}, function(eventObject){
  var externalData = "a=" + eventObject.data.a + ", b=" +
  eventObject.data.b;
 alert (' თქვენ 2-ჯერ დააწკაპუნეთ block კლასის ელემენტზე. '+
      'ამ ხდომილობის დამმუშავებელს გადაეცა მონაცემები: ' +
      externalData );
 });
```

ე) .mousemove()

გამოყენების მაგალითები:

```
// foo-თი  იდენტიფიცირებულ  ელემენტზე ვაყენებთ .mousemove()
// ხდომილობის დამმუშავებელს
$('#foo').mousemove(function(){
  alert ('თქვენ დაძარით თაგვი.');
});

// foo ელემენტისათვის mousemove ხდომილობის გამოძახება
$('#foo'). mousemove ();
```

```
// აქ block კლასის ელემენტებისათვის ვაყენებთ კიდევ ერთ
// ხდომილობის დამმუშავებელს, რომელსაც გადავცემთ მონაცემებს.
$('.block'). mousemove ({a:12, b:"abc"}, function(eventObject){
 var externalData = "a=" + eventObject.data.a + ", b=" +
 eventObject.data.b;
alert ('გაადგილებული იქნა თაგვის კურსორი. '+
     'ამ ხდომილობის დამმუშავებელს გადაეცა მონაცემები: ' +
     externalData );
 });
```

გ) .mousedown()

გამოყენების მაგალითები:

```
// foo-თი იდენტიფიცირებულ ელემენტზე ვაყენებთ .mousedown()
// ხდომილობის დამმუშავებელს
$('#foo'). mousedown (function(){
 alert ('თქვენ თაგვის ღილაკზე დააჭირეთ თაგვის კურსორის foo
ელემენტზე ყოფნისას. თაგვის დაჭერილი ღილაკის კოდია – .);
});

// foo ელემენტისათვის mousedown ხდომილობის გამოძახება
$('#foo'). mousedown ();

// აქ block კლასის ელემენტებისათვის ვაყენებთ კიდევ ერთ
// ხდომილობის დამმუშავებელს, რომელსაც გადავცემთ მონაცემებს.
$('.block'). mousedown ({a:12, b:"abc"}, function(eventObject){
 var externalData = "a=" + eventObject.data.a + ", b=" +
 eventObject.data.b;
alert ('დაჭერილი იქნა თაგვის ღილაკი. '+
```

```
      'ამ ხდომილობის დამმუშავებელს გადაეცა მონაცემები: ' +
      externalData );
  });
```

ზ) .mouseup()

გამოყენების მაგალითები:

```
// foo-თი   იდენტიფიცირებულ   ელემენტზე ვაყენებთ .mouseup()
// ხდომილობის დამმუშავებელს
$('#foo'). mouseup (function(){
 alert ('თქვენ მოხსენით დაჭერა თაგვის ღილაკზე.  აშვებული
ღილაკის   კოდია – .);
});

 // foo ელემენტისათვის mouseup ხდომილობის გამოძახება
$('#foo'). mouseup ();

// აქ block კლასის ელემენტებისათვის ვაყენებთ კიდევ ერთ
// ხდომილობის დამმუშავებელს, რომელსაც გადავცემთ მონაცემებს.

$('.block'). mouseup ({a:12, b:"abc"}, function(eventObject){
 var externalData = "a=" + eventObject.data.a + ", b=" +
 eventObject.data.b;

 alert ('თქვენ აუშვით თაგვის ღილაკი. '+
      'ამ ხდომილობის დამმუშავებელს გადაეცა მონაცემები: ' +
      externalData );
  });
```

3.2. ფორმებთან დაკავშირებული ხდომილობები

ა) .submit()

გამოყენების მაგალითები:

```
// foo-თი იდენტიფიცირებულ ელემენტზე ვაყენებთ .submit()
// ხდომილობის დამმუშავებელს, რის შემდეგაც ვკრძალავთ
// სერვერზე მონაცემების გაგზავნას
$('#foo'). submit (function(){
  alert (' foo ფორმა გაიგზავნა სერვერზე.');
  return false;
});

// foo ელემენტისათვის submit ხდომილობის გამოძახება
$('#foo'). submit ();

// აქ block კლასის ელემენტებისათვის ვაყენებთ კიდევ ერთ
// ხდომილობის დამმუშავებელს, რომელსაც გადავცემთ მონაცემებს.
$('.block'). submit ({a:12, b:"abc"}, function(eventObject){
 var externalData = "a=" + eventObject.data.a + ", b=" +
 eventObject.data.b;
 alert (' foo ფორმა გაიგზავნა სერვერზე. '+
     'ამ ხდომილობის დამმუშავებელს გადაეცა მონაცემები:' +
     externalData );
 });
```

ბ) .focus()

თავიდანვე შევნიშნავთ, რომ როგორც კი ფორმის ელემენტი მოხვდება ფოკუსში, ადგილი ექნება focus ხდომილობას.

გამოყენების მაგალითები:

```
// foo-თი იდენტიფიცირებულ ელემენტზე ვაყენებთ .focus()
// ხდომილობის დამმუშავებელს
```

```
$('#foo'). focus (function(){
   alert (' foo  ელემენტი ფოკუსშია. ');
});

// foo ელემენტისათვის focus ხდომილობის გამოძახება
$('#foo'). focus ();

// აქ block კლასის ელემენტებისათვის ვაყენებთ კიდევ ერთ
// ხდომილობის დამმუშავებელს, რომელსაც გადავცემთ მონაცემებს.
$('.block'). focus ({a:12, b:"abc"}, function(eventObject){
  var externalData = "a=" + eventObject.data.a + ", b=" +
  eventObject.data.b;
 alert (' block  კლასის ელემენტი მოხვდა ფოკუსში. '+
      'ამ ხდომილობის დამმუშავებელს გადაეცა მონაცემები: ' +
      externalData );
 });
```

გ) .blur()

შევნიშნავთ, რომ როგორც კი ფორმის ელემენტი დაკარგავს ფოკუსს, ადგილი ექნება blur ხდომილობას.

გამოყენების მაგალითები:

```
// foo-თი  იდენტიფიცირებულ  ელემენტზე ვაყენებთ .blur()
// ხდომილობის დამმუშავებელს
$('#foo'). blur (function(){
   alert (' foo  ელემენტმა დაკარგა ფოკუსი. ');
});

// foo ელემენტისათვის blur ხდომილობის გამოძახება
$('#foo'). blur ();

// აქ block კლასის ელემენტებისათვის ვაყენებთ კიდევ ერთ
```

```
// ხდომილობის დამმუშავებელს, რომელსაც გადავცემთ მონაცემებს.
$('.block'). blur ({a:12, b:"abc"}, function(eventObject){
 var externalData = "a=" + eventObject.data.a + ", b=" +
 eventObject.data.b;
 alert (' block კლასის ელემენტმა დაკარგა ფოკუსი. '+
     'ამ ხდომილობის დამმუშავებელს გადაეცა მონაცემები: ' +
     externalData );
 });
```

დ) .change()

ხდება ინფორმაციის მიღება ფორმის ნებისმიერ ელემენტში ცვლილებების მოხდენისას, ამის შემდეგ საჭიროების შემთხვევაში შესაძლებელია ხდომილების დამუშავება და ამ მიზნით დამმუშავებელში მონაცემების გადაგზავნაც.

გამოყენების მაგალითები:

```
// foo-თი იდენტიფიცირებულ ელემენტზე ვაყენებთ .change()
// ხდომილობის დამმუშავებელს
$('#foo'). change (function(){
  alert (' მოხდა foo ელემენტის ცვლილება. ');
});

// foo ელემენტისათვის change ხდომილობის გამოძახება
$('#foo'). change ();

// აქ block კლასის ელემენტებისათვის ვაყენებთ კიდევ ერთ
// ხდომილობის დამმუშავებელს, რომელსაც გადავცემთ მონაცემებს.
$('.block'). blur ({a:12, b:"abc"}, function(eventObject){
 var externalData = "a=" + eventObject.data.a + ", b=" +
 eventObject.data.b;
```

```
alert (' block  კლასის  ელემენტში  მოხდა  ცვლილება.  '+
    'ამ  ხდომილობის  დამმუშავებელს  გადაეცა  მონაცემები:' +
    externalData );
});
```

3.3. კლავიატურასთან დაკავშირებული ხდომილობები

ა) .keypress()

გამოყენების მაგალითები:

```
// foo-თი  იდენტიფიცირებულ  ელემენტზე  ვაყენებთ  .keypress()
// ხდომილობის  დამმუშავებელს
$('#foo'). keypress (function(){
  alert (' თქვენ  კლავიატურიდან  შეიყვანეთ  სიმბოლო,  რომლის
კოდია  ' + eventObject.which);
});

// foo ელემენტისათვის keypress ხდომილობის გამოძახება
$('#foo'). keypress ();

// აქ block  კლასის  ელემენტებისათვის  ვაყენებთ  კიდევ  ერთ
// ხდომილობის დამმუშავებელს, რომელსაც გადავცემთ მონაცემებს.

$('.block'). keypress ({a:12, b:"abc"}, function(eventObject){
 var externalData = "a=" + eventObject.data.a + ", b=" +
 eventObject.data.b;

alert (' თქვენ  კლავიატურიდან  შეიყვანეთ  სიმბოლო.  '+
    'ამ  ხდომილობის  დამმუშავებელს  გადაეცა  მონაცემები:' +
    externalData );
});
```

ბ) .keydown()

გამოყენების მაგალითები:

```
// foo-თი   იდენტიფიცირებულ   ელემენტზე ვაყენებთ .keydown()
// ხდომილობის დამმუშავებელს და ვამოწმებთ, რომელი კლავიში
// არის   დაჭერილი

$('#foo'). keydown (function(){

   alert (' დაჭერილია სიმბოლო კლავიატურაზე. შესატანი

 სიმბოლოს კოდია  ' + eventObject.which);

});


// foo ელემენტისათვის keydown ხდომილობის გამოძახება

$('#foo'). keydown ();

// აქ block კლასის ელემენტებისათვის ვაყენებთ კიდევ ერთ

// ხდომილობის დამმუშავებელს, რომელსაც გადავცემთ მონაცემებს.

$('.block'). keydown ({a:12, b:"abc"}, function(eventObject){

 var externalData = "a=" + eventObject.data.a + ", b=" +

 eventObject.data.b;

alert (' თქვენ მიერ კლავიატურაზე დაჭერილია სიმბოლო.  '+

     'ამ ხდომილობის დამმუშავებელს გადაეცა მონაცემები: ' +

     externalData );

 });
```

გ) .keyup()

გამოყენების მაგალითები:

```
// foo-თი   იდენტიფიცირებულ   ელემენტზე ვაყენებთ .keyup()

// ხდომილობის დამმუშავებელს და ვამოწმებთ, რომელი კლავიში

// იქნა აშვებული

$('#foo'). keyup (function(){
```

```
  alert (' დაჭერისაგან გათავისუფლდა  კლავიატურაზე სიმბოლო,
რომლის კოდია ' + eventObject.which);
});

// foo ელემენტისათვის keyup ხდომილობის გამოძახება
$('#foo'). keyup ();

// აქ block კლასის ელემენტებისათვის ვაყენებთ კიდევ ერთ
// ხდომილობის დამმუშავებელს, რომელსაც გადავცემთ მონაცემებს.

$('.block'). keyup ({a:12, b:"abc"}, function(eventObject){
 var externalData = "a=" + eventObject.data.a + ", b=" +
 eventObject.data.b;

alert (' კლავიატურაზე აშვებული იქნა კლავიში.  '+
     'ამ ხდომილობის დამმუშავებელს გადაეცა მონაცემები: ' +
     externalData );
 });
```

3.4. ბროუზერის ფანჯარასთან დაკავშირებული ხდომილობები

ა) .load()

ხდება ყველა ამორჩეული ელემენტის მთლიანად ჩატვირთვის ხდომილობისათვის დამმუშავებლის დაყენება. ცხადია, რომ შესაძლებელია იგივე მოხდეს მთლიანი Web-ფურცლის ჩატვირთვის ხდომილობისთვისაც:

გამოყენების მაგალითები:

ვთქვათ, Web-ფურცელზე განთავსებულია რაიმე ნახატი:

```
<img src="book.png" alt="Book" id="book" />
```

შესაძლებელია მისი ჩატვირთვის შემთხვევაში მართვა გადავცეთ ამა თუ იმ *ხდომილების დამმუშავებელს*:

```
$('#book').load(function() {
  // აქ განთავსდება ნახატის ჩატვირთვის ხდომილების
  // დამმუშავებლის კოდი
});
```

Web-ფურცლის ჩატვირთვა კი ასე დამუშავდება;

```
$(window).load(function () {
  // აქ განთავსდება Web-ფურცლის ჩატვირთვის
  // ხდომილების დამმუშავებლის კოდი
});
```

ბ) **.**resize()

გამოყენების მაგალითები:

```
// ბროუზერის ფანჯრის ზომების ცვლილების resize
// ხდომილობისათვის ყენდება  შესაბამისი დამმუშავებლის კოდი
$(window).resize(function(){
  alert('ბროუზერის ფანჯრის ზომები შეიცვალა.');
});
// აქ განთავსდება ბროუზერის ფანჯრის ზომების ცვლილების
// ხდომილების დამმუშავებლის კოდი
$(window).resize();
```

გ) **.**scroll()

გამოყენების მაგალითები:

```
// foo-თი  იდენტიფიცირებულ  ელემენტზე ვაყენებთ .scroll()
// ხდომილობის დამმუშავებელს
$('#foo').scroll(function(){
  alert('' ჩატარდა foo ელემენტის სკროლინგი.');
});

// foo ელემენტისათვის scroll ხდომილობის გამოძახება
```

```
$('#foo').scroll();
  // აქ block  კლასის ელემენტებისათვის ვაყენებთ კიდევ ერთ
  // ხდომილობის დამმუშავებელს, რომელსაც გადავცემთ მონაცემებს.
$('.block').scroll({a:12, b:"abc"}, function(eventObject){
 var externalData = "a=" + eventObject.data.a + ", b=" +
eventObject.data.b;
 alert(' block კლასის ელემენტისათვის განხორციელდა სკროლინგი. '+
     'ამ ხდომილობის დამმუშავებელს გადაეცა მონაცემები: ' +
     externalData );
});
```

დ) .unload()

მისი მეშვეობით ყენდება Web-ფურცლიდან გამოსვლის ხდომილების დამმუშავებელი Window ობიექტისათვის (ბროუზერის დახურვისას, დაყრდნობით გადასვლისას და სხვ.)

გამოყენების მაგალითები:

```
$(window).unload(function(){
 alert("მომავალ შეხვედრამდე!");
});
```

დანართი № 1

ფრეიმვორკები

ფრეიმვორკ (framework) ტერმინის სინონიმად მიიჩნევა ჩვენთვის უფრო გასაგები სიტყვა - კარკასი.

ფრეიმვორკი (კარკასი) შეიძლება განიმარტოს, როგორც პროგრამული უზრუნველყოფა, რომელიც ამარტივებს დიდი პროგრამული პროექტების დამუშავების პროცესს აღნიშნულ კარკასში ამ პროექტებისათვის საჭირო კომპონენტებისა და მათ შორის "საერთო ენის გამოსანახად" შესაბამისი წესების ერთ სტრუქტურაში მოქცევის გზით.

ტერმინი კარკასი აქ კიდევ ერთი შინაარსობრივი დატვირთვის მატარებელია:

თანამედროვე პროგრამები, როგორც წესი, იგება ორი კომპონენტისაგან - უცვლელი (კარკასი და მისაერთებელი ბუდეები) და ცვლადი (მოდულები) ნაწილების ერთობლიობის სახით.

შეიძლება ითქვას, რომ ფრეიმვორკები შემდგომი ნაბიჯია პროგრამული სისტემების განვითარებაში - ბიბლიოთეკური მიდგომიდან, რაც გულისხმობდა მსგავსი ფუნქციონალობის მქონე ქვეპროგრამების ერთი სახურავის ქვეშ განთავსებას, გადავდივართ ახალ ეტაპზე – პროგრამული სისტემის კარკასში უკვე მოქცეულია ბიბლიოთეკა საკუთარი კოდისათვის, მთელი რიგი სხვადასხვა დანიშნულების მქონე ბიბლიოთეკებისა, სცენარების ენა და დიდი, მრავალკომპონენტური პროექტის შესაქმნელად აუცილებელი სხვა საშუალებები. ამ მეურნეობის მართვა კი, როგორც წესი, ერთიანი API (Application programming interface)-ის მეშვეობით ხორციელდება *(იხ. დანართი №2).*

აღსანიშნავია, რომ ფრეიმვორკული მიდგომა გამოიყენება არა მარტო დიდი პროექტების შესაქმნელი პროგრამული სისტემების შემუშავებისას, არამედ დღეს მათი პროდუქტებიც – ცალკეული პროგრამული გამოყენებებიც (სხვაგვარად, დანართები) ამ კონცეფციის მიმდევრები არიან (ფრეიმვორკად მოხსენიებისათვის საჭირო პირობების მეტ-ნაკლები დაცვით).

ფრეიმვორკების რეალიზაცია ხდება კონკრეტული და აბსტრაქტული კლასების შემუშავების, ასევე მათი განსაზღვრისა და ურთიერთქმედების წესების დადგენის შედეგად. აქვე აღვნიშნოთ, რომ კონკრეტულ კლასებს შორის "საუბრის წესები", ჩვეულებრივ, უკვე შემუშავებულია. აბსტრაქტული კლასებისათვის კი შემოღებულია ე.წ. *გაფართოების წერტილები*, რომლებისთვისაც ასეთი რამ მხოლოდ გარკვეული მოსამზადებელი სამუშაოების ჩატარების შემდგომ იქნება შესაძლებელი.

პროგრამული ფრეიმვორკული სახის სისტემის მაგალითად შეიძლება დავასახელოთ ვებ-პროგრამირებაში კარგად ცნობილი კონტენტის მართვის სისტემები (CMS), ხოლო შესაბამისი სახის დანართებისათვის შესაქმნელად ფართოდ გამოიყენება მაიკროსოფტის პროდუქტი - .NET Framework.

(იხ. დანართი № 2).

გამოყენებითი დაპროგრამების ინტერფეისი API (application programming interface)

API (application programming interface) – წარმოადგენს გამოყენებითი დაპროგრამების ინტერფეისს, რომელიც საშუალებას გვაძლევს ჩვენ მიერ შესაქმნელ გარე პროგრამულ პროდუქტში გამოვიყენოთ ამა თუ იმ სტანდარტულ გამოყენებაში (ბიბლიოთეკაში, სერვისში) არსებული მზამზარეული კლასები, ფუნქციები, სტრუქტურები, კონსტანტების კრებული.

API ცნება ახლოს არის ოქმის ცნებასთან. ეს უკანასკნელი გამოიყენება, მაგალითად, ინტერნეტში 7-დონიანი სქემის მეზობელ დონეებს შორის ურთიერთობისათვის, რაც გამოიხატება მონაცემების გაცვლაში. API კი უზრუნველყოფს გამოყენებებს შორის ურთიერთ-ქმედებებს.

არსებობს მთელი API ბიბლიოთეკები მომხმარებლის უზრუნველ-საყოფად ფუნქციებითა და კლასებით. მათში აღიწერება ფუნქციების სიგნატურა და სემანტიკა.

დანართი № 3

პლაგინი

პლაგინი (plug-in) წარმოადგენს სხვა პროგრამებისაგან დამოუკი-დებლად კომპილირებად პროგრამულ მოდულს, რომელიც ამა თუ იმ პროგრამას მისი შესრულების დროს დინამიკურად შეიძლება მიუერთდეს. შედეგად ხდება ამ უკანასკნელის მოქმედების შესაძლებლობების გაფართოება. (დინამიკური ბიბლიოთეკის ფაილებისათვის Windows ოპერაციული სისტემების ოჯახში გათვალისწინებულია **.dll** გაფართოება). აქვე შევნიშნოთ, რომ პლაგინს ხშირად მოდულადაც მოიხსენიებენ.

თუ პლაგინი ოპერაციული სისტემის მეხსიერებაში (კეშში) ჩაიტვირთა, ჩვეულებრივ, მისი ერთადერთი ასლით რამდენიმე პროგრამას შეუძლია ისარგებლოს. ამ შემთხვევაში პლაგინი ასრულებს ე.წ. *გაყოფადი ბიბლიოთეკის* როლს. ასეთი ბიბლიოთეკების დიდი ღირსება არის მეხსიერების ეკონომია. მათგან განსხვავებით, *სტატიკური ბიბლიოთეკები* (მათი გაფართოება Windows-ში გახლავთ **.lib**) ძირითად (გამომძახებელ) პროგრამულ მოდულს კომპილაციის ეტაპზე უერთდება. შედეგად ეს პროგრამა ავტონომიური ხდება, მაგრამ მთლიანობაში, ამგვარი პროგრამების მოცულობა მათში ბიბლიოთეკების დუბლირების გამო იზრდება.

დანართი № 4

პროქსი-სერვერი

პროქსი-სერვერი (ინგლ. proxy – წარმომადგენელი, უფლებამოსილი) არის კომპიუტერულ ქსელებში მეტად ხშირად გამოყენებული სამსახური (პროგრამების კომპლექსი), რომელიც კლიენტებს შესაძლებლობას აძლევს მიღებული იქნეს მათი მოთხოვნები სხვა სერვერებზე არსებულ რესურსებზე, მაგალითად, განაცხადი ამა თუ იმ სახის საფოსტო მომსახურებაზე. ამასთან, ხშირად შესაძლებელია მოთხოვნა პროქსი-სერვერის კეშ-მეხსიერებიდანაც დაკმაყოფილდეს, მაგრამ თუ ეს ვერ ხერხდება მოთხოვნა გადაიგზავნება შესაბამის სერვერზე.

საინტერესოა, რომ საჭიროების შემთხვევაში პროქსი-სერვერს შეუძლია მოახდინოს კლიენტის მოთხოვნისა და/ან მოთხოვნაზე სერვერის პასუხის კორექტირებაც.

ძალიან მნიშვნელოვანია, რომ პროქსი-სერვერი შესაძლებლობას იძლევა დაცული იქნეს კლიენტის ანონიმურობა, ასევე, რიგ შემთხვევებში – უზრუნველყოფილი იქნეს კომპიუტერის დაცვა არასანქცირებული შეღწევებისაგან (ქსელური შეტევებისაგან).

დაბოლოს, ქვემოთ მოგვყავს ზოგიერთი სხვა ტერმინის განმარტებაც:

DOM არის ბროუზერისათვის ცნობილი ობიექტებისაგან აგებული WEB-დოკუმენტების სტრუქტურული მოდელი. მისი რამდენიმე სპეციფიკაცია არსებობს, თუმცა W3 კონსორციუმისაგან ამათგან სანქცირებული მხოლოდ ერთადერთია.

WEB-სერვისი, როგორც წესი, განიმარტება, როგორც ოპერაციული სისტემის, WEB-დანართის (გამოყენების), მონაცემთა რელაციური ბაზის სერვერისათვის სკრიპტული ენისა და HTML, CSS და Javascript-ის ერთიანობა.

Ajax (ასინქრონული Javascript+XML) ტექნოლოგიაა, რომელიც ამარტივებს ვებ-დაპროგრამებას ინტერფეისის (API)GetXMLHttpRequest ელემენტზე დაყრდნობის შედეგად.

DTD – დოკუმენტის ტიპის გამოცხადება – მიგვითითებს გამოყენებული HTML-ის ვერსიაზე.

URI (Uniform Resource Identifiers)

საინტერესოა, რომ URI (Uniform Resource Identifiers) და URL (Uniform Resource Locators) ცნებებს ხშირად აიგივებენ, მაგრამ ეს მთლად მართებული არ არის. URI გამოიყენება ინტერნეტში სასურველ რესურსთან მისადგომად. იგი ფრიად "ჩახუჭუჭებული" სტრუქტურისაც შეიძლება იყოს, რის გამოც ითვლება, რომ URI უფრო კომპიუტერის (და არა მომხმარებლის) მიერ წაკითხვადობაზეა ორიენტირებული. შესაბამისად, სასურველად მიიჩნევა, მოხდეს URI-ის დამალვა და მომხმარებლებისთვის სასურველი სახის მისამართის ფორმირება კომპიუტერს დაეკისროს. ამასთან, შესაძლებელია ერთსა და იმავე რესურსს სხვადასხვა URI-თაც მივადგეთ. უფრო გრძელი მისამართი რიგ შემთხვევებში აადვილებს რესურსთან შეღწევადობას და/ან რესურსის ძირითად შემცველობასთან ერთად ზოგ დამატებით ინფორმაციისთან გაცნობასაც უზრუველყოფს.

თუ რესურსს ინტერნეტში URI გააჩნია, როგორც წესი, მასთან მიდგომა გარანტირებულია, გამონაკლის შემთხვევებში კი ვღებულობთ შეტყობინებას ნაცნობი 404 კოდით.

რაც მთავარია, URI უფრო ზოგადი ცნებაა, ვიდრე URL, რადგანაც შესაძლებელია იგი რამდენიმე ფაილსაც მოიცავდეს ან იყოს, ფაქტობრივად, ნებისმიერი დოკუმენტის (მაგალითად, გრაფიკულის, მუსიკალურის) ან მისი ნაწილის იდენტიფიკატორი, ასევე – მონაცემთა ბაზისადმი წაყენებული მოთხოვნის შედეგად განხორციელებული ძიების შედეგიც.

URI-ში მოწოდებულ ინფორმაციას, რომელიც ხშირად შეიცავს წყვილებს: *პარამეტრი/მნიშვნელობა*, ამუშავებს HTTP ოქმი.

ლიტერატურა

1. გ. ღვინეფაძე. WEB-დაპროგრამება - Javascript. *"ტექნიკური უნივერსიტეტი".* 2009 წ. ISBN 99940-14-80-3.

2. . JavaScript, 2-е изд. — СПб.: Питер, 2005. 395 с. ISBN 5-469-00804-5.

3. გ. ღვინეფაძე. WEB-დაპროგრამება - PHP. *"ტექნიკური უნივერსიტეტი".* **2009წ.** ISBN 978-9941-14-447-9.

4. JavaScript 24 . , « » , 2002.

5. . - , . - , JavaScript 1.5, , « » , 2007.

6. გ. ღვინეფაძე. WEB-დაპროგრამება WEB 2.0, XML, AJAX. *"ტექნიკური უნივერსიტეტი".* **2013 წ.**

7. http://www.intuit.ru/

8. http://www.wisdomweb.ru/AJAX/json.php

9. http://learn.javascript.ru/json

10. http://www.webmasterwiki.ru/jQuery

11. WEB-ტექნოლოგიების სტანდარტების საიტი http://www.w3schools.com

შინაარსი

- **JavaScript**. შესავალი -- 3
- ჩვენი პირველი სცენარები --- 4
- მივცეთ Web-ფურცელს უფრო მიმზიდველი სახე! ---------------- 11
- ფუნქციები და ობიექტები -- 13
- ხდომილობები და მათი დამუშავება -------------------------------- 22
- JavaScript-ზე დაპროგრამების ძირითადი საშუალებები ---------- 24
- ცვლადები -- 24
- მონაცემთა ტიპები JavaScript-ში ---------------------------------- 27
- მასივები --- 37
- პირობითი ოპერატორები -- 41
- ციკლები --- 46
- ობიექტები --- 50
- ბროუზერის ობიექტების მოდელი --------------------------------- 57
- მომხმარებლის ობიექტები -- 63
- ჩაშენებული ობიექტების გაწყობა ----------------------------------- 66
- კვლავ ხდომილობების შესახებ. თაგვთან დაკავშირებული და სხვა ხდომილობების დამმუშავებელნი ---------------------------- 70
- ამოცანების ნიმუშები --- 77
- **JSON** ფორმატი, **toJSON** მეთოდი ------------------------------ 90
- `JSON.parse` მეთოდი -- 90
- `JSON.stringify` მეთოდი, სერიალიზაცია -------------------- 94
- თვისებების გამორიცხვა --- 96
- გავხადოთ დაფორმატება უფრო მიმზიდველი სახის! ------------- 98
- **AJAX** ტექნოლოგია --- 101

- AJAX-ის გამოყენების მაგალითი ------------------------------------ 102
- AJAX-ის სერვერული ფურცლები ASP-ისა და PHP-ისთვის ----- 108
- AJAX-ის მონაცემთა ბაზასთან დაკავშირების მაგალითი --------- 112
- AJAX-ის მეშვეობით XML ფაილიდან მონაცემების ამორჩევა --- 116
- AJAX-ისთვის XMLHttpRequest ობიექტის გამოყენება -------------- 119

- **jQuery**. შესავალი -- 123
- ელემენტების ამორჩევები --- 128
- მოქმედებები ელემენტებზე -- 133
- ხდომილობებზე ელემენტების რეაქცია ------------------------------ 142

- **დანართები** -- 156
- **ლიტერატურა** --- 161