ბ. ღვინეფაძე

WEB-დაპროგრამება

JQuery

“ტექნიკური უნივერსიტეტი”

საქართველოს ტექნიკური უნივერსიტეტი

გ. ღვინეფაძე

WEB-დაპროგრამება

JQuery

რეკომენდებულია სტუ-ს
სარედაქციო-საგამომცემლო
საბჭოს მიერ.
03.04.2013, ოქმი № 2

თბილისი
2013

შაკ 681.3.06

წიგნი წარმოადგენს პრაქტიკულ სახელმძღვანელოს JavaScript ენისათვის განკუთვნილი jQuery ბიბლიოთეკის შესაძლებლობების შესასწავლად.

იგი განკუთვნილია ინფორმატიკის 22.02, 22.01 და 0719.08 სპეციალობათა შემსწავლელი სტუდენტებისა და ამ საკითხით დაინტერესებული პირებისთვის.

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

ISBN 978-9941-20-228-5

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

jQuery

შესავალი

jQuery წარმოადგენს მისაერთებელ ბიბლიოთეკას JavaScript-ისათვის. იგი მნიშვნელოვნად აადვილებს JavaScript-სა და HTML-ს შორის კავშირს. jQuery ასევე მოიხსენიება როგორც JavaScript ენის ბაზაზე შექმნილი ფრეიმვორკიც *(იხ. დანართი №1).*

jQuery-ის შექმნის საჭიროება განაპირობა შემდეგმა გარემოებებმა:

- ცნობილია, რომ სხვადასხვა ბროუზერი JavaScript-ზე დაწერილ სცენარებს არცთუ იშვიათად სხვადასხვაგვარად ასრულებს, მაგალითად, განსხვავებულია დოკუმენტის ობიექტურ მოდელთან (DOM) მათი მუშაობის წესები, რაც, ცხადია გარკვეულ პრობლემებს წარმოშობს.
- თვით JavaScript ენასაც საკმაოდ ბევრი ხარვეზი აქვს. მაგალითად, იგი ობიექტთან (ტეგთან) მისადგომად იყენებს ამორჩევის მხოლოდ ორ საშუალებას:
    - ტეგის სახელწოდებას;
    - ტეგის იდენტიფიკატორს.

  ამ დროს გამოუყენებელი რჩება დღეს ძალიან ფართოდ გავრცელებული **სტილის კასკადური ცხრილების (CSS),** ანუ WEB ფურცლის ცალკეული უბნებისათვის საკუთარი სტილის მაფორმირებელი ცხრილების მეშვეობით კვალიფიცირებული *ტეგების კლასებისადმი* მიდგომის შესაძლებლობა. jQuery კი საშუალებას იძლევა CSS სელექტორების მეშვეობით მარტივად იქნეს ამორჩეული როგორც ცალკეული ტეგები, ისე მათი კლასები. ამასთან ერთად, დასაშვებია ამ პროცესში სხვადასხვა კრიტერიუმის გამოყენებაც.
- წლების განმავლობაში JavaScript-ზე დაწერილი სცენარების გაანალიზებიდან ნათელი გახდა, რომ საკმაოდ ხშირად საქმე გვაქვს სტანდარტული ამოცანების გადაწყვეტასთან, რომლების-

თვისაც აზრი აქვს ცალკე მოდულების შექმნას. მაგალითად, jQuery-ში გათვალისწინებული ავტომატური ციკლების მეშვეობით ადვილად ხდება მასივებში ელემენტების დათვალიერება.

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

- jQuery-ის შესაძლებლობები უფასოდ შეიძლება გამოვიყენოთ. მის ჩამოსატვირთად უნდა მივმართოთ jquery.com საიტს.
- ფაილის ზომა მცირეა – იგი რამდენიმე ათეულ კილობაიტს არ აღემატება (შეკუმშული სახით 30 კბ-ზე ცოტა მეტია).
- ინტერნეტში შესაძლებელია მოვიძიოთ jQuery-ის შესაძლებლობების გამოყენების ძალიან ბევრი მაგალითი.
- jQuery ბიბლიოთეკა მოიცავს მრავალ პლაგინს.
- იგი, ფაქტობრივად, მუდმივად განახლების პროცესშია.
- jQuery-ის გამოყენებისათვის საჭირო არ გახლავთ რაიმე განსაკუთრებული მომზადება, ახლის დაუფლება – საკმარისია ვერკვეოდეთ CSS-სა და Java Script-ის შესაძლებლობებში.

დასასრულ, 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 განახლებისას ფურცლის მთლიანი გადატვირთვა არ ხდება, რაც მნიშვნელოვნად ამაღლებს ფაილთან მუშაობის სისწრაფესა და უფრო მოხერხებულს ხდის მას.

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

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

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

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

ესენი წარმოადგენენ JavaScript-პლაგინებსა და Ajax-დამატებებს, რომლებიც უზრუნველყოფენ ხდომილობების დამუშავებას, ვიზუალურ ეფექტებს, აგრეთვე "მოგზაურობას" 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 ასევე იქცევა 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 გახლავთ jQuery ბიბლიოთეკის ორგანიზების კონცეფცია. ეს ბიბლიოთეკა წარმოგვიდგება კომპაქტური უნივერსალური

ბირთვისა და პლაგინების ერთობლიობად, რის შედეგადაც 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 მდგარი 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 სახელწოდების მქონე საქაღალდეში. რაც შეეხება თვითონ jQuery ბიბლიოთეკის შემცველ ფაილს, უმჯობესია იგი Google-იდან გადმოვწეროთ. ასეთ შემთხვევაში საკმაოდ სწრაფად მოვიპოვებთ gzip ფორმატში მჭიდროდ დაარქივებულ, კომპაქტური ზომის 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 კლიენტისათვის ინფორმაციის დროებით შენახვისათვის განკუთვნილ საცავში – კეშში. ასეთ შემთხვევაში იგი ფაქტობრივად მომენტალურად ჩამოიტვირთება. 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.)
- გასათვალისწინებელია 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 განმეორებითი შეკუმშვით 26 კილობაიტამდე იქნეს დაყვანილი.

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

- ავირჩიოთ საჭირო ელემენტები;
- დავაკავშიროთ მათთან ხდომილებები;
- განვახორციელოთ მათთზე 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. ელემენტის ამორჩევა

1.1. ელემენტის 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 ჯგუფის აღმწერ ინფორმაციას და ამ ფრჩხილების წინ რაიმე ელემენტის, კლასის ან იდენტიფიკატორის სახელწოდებას. ამასთან, თუ თვით ამ სახელწოდების წინ დასმულია წერტილი, საქმე გვაქვს კლასთან, გისოსის (#) შემთხვევაში – იდენტიფიკატორთან, ხოლო იმ შემთხვევაში, თუ სახელწოდების წინ არანაირი სიმბოლო არ ფიგურირებს, მაშინ – ტეგთან.

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

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

განვიხილოთ თითოეული მათგანი:

ა) ელემენტის ამორჩევა სახელწოდების 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

ელემენტის ამორჩევა იდენტიფიკატორის მიხედვით jQuery-შიც ხდება CSS-სათვის მიღებული წესით. აქაც საძებნი ელემენტების იდენტიფიკატორის სახელწოდების წინ ფიგურირებს # სიმბოლო. მაგალითად, jQuery-ში ფორმირებული ნოტაცია:

```
$('#vardi');
```

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

გ) ელემენტის ამორჩევა კლასის მიხედვით

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

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

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

ამრიგად, jQuery იყენებს ელემენტების ამორჩევის 3 ძირითად წესს:

```
$('p');          // ტეგის სახელწოდების მიხედვით
$('#element');   // იდენტიფიკატორის სახელწოდების მიხედვით
$('.greened');   // კლასის სახელწოდების მიხედვით
```

1.2. ელემენტის 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 შემოთავაზებულია შემდეგი კონსტრუქცია:

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

გ) შვილობილი ტეგი

შვილობილი ტეგის ამოსარჩევად კი შესაძლებელია ასეთი მიდგომის გამოყენება:

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

1.3. ჩადგმული ელემენტები

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

ძალიან ხშირად საჭირო არის ისეთი ელემენტის ამორჩევა, რომელშიც ფიგურირებს მოცემული მნიშვნელობის 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 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 სახელწოდება (ან მისამართი) იწყება სიტყვა photo-თი, მაგალითად, – “photo007.jpg”.

გ) ელემენტის ამორჩევა მისი ატრიბუტის მნიშვნელობის დაბოლოების მიხედვით

ელემენტები ამოირჩევა წყაროს სახელწოდების დაბოლოების მიხედვით. მაგალითად, ოპერატორი:

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

ამოარჩევს ყველა იმ ნახატ-ელემენტს, რომელთა წყაროს სახელწოდება მთავრდება სიტყვა 001.jpg-თი, მაგალითად, – “photo001.jpg”-ს.

დ) ამორჩევა ატრიბუტის მნიშვნელობაში ყოფნა-არყოფნის მიხედვით

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

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

ამოარჩევს ყველა იმ ნახატს, რომელთა წყაროს სახელწოდებაში (მისამართში) ფიგურირებს სტრიქონი 001.

1.4. ამორჩევის შედეგების ფილტრაცია

ა) ფილტრაცია ჩამონათვალში ნომრის ლუწობა-კენტობის შემოწმების მიხედვით

მოგვყავს მაგალითი კოდში შეტანილი ნახატების “პირველ-მეორეზე გათვლისა”:

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

ბ) ყველა, 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

კრიტერიუმის მიხედვით კვალიფიცირებული, კოდში არსებული ელემენტებიდან აირჩევა პირველი (ან ბოლო) ელემენტი:

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

ვ) ხილული/დამალული ელემენტის 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.

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

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

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

გამოვიტანოთ ეს ტექსტი ეკრანზე:

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

არსებული ტექსტის 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 (n მილიწამი);

- იმ ფუნქციის სახელწოდება, 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 გამოსატანად კი ვიყენებთ ანალოგიური პარამეტრების მქონე show() მეთოდს:

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

2.3. ჯაჭვური ფუნქციები

ჯაჭვური ფუნქციების მეშვეობით 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 იმ შემთხვევებში, როდესაც ერთდროულად არის საჭირო რამდენიმე ელემენტის დამალვა-გამოყვანა?

jQuery დასახული მიზნის რეალიზაციას უზრუნველყოფს ციკლების გამოყენების გარეშე – საჭიროა მხოლოდ ასეთი ელემენტების ამორჩევა

განვახორციელოთ იდენტიფიკატორის მეშვეობით 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 ვაძლევთ სასურველ მნიშვნელობებს:

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

დასასრულ, აქაც შესაძლებელია აღნიშნული ქმედებების ერთ ჯაჭვში მოქცევა:

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

2.6. ელემენტისათვის 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>თქვენ წინაშეა სქელშრიფტიანი აბზაცი</strong></p>
```

text() ფუნქციის გამოყენებით ჩვენს განკარგულებაში გადმოდის მხოლოდ ტექსტი "თქვენ წინაშეა სქელშრიფტიანი აბზაცი", მაშინ როდესაც HTML() ფუნქციის მეშვეობით:

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

ამორჩეული იქნება შემდეგი კოდის ფრაგმენტი:

" **თქვენს წინაშეა სქელშრიფტიანი აბზაცი "**

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

2.7. ელემენტის მდოვრე გაქრობა-გამოჩენა

ზემოთ განხილული hide() და show() ფუნქციები ელემენტის გაქრობა-გამოტანას ახდენდნენ ყოველგვარი ვიზუალური ეფექტების გარეშე, fadeOut() და fadeIn() ფუნქციებისაგან განსხვავებით. 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 დანიშნულებაა 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-გამოტანისას პროცესებისათვის მიმართულების მითითებაც – გაქრობა ქვემოდან ზემოთკენ, ხოლო გამოტანა, ცხადია, – საპირისპირო მიმართულებით:

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

2.8. ელემენტის 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/გამოკლება

დავუშვათ, მოცემული საიტისათვის CSS-ში განსაზღვრულია ასეთი კლასი:

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

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

დავუშვათ, ჩვენს განკარგულებაშია შემდეგი აბზაცი:

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

მოვიყვანოთ მისი კლასში ჩარიცხვა-გამოკლების 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 შერჩეული 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, ფერის განმსაზღვრელ ატრიბუტს), საჭიროა, CSS ფუნქციის პარამეტრებად ვუჩვენოთ ატრიბუტის სახელწოდება და მძიმით გამოყოფილი მისი მნიშვნელობა:

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

ახლა კი მოვიყვანოთ რამდენიმე ატრიბუტისათვის ჯაჭვური წესით მნიშვნელობების შეცვლის მაგალითი:

```
$(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'
  });
});
```

თუ გვსურს, ესა თუ ის ქმედება დროში გავწელოთ, ამ შემთხვევაშიც 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 არის შესაძლებელია:

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

ხოლო მის უკან – ამგვარი წესით:

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

2.12. ელემენტების ციკლში შემოწმება

ჯერ მოვიყვანოთ კოდის შესაბამისი ფრაგმენტი:

```
// სიის პუნქტების შემცველობა ციკლში გამოვიტანოთ 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 გამოტანა. clone() ფუნქციით ვახდენთ მის კლონირებას და ვიმახსოვრებთ რომელიმე 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 ელემენტების ამორჩევა

არცთუ იშვიათად მოითხოვება, ამორჩეული 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 ამორჩევა
  s.hide();                        // მათი დამალვა
});
```

3. ელემენტების რეაქცია ხდომილობებზე

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 ელემენტს.);
});
// 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 ელემენტს. '+
     '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 );
 });
```

ვ) .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 ღილაკზე. '+
     '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. ქვემოთ მოგვყავს 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 ხდომილობის
  // 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  დამმუშავებლის კოდი
$(window).resize(function(){
   // ბროუზერის ფანჯრის ზომების ცვლილების ხდომილობის
   // დამმუშავებლის კოდის მაგალითი
  alert('ბროუზერის ფანჯრის ზომები შეიცვალა.');
});
// resize ხდომილობის დამმუშავებლის გამოძახება
```

```
$(window).resize();
```

გ) .scroll()

გამოყენების მაგალითები:

```
// foo-თი იდენტიფიცირებულ ელემენტზე ვაყენებთ .scroll()
// ხდომილობის დამმუშავებელს
$('#foo').scroll(function(){
  alert('' ჩატარდა foo ელემენტის სკროლინგი.');
});

// foo-ისთვის scroll ხდომილობის დამმუშავებლის 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 ფუნქციის ქვეპროგრამების ერთი სახურავის ქვეშ განთავსებას, გადავდივართ ახალ ეტაპზე – პროგრამული სისტემის კარკასში უკვე მოქცეულია ბიბლიოთეკა საკუთარი კოდისათვის, აგრეთვე რიგი სხვადასხვა დანიშნულების ბიბლიოთეკისა, 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.

ფრეიმვორკული ტიპის პროგრამული სისტემის მაგალითად შეიძლება დავასახელოთ ვებ-პროგრამირებაში კარგად ცნობილი კონტენტის მართვის სისტემები (CMS), ხოლო შესაბამისი სახის დანართების 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 დანართში (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 მრავალი API ბიბლიოთეკა მომხმარებლის უზრუნველსაყოფად ფუნქციებითა და კლასებით. მათში აღწერილია ფუნქციების სიგნატურა და სემანტიკა.

დანართი № 3

პლაგინი

პლაგინი (plug-in) წარმოადგენს სხვა პროგრამებისაგან დამოუკიდებლად კომპილირებად პროგრამულ მოდულს, რომელიც შეიძლება დინამიკურად მიუერთდეს ამა თუ იმ პროგრამას მისი შესრულების დროს. 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 ასლი რამდენიმე პროგრამას შეუძლია გამოიყენოს. ამ შემთხვევაში პლაგინი ასრულებს ე.წ. *გაყოფადი ბიბლიოთეკის* როლს. ასეთი ბიბლიოთეკების დიდი უპირატესობაა 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).

დასასრულ, 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, სასურველია, დაიმალოს URI, ხოლო მომხმარებლებისთვის სასურველი სახის მისამართის ფორმირება კომპიუტერს დაეკისროს. ამათან, შესაძლებელია ერთსა და იმავე რესურსს სხვადასხვა URI-თაც მივადგეთ. უფრო გრძელი მისამართი ხშირ შემთხვევაში აადვილებს რესურსთან შეღწევადობას და/ან რესურსის ძირითად შემცველობასთან ერთად ზოგი დამატებითი ინფორმაციის გაცნობასაც უზრუველყოფს.

თუ რესურსს ინტერნეტში URI გააჩნია, როგორც წესი, მასთან მიდგომა გარანტირებულია, გამონაკლის შემთხვევებში კი ვიღებთ შეტყობინებას ნაცნობი 404 კოდით.

რაც მთავარია, URI უფრო ზოგადი ცნებაა, ვიდრე URL, რადგან 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. http://www.webmasterwiki.ru/jQuery
2. WEB-ტექნოლოგიების სტანდარტების საიტი http://www.w3schools.com
3. გ. ღვინეფაძე. WEB-დაპროგრამება. Javascript. სახელმძღვანელო. თბილისი. "ტექნიკური უნივერსიტეტი". 2009. ISBN 99940-14-80-3. http://gtu.ge/books.php/ 681.3(06) /203
4. Освой самостоятельно JavaScript за 24 часа. Майкл Монкур, «Вильямс» , 2002.

შინაარსი