|  |
| --- |
| მიმდინარე პარაგრაფის თემატიკა   * თანამედროვე ბიბლიოთეკები * ბიბლიოთეკების გამოყენების მიზანი * შესატყვისი რედაქტირება და მათი დადებითი მხარეები * კოდის სტრუქტურა |

### 4.2 მოძიებული ბიბლიოთეკის ინტეგრაცია ვებგვერდთან

|  |
| --- |
| მიმდინარე პარაგრაფის თემატიკა   * ბიბლიოთეკის დამაკავშირებელი მისამართები * ბიბლიოთეკასთან თანდართული ფაილები * html ელემენტების მზა ჩონჩხი * სტილური ფაილი |

### 4.3 მზა კოდში ცვლილებების შეტანა

|  |
| --- |
| მიმდინარე პარაგრაფის თემატიკა   * ტექსტის ცვლილება * სურათის პარამეტრები * ბმულის მისამართები * ელემენტების ზომის პარამეტრები * ეფექტების სიჩქარე * ეფექტის ტრაექტორია |

## თავი 5. საიტის ინტერაქტივისა და ეფექტების შემუშავება - JavaScript

Javascript მსოფლიოს ყურადღების ცენტრში 1995 წლიდან მოექცა. მისი მეშვეობით შესაძლებელი იყო

ვებ გვერდზე პროგრამული ნაწილის განთავსება. საწყის ეტაპზე იგი აღიქმებოდა მხოლოდ ბრაუზერ

Netscape Navigator\_ში.

Javascript ენაზე მოთხოვნის სწრაფმა ზრდამ გამოიწვია უმოკლეს დროში მისი ინტეგრაცია წამყვან ბრაუზერებში. ამ პროგრამირების ენამ საშუალება მისცა დეველოპერებს შეექმნათ ვებ საიტებისთვის სხვადასხვა პროგრამული დანამატები (მაგ. რუკები), მაგრამ ძირითადად იგი გამოიყენებოდა ვებ გვერდზე ინტერაქციისა და სხვადასხვა ეფექტების შესამუშავებლად.

გასათვალისწინებელია, რომ Javascript პროგრამირების ენას არავითარი საერთო ( წარმოების თვალსაზრისით ) არ გააჩნია JAVA პროგრამირებასთან. ხშირ შემთხვევაში იგი აღიქმება ხან Java პროგრამირების შვილობილ ენად, ხანაც უშუალოდ java\_დ.დაუშვებელია Javascript\_ის java\_დ მოხსენიება. ეს ორი ერთმანეთისაგან განსხვავებული პროგრამირების ენებია. უნდა აღინიშოს, რომ მათ შეუძლიათ თანადროული მუშაობა.

მას შემდეგ, რაც არსებული პროგრამირების ენის მოხმარება გასცდა Netscape Navigator და გახდა უფრო ფართოდ გამოსაყენებელი ორგანიზაცია ECMA\_ს მიერ შემუშავებულ იქნა სტანდარტი ECMAScript - დოკუმენტი, რომელშიც აღწერილი იყო ენის მუშაობის პრინციპები.

საწყის პერიოდში, როდესაც ECMAScript სტანდარტი იწყებდა განვითარებას მსოფლიოში დომინირებდა პროგრამირების ენის Java. Java იმდენად პოპულარული გახლდათ - მიიჩნეოდა, რომ მომავალში ყველა პროგრამიერბის ენა ჩაიყლაპებოდა ამ გიგანტის მიერ. სწორედ არსებული სიტუაციიდან გამომდინარე ორგანიზაცია ECMA\_მ განახორციელა გამართლებული მარკეტინგული სვლა და პროდუქტს უწოდა Javascript\_ი ( თავდაპირველი სახელწოდება არსებული LiveScript). ამ პერიოდიდან დოკუმენტაცია ECMAScript მოიხსენიება Javascript\_ად. არსებულ ენას ECMAScript\_ის სტანდარტი გააჩნია დღესაც უბრალოდ ფართო მასებისთვის ის რჩება Javascript ენად.

პროგრამების შესასრულებლად არა აქვს მნიშვნელობა, თუ რომელ პროგრამირების ენაზე ისინი დაწერილი, გამოიყენება 2 მეთოდი : „კომპილაცია“და „ინტერპრეტაცია“.

* კომპილაცია - ეს არის შემთხვევა როცა პროგრამის კოდს იღებას სპეციალური პროგრამა „კომპილატორი“ , გარდაქმნის სხვა ენად, როგორც წესი მანქანურ ენად (ნულებისა და ერთების ენად 0-1) . ეს კომპილირებული კოდი დამოუკიდებლად Source (საწყისი) კოდისგან ეშვება პროგრამის სახით.

* ინტერპრეტაცია - ეს არის შემთხვევა, როცა პროგრამის კოდს იღებას ინსტრუმენტი, რომელიც მოიხსენიება „ინტერპრეტატორად“ და ასრულებს ისე როგორი სახითად მიუვიდა სკრიპტი. Javascript სწორეად ასეთი ტიპის პროგრამირების ენაა. ეს მიდგომა გამოიყენება ბრაუზერებში javascript\_ის კოდი მიღებისას.

თანამედროვე ინტერპრეტატორები გარდაქმნისა და უშუალოდ გაშვების პორეცსშიც ახდენენ javascript კოდის ოპტიმიზაციას, ამიტომ Javascript მუშაობს ძალიან სწრაფად.

დღეს ფაქტობრივად ყველა ბრაუზერში ჩაშენებულია Javascript ინტერპრეტატორი, ამიტომ მისი შესაძლებელია მისი აქტიური გამოყენება ვებ გვერდებზე. რა თქმა უნდა შესაძლებელია Javascript გამოყენება ბრაუზერის გარეშეც.

რა შეუძლია Javascript\_ს ?

Javascript\_ის შესაძლებობლები დამოკიდებულია გარემოზე რომელშიც უწევს ფუნქციონირება. მაგ.: ბრაუზერში მას შეუძლია ყველაფერი რაც შეეხება ვებ გვერდის მანიპულაციას, მომხმარებელს, ურთიერთობას სერვერთან გარკვეული დოზით:

* ახალი HTML ტეგების შექმნა, არსებულის წაშლა, ტეგებისათვის სტილების გაწერა, მათი დამალვა/გამოჩენა;
* მომხმარებლების მოქმედებებზე რეაგირება, მაუსისა და კლავიატურის ხდომილებებზე (event) კონრეტული ფუნქციონალის მინიჭება;
* სერვერზე მოთხოვნების გაგზავნა, ინფორმაციის ჩატვირთვა ვებ გვერდის განახლების გარეშე

(ajax ტექნოლოგია);

* Cookie\_თან ურთიერთობა;

ეს მხოლოდ ნაწილია იმ შესაძლებლობისა, რომელსაც ფლობს პროგრამირების ენა Javascript.

მიმდინარე მოდულის ფარგლებში ჩვენ შევეხებით javascript პროგრამირებას, რომელიც ორიენტირებულია მომხარებლის მხარეზე, შესაბამისად მისი მოქმედების გარემო იქნება ბრაუზერი.

### 5.1 მარტივი ამოცანის გადაწყვეტა JavaScript ენის ძირითადი ელემენტების გამოყენებით

|  |
| --- |
| მიმდინარე პარაგრაფის თემატიკა   * დეკლარაციების გამოცხადება * არითმეტიკული, ლოგიკური, ბინარული, სტრიქონული და პირობითი ოპერატორები * შედეგის გამოტანის ოპერატორები |

როგორც შესავალში აღვნიშნეთ javascript ფართო შესაძლებლობების მქონე პროგრამირების ენაა. ჩვენს შემთხვევაში ორიენტირებულნი ვიქნებით javascript\_ის გამოყენებაზე ვებ გვერდში, შესაბამისად სამუშაო გარემო html დოკუმენტთან უშუალო კავშირში იქნება.

javascript კოდის არეალის გამოყოფა html დოკუმენტის სხვადასხვა ადგილასაა შესაძლებელი. გამომდინარე დავალების ინტერესებიდან ის შეიძლება გამოძახებულ იქნას:

* <head> JS </head>ტეგში;
* <body> JS </body>ტეგში;
* შესაძლებელია პროგრამული კოდის სხვა დოკუმენტში ფორმირება და შემდგომში მისი იმპორტორება html დოკუმენტში;

თითოეულ ამ ხერხს დაწვრილებით გზადაგზა გავარჩევთ. განვიხილავთ თუ რა დატვირთვის მატარებელია ზემოთ ჩამოთვლილი გზებით სკრიპტის ასახვა. მაგრამ საწყის ეტაპზე შევეცადოთ მოვამზადოთ Javascript არეალი <body>ტეგში. ამისათვის საჭიროა გავხსნათ და დავხუროთ <script> … </script>ტეგი. არსებულ ტეგში განთავსებული ინფორმაცია წარმოადგენს პროგრამულ კოდს.

|  |
| --- |
| <!DOCTYPE HTML>  <html> <head>  <meta charset="utf-8">  </head>  <body>    <script>  .......................................  </script>    </body>  </html> |

მიმდინარე ეტაპზე დასაშვებია <script>ტეგის უატრიბუტოდ გამოცხადება.

შესაძლებელია <script type="text/javascript"> გამოყნებაც, რომელიც არც თუ ისე შორეულ წარსულში აქტიურად გამოიყენებოდა. დეველოპერთა დიდი ნაწილი არსებულ ჩანაწერს დღესაც ინტენსიურად იყენებს.

შესაძლოა რომელიმე საიტთან შეხებისას <script language="JAVASCRIPT">გახსნის შემდეგნაირი ჩანაწერი აღმოაჩინოთ. ის საკმაოდ ძველმოდურია და ფაქტობრივად აღარსად გამოიყენება. თუ მეორე ჩანაწერზე აღვნიშნეთ რომ ჯერ კიდევ აქტიურად მოიხმარება, ამ ტიპის ჩანაწერი არც თუ ისე დადებითად დაგახასიათებთ - ის არავალიდურია.

შესაძლოა Javascript\_ით გაფორმებულ დოკუმენტში თვალი მოკრათ მსგავსი ტიპის ჩანაწერს

<script type="text/javascript"><!--

...

//--></script>

სადაც <script>გამხსნელ - დამხურავ ტეგებს შორის მოთავსებულია html კომენტარები. ეს მეთოდი გამოყენებოდა იმისათვის, რომ თუ ბრაუზერს არ ექნებოდა Javascript\_ის მხარდაჭერა მასში არსებული კოდი არ გამოისახებოდა ეკრანზე ის ავტომატურად გადაიქცეოდა კომენტარად.

შესაბამსად დღევანდელ გარემოში არსებულ ყველა ბრაუზერს აქვს Javascript\_ის მხარდაჭერა და მიმდინარე კოდის გამოყენება აღარაა აქტუალური.

javascript\_ში ინფორმაციის გამოტანის რამოდენი მემეთოდი არსებობს. მაგ.:

|  |  |
| --- | --- |
| • | alert('შეტყობინება 1')- ინფორმაციის გამოტანადი ალოგური ფანჯარის მეშვეობით; |

* document.write('შეტყობინება 2') - ინფორმაციის გამოტანა ვებ გვერდის შემცველობაში;
* console.log('შეტყობინება 3') - ინფორმაციის გამოტანა კონსოლში;

სანამ უშუალოდ სათითაო მეთოდს გავარჩევთ ზოგადად ავხსნათ მეთოდის არსი. მეთოდი იგივე ფუნქიციაა. მისი მოხსენიება ორივენაირად შეიძლება. ჩემს მიერ გამოყენებული იქნება ორივე მიმართება და იმედია ეს არ გამოიწვევს რაიმე გაურკვევლობას.

პროგრამირებაში ამომავალი წერტილი ობიექტია. ნებისმიერი მოქმედება დაკავშირებულია კონკრეტულ ობიექტთან. ობიექტებზე ვრცლად შემდეგ ქვეთავებში ვისაუბრებთ ახლა მხოლოდ მცირეოდენი რაკურსით შემოვიფარგლოთ, რათა გაგიადვილდეთ ფუნქციის/მეთოდის არსის გაგება და არსებული ჩანაწერების დაზეპირება გაუაზრებლად არ მოახდინოთ.

მაგ.: გვაქვს ობიეტი ტელეფონი. მას გააჩნია თვისებები და მეთოდები. თვისებებში შეიძლება ვიგულისმოთ წონა, ფერი, მეხსიერების ზომა და ა.შ. აი მეთოდები წარმოადგენს მის შესაძლებლობებს. ვთქვათ ტელეფონის მეშვეობით შესაძლებელია ფოტოსურათის გადაღება, შეტყობინების გაგზავნა.

რომ შევაჯამოდ ეს არც თუ ისე ვრცელი აბზაცი მივიღებთ შემდეგს:

* ობიექტის თვისების გასაგებად თქვენ სთხოვთ ობიექტს გამოგიტანოს/გიჩვენოს უკვე არსებული თვისება;მაგ.: გამოსახოს ეკრანზე ტელეფონის წონა.
* ობიექტის მეთოდის გამოყენებით შენ ზემოქმედებ ამ კონკრეტულ ობიეტზე და ახდენ რაიმე ტიპის მანიპულაციას; მაგ.: გაგზავნოს შეტყობინება თქვენს მიერ აკრეფილი ტექსტით.

შესაძლოა ეს ყოველივე ამ ეტაპზე სრულად გასაგები არ იყოს, ასევე არ წარმოადგენდეს იდეალური ახსნა ობიექტის თვისებასა და მეთოდს შორის განსხვავებისა, მაგრამ იქება მცირეოდენი ბიძგი მომავალში ყველაფრის უკეთ აღსაქმელად.

თვისებასა და მეთოდს გააჩნია ერთი ფუნდამენტალური პრინციპი ორივე მიეკუთვნება კონკრეტულ ობიექტს. კუთვნილება გამოისახება - წერიტილით „ . “ ;

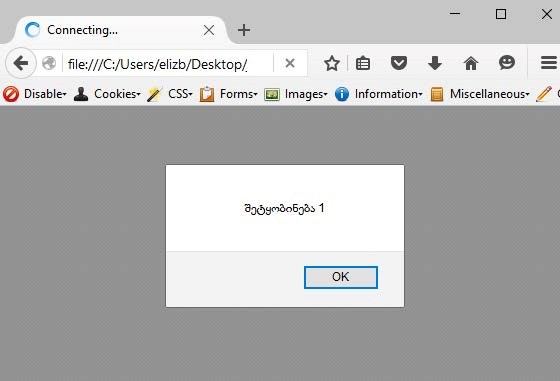
ასევე ყველა მეთოდი გამოსახება ფრჩხილებით-( ); ამ ეტაპზე არ ვეხებით თვისებებს და მათ გამოსახვას.

ჩვენს მიერ ჩამოთვლილ გამოტანის ოპერატორებზე ყურადღების გამახვილება კიდევ ერთხელ შეიძლება - document.write(), console.log() , alert() ; როგორც თეორიული მასალიდან გავიგეთ პირობა:

* „ყველა მეთოდი აისახება ფრჩხილებით“ - ჭეშმარიტია.
* „მეთოდი მიეკუთვნება ობიექტს და გამოიყოფა წერტილით“ - პირველ ორ შემთხვევაში პირნათლად სრულდება. ანუ:
  + - document - წარმოადგენს ობიექტს და შესაბამისად write() მისი ერითი-ერთი რიგირთი მეთოდია - რომელიც პასუხისმგებელია ინფორმაციის გამოტანაზე ვებ გვერდის ტანში;
    - console -ასევე წარმოადგენს ობიექტს და მისიერთ-ერთიმეთოდია log()- ინფორმაციის გამოტანა კონსოლში;
    - რაც შეეხება alert() მეთოდს ჩანაწერში არ ფიგურირებს ობიექტი და მასზე მიმაგრება წერტილის მეშვეობით. alert() მეთოდის სრყლყოფილი ჩანაწერი შემდეგნაირია window.alert(); ობიექტ window\_ს მომავალში დაწვრილებით შევეხებით ამ ეტაპზე არსებული ჩანაწერით წარმოგიდგინეთ, რომ alert() არაფრით განსხვავდება ზემოხსენებული მეთოდებისაგან და ისიც ობიექტის მეთოდია უბრალოდ მასი შემოკლებული ჩანაწერი დასაშვებია ( ამ დეტალსაც შემდგომში გავარჩევთ );

Javascript რეგისტრზე დამოკიდებული ენაა შესაბამისად Alert(), ALERT() და ა.შარსებული მეთოდების სხვადასხვაგვარი ინტერპრეტაციით დაწერა არ გამოირანს სასუველ შედეგს. არსებული მეთდები გამოიყენეთ მხოლოდ ასოთა იმ რეგისტრით, როგორიც მოცემულია მაგალითად.

|  |
| --- |
| <!DOCTYPE HTML>  <html> <head>  <meta charset="utf-8">  </head>  <body>    <script>  alert('შეტყობინება 1');  </script>    </body>  </html> |

დავუბრუნდეთ ჩვენს მიერ უკვე აღწერილ მეთოდ alert() \_ ს და ვნახოთ მისი გამოძახებით მიღებული შედეგი. ( იხ.სურ.5.1 )

გასათვალისწინებელია, რომ დიალოგური ფანჯარა alert() თანამედროვე ბრაუზერში გასხვავებული დიზაინით და მდებარეობით გამოიხატება.

5.1 alert() დიალოგური ფანჯარა

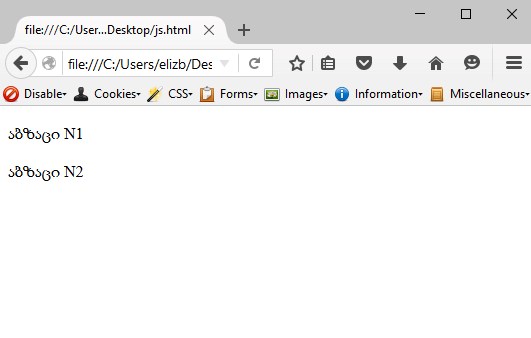
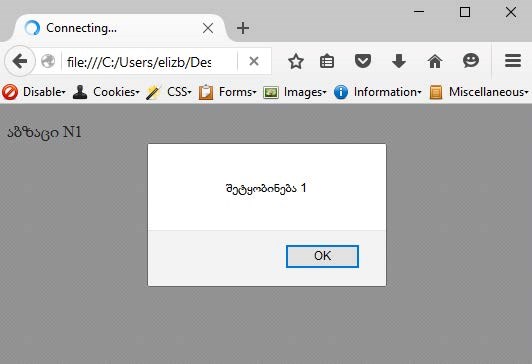
ყურადღება გაამახვილეთ alert() მეთოდში ჩაწერილ სიტყვაზე - მას ფუნქციის არგუმენტი ეწოდება. მიმდინარე ეტაპზე მისი დამახსოვრება დიდ მნიშვნელობას არ წარმოადგენს, რეალურ არსს ფუნქციების განხილვისას გაეცნობით.

როგორც ატყობთ ტექსტი ბრჭყალებშია მოთავსებული. დასაშვებია როგორც ერთმაგი ასევე ორმაგი ბრჭყალების გამოყენება. მაგ.: 'შეტყობინება 1' და "შეტყობინება 1" ორივე მართებული ჩანაწერია. თუ მხოლოდ რიცხვითი მნიშვნელობა გამოსატანი დასავებია ბრჭყალების გარეშე მითითება. მაგ: alert( 150 );

alert() წარმოადგენს მეთოდს, რომელიც იწვევს html კოდის ჩატვირთის შეყოვნებას.

|  |
| --- |
| <!DOCTYPE HTML>  <html> <head>  <meta charset="utf-8">  </head>  <body>  <p>აბზაცი N1 </p>  <script>  alert('შეტყობინება 1');  </script>  <p>აბზაცი N2</p>  </body>  </html> |

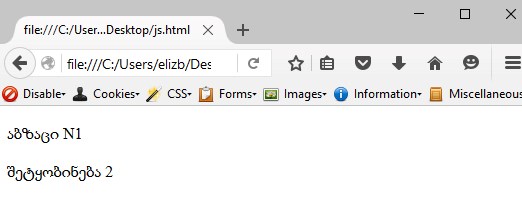
კოდის მართლზომიერი მუშაობის შედეგად მონიტორზე გამოისახე alert() დიალოგური ფანჯარის მეშვეობით 'შეტყობინება 1'. თუ უკან ფონს დაუკვირდებით ბრაუზერის <body>ტეგში გამოსახულია წარწერა „აბზაცი N1“ (იხ.სურ. 1.2.1),შესაბამისად „აბზაცი N2“ არსად ფიგურირებას. ასევე ბრაუზერის ფანჯრის ზედა title ველში ჩატვირთვის ნიშანი აქტიურია, რაც მიუთთებს რომ ვებ გვერდის ჩატვირთვა არ დასრულებულა. მხოლოდ alert() ფანჯრის „OK“ღილაკით დადასტურება გამოიწვევს დანარჩენი კოდის ჩატვირთვას შემდგომ alert() ფუნქციამდე, თუ იგი კიდევ სადმე იარსებებს მიმდინარე ვებ გვერდზე. (გამომდინარე იქიდან, რომ ჩვენს შემთხვევაში გამოყენებულია მხოლოდ ერთი alert() ვებ გვერდი ბოლომდე ჩაიტვირთება ). გამოჩნდება ტექსტი „აბზაცი N2“. (იხ.სურ. 1.2.2)



1.2.1 კოდის ჩატვირთვის წყვეტა alert() მეთოდის მიერ 1.2.2 alert() მეთოდის დასტურის შედეგად მიღებული შედეგი

alert() მეთოდისაგან განსხვავებით document.write('შეტყობინება 2') არ წარმოადგენს მცურავ ფანჯარას, მას გამოაქვს ინფორმაცია ბეჭდური სახით, იმ კონკრეტულ სტრიქონზე, რომელზეც ის არის განთავსებული.

|  |
| --- |
| <!DOCTYPE HTML>  <html> <head>  <meta charset="utf-8">  </head>  <body>  <p>აბზაცი N1 </p>  <script>  document.write('შეტყობინება 2');  </script>  </body>  </html> |

დავალების შესაბამისად „აბზაცი N1“ ფორმირდება <p></p>ტეგის მეშვეობით, ხოლო „შეტყობინება 2“\_ის გამოტანა ხორციელდება document.write() მეთოდის დახმარებით (იხ.სურ.1.3)

1.3document.write() ფუნქციის გამოტანილი „შეტყობინება 2“

Javascript სინტაქსი ხაზოვანი ტიპის კითხვადობის რეჟიმის მიმდევარია. თითოეული პროგრამული სტრიქონის ბოლოს რეკომედირებულია წერტილმძიმის - „ ; “ გამოყენება, რაც სტრიქონის დასასრულის აღმნიშვნელია. (სხვადასხვა პროოგრამირების ენაში - მაგ. php, c# „;“ გამოყენება სავალდებულოა ). სტრიქონში ედიტორის ერთი კონკრეტული ხაზზე განთავსებული ინფორმაცია არ მოაზრება, პროგრამირებაში სტრიქონად ერთი უწყვეტი კოდი იგულისხმება. იხილეთ მოცემული მაგალითი

1. <script>
2. alert('შეტყობინება 1');
3. document.write(' აქ განთავსდება ნებისმიერი სიგრძის ინფორმაციული ტექსტი და ის აღიქმება ერთ სტრიონად მიუხედავ იმისა ედიტორში რამოდენ ხაზზეა იგი განლაგებული');

4.</script>

რეკომენდირებულია ასევე სხვადასხვა აზრობრივი კოდის სხვადასხვა ხაზზე განთავსება. მიუხედავად იმისა, რომ მაგალითზე ნაჩვენები კოდი უპრობლემოდ იმუშავებს უმჯობესია alert() ფუნქციები განთავსდეს სხვადასხვა სტრიქონზე.

<script> alert('შეტყობინება 1'); alert('შეტყობინება 2');

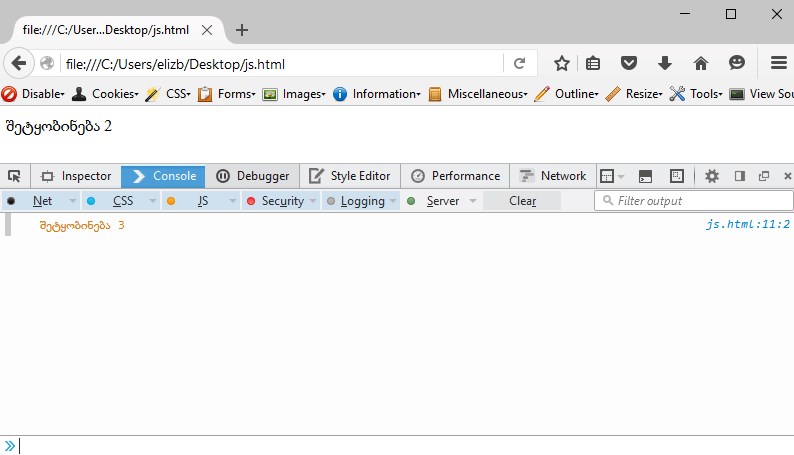
</script>

<script>

alert('შეტყობინება 1'); alert('შეტყობინება 2');

</script>

console.log()\_ ის მეშვეობით გამოტანილი ინფორმაცია გამოისახება ბრაუზერის კონსოლში, რომლის გამოძახებაც შესაძლებელია ბრაუზერზე მაუსის მარჯვენა კლიკი >Inspect Elementან F12\_ის მეშვეობით. (იხ.სურ.1.3.1). რა უპირატესობა გააჩნია კონსოლს თუ ის ხილვადია მხოლოდ ვებში გათვიცნობიერებული ადამიანისათვის. ვებ დეველოპერების უდიდესი ნაწილი მოიხმარს კონსოლს.



1.3

.1

კონსოლი

ს

ვიზუალური

გამოხატულება

რამეთუ კეთების პროცესში პროგრამისტი არ არის დაზღვეული შეცდომებისაგან. კონსოლი საშუალებას გაძლევთ მარტივად დაადგინოთ შეცდომის ტიპი და მდებარეობა მინიშნებების საშუალებით.

<script>

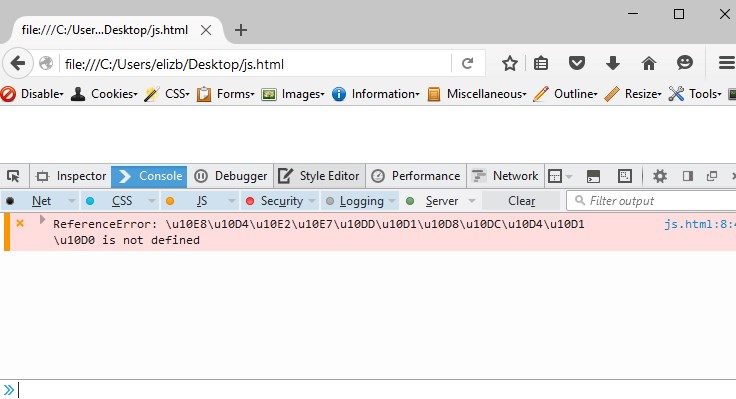
console.log('შეტყობინება 3');

</script>

სურათ 1.3\_ზე მოცემულია კონსოლი სადაც წინასწარ მაგალითზე ნაჩვენები console.log() მეთოდის მეშვეობით აისახა შედეგი.

დავუშვათ განზრახ შეცდომა სკრიპტში, რათა მოვახდინოთ კონსოლის რეაგირების დემონსტირება. console.log() მეთოდში გადაცემული არგუმენტი შეტყობინება 3 ბრჭყალების გარეშეა. როგორც აღვნიშნეთ ეს უპატიებელი შეცდომაა. ვნახოთ კონსოლის რეაგირება (იხ.სურ.1.3.2)

|  |
| --- |
| <!DOCTYPE HTML>  <html> <head>  <meta charset="utf-8">  </head> <body> <script>  console.log(შეტყობინება 3);  </script>  </body>  </html> |



1.3.1. კონსოლის რეაირება შეცდომაზე

ძირითად ნაწილში განსაზღვრულია შეცდომის ტიპი, ხოლო მარჯვენა კიდეში ასახულია სტრიქონის ნომერი, თუ რომელ ხაზზეა შეცდომა დაშვებული js.html:8

კომენტარები

Javascript\_ში ისევე როგორც თქვენს მიერ აქამდე შესწავლილ სხვა ენებში (html, css) აქტიურად გამოიყენება კომენატერბის სისტემა.

დიდი კოდის არსებობის შემთხვევაში ბევრი რამ ბუნდოვანი ხდება მისი შემდგომი გარჩევისას. ამიტომ ხშირად გამოიყენებენ კომენტარების სისტემას, რომელიც მიანიშნებს კონკრეტული სტრიქონის ან ლოგიკური ბლოკის დანიშნულებას.

კომენატრები შესაძლოა გამოყენებულ იქნას სკრიპტის ნებისმიერ ადგილას და ის არანაირ უარყოფით ზეგავლენას არ ახდებს კოდის მუშაობაზე, პირიქით დადებით ტონად გასდევს დეველოპერის ნამუშევარს.

ერთხაზიანი კომენტარები აღინიშნება // სიმბოლოთი და ზემოქმედებს მხოლოდ 1 სტრიქონზე არსებულ ინფორმაციაზე. დაიმახსოვრეთ: ერთ ხაზში მოაზრება უწყეტი ტექსტი, რომელიც ედიტორში შესაძლოა წარმოდგენილი იყოს რამოდენიმე სტრიქონადაც. ახალი სტრიქონის დასაწყისს უზრუნველყოფს კლავისი ENTER აქტივაცია.

// არსებულ მეთოდს გამოაქვს დიალოგური ფანჯარა alert('შეტყობინება 1'); alert('შეტყობინება 2');// მიღებულია კომენტარის ამ ადგილას განთავსებაც

არსებობს ასევე მრავალზახიანი კომენტარი, რომელსაც შესაბამისად გააჩნია გამხსენი ( /\* ) და დამხურავი ( \*/ )სიმბოლოების ერთობლიობა. მოცემულ მაგალითზე შეტყობინება 2 და 3 არ გამოისახება.

|  |
| --- |
| <!DOCTYPE HTML>  <html> <head>  <meta charset="utf-8">  </head> <body>  <script>  /\*  document.write('შეტყობინება 2'); document.write('შეტყობინება 3');  \*/  document.write('შეტყობინება 4');  </script>  </body>  </html> |

ცვლადები და მათი ტიპები

ისევე, როგორც პროგრამირების დანარჩენი ენებში Javascript\_შიც ინტენსიურად გამოიყენება ტერმინი ცვლადები.

რას წარმოადგენს ცვლადი?

ცვლადი არის პირობითად კონტეინერი, რომელსაც შეუძლია შეინახოს გარკვეული მონაცემი. ყოველ ცვლადს გააჩნია სახელი, ხოლო შენახულ მონაცემს ცვლადის მნიშვნელობა ეწოდება.

ახლი ცვლადის გამოსაცხადებლად javascript\_ში გამოიყენება საკვანძო სიტყვა var.ის იშფრება როგორც variable - ცვლადი.

როგორც აღვნიშნეთ ცვლადს გააჩნია სახელი. Javascript\_ში ცვლადის გამოცხადებისას ყურადღება უნდა გაამახვილოთ ცვლადის სახელებზე, რადგანაც სახელი შეიძლება შედგებოდეს მხოლოდ: სიმბოლოების $ და \_ , ასოების, ციფრებისაგან. ამასთანავე დაუშვებელია ცვლადის სახელის დაწყება ციფრით.

მაგ. I\_ზე მოყვანილია ცვლადის გამოცხადების სწორი გზა, ხოლო მაგ. II მცდარი.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| // არსებული მეთოდით ცვლადების გამოცხადება სწორია    var myName;  var test123; var $; var \_; |  |  |  | მაგ. I |

|  |  |
| --- | --- |
| // არსებული მეთოდით ცვლადების გამოცხადება მცდარია    var 123test; // ცვლადის სახელის რიცხვით დაწყება არ შეიძლება; var my-name; // არ შეიძლება სიმბოლო ‘ - ’ ტირეს გამოყენება მხოლოდ ქვედატირე; | მაგ. II |

რაც შეეხება ცვლადების სახელების მიმართ გამოთქმულ რეკომენდაციებს მდგომარეობს შემგდომში:

* გამოიყენეთ მხოლოდ ლათინური ანბანის ასოები;
* უმჯობესია თუ მოერიდებით რამოდენიმე სიტვიან ცვლადებ ან მათი გამოყენებისას მაქსიმალურად შეზღუდეთთუნდაც დასაშვები სიმბოლო „\_“. მაგ. უმჯობესია გამოიყენოთ myFirstName ვიდრე my\_first\_name;
* მაქსიმალურად მიაახლოვეთ ცვლადის სახელი მასში შესანახ მნიშვლობის არსთან;

Javascript\_ში არსებობს რეზერვირებული სიტყვები, რომელთა სახელებად გამოყენება ასევე დაუშვებელია. ცხრილ I\_ში გამოსახულია არსებულ სიტყვათა ნუსხა და პრობლემების თავიდან ასაცილებად შეეცადეთ არ გამოიყენოთ ჩამონათვლიდან არც ერთი. ამ სიტყვების ერთიანად დაზეპირება სავალდებულო არ არის. გზადაგზა უმეტეს მათგანს შეეხებით და დაიმახსოვრებთ. შესაძლოა ასევე მათი ინტერნეტში მოძიება შემდეგი ტექსტის მეშვეობით - „Javascript reserved words“

|  |  |  |  |
| --- | --- | --- | --- |
|  | ცხრილი I | |  |
| abstract boolean break byte case catch char class const continue debugger default delete do double | else  enum  export extends  false final finally float for function goto  if  implements import in | instanceof int interface long native new  null package private protected public return short static super | switch synchronized  this  throw throws transient true try typeof var void volatile while with |

როგორ ზედა ქვეთავში იყო განხილული ცვლადი შედგება 2 ძირითი ნაწილისაგან. ჩვენს მიერ გარჩეულ იქნა ცვლადის სახელი. ახლა გადავიდეთ ცვლადის მნიშვნელობაზე.

მაგალითად ისეთ პორგამირების ენებში, როგორებიცაა : C++, C# ცვლადის გამოცხადებისას ხდება მისი ტიპის განსაზღვრა და მასში მხოლოდ მითითებული ტიპის მონაცემების ჩალაგებაა შესაძლებელი. იხ.

მაგალითი

int n; // გამოცხადებული ცვლადი n შეიძლება შეიცავდეს მხოლოდ მთელ რიცხვებს

string txt; // ცვლადი txt შეიძლება შეიცავდეს მხოლოდ ტექსტუალურ მნიშვნელობას

// და ა.შ;

Javascript\_ში არ გვხვდება მკაცრად გამოხატული ტიპიზაცია. რას ნიშნავს ეს ყოველივე ?!

ეს ნიშნავს შემდგომს - Javascript\_ში შესაძლებელია ერთი და იგივე ცვლადისათვის სხვადასხვა ეტაპზე სხვადასხვა ტიპის მნიშვნელობის მინიჭება, ისე რომ არავითარი „ტიპზე დაყვანის ოპერაციები“ არ არის საჭირო, ის ავტომატურად გარდაიქმნება იმ ტიპის ცვლადად რა ინფორმაციასაც მიიღებს.

ზოგადად პროგრამირების ენები, რომლებიც არ ხასიათდება მკაცრად გამოხატული ტიპიზაციით მაგ. Javascript, php ამ ამოცანას აკისრებენ ინტერპრეტატორს, ამიტომაც Javascript\_ში მათ გამოსაცხადებლად საკმარისია მხოლოდ უკვე ნახსენები საკვანძო სიტყვა var გამოყენება.

|  |  |
| --- | --- |
| var myName; | // მისთვის შესაძლებელია ნებისმიერი ტიპის მნიშვნელობის მინიჭება |

პროგრამირების ენებში არსებობს სხვადასხვა ტიპის ოპერატორები: არითმეტიკული, ლოგიკური, ბინარული, სტრიქონული, პირობითი. მათ სრულყოფილად ორიოდ წუთში მივუბრუნდებით ამ ეტაპზე კი მხოლოდ მინიჭების ოპერატორ „=“ შევეხოთ. დაიმახსოვრეთ ეს ნიშანი ‘=’ პოგრამირებაში არ გამოხატავს „ტოლობას“ ის არის მინიჭება.

ცვლადში დროებითი მნიშვნელობის განსათავსებლაბლად საჭიროა დავწეროთ მაგალითის შესაბამისი გამოსახულება;

var myCar = 'Ford Mustang GT 500 Eleanor';

var price = 550000;

ყურადღება გაამახვილეთ შემდეგზე: ცვლადი myCarარ უდრის, არ არის ტოლი, მას უბრალოდ ენიჭება'Ford Mustang GT 500 Eleanor' მნიშვნელობა. თანდათან უფრო ცხადი წარმოდგენა შეგექმნებათ მინიჭების ოპერატორზე.

ახლა უშუალოდ რაც შეეხება ამ კონკრეტულ ჩანაწერთა ერთობლიობას. ერთი შეხედვით სხვაობა არ არის არანაირი უბრალოდ myCar\_ის მნიშვნელობა მოთავსებულია ბრჭყალებში, ხოლო price\_ის არა. ეს მათი ტიპების დამსახურება.

Javascript\_ში ტიპებიდაყოფილია 2 ჯგუფად:

1. პრიმიტიული ანუ მარტივი ტიპები:
   1. String
   2. Number
   3. Boolean
   4. Null
   5. Undefined
2. შედგენილი ტიპი - Object;

Nullდა Undefined ტიპებს ხშირად სპეციალურ ტიპებადაც მოიხსენიებენ.

გავეცნოთ პრიმიტიულ ტიპებს დაწვრილებით, ხოლო Objectგადავდოთ მომავალისათვის.

String - ტექსტური ტიპი

ტექსტური ტიპი აერთიანებს ყველა სახის ტექსტს. კანონია ყველა ტექსტური ტიპის მნიშვნელობა უნჯა იჯდეს ერთმაგ ან ორმაგ ბრჭყალებში.Javascript\_არ არის სხვაობა არსებულ ბრჭყალებს შორის. იხ. მაგ.

|  |  |
| --- | --- |
| var txt ="აქ განთავსდება ტექსტური მნისჰვნელობა"; | // ორმაგი ბრჭყალი |
| txt ='აქ განთავსდება ტექსტური მნისჰვნელობა'; | // ერთმაგი ბრჭყალი |
|  |  |
| Number - რიცხვითი ტიპი |  |

რიცხვითი ტიპი თავის თავში აერთიანებს, როგორც მთელ, ასევე ათწილად რიცხვებს. არსებობს ასევე სპეციალური რიცხვითი ტიპები -Infinity , +Infinity , NaN.

var n=123; n =12.345;

Infinity წარმოადგენს უსასრულობას;

|  |
| --- |
| // მაგ.: 1\_ის ნულზე გაყოფის შედეგად მიიღება უსასრულობა;    alert(1/0);// +Infinity alert(-1/0);// -Infinity |

NaN - იშიფრება როგროც - Not a Number, ანუ რიცხვითი მნიშვნელობა, რომელიც არ უდრის არც ერთ სხვა რიცხვს, ასევე არ უდრის თავის თავსაც. შესაძლოა ცოტა გაუგებარია?! ეს არის ტიპი რომელიც რამენაირ კავშირშია რიცხვთან მაგრამ ამავდროულად არ არის რიცხვი. მაგ.: რიცხვისა და ტექსტის არითმეტიკული ნამრავლი არის NaN, ასევე მსგავსი ტიპი მიიღება კვადრატული ფესვი -1 დან და ა.შ.

alert("ტექსტი"\*15);// NaN

Boolean - ბულის ანუ ლოგიკური ტიპი

ის თავის თავში აერთიანებს მხოლოდ 2 მნიშვნელობას true - ჭეშმარიტია , false - მცდარია;

var checked =true;// მიმდინარე სტრიქონზე ცვლადი ჭეშმარიტია checked =false;// მიმდინარე სტრიქონზე ცვლადი მცდარია

Null - მნიშვნელობა, რომელიც არ არსებობს

არსებული ტიპი არ განეკუთვნება არც ერთ ტიპს. ზოგადა მიჩნეულია, რომ null წარმოადგენს ობიექტის ტიპს და მიუთითებს ობიექტის არ არსებობას.. დაიმახსოვრეთ null არ არის 0. ის არის შემთხვევა როცა კონკრეტული ცვლადი უდრის არაფერის;

var age =null;

Undefined - მნიშვნელობა განუსაზღვრელია

არსებული ტიპი მხოლოდ ერთ მნიშვნელობას შეიცავს და ეს არს „მნიშვნელობა განუსაზღვრელია“.

თუ მოვახდენთ ცვლადის დეკლალირებას მაგრამ მას არ მივანიჭებთ საწყის ეტაპზე არაფერს არ წარმოადგენს არანაირ პრობლემას. უბრალოდ თუ შევეცდების მის ეკრანზე გამოტანას მნიშვნელობად გვიჩვენებს undefined

|  |  |
| --- | --- |
| var x;  alert( x );// გამოიტანს "undefined" |  |
| არსეობსსპეციალურიოპერატორი და რომელსაცშეუძლიამონაცემისტიპისდადგენადამისიჩანაწერიშემდეგნაირია:   * ოპერატორის სინტაქსი typeofx; * ფუნქციის სინტაქსი typeof(x); ორივე შემთხვეაში typeofმუშაობს იდენტურად. გამოიყენეთ რომელიც გსურთ | ფუნქციაtypeof, |
| alert(typeof undefined)// "undefined" |  |

alert(typeof0)// "number"

alert(typeoftrue)// "boolean"

alert(typeof"foo")// "string"

alert(typeofnull)// "object"

შესაძლოა კონკრეტული მნიშვნელობის მაგირად გადავაწოდოთ ცვლადად

)// " number "

ცვლადის გამოცხადების გზები

ჩვენს მიერ განხილულ იქნა ცვლადის სახელის შერჩევა, var საკვანძო სიტყვა, ცვლადის მნიშვნელობა შესაბამისად შეგვიძლია ცვლადის გამოცხადების გზებზე საუბარი.

შესაძლებელია ცვლადის გამოცხადება მნიშვნელობის გარეშე და ასევე მასთან ერთად

|  |
| --- |
| var x; // ცვლადის გამოცხადება მნიშვნელობის გარეშე var y; // ცვლადის გამოცხადება მნიშვნელობის გარეშე    var age = 15; // ცვლადის გამოცხადება - მნიშვნელობის მინიჭებით var city = "თბილისი"; // ცვლადის გამოცხადება - მნიშვნელობის მინიჭებით |

შესაძლებელია ასევე ცვლადების ჩამოთვლა და მათი ერთდოული დეკლარირება 1 var სიტყვის საშუალებით, გამყოფებად კი გამოიყენება მძიმე“,” .

|  |  |
| --- | --- |
| var x, | age = 15, city = "თბილისი"; |

მრავალჯერ აღნიშნული მიუხედავად კვლავ დავკონკრეტდები, რომ Javascript ხაზოვანი კითხვადობის მიმდევარია.

varage = 15; // ცვლადისთვის პირველადი დეკლარირება;

alert(age); // შედეგი ამ ეტაპზე 15

age = 103; // ცვლადისთვის ყოველ შემდეგ ჯერზე მნიშვნელობის მინიჭებისას სიტყვა varაღარ არის საჭირო, არსებული სტრიქონის ქვემოთ გამოყენებული ცვლადის მნიშვნელობა ყველგან იქნება 103.

ახალი მნიშვნელობის მინიჭებამდე

alert(age); // შედეგი ამ ეტაპზე 103

არითმეტიკული ოპერატორები

არითმეტიკული მოქმედებები გამოიყენება რიცვითი ტიპის ცვლადებზე. ცხრილ II მოცემულია Javascript\_ში არსებული არითმეტიკული მოქმედებები.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  | ცხრილი II |
| ოპერატორი | აღწერა | მაგალითი | y მნიშვნელობა | შედეგი |
| + | მიმატება | x = y + 2 | y = 5 | x = 7 |
| - | გამოკლება | x = y - 2 | y = 5 | x = 3 |
| \* | გამრავლება | x = y \* 2 | y = 5 | x = 10 |
| / | გაყოფა | x = y / 2 | y = 5 | x = 2.5 |
| % | მოდულით გაყოფა | x = y % 2 | y = 5 | x = 1 |
| ++ | ინკრიმენტი | x = ++y | y = 5 | x = 6 |
| x = y++ | y = 5 | x = 5 |
| -- | დეკრიმენტი | x = --y | y = 5 | x = 4 |
| x = y-- | y = 5 | x = 5 |

პირველი ოღხი ოპერატორი არ საჭიროებს დამატებით განმარტებას, მათთან შეხება სკოლაში საწყისი ასაკიდანვე გიწევდათ.

რაც შეეხება % ნიშანს მისი მოქმედება ოდნავ განსხვავებულია და ბევრისთვის შეიძლება გამოცანად რჩება ტერმინი მოდულით გაყოფა. მარტივათ, რომ ვთქვათ მოდულით გაყოფისას შედეგი მიიღება ნაშთი. თუ პასუხად მოგვცა 0 ე.ი გასაყოფი გამყოფზე იყოფა უნაშთოდ.

|  |  |
| --- | --- |
| varx = 19 %4; | // 3 - ნაშთი |
| x = 20 %4; | // 0 და არა 5 რადგანაც გაყოფა მოხდა უნაშთოდ |

ინკრიმენტი და დეკრიმენტი პროგრამირებაში ფართოდ გამოყენებადი ოპერატორებია. რეალურ რეჟიმში ისინი შემოკლებულ ჩანაწერს წარმოადგენენ. მაგ. ინკრიმენტი: i++იგივეა, რაც i+1, ასევე ++i იდენტურია i+1ერთი არც თუ ისე უმნიშვნელო განსხვავებისა: i++ზრდის ცვლად i მნიშვნელობას 1\_ით მხოლოდ მომავალ სტრიქონზე, ხოლო ++I მიმდინარეზე, სადაც მოხდა მისი გამოყენება;

მაგ. I vari = 5; // საწყის მნიშვნელობად i = 5 alert(i);// 5

alert(i++);// ამ ეტაზე i ისევ 5\_ია ის განიცდის მომატებას და გაზრდილ მნიშვნელობად მხოლოდ ზემდეგი სტრიქონიდან ფიქსირდება

alert(i);// აქ უკვეi გახდა 6 და ყველა შემდეგ ხაზზე მისი მნიშვნელობა 6\_ია თუ უცვლელად დავტოვებთ

მაგ. II var i = 5; // საწყის მნიშვნელობად i = 5 alert(i);// 5

alert(++i);// აქ უკვე i გახდა 6 და ყველა შემდეგ ხაზზე მისი მნიშვნელობა 6\_ია თუ უცვლელად დავტოვებთ

alert(i);// 6

რაც შეეხება დეკრიმენტს ინკრიმენტის მსგავსად მუშაობს უბრალოდ ახორციელებს 1\_ით კლებას.

მაგ. I vari = 5; // საწყის მნიშვნელობად i = 5 alert(i);// 5

alert(i--);// ამ ეტაზე i ისევ 5\_ია ის განიცდის კლებას და დაკლებულ მხოლოდ ზემდეგი სტრიქონიდან ფიქსირდება alert(i);// აქ უკვეi გახდა 4 და ყველა შემდეგ ხაზზე მისი მნიშვნელობა 4\_ია თუ უცვლელად დავტოვებთ

მაგ. II vari = 5; // საწყის მნიშვნელობად i = 5 alert(i);// 5

alert(--i);// აქ უკვეi გახდა 4 და ყველა შემდეგ ხაზზე მისი მნიშვნელობა 4\_ია თუ უცვლელად დავტოვებთ

alert(i);// 4

როგორც ქვეთავის დასაწყისში აღვნიშნეთ არსებული არითმეტიკული მოქმედებები იდეალურად მუშაობენ მხოლოდ ციფრებთან მიმართებაში. რაც შეეხება მათ მუშაობას ტექსტური (string) ტიპიც ცვლადებთან ცოტა განსხვავებულად მოქმედებენ.

მაგალითად თუ მოცმულია შემთხვევა

|  |  |
| --- | --- |
| vari = "თბილისი"\*5; | // NaN |

ყველა არითმეტიკული მოქმედება გარდა „+“ შედეგად დააბრუნებს NaN, რაც ცხადზე ცხადია გამომდინარე იქიდან, რომ ტექსტუალური ტიპის ინფორმაციაზე არითმეტიკული მოქმედებების გამოყენება დაუშვებელია. რაც შეეხება „+“ ორიოდ წუთში დავუბრუნდებით. მაგრამ ვთვქვათ ტექსტუალური ტიპით წარმოვადგინეთ რიცხვი რა მოხდება ამ შემთხვევში? ინტერპრეტატორი ავტომატურად მოახდენს ტიპის დაყვანას თუ ის შესაძლებეალია

|  |  |
| --- | --- |
| vari = "6"\*5; i = "11"-3; // 8 i = "10"/2; //5 i = "10"/4; //2 | //30 |

რაც შეეხება “+” ნიშანს გარდა არითმეტიკული მოქმედებისა javascript\_ში ის წარმოადგენს კონკატინაციის / მიერთების ნიშანს. თუ ეს ნიშაში მოხვდება ტექსტუალურ მონაცემთან ის ავტომატურად მიერთებას მოახდენს ხოლო თუ რიცხვით ტიპთან არითმეტიკულ მოქმედებას. ამიტომ საწყის ეტაპზე ხშირად იწვევს დაბნეულობას როცა არითმეტიკული შეკრება სულ სხვა შედეგით სრულდება ვიდრე ამას მოელოდით. მაგ.:

vari = "6"+5; //65 i = "25"+"23"; //2523

როგორც ზედა მაგალითიდან გამოჩნდა „+“ ნიშნის გამოყენებით მიღებული შედეგი NaN არ არის, არამედ მოხდა ტექსტების ერთმანეთის გვერდიგვერდ დალაგება. რა თქმა უნდა შესაძლებელია ტექსტური ინფორმაციის რიცხვითში გადაყვანა სპეციალური ფუნქციების გამოყენებით, მაგრამ არსებობს ასევე საკმაოდ მარტივი გზაც, რომელიც მაგალითზეა ნაჩვენები

vari = +"6"+5; //11 i = +"25"++"23"; //48

„+“ ნიშანი, როგორც აღვნიშნეთ კონკატინაციას წარმოადგენს და მისი გამოყენება საკმაოდ ინტენსიურად ხორცილედება. მაგ.: მოცემული ცვლადებით მიიღეთ სრულფასოვანი წინადადება და დაბეჭდეთ ეკრანზე.

|  |
| --- |
| Var name = 'თედო'; var txt = 'საოცნებომანქანა'; varmyCar = 'Ford Mustang GT 500 Eleanor';    //საოცნებო მანქნაFord Mustang GT 500 Eleanor document.write(txt + myCar ); //ორი ცვლადის ტექსტი წვეტის გარეშე მიეწებება ერთმანეთს    //საოცნებო მანქნა Ford Mustang GT 500 Eleanor document.write(txt +' - '+myCar );    //Ford Mustang GT 500 Eleanor - საოცნებო მანქნა document.write(myCar + ' - '+txt );    //Ford Mustang GT 500 Eleanor ჩემი საოცნებო მანქნაა document.write(myCar + ' ჩემი '+txt+ ' ა' );    //თედოს საოცნებო მანქნა არის Ford Mustang GT 500 Eleanor document.write(name + 'ს '+ txt+ ' არის '+myCar ); |

მინიჭების ოპერატორები

ზედა ნაწილში განხულულ იქნა მინიჭების ოპერატორი „=“. არსებობს სხვა ოპერატორებიც, რომლებით აერთიენებს მინიჭების და არითმეტიკულ ოპერატორებს და საკმაოდ ამარტივებს მიფგომას. ცხრილი III განხილულია შემთხვევა სადაც საწყის მნიშვნელობებად x = 10 და y = 5.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | ცხრილი III |
| ოპერატორი | მაგალითი | მსგავსი გამოსახულება | შედეგი |
| = | x = y | x = y | x = 5 |
| += | x += y | x = x + y | x = 15 |
| -= | x -= y | x = x - y | x = 5 |
| \*= | x \*= y | x = x \* y | x = 50 |
| /= | x /= y | x = x / y | x = 2 |
| %= | x %= y | x = x % y | x = 0 |

შედარების ოპერატორები

პროგრამირებაში დიდია შედარების ოპერატორების გამოყენების ინტენსივობა. ხშირია პროგრამის შემუშავების პროცესში პირობის წამოჭა, რომლის ჭეშმარიტების დასადგენად შედარების პრინციპის გამოყენებაა საჭირო. ნებისმიერი შედარების შედეგად მიღებული შედეგი წარმოადგენს Boolean ტიპის მნიშვნელობას: ის არის ჭეშმარიტი trueან მცდარიfalse.იხ.ცხრილი IV

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | ცხრილი IV |
| ოპერატორი | აღწერა | პირობა | შედეგი |
| == | ტოლობა | 5 == 8 | false |
| 5 == 5 | true |
| === | ექვივალენტურია, ანუ ტოლია როგორც  მნიშვნელობით ასევე ტიპიც საერთო აქვთ | 5 === "5" | false |
| 5 === 5 | true |
| != | არ უნდრის | 5 != 8 | true |
| !== | არაექვივალენტურია | 5 !== "5" | true |
| 5 !== 5 | false |
| > | მეტია | 5 > 8 | false |
| < | ნაკლებია | 5 < 8 | true |
| >= | მეტია ან უდრის | 5 >= 8 | false |
| <= | ნაკლებია ან უდრის | 5 <= 8 | true |

|  |  |
| --- | --- |
| “a” >“aba” | false |
| “b” >“aba” | true |
| “abas” >“aba” | true |

რიცხვითი მნიშვნელობების დროს შედარების ოპერატორებზე ფაქტობრივას სასაუბრო არც არაფერი იმდენად მარტივია ყველაფერი. რაც შეეხება ტექსტური ტიპის მნიშვნელობებს არც მათი შედარებაა წარმოუდგენლად რთული. გამომდინარე იქიდან, რომ თითოეულ

სიმბოლოს საკუთარი ASCII კოდი შეესაბამება სირთულეც მოხსნილია. ყურადღება გაამახვილეთ, რომ დადარების პროცესი ხდება ინდივიდუალურად თითოეული სიმბოლოს მიხედვით და ტექსტის დიდი სიგრძე ყოველთვის უპირატესობას არ წარმოადგენს. როგორც მაგალითზეა მოცემული ერთ შემთხვევაში “a” > “aba” მცდარია, ხოლო “b” > “aba” ჭეშმარიტი. სწორედაც რომ ასეა. გამომდინარე იქიდან რომ დადარება უშალოთ თითოეული სიმბოლოსი ხდება მეორე შემთხვევაში “b” შეედარა “aba” სიტყვის პირველ ასო “a” აღმოჩნდა მასზე მეტი და დაჯაბნა მთლიანი სიტყვა დანარჩენ ასეობზე დადარება საერთოდ აღარ ცათვალა საჭიროდ.

ლოგიკური ოპერატორები

ლოგიკურ ოპერატორებს საკმაოდ მნიშნველოვანი როლი ენიჭება პირობით ოპერატორებში. მათი დახმარებით დეველოპერებს ეძლევათ საშუალება პირობა უფრო დააკონკრეტონ. ლოგიკური ოპერატორები წარმოდგენილია ცხრილი V სადაც x = 6 და y = 3;

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  | |  | | ცხრილი V | |
| ოპერა | | ტორი | აღწერა | პირობა | |  | შედეგი |
| ! | |  | უარყოფა | !(x == y) | |  | true |
|  | && |  | ლოგიკური „და“ |  | |  | true |
|  | (x < 10 && y > 1) |
|  |  |
| || | |  | ლოგიკური „ან“ | (x == 5 || y == 5) | |  | false |

### 5.2 ამოცანის გადაჭრა ძირითადი კონსტრუქციების გამოყენებით

|  |
| --- |
| მიმდინარე პარაგრაფის თემატიკა   * ამოცანის გადაჭრის ალგორითმის შემუშავება * ენის ძირითად კონსტრუქციები |

პირობითი ოპერატორები

პროგრამირებაში საკმაოდ დიდი როლი უკავიათ პირობით ოპერატორებს. ამოცანების უდიდეს ნაწილში პირობის დასმა და შესაბამისი ლოგიკის მიღება ფაქტობრივად წინა პლანზეა წამოწეული. Javascript\_ში გვხვდებაifდაswitch კონსტრუქციები. ამათგან if უფრო ფართოდ გამოყენებადი და ბევრისთვის საყვარელი ოპერატორია, მაგრამ არც switch\_ს უნდა დავუკარგოთ ის დადებითი რაც გააჩნია. გზადაგზა უფრო ვრცლად მაგალითებზე დაყრდნობით გავეცნობით თითეულ მათგანს.

if ოპერტორის რამოდენიმე სახის ჩანაწერი არსებობს. ყველაზე გავრცელებული შემდეგი სკრიპტი

if( x >10) {

alert('გამოიტანოს შედეგი')

}

ნებისმიერი სახით if ოპერატორის გამოყენება, იქნება ეს ფიგურული ფრჩხილებით, მის გარეშე თუ სხვა მეთოდით უცვლელი რჩება არსი: if ოეპრატორს ჭირდება სტანდარტული ფრჩხილები ( ... ) პირობის დასასმელად;

if(პირობა) {

ბლოკი რომელიც სრულდება პირობის დაკმაყოფილების შემთხვევაში

}

ბლოკში არსებული ინფორმაცია იმუშავებს მხოლოდ იმ შემთხვევაში თუ შესრულება if\_ის ფრჩხილებში () დასმული პირობა, ანუ პასუხი იქნება ჭეშმარიტი - true.

როგორც დასაწყისში აღვნიშნეთ შესაძლებელია if ჩაწერა ბლოკის - ფიგურული ფრჩხილების გარეშე თუ მხოლოდ ერთი ხაზია მნიშვნელობა

if( 20 >10) alert('შედეგი 1') // ეკრანზე გამოისახება შედეგი 1

თუ შედეგი რამოდენიმეა მაგ.:

if( 20 >10) alert('შედეგი 1') // ეკრანზე გამოისახება შედეგი 1 alert('შედეგი 2') // ეკრანზე გამოისახება შედეგი 2

მაგრამ იმ შემთხვევაში თუ if პირობა არ დაკმაყოფილდა

if(3 >10) alert('შედეგი 1') // ეკრანზე არ გამოისახება შედეგი 1 alert('შედეგი 2') // ეკრანზე გამოისახება მხოლოდშედეგი 2 დასკვნა: თუ რამოდენიმე ხაზიანი შედეგის მიღება გვესაჭიროება if პირობის შესრულების დროს ფიგურული { ... } ფრჩხილების გარეშე არსებული შედეგი არ მიიღება.

ზოდად რეკომენდაციის სახით: უჯობესია ყოველთვის გამოიყენოთ ფიგურული ფრჩხილები. ის უფრო ნათელს ხდის კოდის ბლოკის დასაწყისსა და დასასრულს.

if პირობის დასაკმაყოფილებლად საჭიროა პირობაში იყოს შედეგი - true. პირობაში განთავსებული მნიშვნელობიდან გარდა 0, null, undefined, '', NaN, false დანარჩენი ყველა მნიშვნელობა ითვლება ჭეშმარიტად.

|  |
| --- |
| if(0) { // კონსტუქცია არ იმუშავებს } if(null) { // კონსტუქცია არ იმუშავებს } if(undefined) { // კონსტუქცია არ იმუშავებს } if('') { // კონსტუქცია არ იმუშავებს } if(NaN) { // კონსტუქცია არ იმუშავებს } if(false) { // კონსტუქცია არ იმუშავებს }    // გარდა ზემოთ ჩამოთვლილი შემთხვევებისა სხვა ყველა შემთხვევაში პირობა აღიქმება დადებითად    if(-1) { // კონსტუქცია იმუშავებს} |

დაიმახოსვრეთ Javascript\_ში if კონსრუქციის პირობის დაკმაყოფილების შედეგად კოდის კითხვა გრძელდება ბლოკის ფრჩხილებში და მხოლოდ ამის შემდგომ გადადის მის ქვემოთ არსებულ სკრიტების შესრულებაზე.

მაგ.: ამოცანა მდგომარეობს შემდეგში. შემოვიტანოთ ცვლადი რომელშიც განვსაზღვრავთ პიროვნების ასაკს. მოვახდინოთ if ოპერატორის მეშვეობით დადგენა სრულწლოვანია თუ არასრულწლოვანია არსებული პიროვნება. პირობა დაისმის შემდეგნაირად: თუ ადამიანის ასაკი აღემატები 18 წელს ის წარმოადგენს სრულწლოვანს.

|  |
| --- |
| <!DOCTYPE HTML>  <html> <head>  <meta charset="utf-8">  </head> <body> <script>  Var age = 25; var age1 = 10; |
| var user = 'სრულწლოვანი'; // საწყის ეტაპზე განისაზღვრა, რომ ეს მომხმარებელი არის 18\_ზე მეტი წლის.    if(age <18) { user = 'არა სრულწლოვანი';  }    // თუ if პირობა შესრულდებოდა user ცვლადი მიიღებდა მნიშვნელობად 'არა სრულწლოვანი' და შემდეგ ხაზზე დაიბეჭდებოდა სწორედაც ეს სიტყვა, გამომდინარე იქიდან, რომ 25 ნაკლები არ არის  18\_ზე if კონსტრუქცია აღარ გააქტიურდა და ცვლად user \_ მა შეინარჩუნა პირველადი მნიშვნელობა    document.write(user); // სრულწლოვანი      if(age1<18) { user = 'არა სრულწლოვანი';  }    // არსებულ შემთხვევაში 10 ნაკლებია 18ზე პირობა დაკმაყოფილებულია. user ცვლადმა მიიღო მნიშვნელობა 'არა სრულწლოვანი'    document.write(user); // შედეგია: არასრულწლოვანი    </script>  </body>  </html> |

გარდა ცალკეული if კონსტრუქციისა არსებობს if-else ოპერატორი და მისი ჩანაწერი შემდეგია

if(age1<18) {

// შესრულდება ყოველთვის, როცა დაკმაყოფილდება if პირობა

} else {

// შესრულდება ყოველთვის, როცა არ დაკმაყოფილდება if პირობა

}

თუ ყურადღებით დაუკვირდებით else პირობის ფრჩხილები (...) არ გააჩნია, რადგანაც ის if\_ის უარყოფას წარმოადგენს და დამატებით პირობებს არ საჭიროებსდ და if ოპერატორში დასმული პირობის „მცდარობის“ შემთხვევაში ყველეთვის შესრულდება.

დაიმახსოვრეთ: else ოპერატორს დამოუკიდებლად არსებობა არ შეუძლია, განსხვავებით if ოპერატორისაგან.

if(age1 <18) {

document.write('არა სრულწლოვანი');

} else {

document.write('სრულწლოვანი');

}

არსებობს if-else კონსტრუქციის შემოკლებული ჩანაწერი, რომელსაც დეველოპერები საკმაოდ აქტიურად გამოიყენებენ მცირე კოდის შემთხვევაში

(age1 <18) ?document.write('არა სრულწლოვანი') :document.write('სრულწლოვანი');

სადაც სიტყვიერი if არ იწერება, იხსნება პირობის ფრჩხილები, მას მოსდევს „?

„ რომელიც აღნიშნავს if\_ის ბლოკს, ხოლო “:” else \_ს ბლოკია .

არსებობს ასევე შედაერბით რთული კონსტრუქცია, რომელიც else-if\_ად მოიხსენიება. ეს არის კონსტრუქცია, რომელშიც if პირობის უარყოფასთან ერთად ხდება ახალი პირობის წამოყენება.

if(age <18) {

document.write('მოზარდი');

} elseif (age <60) {

document.write('ზრდასრული');

}

მოცემულ მაგალითი შემდეგნაირად იშიფრება: თუ ცვლადი age ნაკლებია 18\_ზე მაშინ დაბეჭდოს 'მოზარდი', წინააღმდეგ შემთხვევაში თუ age ნაკლებია 60\_ზე დაბეჭდავს'ზრდასრული'.

ბევრს შეიძლება გაუჩნდეს კითხვა,თუ age = 15 მაშინ შესრულდება if ბკლოკში არსებული კოდი ( დაიბეჭდება 'მოზარდი') , მაგრამ 15 ხომ 60\_ზეც ნაკლებია? რა თქმა უნდა ნაკლებია მაგრამ else if პირობაზე გადასხვა მხოლოდ იმ შემთხვევაში ხდება, როდესაც არ სრულდება if პირობა.

თუ if პირობა შესრულდა არა აქვს მნიშვნელობა რამდენი else-if იარსებებეს მის ქვემოთ ისინი არასდროს დაკმაყოფილდებიან. იხ.მოცემული მაგალითი

var age =15;

if(age <18) {

// 15<18 შესაბამისად შესრულდება მხოლოდ ეს კოდი document.write('მოზარდი');

} elseif (age <30) {

|  |
| --- |
| document.write('ზრდასრული');  } elseif (age <60) {  document.write('ასაკოვანი');  } elseif (age >60) {  document.write('ხნიერი');  } |

შესაძლებელია else-if კონსტრუქციის გამოყენებისას ის დასრუდეს else ოპერატორით, რომელიც ყველა შესაძლო else if პირობის უარყოფას წარმოადგენდეს

|  |
| --- |
| var car ='acura'; if(car =='bmw') {  document.write('bmw');  }elseif (car =='volvo') {  document.write('volvo');  }elseif (car =='opel') {  document.write('opel');  }else {  document.write('მარკა სრულიად განსხვავებულია ის არის '+car);  } |

switch – case კოსნტრუქცია

ბოლო მოგალითზე დაკვირვებით სია, რომელთანაც არსებულ car \_ს ვადარებთ შესაძლოა საკმაოდ დიდი იყოს. შემთხვევა, როცა ერთი კონკრეტული მნიშვნელობის რამოდენიმესთან ერთდროულ შედარებაა საჭირო გამოიყენება switch – case კოსნტრუქცია რომელიც ბევრად მოხერხებულია, გამომდინარე მისი სინტაქსიდან.

|  |
| --- |
| switch(შესადარებელიმნიშვნელობა){    case'მნიშვნელობა 1': შესასრულებელი არეალი .........  break; // თუ case არსებული მნიშვნელობის თანხვედრა მოხდა break აჩერებს დანარჩენი დადარების ოპერაციებს.    case'მნიშვნელობა 2': შესასრულებელი არეალი ......... break;    default: შესასრულებელი არეალი ......... |

break;

}

|  |
| --- |
| var car ='acura';    switch(car){ case'bmw':// if (x === ' bmw')  document.write('bmw'); break;    case'volvo':// if (x === 'volvo')  document.write('volvo'); break;    case'opel':// if (x === 'opel')  document.write('opel'); break;    default:  document.write('მარკა სრულიად განსხვავებულია ის არის '+car); break;  } |

ოპერატორი breakწყვეტს კოდის კითხვადობას თუ მოხდა გადმოცემული მნიშვნელობის და case\_ის გატოლება (ექვივალენტურად). არსებული ოპერატორის გამოტოვების შემთხვევაში გამოვა switch\_ში არსებული უკანასკნელი მნიშვნელობა, რაც არასასურველ შედეგამდე მიგიყვანთ. მინიშნება არ დაგავიწყდეთ პირობის შემდეგbreakოპერატორის გამოყენება.

რაც შეეხება ოპერატორ default მის „ტანში“ გამოტანილი მოქმედება სრულდება მაშინ, როცა არ მოხდება ცვლადის არცერთ case მნიშვნელობასთან გატოლება (ექვივალენტურად).

ფაქტობრივად switch-case კონსტრუქცია, რაღაც კუთხით else-if კონსტრუქციის გამარტივებულ ვარიანტს წარმოადგენს (ყველა შემთხვევაში არა), შესაბამისად თუ მას else-if \_თან გავაიგივებთ ოპერატორი default წარმოადგენს ყველა პირობის უარმყოფ else ოპერატორს, რომელიც სრულდება დანარჩენი პირობის დაუკმაყოფილებლობის შემთხვევაში.

default გამოყენება არ არის სავალდებულო.

switch-case კონსტრუქციაში შესაძლებელია რამოდენიმე case მნიშვნელობის დაჯგუფება შემდეგი სახით

var a =2+2;

|  |
| --- |
| switch(a){ case4: alert('სწორია!'); break;    case3:// თუ a === 3 ან a ===5 გამოვიდოდა მიმდინარე შეტყობინებები case5:  alert('შეცდომაა!'); alert('დაგაკლდათ მცირეოდენი'); break;    default:  alert('თქვენ საერთოდ არ ხართ არითმეტიკულ მოქმედებებთან ახლოს '); break;  } |

ციკლები

სკრიპტის შემუშავების პროცესში ხშირად დგება ამოცანა ერთგვაროვანი მოქმედება განმეორდეს რაიმე ინტენსივობით გარკვეულის ლოგიკის შესაბამისად. კოდის ნაწილის გამეორებადობისათვის Javascript პროგრამირების ენა იყენებს ციკლებს. ადვილად აღწმადი, რომ გამოვიდეს კიდევ ერთხელ განვმარტოთ: ციკლი წარმოადგენს პროცესს, რომელიც რაღაც კანონზომიერებით მეორდება.

მაგ. 1: წელიწადში 4 დროა: გაზაფხული, ზაფხული, შემოდგომა, ზამთარი. ისინი სწორედ იმ თანმიმდევრობით მონაცვლეობენ, როგორც ჩამოვთვალეთ ანუ ემორჩილებიან კონკრეტულ კანონზომიერებებს. ეს არის ციკლი, რომელიც არასდროს (ყოველ შემთხვევაში იმედია) დასრულდება.

მაგ. 2: დღე-ღამე შედგება 24 საათისაგან. სწორედ 24 საათიანი ინტერვალის გავლიშ შემდგომ იწყება ახალი დღე ანუ საათის ათვლა უბრუნდება ნიშნულ 0\_ს.

ამ ყოველივეს გასამარტივებლად Javascript\_ში გაოიყენება ციკლის ოპერატორები while, do while, for. განვიხილოთ თანმიმდევრულად.

ციკლი while სინტაქსი მოცემულია მაგალითზე.

while(პირობა){

მოქმედების შესასრულებელი არეალი

}

როგორც მონახაზიდან ჩანს ციკლი while ვიზუალი ძალიან ახლოსაა if კონსტრუქციასთან. მასაც გააჩნია პირობის ჩასაწერი ფრჩხილები ( ... ) და იდენტური ბლოკის გამხსენი და დამხურავი ფიგურული ფრჩხილები { ... } . შესაბამისად არსიც იგივეა თუ პირობა სრულდება ანუ პასუხი ჭეშმარიტია true, მაშინ სრულდება ბლოკის ფრჩხილებში მოთავსებული სკრიპტი და ეს პროცესი გრძელდება იქამდე სანამ პირობის ჭეშმარიტება ძალაშია. მაგ.:

var a = 5;

while(a < 9){

alert('გამარჯობა მე ვარ ციკლი');

}

მოცემული ციკლი იმუშვებს უსასრულოდ. ტექსტი 'გამარჯობა მე ვარ ციკლი' alert დიალოგური ფანჯრის მეშვეობით გამოტანება უწყვეტ რეჟიმში და თითოეულის გათიშვა გამოიწვევს საპასუხოდ ახლის გამოჩენს. რატომ ? გამომდინარე იქიდან, რომ 5 ყოველთვის ნაკლებია 9\_ზე ეს პირობა არასდორს შეიცვლება. ვთქვათ ამ პოცესის გამეორება გვესაჭიროება მხოლოდ 4\_ჯერ. როგორ მოვიქცეთ? შევეცადოთ დავხვეწოთ კოდი და მოვარგოთ ჩვენს მოთხოვნილებებს.

|  |  |
| --- | --- |
| var a = 5;    while(a < 9){  alert('გამარჯობა მე ვარ ციკლი');  a++;  } | // გამოვიყენე ინკრიმენტი |

a ცვლადის ზრდადობა გამოიწვევს იმას, რომ როდესღაც a მიაღწევს მნიშვნელობას როცა ის აღარ იქნება ნაკლები 9\_ზე და ციკლი შეწყვეთს მუშაობას. არსებულ შემთხვევაში ეს იქნება მეხუთე ნაბიჯზე, შესაბამისად შეტყობინება ეკრანზე გამოვა 4\_ჯერ.

var a = 5;

##### // ეტაპი I

// a = 5 პირობა სრულდება

while(a < 9){

alert('გამარჯობა მე ვარ ციკლი'); // გამოდის ეკრანზე

a++; // a ცვლადმა განიცადა ინკრიმენტაცია - შემდეგი ხაზიდან a=6

// აქ უკვე a არის 6 – a=6

}

// ციკლის უზრუნველყოფს ინფორმაციის გამეორებადობას ე.ი ის აბრუნებს კოდის მნიშვნელობას დასაწყისში და ეკითხება არის a < 9\_ზე? გამომდინარე იქიდან, რომ a =6 ის აშკარად ნაკლები გამოდის

9\_ზე და პროცესი კიდევ ერთხელ მეორებდება

|  |
| --- |
| // ეტაპი II  // a = 6 პირობა სრულდება  while(a < 9){  alert('გამარჯობა მე ვარ ციკლი'); // გამოდის ეკრანზე  a++; // a ცვლადმა განიცადა ინკრიმენტაცია - შემდეგი ხაზიდან a=7  // აქ უკვე a არის 7 – a=7  }      // ეტაპი III  // a = 7 პირობა სრულდება  while(a < 9){  alert('გამარჯობა მე ვარ ციკლი'); // გამოდის ეკრანზე  a++; // a ცვლადმა განიცადა ინკრიმენტაცია - შემდეგი ხაზიდან a=8  // აქ უკვე a არის 8 – a=8  }      // ეტაპი IV    // a = 8 პირობა სრულდება  while(a < 9){  alert('გამარჯობა მე ვარ ციკლი'); // გამოდის ეკრანზე  a++; // a ცვლადმა განიცადა ინკრიმენტაცია - შემდეგი ხაზიდან a=9  // აქ უკვე a არის 9 – a=9  }      // ეტაპი V  //a = 9 პირობა არ სრულდება.9 არ არის ნაკლები 9\_ზე  // შესაბამისად სკრიპტი აღარ კითხულობს ბლოკის შიგთავს და ეკრანზე alert-აღარ გამოვა. while(a < 9){  alert('გამარჯობა მე ვარ ციკლი'); a++;  } |

ციკლის გამეორებადობის პროცესს იტერაცია ეწოდება. მაგალითის შემთხვევაში ციკლი ასრულებს 4 ოთხ იტერაციას.

უსასრულო ციკლის საჭიროების შემთხვევაში არ არის აუცილებელი ვრცელი გზის გამოყენება

var a = 5;

while(a < 9){

alert('გამარჯობა მე ვარ ციკლი');

}

შესაძლებელია პირობაში გაიწეროს true

while(true){

alert('გამარჯობა მე ვარ ციკლი');

}

დავალება 1:

გამოვსახოთ ეკრანზე 1\_დან 10\_მდე ყველა ლუწი რიცხვი შემდეგი სახით. ამ ეტაპზე დავუშვათ , რომ ბოლო მნიშვნელობასაც ექნება გამყოფად ტირე „-“

2 – 4 – 6 – 8 -

დავალების ამოსახსნელად გვჭირედბა ციკლი, ინფორმაციის გამოტანის ოპერატორი, და ცვლადი რომელიც ყველა ჯერზე მოიმატებს 2\_ით.

var a = 2;

while(a < 10){ document.write( a + ' - ' ); // დაბეჭდავს 2 - ...... და აშ ყველა ნაბიჯზე a += 2; // იგივეა რაც a = a + 2 (მინიჭების ოპერატორის შემოკლებული ჩანაწერი )

}

დავალება 2: ციკლის მეშვეობით გამოვიტანოთ ეკრანზე ყველა რიცხვის ჯამი 1\_დან 5\_მდე.

15

დავალების ამოსახსნელად გვჭირედბა ციკლი, ინფორმაციის გამოტანის ოპერატორი, და ცვლადი, რომელიც ყველა ჯერზე მოხდება არსებული რიცხვის მიმატება წინა ჯამთან.

var i = 1; var sum = 0;

while(i <= 5){

sum += i; //sum = sum+i I ეტაპი 0+1, II ეტაპი 1+2, II I ეტაპი 3+3, IV ეტაპი 6+4, V ეტაპი 10+5 i++; //უზრუნველყოფს ციკლის მუშაობას

}

document.write(sum); // 15

განსხვავევით დავალება1\_ის ამოხსნისაგან სადაც document.write() მეთოდი უშუალოდ while ციკლის ტანში { ... } იყო გამოძახებული, დავალება2 შედეგის გამოტანის მეთოდი გარეთაა გამოძახებული. ეს რა თქმა უნდა დავალებიდან გამომდინარეა. პირველ შემთხვევაში საჭირო იყო ყველა ეტაპზე შედეგის გამოტანა, ხოლო მეორე შემთხვევაში მხოლოდ საბოლოო შედეგის. მეორე დავალების ციკლი უბრალოდ უზრუნველყოფდა დაჯამებას თითოეულ ეტაპზე (ეტაპები კომენტარის ველშია გაშლილი) და გამოტანა ერთჯერადი პროცედურით შემოიფარგლებოდა.

ცილში შესაძლებელია break და continue ოპერატორების გამოყენება. break\_ის შესახებ ინფორმაცია მოწოდებული იყო switch-case კონსტრუქციის განხილვისას. ის გამოიყენებოდა პროცესის შესაჩერებლად / გასაწყვეტად. იგივე ფუნქციონალი აკისრია მას ციკლთან მიმართებაშიც.

break ციკლში მოქმედების უკეთ გასაგებად დავალება 2 შევასრულოდ სახეცვლილი პირობით

დავალება 3:

ეკრანზე დაბეჭდოს რიცვხთა ჯამი 1\_დან 5\_მდე, სანამ ის იქნება 12\_ზე ნაკლები. ანუ ეკრანზე უნდა გამოიტანოს 10 ( I ეტაპი 0+1, I I ეტაპი 1+2, I I I ეტაპი 3+3, IV ეტაპი 6+4, V ეტაპი 10+5 ), რადგანაც ბოლო ეტაპზე მიიღება რიცხვი 15, რომელიც შესაბამისად აღემატება 12\_ს.

|  |
| --- |
| var i = 1; var sum = 0;    while(i <= 5){    // არსებულ ეტაპზე ვამოწმებთ ჯამს დამატებული მიმდინარე რიხვი მეტი ხომ არ არის 12, თუ პირობა შესრულდა sum ცვლადში აღარ მოხდეს მიმდინარე ბიჯის დამატება და ციკლმა შეწყვიტოს მუშაობა;    if( (sum + i)> 12){  break;  }  sum += i; //sum = sum+i I ეტაპი 0+1, II ეტაპი 1+2, II I ეტაპი 3+3, IV ეტაპი 6+4  i++; //უზრუნველყოფს ციკლის მუშაობას  }    document.write(sum); // 10 |

break ოპერატორისაგან გასნხსვავებით continue არ წყვეტს ციკლის მუშაობას ის უბრალოდ ახტება იმ ნაბიჯს, რომელზეც გვაქვს შესაბამისი პირობა დადებული.

დავალება 4:

გამოვსახოთ ეკრანზე 1\_დან 20\_მდე ყველა ლუწი რიცხვი,გარდა 6 და 14 შემდეგი სახით

2 – 4 –8 – 10 – 12 – 16 – 18

|  |
| --- |
| var a = 2;    while(a < 20){    // პირობა: თუ a მნიშვნელობა ტოლია 6 ან 14 მოახდინოს a გაზრდა 2\_ით და დაბრუნდეს ციკლის პირობაში, შესაბამისად ტოვებს document.write()\_ს რომელიც უზრუნველყოფს დაბეჭვდას    if(a==6 || a==14 ){  a += 2; continue;  }    document.write( a + '-' );//დაბეჭდავს 2 - ...... და აშ ყველა ნაბიჯზე a += 2; // იგივეა რაც a = a + 2 (მინიჭების ოპერატორის შემოკლებული ჩანაწერი )  } |

როგორც ქვეთავის დასაწყისში აღვნიშნეთ Javascript\_ში გვხვდება do…while ციკლიც. მისი მუშაობის პრინციპი ოდნავ განსხვავებულია სტანდარტული while ციკლისაგან, ასევე მცირეოდენ შეცვლილია წერის სინტაქსიც, რამეთუ ჯერ მოდის შესასრულებელი მოქმედების ბლოკი ხოლო შემდგომ პირობა. ეს ყოველივე კი უზრუნველყოფს იმას, რომ მოქმედება მინიმუმ ერთჯერადად მაინც სრულდება.

|  |
| --- |
| var i = 2;    do {    document.write( i + ' - ' ); i++;    } while(i > 5)    //ეკრანზე მიღებული შედეგი 2 - |

ციკლი ასრულებს შემდეგ მოქმედებას :

* do\_ს მეშვეობით შედის ციკლის ტანში {...} ასრულებს ლოგიკურ ოპერაციას ( 2- );
* ახდენს i ცვლადის გაზრდას 1\_ით;
* ამის შემდგომ უყურებს whileციკლის პირობას, გამომდინარე იქიდან, რომ პირობა არ კმაყოფილდება ( 3>5 ), ხელახალი პროცესის გამეორებადობა არ მოხერხდა;

სტნდარტული while ციკლის გამოყენების შემთხვევაში შედეგი საერთოდ არ გამოსახებოდა ეკრანზე, რადგანაც იქ ჯერ შედარების ოპერაცია ხორციელდება

|  |
| --- |
| var i = 2;    while(i > 5){    document.write( i + '-' ); i++;    }    //ეკრანზე არავიტარი შედეგი არ გამოისახება |

მაგრამ მოცმეული მაგალითს თუ შემდეგ პირობას წავუყენებთ i < 5 , არავითარი განსხვავება არ იქნება, რომელი ციკლის მეშვეობით შევასრულებთ მას იქნება ეს whileთუdo…while ციკლები

|  |
| --- |
| //მაგალითი while - ციკლით    var i = 2;    while(i < 5){    document.write( i + '-' ); //ეკრანზე მიღებული შედეგი 2 – 3 - 4 –  i++;    }      //მაგალითი do…while - ციკლით    var i = 2;    do {    document.write( i + ' - ' ); // ეკრანზე მიღებული შედეგი 2 – 3 - 4 - |
| i++;    } while(i < 5) |

პროგრამისტეიბს მიერ ციკლებიდან ყველაზე ხშირად - for კონსტრუქციაა, რომელიც თავისი სინტაქსით მეტად მობილურია და შემდეგნაირად გამოიყურება. ის ფაქტობრივად იყენებს შაბლონს, რომელშიც თავისთავად მოიაზრება ციკლი while.

for(დასაწყისი; პირობა; ბიჯები){ // ... ციკლის ტანი ...

}

while ციკლისაგან განსხვავევით for\_ის ფრჩხილებში ( ... ) კომპაქტურადაა მოთავსებული ცვლადის გამოცხადება, პირობა და იტერაცია რაც თვალნათელს ხდის თითოეულ ბიჯს.

|  |  |
| --- | --- |
| for(var i =0; i <3; i++){  alert( i );  } | // alert(0), alert(1), alert(2) |

როგორც while და do…while შემთხვევაში იყო შესაძლებელი ინდენთურად მათი ამუშავებისა ასევე შესაძლებელია forციკლი ვამუშაოთ while\_ის მსგავსად. for ციკლში დასაშვებია ნებისმიერი მნიშვნელობის გამოტოვება.

|  |
| --- |
| //ვარიანტი I    var i =0 for(; i <3; i++){  // ... ციკლის ტანი ...  }      //ვარიანტი II var i =0 for(; i <3;){  // ... ციკლის ტანი ...  i++;  }    //ვარიანტი III  for(;;){ |

// ... ციკლის ტანი ...

}

ვარიანტი I- ფაქტობრივადარგანსხვავდებაfor სტანდარტულიჩანაწერისაგანუბრალოდსაწყისიმნიშვნელობაამოღებულიაფრჩხილებიდან (...) დაგატანილიაცალკე.

ვარიანტი II - ანალოგურია while ციკლისა ვარიანტი III - ციკლი, რომელიც იმუშავებს უსასრულოდ.

Javascript\_ში არსებობს for in კონსტრუქცია ამ ეტაპზე მისი განხილვა აზრს მოკლებულია, დამხმარე თემებზე, რომელთანაც ის კავშირშია ჯერ არ გქონიათ შეხება, შესაბამიად შემდეგ ქვეთავებში მაგალითებზე დაყრდნობით გავარცევთ მასასც დეტალურად.

### 5.3 მასივებთან მუშაობა

|  |
| --- |
| მიმდინარე პარაგრაფის თემატიკა   * მასივებთან მუშაობის ძირითად ფუნქციები * ამოცანის გადაჭრის გზები მასივზე დაფუძნებული აბსტრაქტული მონაცემთა სტრუქტურების მეშვეობით * ამოცანის იმპლემენტაცია მასივების მეშვეობით |

მასივი ეს არის მონაცემთა ერთგვარი ტიპი, რომელიც შეიცავს ერთ ან რამოდენიმე მნიშვნელობას ერთდროულად, რომლებიც მასივში განთავსებულნი არიან საკუთარ პოზიციებზე. არსებულ პოზიციებს ინდექსები ეწოდებათ და ისინი წარმოადგენენ რიცხვით მნიშვნელობას 0\_დან დადებითი მიმართულებით. გამომდინარე Javascript\_ის არამკაცრი ტიპიზაციიდან მასივს მკვეთრად გამოხატული ტიპიზება არ სჭირდება, ასევე მისი თითოეული მნიშვნელობა შეიძლება ერთმანეთისაგან რადიკალურად განსხვავებული ტიპის გახლდეთ. არის შესაძლებლობა მასივის მნიშვნელობა ასევე მასივი იყოს.

მასივის შესაქმნელად ერთ ერთი გზა შემდეგი. საჭიროა ახალი ცვლადის გამოცხადება მისთვის სახელის მინიჭება და კვადრატული ფრჩხილები [....]. მაგალითზე მოცემულია ცარიელი მასივი.

var arr =[];

მასივის ელემენტების შესავსებად საკმარისია კვადრატულ ფრჩხილებში მძიმით მოხდეს ჩამონათვლის გაკეთება

var arr =['მარწყვი', 'ატამი', 'ფორთოხალი'];

არსებული მასივი შედგება 3 ელემენტისაგან, შესაბამისად ისინი განთავსებულნი არაინ პოზიციებზე რომელიც ათვლას ყოველთვის 0 დან იწყებს. თითოეულ მათგანთან წვდომისათვის უნდა მიმართოთ მასივს სახელწოდებით , კვადრატული ფრჩხილები და იმ კონკრეტული მნიშვნელობის ინდექსით რომელის გამოტანაც გსურთ.

var arr =['მარწყვი', 'ატამი', 'ფორთოხალი'];

alert(arr[ 0 ]); // მარწყვი alert(arr[ 1 ]); // ატამი alert(arr[ 2 ]); // ფორთოხალი

იმ ინდექსით მასივისათვის მიმართვა რომელზეც არანაირი მნიშვნელობა არ არის განთავსებული შედეგად დაგიბრუნებთ undefined

var arr =['მარწყვი', 'ატამი', 'ფორთოხალი'];

alert(arr[ 3 ]); // undefined alert(arr[ 80 ]); // undefined

თუ მასივის ელემენტების ჩამოთვლისას მათ შორის ხელოვნურად ან შემთხვევით ჩავსვამთ დამატებით მძიმეებს ისინი გამოყოფს მასივის ელემენტისათვის ინდექსებს და მათმნიშვნელობად undefined მოიაზრებს. მაგალითზე მოყვანილი მასივის მნიშვნელობა 'ატამი'\_თან წვდომისათვის ინექსი 1 აღარ გამოდგება, რადგანაც არსებულ ინდექსე მასივმა განალაგა მნიშვნელობა რომელიც ამ ეტაპზე undefined არის. მაგალითზე ხელოვნურად არის ადგილები გამოტოვებული ინდექსებიის ზუსტად აღსაქმელად.

|  |
| --- |
| var arr =['მარწყვი', , ,'ატამი', ,'ფორთოხალი'];  // 0 1 2 3 4 5    alert(arr[ 1 ]); // undefined alert(arr[ 3 ]); // ატამი |

მასივისათვის ახალი ელენებტის მინიჭება ან არსებულის ჩანაცვლება ყველთვის შესაძლებელია, საჭიროა მხოლოდ მასივის კონკრეტულ ინდექსს მივანიჭოთ მნიშვნელობა. იმ შემთხვევაში თუ ამ კონკრეტულ ინდექსზე უკვე არსებობს მნიშვნელობა ის ჩანაცვლდება წინააღმდეგ შემთხვევაში დაემატება ახალი.

var arr =['მარწყვი' , 'ატამი','ფორთოხალი'];

##### // 0 1 2

arr[ 0 ]='ვაშლი'; // arr =['ვაშლი' , 'ატამი','ფორთოხალი']; arr[ 3 ]='ყურძენი'; // 'ვაშლი' , 'ატამი','ფორთოხალი', 'ყურძენი'];

როგორც ქვეთავის დასაწყისში აღვნიშნეთ აუცილებელი არ არის მასივი შეიცავდეს ერთგვაროვანი ტიპის ელემენტევს. Javascript\_ში შესაძლებელია შერეული ტიპის მნიშვნელობების მინიჭება ერთი კონკრეტული მასივისათვის. აქამდე განხილულ მაგალითებში საქმე მხოლოდ ტექსტური (string) ტიპის მნიშვნელობები ფიგურირებდნენ. გასათვალისწინებელია ის რომ ნასივის ინდექზე მნიშვნელობის განთვსებისას ტიპი სტანდარტული ჩანაწერით (იქნება ის ბრჭყალებით - string, მის გარეშე-number, იქნება ობიექტი და ა.შ) გამოისახება.

var arr =['მარწყვი' , true , null, 'კივი', 155];

მასივის ინდექსაცია ყოველთვის იწყევა 0\_დან. ხელოვნურად მისი ცვლილება შეუძლებელია. მაგ.: ინდექსად 5\_ის მნიშვნელობის განსაზღვრა არ განაპირობებს მის ამთვლელ ინექსს. საწყისი ინდექსები ავტომატურად შეივსება undefined მნიშვნელობებად.

|  |  |
| --- | --- |
| arr[ 5 ]='საქართველო'; | // arr =[ , , , , , 'საქართველო']; |

როგორც აღვნიშნეთ მასივის კონრეტული მნიშვნელობის გამოსატანად სჭიროამასივის სახელიდა და მისი ინდექსის გამოტანის ოპერატის ატრიბუტად განსაზღვრა ( მაგ.: document.write(arr[5]) ), მაგრამ Javascript\_ში შესაძლებელია მასივის სრული ვიზუალიზაცია უშუალოდ მისი სახელის გამოტანით. დაიმახსოვრეთ არსებული გზა ემსახურება მხოლოდ მასივის ვიზუალური შედეგის მიღებას და ძირითადად სატესტოდ გამოიყენება.

var arr =['მარწყვი' , true , null, 'კივი', 155];

document.write(arr)// 'მარწყვი' , true , null, 'კივი', 155;

არსებობს მასივის შექმინის კიდევ ერთი სახის სინტაქსიnewArray(); ის შედარებით იშვიათად გამოიყენება გამომდინრე იქიდან, რომ კვადრატული ფრჩხილებით [...] ჩანაწერი ბევრად შემოკლებულია.

var arr =newArray('მარწყვი' , true , null, 'კივი', 155);

არსევე არსებობს ერთი თავისებურება და სხვაობა ამ ორ სინტაქსს შორის. თუ მასივის შედგება რიცხვითი მნიშვნელობისაგან და ის მხოლოდ 1 ცალია, newArray() მეშვეობთ მისი გამოცხადება მოგცემთ განსხვავებულ შედეგეს. მაგ.:

|  |  |
| --- | --- |
| var arr =[155]; alert( arr[ 0 ] ); | // 155; |
| var arr1 =newArray(155); alert(arr1[ 0 ]); | // undefined |

მაშინ რა მისია აკისრია ამ შემთხვევაში 155 - newArray(155 ) ? ის განსაზღვრავს მასივის სიგრძეს. (არსებულ საკითხს დავუბრუნდებთ) სხვა ყველა დანარჩენ შემთხვევაში ეს 2 ჩანაწერი მუშაობს ინდენტურად იქნება ეს სხვა ტიპის 1 ან 1\_ზე მეტი მნიშვნელობა, თუ რიცვითი ცვითი მნიშვნელობა მხოლოდ ერთ ელემენტზე მეტი.

// შედეგი იდენტური

var arr =newArray('მარწყვი' ,true,null, 'კივი', 155); var arr =['მარწყვი' ,true,null, 'კივი', 155];

// შედეგი იდენტური

var arr =['მარწყვი'];

var arr =newArray('მარწყვი');

// შედეგი იდენტური

var arr =[5 , 120]; var arr =newArray(5 , 120);

// შედეგი განსხვავებული

var arr =[10]; var arr =newArray( 10 );

რა არის მასივის სიგრძე, რომელმაც მცირეოდენი გაუგებრობა წარმოშვა?

მასივის სიგრძე ეს არის მასისვის თვისება რომელიც განსაზღვრავს თუ რამდენი ელემენტისაგან შედგება ესა თუ ის კონკრეტული მასივი. მისი სინტაქსი შემდეგია მასივის სახელი.length

var arr =['მარწყვი' , true , null, 'კივი', 155];

alert( arr.length ); // 5

მასივის სიგრძე დინამიურ თვისებას წარმოადგენს და იცვლება მასივის ელემენტების რაოდენობიდან გამომდინარე- უფრო ზუსტად, რომ აღვნიშნოთ მასივის ინდექსების რაოდენობიდან გამომდინარე. შემთხვევას როცა კონკრეტულ ინდექსზე (გარდა 0 ) ვათავსებთ ერთ ელემენტს მასივის სიგრძეს განსაზღვრავს ინდექსი და არა ელემენტი, რადგანაც ინდექზე განთავსება ნიშნავს იქამდე არსებული გამოტოვებული ინდექსების ავტომატურ შევსებას undefined მნიშვნელობებით.

.length თვისების გამოყენებით შესაძლებელია არსებული მასივის დამოკლება, რომელიც არც ისე ხშირად გამოყენებადი ამ დანიშნულებით. მაგ.:

|  |
| --- |
| var arr =[1,2,3,4,5];    arr.length =2;// შემცირდა 2 ელემენტრამდე alert( arr );// [1, 2]    arr.length =5;// შევეცადოთ დავაბრუნოთ მასივის საწყისი სიგრძე alert( arr[3]);// undefined: სამწუხაროდ მცდელობამ უშედეგოდ ჩაიარა |

მასივივის ელემენტების უწყვეტ რეშიმში ამოსაღებად შესაძლებელია უკვე ცნობილი რომელიმე ციკლის ოპერატორის გამოყენება, რომელიც შეასრულებს იმდენ იტერაციას რამდენიც იქნება მასივის სიგრძის (arr.length) .ეს ყოველივე უზრუნველყოფს ყველა ელემენტთან წვდომას. მაგ.:

var arr =['მარწყვი' , true , null, 'კივი', 155];

for(var i =0; i <arr.length; i++){ alert(arr[i]); // alert('მარწყვი'), alert(true), ….

}

ციკლების განხილვისას აღვნიშნეთ, რომ არსებობს for\_ის სახეცვლილი ჩანაწერი რომელსაც იმ კონკრეტულ ეტაპზე არ შევეხეთ. ეს არის for…in კონსტრუქცია, რომელიც გამოიყენება ობიექტის ყველა მნიშვნელობის გადასათვლელად. არსებულ შემთხვევაში ზედა მააგალითის პირობის გასამარტივებლად ეს კონსტრუქიცა იდეალური ვარიანტი იქნებოდა, სადაც key მნიშვნელობა ხოლო arr არსებული მასივი.

var arr =['მარწყვი' ,true,null, 'კივი', 155];

for(var key in arr){ alert( arr[key]);// 'მარწყვი' , true , null, 'კივი', 155; }

გარდა ერთგანზომილებიანი მასივებისა Javascript\_ში გვხვდება მრავალგანზომილებიანი მასივები. იმ შემთხვევაში თუ მასივის ინდექსებზე მნიშვნელობებად ასევე მასივები მოგვევლინებიან მაშინ საქმე მრავალგანზომილებიან მასივებთან გქოდეთ. ამ მეთოდის გამოყენებით შესაძლოა მაგ. მატრიცის მიღება. (ვიზუალისათვის წარმოდგენილია სხვადასხვა ხაზზე, არსებილი ერთ ხაზზე განლაგეც არ წარმოადგენს შეცდომას) მის კონკრეტულ ელემენტთან წვდომა შესაძლებელია მთავარი და ქვე მასივის მასივის ინდექსის მეშვეობით

|  |  |
| --- | --- |
| var matrix =[  [1,2,3],  [4,5,6],  [7,8,9]  ];    alert( matrix [1] [2]); | // 6 |

გარდა length თვისებისა არსებობს რიგი ფუნქციების / მეთოდებისა, რომლებიც აქტიურ რეჟიმში გამოიყენებიან მასივებთან სამუშაოდ. იხ ცხრილი VI

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | | ცხრილი VI | |
|  | var arr =['მარწყვი' , 'ატამი','ხილი']; | | |  |  |
| მეთოდი | | აღწერა | მაგალითი | შედეგი | |
| toString() | | ტექსტურ მონაცემად გარდაქმნა | arr.toString() | მარწყვი , ატამი , ფორთოხალი | |
| join() | | მნიშვნელობების შეერთება | arr.join( \* ) | მარწყვი \* ატამი \* ფორთოხალი | |
| pop() | | ბოლო ელმენტის ამოჭრა | arr.pop() | 'მარწყვი' , 'ატამი' | |
| push() | | ელემენტის/ელემენ ტების დამატება მასივის ბოლოში | arr.push('ვაშლი') | 'მარწყვი' , 'ატამი','ხილი', 'ვაშლი' | |
| shift() | | პირველი ელმენტის ამოჭრა | arr.shift('კივი') | 'კივი','მარწყვი' , 'ატამი','ხილი'; | |
| unshift() | | ელემენტის/ელემენ ტების დამატება მასივის ბოლოში | arr.unshift() | 'ატამი','ხილი' | |
| splice() | | შესაძლოა ელემენტების ჩამატება  კონკრეტულ ინდექსზე, არსებულის ამოჭრა ან ჩანაცვლება | arr.splice(0, 2 , 'კივი') | 'კივი','ხილი' | |
| arr.splice(1, 0 , 'კივი') | 'მარწყვი' ,'კივი', 'ატამი','ხილი' | |
| arr.splice(1, 2) | 'მარწყვი' | |
| slice() | | მასივის ინდექსების კონკრეტული დიზაპაზონიდან მნიშვნელობების  ამოღბა | var arr2=arr.slice(0,2) | 'მარწყვი' ,'ატამი' | |
| sort() | | სორტირება / დალაგება | arr.sort() | 'ატამი','მარწყვი','ხილი' | |
| reverce() | | შეტრიალება | arr.reverce() | 'ხილი','ატამი','მარწყვი' | |
| concat() | | რამოდენიმე მასივის შეერთება | arr2= ['კივი' , 'ბუ']; | 'მარწყვი' , 'ატამი','ხილი','კივი' , 'ბუ' | |
| arr.concat(arr2) |

### 5.4 მზა ფუნქციების გამოყენება

|  |
| --- |
| მიმდინარე პარაგრაფის თემატიკა   * დროისა და თარიღის ფუნქციები * სტრიქონული ფუნქციები * ბრაუზერთან სამუშაო ფუნქციები * დოკუმენტთან სამუშაო ფუნქციები * ტაიმერის ფუნქციები |

ობიექტი ამომავალი წერტილია პროგრამირებაში. ყველაფერი მის გარშემო კრავს მოქმედ ჯაჭვს. რალურ რეჟიმში კონკრეტულ ობიექტთან მუშაობის პროცესი საკმაოდ საინტერესო, რამეთუ დიდი აბათობით ობიექტების უმრავლესობა საკმაო ინფორმაციისა და ფუნქციონალის მატარებელია.

ძირითად ობიექტებს, რომლებიც Javascript\_ში აქტიურად გამოიყენება საკუთარი მზა თვისებებისა და მეთოდების / ფუნქციების ერთობლიობა გააჩნია, რომლებიც საკმაოდ ამარტივებს ობიექტთან მუშაობას და ხელს უწყობს დეველოპერს მარტივი ფუნქციონალის გამოყენებით ისეთი შედეგის მიღებაში, რომლის თავად შესაქმნელად შესაძლოა არც ისე მცირე დროითი რესურსის დახარჯვა მოუხდეს.

არსებულ ქვეთავში შეძლებისდაგვარად მაქსიმალური რაოდენობის მეთოდების გარჩევა მოხდება. და თუ შემთხვევით მათ გარდა კიდევ სხვა მეთოდებს მიაკვლიეთ არ იფიქროთ მორიგ მსოფლიო დონის აღმოჩენასთან გქონდეთ შეხება. გამომდინარე იქიდან, რომ ფუნქციათა რაოდენობა არც ისე მცირეა მათი სრულყოფილი ახნა ფაქტობრივად შეუძლებელია.

String\_თან მუშაობის მეთოდები

იმედია შეხსენება არ გჭირდებათ, რომ Javascript\_ში არსებობს ტექსტუალური ტიპის ინფორმაცია. ერთის შეხედით რას წარმოადგენს იგი? String ეს არის სიმბოლოთა ერთობლიობა.

var txt ='21\_ე საუკუნე ინფორმაციული ტექნოლოგიების ერადაა მიჩნეული';//ლიტერალი / მნიშვნელობა

var txt =new String('21\_ე საუკუნე ინფორმაციული ტექნოლოგიების ერადაა მიჩნეული'); //ობიექტი

არსებული შექმნის მეთოდებიდან პირველი მეტად პრიმიტიულია. ამასთანავე Javascript\_ში ლიტერარს შეუძლია ობიექტის მეთოდების მიღება და მათთან მუშაობა, საჭიროების შემთხვევაში ინტერპრეტატორი ავტომატურად ახდენს მის გარდაქმნას ობიექტად.

თუ კარგად დაუკვირდებით შეატყოფთ, რომ ტექსტი შედგება სიმბოლოებისაგან, ე.ი ის თავის მხრივ წარმოადგენს ერთგვარ მასივს. შესაბამისად მასივის თვისება length არსებულო ობიექტის თვისებასაც წარმოადგენს. ის განსაზღვრავს ტექსტის სიგრძეს.

var txt ='21\_ე საუკუნე ინფორმაციული ტექნოლოგიების ერადაა მიჩნეული'; //ლიტერალი / მნიშვნელობა

alert ( txt.length ) //55

მასივის მსგავსად ტექსტის კონკრეტულ სიმბოლოსთან წვდომა შესაძლებელია მის ინდექსზე მიმართვით

var txt ='21\_ე საუკუნე ინფორმაციული ტექნოლოგიების ერადაა მიჩნეული';

alert ( txt[ 0 ] ) //2

alert ( txt[ 3 ] ) //ე

რა თქმა უნდა გარდა ამ მსგავსებისა მასივთან String\_ს გააჩნია მეთოდთა ერთობლიობა რომელიც მოცემულია ცხრილ VII

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | | ცხრილი VII | |
|  | var txt ='21\_ე საუკუნე ინფორმაციული ტექნოლოგიების ერადაა მიჩნეული'; | | |  |  |
| მეთოდი | | აღწერა | მაგალითი | შედეგი | |
| charAt() | | კონკრეტულ ინდექსზე არსებული მნიშვნელობის გამოტანა, იდენტურია txt[…] ჩანაწერისა | txt.charAt(3) | ე | |
| charCodeAt() | | მისი საშუალებით შესაძლებელია დადგინდეს სიმბოლოს კოდური რიცხვითი მნიშვნელობა ASCII | txt.charCodeAt(3) | 4608 | |
| String.fromChar-  Code() | | რიცხვითი მნიშვნელობის შესატყვისი სიმბოლოს მიღება | String.fromCharCode(97) | a | |
| concat() | | შეერთება ერთი ან  რამოდენიმე ტექსტური ინფორმაციის | var txt2 ='ამბობენ, რომ ’ | ამბობენ, რომ 21\_ე საუკუნე ინფორმაციული ტექნოლოგიების ერადაა მიჩნეული | |
| txt.concat(txt2) |

|  |  |  |  |
| --- | --- | --- | --- |
| indexOf() | ახორციელებეს სიმბოლოს ძებნას ტექსტში და გამოაქვს მისი პირველი დამთხვევის ინდექსი, სიმბოლოს ვერ მოძებნის შემთხვევაში პასუხია ( -1 ) | txt.indexOf('საუკუნე') | 5 |
| lastIndexOf() | ახორციელებეს სიმბოლოს ძებნას ტექსტში და გამოაქვს მისი უკანასკნელი დამთხვევის ინდექსი.(ძიება ბოლოდან) სიმბოლოს ვერ მოძებნის შემთხვევაში პასუხია ( -1 ) | txt.lastIndexOf('ულ') | 52 |
| replace () | ტექსტის ჩანაცვლება | txt.replace('21\_ე','XIX') | XIX საუკუნე ინფორმაციული ტექნოლოგიების ერადაა მიჩნეული |
| search() | ფაქტობრივად მუშაობს ისე როგორც მეთოდი indexOf() | txt.search('საუკუნე') | 5 |
| slice() | იღებს ტექსტიდან კონკრეტულ მონაკვეთს მითითებულს ფუნქციის არგუმენტებად | txt. slice(5,11) | საუკუნ |
| substring() | იდენტურია ფუნქციისა  slice() | txt.substring (5,11) | საუკუნ |
| substr() | იღებს ტექსტიდან კონკრეტულ მონაკვეთს პირველი არგუმენტი საწყისია მეორე რაოდენობა | txt.substr(5,11) | საუკუნე ინ |
| toUpperCase() | რეგისტრის ცვლილება ზედა ინდექზე. ქართულ ტექსტზე აზრი არ აქვს გამოყენებას | var txt2 = 'hello' txt2.toUpperCase() | HELLO |
| toLowercase() | რეგისტრის ცვლილება ქვედა ინდექზე. ქართულ ტექსტზე აზრი არ აქვს გამოყენებას | var txt2 = 'HEllo' txt2. toLowercase() | hello |

ტექსტური მნიშვნელობის გარდაქმნა რიცხვით მნიშვნელობად

არსებობს რიგი ფუნქციების რომლებიც უზრუნველყოფენ ტექსტური ტიპის ინფორმაციის გარდაქმნას რიცხვითტიპად. რა თქმენა უნდა თუ ტექსტუარი ტიპის ცლადიდან შესაძლებელია რიცხვითი მნიშვნელობის მიღება, წინააღმდეგ შემთხვევაში მიიღება NaN .არსებული მეთოდებია:

* parseInt()
* parseFloat()
* Number(); მოკლედ მიმოვიხილოდ ისინი:

მეთოდი parseInt() უზრუნველყოფს ტექსტიდან მთელი რიცხვის გამოყოფას, მხოლოდ იმ შემთხვევაში თუ ტექსტი იწყევა რიცხვითი მნიშვნელობით,წინააღმდეგ შემთხვევაში ნებისიმერი სიმბოლო აღიქმება არარიცხვით ტიპად და შედეგი იქნება NaN

|  |  |
| --- | --- |
| alert(parseInt('21\_ე საუკუნე')) | //21 |
| alert(parseInt('21.55')) | //21 |
| alert(parseInt('ტექსტით დაწყებული 21.55')) | //NaN |
| მეთოდი parseFloat() იღებს ათწილად მნიშვნელობებს. მაგ.: |  |
| alert(parseInt('21\_ე საუკუნე')) | //21 |
| alert(parseInt('21.55')) | //21.55 |
| alert(parseInt('ტექსტით დაწყებული 21.55')) | //NaN |

ხოლო მეთოდი Number() ახდენს მხოლოდ იმ ტექსტის დაყვანას როგორც მთელ ასევე აღწილად მნიშვნელობად რომელიც შედგება მხოლოდ რიცხვითი მნიშვნელობისაგან და ასევე წერტილისაგან „.“

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| alert(Number(true)) |  |  |  | //1 |
| alert(Number(false)) |  |  | //0 |  |
| alert(Number('21')) |  |  |  | // 21 |
| alert(Number('21.55')) |  |  | // 21.55 |  |
| alert(parseInt('21 55')) |  |  | //NaN |  |
| alert(parseInt('ტექსტით დაწყებული 21.55')) |  |  | //NaN |  |

დროისა და თარიღის ფუნქციები

Javascript დროსთან სამუშაოდ იყენებს ობიექტ Date(). მიმდინარე დროის შესაქმნელად გამოიყენება new Date() ობიექტი.

var now =newDate();

alert( now );

თუ კონსტრუქტორს გადაეცემა ერთი რიცხვითი არგუმენტი ის აღიქმება, როგორც მილიწამის (1/1000 წამი ) მნიშვნელობა. თარიღის ამთვლელ წერტილად ნაგულისხმევია 1970 წლის 1 იანვარი GMT+0 (დროითი სარტყლის აღმნიშვნელია)

|  |
| --- |
| // 24 საათი 01.01.1970 GMT+0 წლის შემდეგ var getD =newDate(1000\*60 \* 60 \*24);  // მილიწამი \* 1000 = 1 წამი  // 60 \* 60 = 1 საათი // .... \* 24 = 1 დღე alert(getD); // შედეგი 02.01.19702 იანვარი 1970 წლის |

დროის შექმნა შესაძლებელია სხვა არსებული პარამეტრების გადაწოდებით. სადაც წინასწარ არის განსაზღვრული შემდეგი თანმიმდევრობით:

|  |
| --- |
| var now =newDate(წელი, თვე, რიცხვი,საათი, წუთი, წამი, მილიწამი);    // 1 იანვარი 2016, 00:00:00 var date =newDate(2016,0,1,0,0,0,0);    // შედეგი იგივეა საათი,წუთი,წამი,მილიწამი ავტომატურად 0\_ად არის განსაზღვრული var date =newDate(2016,0,1);      // დროის შექმნა მილიწამების სიზუსტით  var date =newDate(2016,0,1,2,3,4,567);// 1.01.2016, 02:03:04.567 |

როგორც მაგალითიდან ჩანს აუცილებელი პირობაა:

* წელი შედგებოდეს 4 სიმბოლოიანი ჩანაწერისაგან ( 2016 და არა 16)
* თვის ათვლა იწყება ნულიდან და არა ერთიდან ( იანვარი - 0 )

გასათვალისწინებელია ისიც, რომ მოცემილი სტილით დროის შექმნისას აუცილებელი პირობა მინიმუმ 2 არგუმენტის (წელი, თვე) გადაცემაა. დანარჩენი არგუმენტების არ მიწოდების შემთხვევაში ისინი ავტომატურად 0\_ებად აღიქმებიან და არაფერი დაშავდება.

დაიმახსოვრეთ Javascript\_ში დროის გამოსახვა ლოკალური დროის მიხედვით ხდება და არა UTC

(Universal Coordinated Time), ანალოგიურია GMT ( Greenwich Mean Time )

დროით ობიექტს გააჩნია მთელი რიგი მეთდებისა, რომელიც უზრუნველყოფს არსებული დროის კონკრეტული მნიშნველობის წარმოჩინებას. მაგ: სასურველია ინფორმირებულნი ვიყოთ მხოლოდ მიმდინარე წლის შესახებ და საერთოდ არ არის საჭირო მთილიანი დროის (წელი,თვე,რიცხვი, საათი,წუთი,წამი,მილიწამი,) გამოსახვა.

ძირითადად დროით ობიექტის მეთდები ორი ფაქტობრივად ერთგვაროვანი სახითაა წარმოდგენილი, რაც უზრუნველყოფს როგორც ლოკალურ ასევე UTC დროსთან მუსაობას. ასევე არსებობს მეთოდთა ერთობლიობარომელიც უზრუნველყოფს ობიექტიდან მონაცემების მიღებას და პირიქით - მინიჭებას. მათგან get… თავსართით დაწყებყლი მეთდები შედეგს იღებს ობიექტიდან, ხოლო set… უზრუნველყოფს მინიჭებას.

ცხრილში VIII მოცემულია დროის ობიექტის მეთოდები განმარტებებითურთ.

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | ცხრილი VIII |
| მაგ.მიმდინარე დრო არის varnow=newDate();1 იანვარი 2016, 02:23:14.557 | | |  |
| მეთოდი | აღწერა | მაგალითი | შედეგი |
| getFullYear() | მიიღება 4 ციფირანი წელი | now.getFullYear() | 2016 |
| getMonth() | გამოაქვს თვე ( გაითვალისწინეთ საწყისი თვე 0\_ით სიმბოლირდება ) | now.getMonth() | 0 |
| getDaTe() | მიიღება თვის დღე  1\_დან 31\_მდე | now.getDay() | 1 |
| getHours() | მიიღება საათი | now.getHours() | 2 |
| getMinutes() | მიიღება წუთი | now.getMinutes() | 23 |
| getSeconds() | მიიღება წამი | now.getSeconds() | 14 |
| getMilliseconds() | მიიღება მილიწამი | now.getMilliseconds() | 557 |
| setFullYear() | წლის მინიჭება, შესაძლოა გადაეცეს თვეც და რიცხვიც | now.setFullYear(2022) | 1 იანვარი 2022,  02:23:14.557 |
| setMonth() | თვის მინიჭება, შესაძლებელია რიცხვისაც | now.setMonth(5) | 1 ივნისი2016, 02:23:14.557 |
| setDate() | რიცხვის მინიჭება | now.setDate(31) | 31 იანვარი  2016,02:23:14.557 |
| setHours() | მიენიჭება საათი, შესაძლოა წუთი,წამი,მილიწამიც | now.setHours(5) | 31 იანვარი  2016,05:23:14.557 |
| setMinutes() | ენიჭება წუთი, შესაძლოა წამი, მილიწამიც | now.setMinutes(35) | 31 იანვარი  2016,02:35:14.557 |
| setSeconds() | ენიჭება წამი, შესაძლოა მილიწამიც | now.setSeconds(19) | 31 იანვარი  2016,02:23:19.557 |
| setMilliseconds() | ენიჭება მილიწამი | now.setMilliseconds(2) | 31 იანვარი 2016,02:23:14.2 |

აღსანიშნავია: ცხრილში ჩამოთვლილ ყველა მეთოდს გააჩნია მსგავსი მორგებული UTC\_სარტყელზე. მაგ: getUTCFullYear().

ბრაუზერთან სამუშაო ფუნქციები

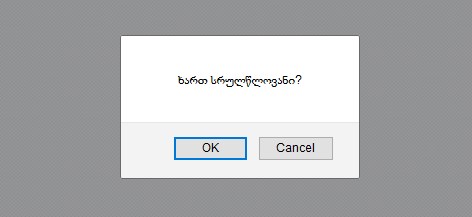
Javascript\_ში window წარმოადგენს როგორც გლობალურ ელემენტს ასევე ბრაუზერის ფანჯარას. რეალურ რეჟიმში არსებულმა ობიექტმა განიცადა სრულუყოფა და დროთა განმავლობაში მიიღო უამრავი დამატებითი ფუნქცია და მეთოდი. ბრაუზერთა სწრაფმა განვითარებამ შეცვალა დამოკიდებულება window ობიექტის ბევრი თვისებისადმი, გამომდინარე იქიდან რომ სხვადასხვა ბრაუზერი სხვადსახვა მეთოდებთან განსხვავებული თავსებადობით გამოიხატება. მაგრამ ეს სულაც არ ნიშანვს window ობიექტის მეთოდების ცოდნიაგან თავის არიდებას.

Javascript\_ის პირველივე შემხებლობისას გავარჩიეთ გამოტანის ოპერატორი alert(). თუ იმ ლოგიკიდან გამოვალთ, რომ window წარმოადგენს სუპერ გლობალურ ობიექტს მაშინ ჩანაწერი window.alert() არ უნდა იყოს ლოგიკას მოკლებული. სინამდვილეში ასეცაა. მაგარამ გამომდინარე იქიდან, რომ window ყველა ობიექტის მშობელია სიმარტივის მიზნით მისი გამოუყენებლობა დასაშვებია. გარდა alert() მეთოდისა window ობიექტს გააჩნია რიგი მეთოდებისა, რომელიც აქტუალურობას არ კარგავს მიმდინარე ეტაპზეც. მათი სრული სია შეგიძლიათ იხილოთ შემდეგ მისამართზე <http://www.w3schools.com/jsref/obj_window.asp>. ჩვენს მიერ კი განსხილული იქნება რამოდენიმე მათთაგანი.(გამოყენებული იქნება სრული ჩანაწერი)

window.print()

არსებული ფუნქცია უზრუნველყოფს მითითებული ადგილის, არეალის ბეჭვდვას.

confirm()

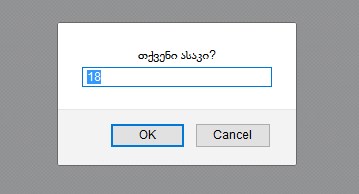
არსებობს დიალოგური ფანჯარა, რომლის მეშვეობითაც შესაძლებელია შეტყობინების გამოტანა ეკრანზე. (იხ.სურ.1.4.1) მისი სინტაქსი შემდგომია confirm("შეკითხვა"), დიალოგურ ფანჯარას გააჩნია ორი ღილაკი OK და Cencel. რომელთა გააქტიურებაც რეზულტატში გვაძლევს

OK->true , Cencel->false.

1.4.1 დიალოგური ფანჯარა confirm()

prompt()

დიალოგური ფანჯარა prompt() შედგება 2 ნაწილისაგან ეს არის ტექსტის გამოტანის არეალი ინფორმაციის შესაყვანი ფორმა (იხ.სურ. 1.4.2). prompt("თქვენი ასაკი","18"). მეორე მნიშვნელობა default\_ად სიცარიელეს წარმოად-გენს და მისი საერთოდ არ მითითებით არაფერი დაშავდება. prompt("თქვენი ასაკი")



1.4.2

.

prompt

დიალოგური

ფანჯარა

window მეთოდებიდან გამოსაყოფია window.open(), რომელიც ახალი ფანჯრის გახსნის ფუნქციონალს წარმოადგენს. მისი სრულყოფილი სინტაქსი შემდეგი საახისაა window.open("საიტის URL მისამართი","გახსნილი ფანჯრის სახელი","პარამეტრები"). window.open() მეთოდს შესაძლოა გადაეწოდოს მხოლოდ საიტის მისამართი დამატებითი პარამეტრებისა და სახელის გარეშე. ის უპრობლემოდ იმუშავებს.

window.open("http://google.ge");

არსებული მეთოდის ასეთი სახით გამოყენებისას გაიხსნება ახალი ფანჯარა ბრაუზერის ახალი ჩანართში - ტაბში . თნამედროვე ბრაუზერების უმეტეს ნაწილში მეთოდის ამგვარი ჩანაწერი ფანჯარას სწორედაც, რომ ახალ ჩანართში ხსნის და არა დამოუკიდებელ popup ფანჯრად.სრულყოფილი სინტაქსი უზრუნველყოფს ფანჯრის დამატებითი პარამეტრეის განსაზღვრას, რაც თავისთავად შესაძლებელს ხდისგახსნილი ფანჯარა დამოუკიდებელი ზომებით და პარამეტრებით მოგვევლინოს.

win = window.open(url, name, params)

url - ის წარმოადგენს გასახსნელი ფანჯრის მირამართს.

name- მიეთითება ფანჯრის სახელი. რამდენიმე ფანრის გახსნის მცდელობისას თუ სახელი ერთი იქნა ახალ ფანჯარაში ჩანაცვლდება საიტის ვიზუალი და ყოველ ჯერზე ახალში არ გაიხსნება.

params - პარამეტრები უზრუნველყოფს ფანჯრის სრულყოფას. აღსანიშნავია, რომ დღეს მათი ნაწილი სხვადასხვა ბრაუზერებში სხვადასხვანაირად გამოისახება, შესაძლოა ზოგმა საერთოდ არ იმუშაოს. ასე რომ ყურადღებით გაარჩიეთ დამოუკიდებლად სხვადასხვა პარამეტრის თავსებადობა ბრაზერებთან.იხ. ცხრილი IX

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | ცხრილი IX. |
| N | მეთოდი | აღწერა | მნიშვნელობა |
| 1 | left | ფანჯრის დაშორება ეკრანის / ბრაუზერის მარცხენა კიდიდან | რიცხვი |
| 2 | top | ფანჯრის დაშორება ეკრანის / ბრაუზერის ზედა კიდიდან | რიცხვი |
| 3 | width | ფანჯრის სიგანე | რიცხვი |
| 4 | height | ფანჯრის სიმაღლე | რიცხვი |
| 5 | menubar | ბრაუზერის მენიუს ზოლის დამალვა/გამოჩენა | yes/no |
| 6 | toolbar | ნავიგაციის პანელის დამალვა / გამოჩენა | yes/no |
| 7 | location | საიტის URL სამისამართე ველის დამალვა / გამოჩენა | yes/no |
| 8 | statusbar | სტატუსის ველი | yes/no |
| 9 | resizable | ბრაუზერის ფანჯრის დაპატარავება / გაზრდა | yes/no |
| 10 | scrollbar | სქროლის დამალვა / გამოჩენა | yes/no |

როგორც აღვნიშნეთ მოცემული პარამეტრები დღევანდელ ბრაუზერებში, არც ისე სრულყოფილად მუშაობს, მაგრამ თუ ახალი ფანჯრის გახსნა არა ბრაუზერის ჩანართად (ტაბი) ჩამატება არამედ ფანჯრად ვიზუალიზება გსურთ აუცილებელია width და height პარამეტრების მინიჭება.

var newWin = window.open("http://google.ge/","google","width=200,height=200");

გახსნილი ფანჯრის დასახურად გამოიყენება window.close() ფუნქცია, რომელიც სხვადასხვა ბრაუზერებში განსხვავებულად მოქმედებს აქტიური ფანჯრის პირობებში, ზოგან ხურავს ავტომატურად, ზოგან გამოაქვს შეკითხვის ფორმა დაახლოებით „დარწმუნებულნი ხართ რომ ნამდვილად გინდათ ფანჯრის დახურვა?“ newWin - ფანჯრის დასახურად გამოიყენეთ შემდეგი კონსტრუქცია newWin.close();

არსებობს ასევე ფანჯრის გადატანის / გადაადგილების, ზომის ცვლილებისა და სქროლის ფუნქციები მეთოდები

* win.moveBy(x,y) - ფანჯრის მოძრაობა - წანაცვლება არსებული პოზიციიდან x , y მნიშვნელობების შესაბამისად. მაგ. ფანჯრის საწყისი მდებარეობაა left-0; top-0; win.moveBy(100,100) მეთოდის გამოყენებთ პირველ ნაბიჯზე ფანჯარა განთავსდება left-100px; top-100; შემდეგი მოქმედებისას left-200px; top-200; და ა.შ

* win.moveTo(x,y) - ფანჯრის გადაადგილება ეკრანის / ბრაუზერის x, y კოორდინატებზე. მაგ. ფანჯრის საწყისი მდებარეობაა left-0; top-0; ხოლოwin.moveTo(100,100) მეთოდის გამოყენებთ ფანჯარა განთავსდება ფიქსეირებულleft-100px;top-100;

* win.resizeBy(width,height) - მითითებული ფუნქციით შესაძლებელია არსებული ფანჯრის ზომების ცვლა მისი საწყისი ზომიდან გამომდინარემრავალჯერადად

* win.resizeTo(width,height) მითითებული ფუნქციით შესაძლებელია არსებული ფანჯრის ზომის ცვლა ერთჯერადად.

* win.resizeTo(width,height) მითითებული ფუნქციით შესაძლებელია არსებული ფანჯრის ზომის ცვლა ერთჯერადად.

* win.scrollBy(x,y) მითითებული ფუნქციით შესაძლებელია სკროლის გადატანა x, y პოზიციებზე მრავალჯერადად

* win.scrollTo(x,y) მითითებული ფუნქციით შესაძლებელია სკროლის გადატანა x, y პოზიციებზე ერთჯერადად

object screen

მომხმარებლის მონიტორის რეზოლუციის დასადგენად გამოიყენება ობიექტი screen. მისი ჩაწერა

დასაშვებია, როგორც window ობიექტთან, ასევე მის გარეშე :

screen window.screen

screen ობიექტის თვისებათა ერთობლიობაწარმოდგენილია ცხრილ X:

|  |  |  |
| --- | --- | --- |
|  |  | ცხრილი X. |
| N | მეთოდი | აღწერა |
| 1 | screen.width | გამოაქვს მონიტორის სრული სიგანე |
| 2 | screen.height | გამოაქვს მონიტორის სრული სიმაღლე |

გამოაქვს მხოლოდ გამოყენებადი არეალის სიგანე ( მაგ.: თუ სტატრის

1. screen.availWidth ზოლი ან დამატებითი ინსტრუმენტების პანელი გამოყენებულია მარჯვენა ან მარცხენა მხარეს )

გამოაქვს მხოლოდ გამოყენებადი არეალის სიმაღლე ( მაგ.: თუ სტანდარტულ მდგომარეობაშია windows ინსტრუმენტების პანელი

1. screen.availWidth განთავსებულია ქვედა პანელზე და მისი სიმაღლეა 40 px რაც აკლდება სრულ სიმაღლეს )
2. screen.colorDepth გამოაქვს ფერთა პალიტრის ბიტური მნიშვნელობა

object Navigator

ობიექტიNavigator შეიცავს ინფორმაციას მომხმარებლის ბრაუზერის შესახებ. არსებული ობიექტის

გამოყენებით შესაძლებელია დადგენა თუ რომელ ბრაუზერს იყენებს მომხმარებელი. navigator ობიექტსგააჩნია შემდეგი თვისებები (იხ. ცხრილ XI):



object History

ობიექტიHistory ინახავს გვერდების ნავიგაციის მონაცემებს. მას გააჩნია 2 მეთოდი, რომელიც უზრუნველყოფს წინა და მომდევნო მისამართებზე გადასვლას:

* history.back() - უკან
* history.forward() - წინ



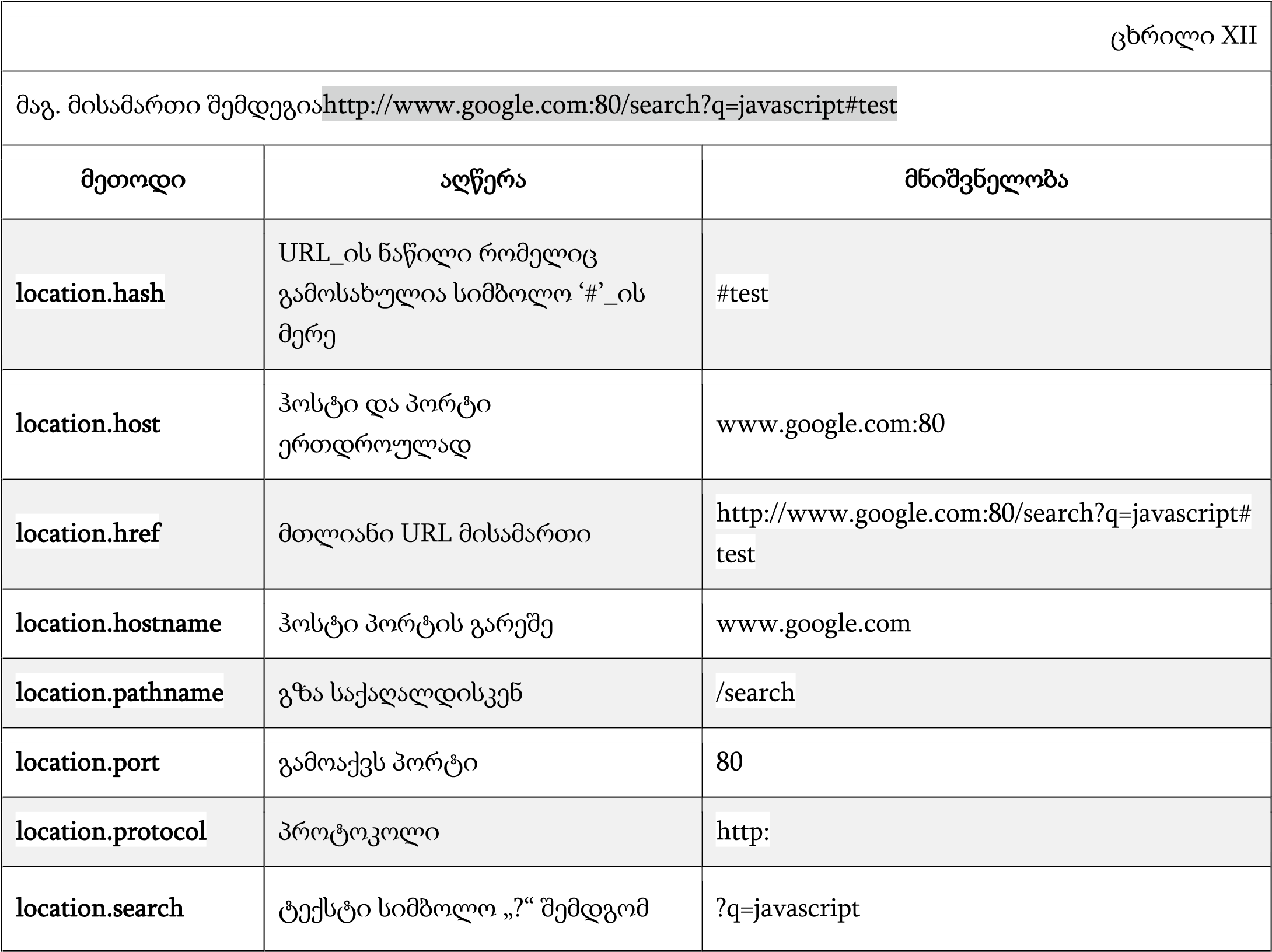
1.5. სანავიგაციო ისრები

ისინი იდენტურ მოქმედებებს ასრულებელ რასაც მოიცავს სურათ 1.5\_ზე მონიშნული ისრები.

object Location

Location ობიექტი უზრუნველყოფს სამისამართე პანელი ინფორმაციის მართვას. მისი ფუნქციაა მიიღოს / მიანიჭოს URL მისამათი და მისი კომპონენტები.

Location ობიექტს გააჩნია რიგი მეთოდებისა და თვისებებისა. ცხრილ XII მოცემულია იმ თვისებების ჩამონათვალი, რომესაც იყენებს Location ობიექტი



გარდა თვისებებია Location ობიექტს გააჩნია შემდეგი მეთოდები:

* location.assign(url) - გადაეცემა url მისამართი, რომელზეც ახდებს გადამისამართებას. ახდენს მითითებული მისამართის დოკუმენტის ჩატვირთვას
* location.replace(url) - ახდენს არსებული დოკუმენტის ჩანაცვლებას იმ დოკუმენტით რომელიც იტვირთება მითითებული url\_ის შემთხვევაში. ( დოკუმენთში მოიაზრება საიტის გვერდი ). replace() მეთოდი assign()\_სგან განსხვავდება იმით, რომ ის არ ინახება ისტორიაში შესაბამისად replace() მეთოდით გახსნილ გვერდს არ გააჩნია back უკან დაბრუნების ღილაკი ის დეაქტივირებულია.
* location.reload() - არსებული მეთოდის მეშვეობით შესაძლებელია მიმდინარე გვერდის განახლება. შესაძლოა მეთოდს გადავცეთ ატრიბუტი true ან false . თუ მნიშვნელობა იქნება true განიცდის განახლებას ყოველთვის სერვერიდან, ხოლო false . ის შემთხვევაში ბრაუზერის ქეშიდან.

ტაიმერის ფუნქციები

ზემოთმულ მასალაზე დაყრდნობით შეგვიძლია დავასკვნათ, რომ ობიექტი window საკმაოდ მნიშვნელოვანი ფუნქციების გამაერთიანებელია. ვებ გვერდზე ხშირია მოცემულობა, როცა საჭიროა გარკვეული ინფორმაციის პერიოდული გამეორება, ან რომელიმე ინფორმაციის ერთჯერადად, მაგრამ კონკრეტული დროის შემდგომ ასახვაა საჭირო. window ობიექტის ძირითად მეთოდებად მოიხსენიება „ტაიმერის“ ფუნქციები: setInterval( ) და setTimeout().

ორივე მეთოდის სინტაქსი ანალოგურია განსხვავებულია მათი მუშაობის პრინციპი. setInterval( ) უნრუნველყოფს მოქმედების მრავალჯერადად გამეორებადობას, ხოლო setTimeout() თვისი არსით ერთჯერადად. რა თქმა უნდა შესაძლებელია ისინი იდენტური ფუნქციონალით ვამუშაოთ, მაგრამ ყოველი აზრს არის მოკლებული, რადგანაც ისინი იმიტომ არსებობენ, რომ გამოიყენოთ დანიშნულებისამებრ.

მეთოდის სინტაქსი შემდეგია: ის შედგება ფუნცქიისა და დროის მნიშვნელობისაგან, შესაძლოა გამოყენებულ იქნას არგუმენთებიც. დრო განისაზღვრება მილიწამებად ( 1 წთ - 1000 მილიწამი)

var timerId =setTimeout(func / code, delay[, arg1, arg2...])

განხილვისას შეგვხვდება ტერმინი ფუნქცია და მისი გამოყენება, რომელიც შემდეგ 5.5 თავშია განხილული, უნჯობესია გაურკვევლობის შემთხვევაში ეწვიოთ მითითებულ თავს.

ამოცანა:

საიტის გახსნიდან 2 წთ\_ში გამოგვიტანოს ერთჯერადად alert("მოგესალმებათ საიტი") ფანჯარა შეტყობინებით

ამოცანის გადასაჭრეად საჭიროა setTimeout() ფუნქციის გამოყენება. არსებული დავალების გადასაწყვეტათ გამოვიყენოთ ყველა შესაძლო გზა, რაც გაამარტივებს setTimeout() ფუნქციის არსის გაგებას.

გზა I - შეიქმნას ანონიმური ფუნქცია უშუალოდ setTimeout() მეთოდში

setTimeout( function (){ alert("მოგესალმებათსაიტი") } , 2000 )

ფუნქცია ვიზუალურად საკმაოდ სასიამოვნოდ გამოიყურება, უბრალოდ დიდი კოდის შემთხვევაში ერთობ მოუხერხებელია. ამისათვის უნჯობესია შეიქმნას ცალკე ფუნქცია.

გზა II ბევრად უფრო მისაღები და კლასიკურია

functionshowMessage(){ alert("მოგესალმებათსაიტი")

}

setTimeout(showMessage, 2000 )

არსებული დავალება იმდენად მარტივია შესაძლებელია ფუნქციის გამოყენებისაგან საერთოდ თავის შეკავება და alert() მეთოდის სტრინგად გამოყენება

გზა III

setTimeout("alert("მოგესალმებათსაიტი")", 2000 )

არსებული ფუნქციაც ანალოგურად იმუშავებს, მაგრამ ამგვარი ჩანაწერი მაინც არარეკომენდირებულია.

თუ ფუნქცია იღებს მნიშვნელობად არგუმენტებს მაშინ დროით მნიშვნელობის შემდეგ მძიმის მეშვეობით შესაძლებელია არგუმენტების გადაცემა. მაგ:

|  |
| --- |
| functionsum(a,b,c){ alert(a+b+c)  }    setTimeout(sum, 2000, 5,25,15 ) // 2 წთ \_ ში ეკრანზე გამოდის შედეგი 45 |

არსებული ფუნქციის განულება, მწყობრიდან გამოყვანა შესახძლებელია clearTimeout()ფუნქციის მეშვეობით. რომელსაც არგუმენტად გადაეცემა ფუნქციის სახელი/ცვლადი რომელსაც ის მიენიჭება გამოცხადება/გამოძახებისას:

|  |
| --- |
| functionsum(a,b,c){ alert(a+b+c)  }    var t = setTimeout(sum, 2000, 5,25,15 )    clearTimeout(t) // ეკრანზე შედეგი აღარ გამოვა |

ანალოგური სინტაქტი აქვს setInterval() მეთოდს. მისი შესაძლებელია clearInterval()მეთოდის საშუალებით.

დავალება: ეკრანზე გამოვიდეს მთვლელი, რომელიც 10\_დან უკუთვლით წამოვა 1\_მდე 1 წამის ინტერვალით. და საბოლოოდ გადაამისამართოს გვერდი google.com\_ზე.

var i = 10; functiontimer(){

alert(i)

if (i>1){ i--; }else{ clearInterval(t); location.assign("http://google.com");

}

}

var t = setInterval (timer, 1000 )

DOM

კლიენტზე ორიენტირებული Javascript\_ის შესაძლებლობებშოი შედის სტატიკური HTML დოკუმენტის დინამიურად გარდაქმნა, ასევე ინტერაქციული ეფექტების შემუშავება. მისი მთავარი ორიენტირი ვებ გვერდების შემცველობასთან მუშაობაა. ცალკე საუბრის თემაა ობიექტები,რომლებიც წარმოადგენს უშუალოდ ვეგ ვერდის შემცველობას.

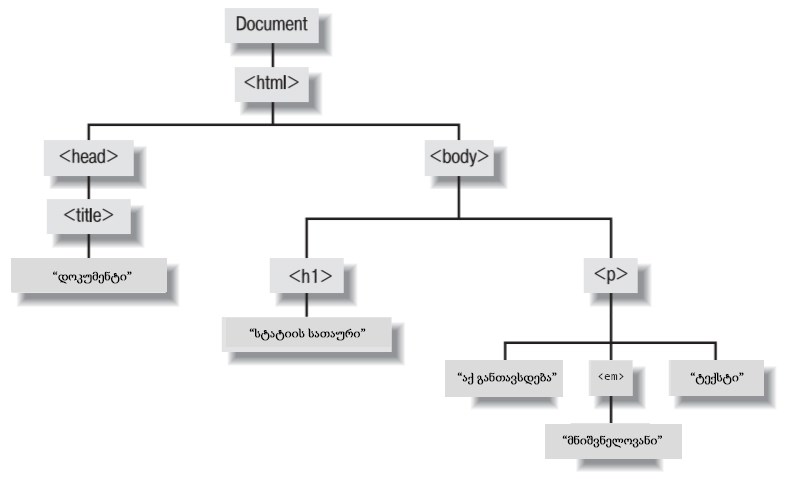
HTML დოკუმენტი შეიცავს ტექსტებს, სურათებს, ჰიპერბმულებს, ფორმის ელემენტებს და ა.შ. Javascript შეუძლია არსებულ ობიექტებთან მუშაობა და მათი მანიპულირება.

DOM (Document Object Model) დოკუმენტის ობიექტური მოდელი - ეს არის დოკუმენტის ობიექტებთან წვდომის განმსაზღვრელი ინტერფეისი.

HTML დოკუმენტი წარმოადგენს ერთმანეთში ჩადგმული ტეგების იერარქიულ სტრუქტურას, რომელიც DOM \_ში წარმოდგენილია, როგორც „ობიექტების ხე“. ხის კვანძები წარმოადგნენ დოკუმენტის სხვადასხვა ტიპის ინფორმაციას, როგორიცაა HTML ელემენტები/ტეგები, ტექსტური ინფორმაცია, კომენტარები და ა.შ

მაგალითზე მოცემულია მარტივი HTML დოკუმენტი რომლის DOM სტრუქტურა წარმოდგენილია სურათ 1.6\_ზე

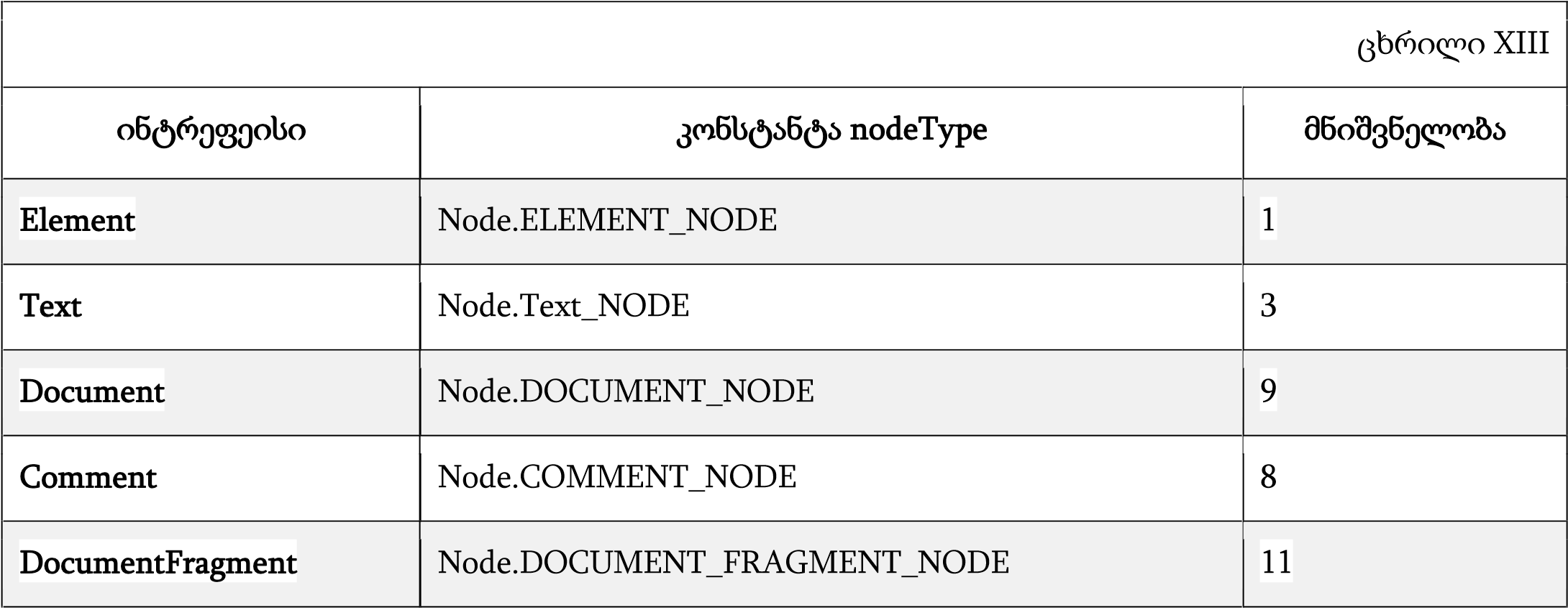
|  |
| --- |
| <html> <head>  <title>დოკუმენტი</title>  </head>  <body>  <h1>სტატიის სათაური</h1>  <p>აქ განთავსდება <em>მნიშვნელოვანი</em> ტექსტი</p>  </body>  </html> |



1.6 დოკუმენტის DOM სტრუქტურა

კვანძები ერთმანეთთან მიმართბაში შეიძლება „გენიალოგიურ“ ხის სტრუქტურას მივამსგავსოთ, სადაც კვანძი შესაძლოა წარმოადგენდეს მშობელ კვანძს (მაგ.:html). კვანძები რომლებიც განლაგებულნი არიან ერთი დონით ქვემოთ მშობელ კვანძებზე იწოდებიან შვილეული კვანძებად (html კვანძსთვის ->head და body). ერთდონეზე მდებარე კვანძებს რომლებსაც გააჩნიათ ერთი მშობელი ერთმანეთისათვის წარმოადგენენ დედმამიშვილ კვანძებს ( მაგ.: head და body ერთმანეთისათვის, ასევე h1 და p )

DOM სტრუქტურაში კვანძი მოხსენიებულია როგორც - Node. სურათ 1.6\_ზე წარმოდგენილია სხვადასხვა ტიპის Node\_ები ( კვანძები ). ესენია: დოკუმენტი, ელემენტი, ტექსტი. ცხრილ XIII წარმოდგენილია ძირითადი კვანძები შესაბამისი ტიპის მნიშვნელობით. სრული სია იხილეთ <http://www.w3schools.com/jsref/prop_node_nodetype.asp>



|  |  |  |
| --- | --- | --- |
| Attr | Node.ATTRIBUTE\_NODE | 2 |

DOM სტრუქტურის ფუძისეულ კვანძს წარმოადგენს Document ობიექტი.შევეცადოთ გადავიდეთ უშუალოდ სხვა ელემენტების მიმართიანობაზე და ვნახოთ როგორ ხდება მათთან წვდომა.

document.documentElementახდენს წვდომას <html>ტეგთან. არსებობს თვისება რომელიც გამოსახავს ეკრანზე კვანძის სახელს nodeName.

|  |
| --- |
| <html> <head>  <title>დოკუმენტი</title>  </head>  <body>  <h1>სტატიის სათაური</h1>  <script>  alert( document.documentElement.nodeName ) // html  </script>  </body>  </html> |

ძირითადად მიმართვიანობა არა html არამედ body ტეგზე ხდება ამისათვის გამოიყენება შემდეგი ჩანაწერიdocument.body

უშუალოდინდივიდუალურადელემენტეზემიმართვისათვიშესაძლოაგამოიყენოთ:

* document.getElementById()- ელემენტზე მიმართვა კონკრეტული id\_ის მიხედვით;
* document.getElementsByClassName()-მიმართვა ელემენტზე class სელექტორის მიხედვით
* document.getElementsByTagName()-ელემენტებზე ტეგების სახელწოდებებით მიმართვა
* document.getElementsByName()- ელემენენტებზე ატრიბუტ name\_ის მიხედვით მიმართვა

გასათვალისწინებელია, რომ getElement მხოლობითში იხმარება მხოლოდ ID\_სთან მიმართებაში, რადგანაც საიტზე შესაძლებელაი მხოლოდერთი განუმეორებელი id მქონე ელემენტის არსებობა. დანარჩენ შემთხვევებში ერთი და იმავე კლასის, ტეგის სახელისა და ატრიბუტ name\_ის ელემენტები მრვალად შეგხვდეს. მათ ოდნავ მოგვინაებით შევეხოთ. იქამდე გავიგოთ არსი უშუალოდ მიმართვიანობისა.

დავალება:

Javascript\_ის მეშვეობით შევიტანოთ ტეგში ტექსტი.

მოცემული დავალების შესასრულებლად საჭიროა მოვახდინოთ ტეგზე მიმართვიანობა, და გამოვიყენოთ მეთოდიinnerHTML მისთვის ინფორმაციის მისანიჭებლად:

|  |
| --- |
| <html> <head>  <title>დოკუმენტი</title> |

</head>

<body>

<h1id="caption"></h1>// საწყის ეტაპზე h1 ცარიელია

<script>

document.getElementById("caption").innerHTML = "საიტის სათაური";

// არსებული ჩანაწერის გამოყენებით ტექსტ - „საიტის სათაური“ ხელოვნურად მოვათავსებთ ტეგ

<h1>\_ში და მიიღება შედეგი <h1id="caption">საიტის სათაური</h1>

</script>

</body>

</html>

კონკრეტული თვისების უმრავლესობა ორმაგი ხასიათისაა და როგორც მინიჭების ასევე არსებული ელემენტის თვისების ამოღების უნარი გააჩნია. მაგ.: იმავე innerHTML\_ის გამოყენება შემდეგ სახეს მოგვცემს

|  |
| --- |
| <html> <head>  <title>დოკუმენტი</title>  </head>  <body>  <h1id="caption">განთავსებულია სათაური </h1>  <script>  alert( document.getElementById("caption").innerHTML )  // არსებული ჩანაწერის გამოყენებით ეკრაბზე გამოვა შედეგი “განთავსებულია სათაური”  </script>  </body>  </html> |

არსებული მოცემულით შესაძლებელია ასევე საკმაოდ მარტივი და საინტერესო დავალების შესრულება.: მოადინეთ ერთ ტეგში არსებული ტექსტის მეორეში კოპირება და ჩანაწერის „ეს საინტერესოა“\_ს თნდართვა;

|  |
| --- |
| <html> <head>  <title>დოკუმენტი</title>  </head>  <body>  <h1id="caption">განთავსებულია სათაური </h1>  <em id="copyText"></em>  <script> document.getElementById("copyText").innerHTML = document.getElementById("caption") .innerHTML + "ეს საინტერესოა"  </script> |

</body>

</html>

შედეგი ბრაუზერში

<h1>

<em>

Javascript\_ის მეშვეობით შესაძლებელია კონკრეტული ელემენტების სტილირება. რა თქმა უნდა სტილების სრულფასობანი გაწერა აზრს მოკლებულია და მიის გამოყენება მხოლოდ კონკრეტული ლოგიკის შესაბამისადაა საჭირო. მაგ. საჭიროა ღილაკზე ხელის დაჭერისას ბლოკის გამოცენა / გაქრობა. რა თქმა უნდა ასეთ შემთხვაში დასაშვებბია სტილების მინიჭება.

არსებობს სტილის მინიჭების ორი გზა. ეს არის ობოექტ styleგამოყენება ან ატრიბუტის მინიჭება. ორივე

შემთხვევაში სტილები ელემენტს ენიჭება ხაზოვანი სტილის მინიჭების მეშვეობით (Inline style):

<h1id="caption">განთავსებულია სათაური </h1>

<script> document.getElementById("caption ").style.color ="red" // h1 ტექსტი გახდება წითლი

</script>

მაგალითზე მოცემული ჩანაწერით შესაძლებელია 1 სტილის მინიჭება და ყოველი ახალი სტილის მიცემისათვის გამოყენებული უნდა იქნას ახალი ჩანაწერი.

<h1id="caption">განთავსებულია სათაური </h1>

<script> document.getElementById("caption ").style.color ="red" // h1 ტექსტი გახდება წითლი document.getElementById("caption ").style.backgroundColor ="#fff44c" // h1 ფონი - ყვითელი </script>

Javascript\_ით სტილების ბრძანენების სრული ჩამონათვალი შეგიძლიათ იხილოთ შემდეგ მისამართზე <http://www.w3schools.com/jsref/dom_obj_style.asp>

DOM სტრუქტურის მეშევეობით შესაძლებელია კონკრეტულ ელემენტის ატრიბუტზე წვდომა, მისი მინიჭება ან კითხვადობა. style წარმოადგენს ელემენტის ატრიბუტს როცა ის გაწერილია ხაზოვან რეჟიმში. თუ ერთდორულად რამოდენიმე სტილის მინიჭება გჭირდებათ სასურველია გამოიყენოთ მეორე გზა.

<h1 id="caption">განთავსებულია სათაური </h1>

<script> document.getElementById("caption ").setAttribute("style","color: red; background-color: #fff44c; font-size: 15px" )

</script>

setAttribute() მეთოდის გამოყენებით შესაძლებელაი ნებისმიერი ატრიბუტი მინიჭება html ტეგებისათვის. არსებული მეთოდი მოითხოვს ორ არგუმენტს: ეს არის ატრიბუტის სახელი რომელსაც ანიჭებთ და მნიშვნელობა რომელიც მიეკუთვნება ამ ატრიბუტს.

მაგ.: setAttribute() მეთოდის მეშვეობით მივანიჭოთ ელემენტს class="logo"

<h1id="caption">განთავსებულია სათაური </h1>

<script> document.getElementById("caption ").setAttribute("class","logo" ) </script>

მიღებულ შედეგში ტეგ h1 ექნება დამატებული ატრიბუტი class

<h1 id="caption" class="logo" >განთავსებულია სათაური </h1>

გარდა ელემენტის ატრიბუტის მინიჭებისა შესაძლებელია მოხდეს ელემენტის ატრიბუტის დადგენა getAttribute() მეთოდის მეშვეობით, რომელიც თავის მხრივ ერთ არგუმენტს საჭიროებს, თუ რომელი ატტრიბუტის მნიშვნელობის დადენა გვინდა:

მაგ.:

<h1 id="caption" class="logo" >განთავსებულია სათაური </h1>

<script> alert(document.getElementById("caption ").getAttribute("class" )// logo

)

</script>

გარდა document.getElementById() მიმართვისა შესაძლებელია ელემენტებზე მიმართვა ტეგის სახელისა, კლასებისა და ატრიბუტ name\_ის მეშვეობით. ისინი თავის მხრივ საიტზე რამოდენიმე ერთფგვაროვანი შესაძლოა იყოს. ამიტომ ისინი აღიქმებიან როგორც მასივი და უშუალოთ მათგან ერთ ან რამოდენიმეზე მიმართვისას საჭიროა დაკონკრეტებია იმ ინდექსისა თუ რომელს უკავშირდებით. მაგ.:

|  |
| --- |
| <html> <head>  <title>დოკუმენტი</title>  </head>  <body>  <div class="container">ტექსტი 3</div>  <h1id="caption">განთავსებულია სათაური </h1>  <h2>განთავსებულია მეორე სათაური </h2>  <p>ვრცელი ტექსტი 1 </p>  <h2>განთავსებულია მესამე სათაური </h2>  <p>ვრცელი ტექსტი 2 </p> |

<div class="container">ტექსტი 5</div>

<div class="about"> ტექსტი კომპანიის შესახებ </div>

<input type="radio" name="gender" />

<input type="radio" name="gender" />

<div class="container">ტექსტი 4</div>

</body>

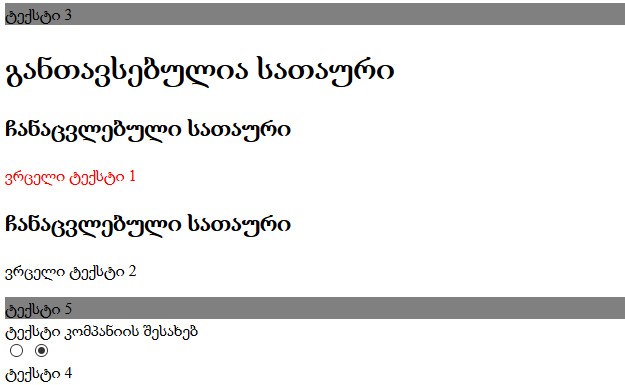
</html>

არსებულ კოდზე დაყრდნობით სასურველია შემდეგი სტილის დოკუმნტის მიღება:

* ყველა h2\_ში გამოისახოს ტექსტი ‘ჩანაცვლებული სათაური’;
* მხოლოდ პირველი <p> ტეგის ფერი გახდეს წითელი;
* საიტზე ნებისმიერ ადგილას განთავსებული პირველი და მეორე class="container"ფონად გამოესახოს სერი ფერი;
* მხოლოდ მეორე რადიო name\_ით genderსაწყის მნიშვნელობადიყოს მონიშნული(checked).

|  |
| --- |
| <script>  // 1 - ყველა h2\_შიგამოისახოსტექსტი ‘ჩანაცვლებულისათაური’;    // გამომდინარე იქიდან რომ არ ვიცით წინასწარ რამდენი ელემენტი h2 იქნება საიტზე გამოყენებული უმჯობესია ციკლის მეშვეობთ ჩავწვდეთ და ყველა h2 მივაგნოთ ეს მეტად დინამიური იქნება :      for(var i =0; i <document.getElementsByTagName("h2").length; i++){  document.getElementsByTagName("h2")[i].innerHTML = "ჩანაცვლებული სათაური"  }    // 2- მხოლოდ პირბელი p ტეგის გერი გახდეს წითელი document.getElementsByTagName ("p")[0].style.color = "red"    // 3- მხოლოდ პირბელი p ტეგის გერი გახდეს წითელი document.getElementsByClassName("container")[0].style.backgroundColor = "gray"; document.getElementsByClassName("container")[1].style.backgroundColor = "gray";    შესაძლოა ეს ჩანაწერი შემდეგნაირადაც გამოისახოს document.getElementsByClassName("container")[0].style.backgroundColor = document.getElementsByClassName("container")[1].style.backgroundColor = "gray";      // 4 - მხოლოდ მეორე რადიო იყოს cheked document.getElementsByName("gender")[1].setAttribute("checked","checked") |

// უფრო მოკლე გზაა document.getElementsByName("gender")[1].checked = true </script>

მოცემილი კოდის მანიპულირებით მიღებული შედეგი გამოსახულია 1.6.2 სურათზე. შეეცადეთ გაარჩიოთ მოცემილი მაგალითი, ააწყოთ თქვენთვის საურველი html დოკუმეტის ჩონჩხი და მოახდინოთ წვდომა ტეგებთან, ცვალოთ მათი სტილები და ინფორმაცია.

1.6.2 სტილშეცვლილი დოკუმენტი

Dom სტრუქტუს მეშვეობით შესაძლებელია ახალი ელემენტების, ტექსტური node\_ების შექმნა მათთვის ატრიბუტების გენერირება და მინიჭება, ასევე უკვე არსებული ელმენტების წაშლა. მიდგომა საკმაოდ ხშირად გამოყენებადია. მისი საშუალებით სესაძლებელია html დოკუმენტის ჩონჩხის ცვლილება.

ახალი ელემენტის შესაქმნელად გამოიყენეთ მეთდი createElement(). ის მოითხოვს ატრიბუტად მხოლოდ ტეგის დასახელებას და თავად ახდენს მის ტეგად გარდაქმნას.

<script>

document.createElement("div");

</script>

მაგრამ ტეგის შექმნა მის ავტომატურ დამატებას html ჩონჩხში არ ნიშნავს, ასევე ის ცარიელია, შევეცადოთ შევუქმნათ მას ტექსტური შემცველობა, მივანიჭოთ ატრიბუთი id="newElement". და მხოლოდ ამის შემდეგ განვათავსოთ საჭირო ადგილას ჩვენს მიერ წინასწარ შექმნილ html დოკუმენტში

|  |
| --- |
| <script>  //<div>ელემენტის შექმნა var elem = document.createElement("div");    // ატრიბუტ id შექმნა  var attr = document.createAttribute("id");    // ატრიბუტ id\_ის მნიშვნელობის განსაზღვრა attr.value = " newElement";    // ატრიბუტის div ელემენტზე მიბმა / მინიჭება elem.setAttributeNode(attr); // არსებულ დონეზე მიღებული შედეგია <div id=’newElement’></div> |

|  |
| --- |
| // ტექსტური კვანძის - ტექსტური ინფორმაციის შექმნა var txt = document.createTextNode("ტექსტი ახალ ელემენტში");    // ტექსტური ინფორმაციის div \_ში განთავსება elem.appendChild(txt) // არსებულ დონეზე მიღებული შედეგია <div id=’newElement’>"ტექსტი ახალ ელემენტში"</div>      // div\_ის ბოლო ელემენტად განთავსება body\_ტეგში document.body.appendChild(elem);  </script> |

მოცემული სკრიპტი შემდეგნაირად იმუსავებს უშუალოდ html დოკუმენტში:

|  |
| --- |
| <html> <head>  <title>დოკუმენტი</title>  </head>  <body>  <div class="container">ტექსტი 3</div>  <h1id="caption">განთავსებულია სათაური </h1>  <div class="about"> ტექსტი კომპანიის შესახებ </div>    // body\_ის უკანასკნელ ელემენტად დააამატებს  //--------------------------------------------------------------------  <div id="newElement"> ტექსტი ახალ ელემენტში </div>  //------------------------------------------------------------  <script> var elem = document.createElement("div"); var attr = document.createAttribute("id"); attr.value = " newElement"; elem.setAttributeNode(attr);  var txt = document.createTextNode("ტექსტი ახალ ელემენტში"); elem.appendChild(txt) document.body.appendChild(elem);  </script>    </body>  </html> |

თუ არსებული ბლოკის არა ბოლოში არამედ კონკრეტულ ადგილას განთავსებაა საჭირო საჭიროა მეთოდ insettBefore() გამოყენება. ის მოითხოვს ორ ატრიბუტს, პირველი ეს არის რას ვსვამტ, მეორე რომელი ბრძანების წინ.

ზედა მაგალითზე დაყრდნობით შექმნილი ბლოკი არა body ტეგის ბოლოში არამედ <div class="container"></div>....................... <h1 id="caption"></h1>ტეგებს შორი განთვსდეს (წერტილების ადგილას).

ამისათვის შაჭიროა scritp\_ის ბოლო ბრძანება document.body.appendChild(elem) შეიცვალოს

document.body.insertBefore(elem, document.getElementById("caption"))

Dom\_ის მეშვეობით შესაძლებელია კონკრეტული ელემენტის პირველ(firstChild) და უკანასკნელ (lastChild) ელემენტებზე წვდომა. ასევე ერთ დონეზე განლაგებულ შემდეგ (nextSibling) და წინა (previousSibling) ელემენტებზე.

|  |
| --- |
| <html> <head>  <title>დოკუმენტი</title>  </head>  <body>  <div>ტექსტი 1 </div>  <ul id="partner">  <li> განათლების სამინისტრო </li>  <li> ჯანდაცვის სამინისტრო </li>  <li> სპორტისა და ახალგაზდობის სამინისტრო </li>  <li> ენერგეტიკის სამინისტრო </li>  </ul>  <div>ტექსტი 2 </div>    <script> var elem = document.getElementById("partner"); alert(elem.firstChild.innerHTML);// განათლების სამინისტრო alert(elem.lastChild.innerHTML);// ენერგეტიკისსამინისტრო alert(elem.nextSibling.innerHTML);// ტექსტი 2 alert(elem.previousSibling.innerHTML);// ტექსტი 1 </script>    </body>  </html> |

გარდა ელემენტების შექმნისა არსებობს ელემენტების წაშლის removeChild() და ჩანაცვლების replace.Child() მეთოდები.

|  |
| --- |
| <html> <head>  <title>დოკუმენტი</title> |

</head>

|  |
| --- |
| <body>  <div>ტექსტი 1 </div>  <ul id="partner">  <li> განათლების სამინისტრო </li>  <li> ჯანდაცვის სამინისტრო </li>  <li> სპორტისა და ახალგაზდობის სამინისტრო </li>  <li> ენერგეტიკის სამინისტრო </li>  </ul>  <div id="txt">ტექსტი 2 </div>  <script> var elem = document.getElementById("partner"); document.body.removeChild(elem);// წაიშლება მთლიანი <ul>თავისი li\_ებით    var elem2 = document.getElementById("txt"); var txt = document.createTextNode("შეცვლილი ტექსტი "); document.body.replaceChild(txt, elem2);// შედეგი <div id="txt">შეცვლილი ტექსტი </div> </script>  </body>  </html> |

### 5.5 ფუნქციებთან მუშაობა

|  |
| --- |
| მიმდინარე პარაგრაფის თემატიკა   * ფუნქციების შექმნა და გამოყენება * „ლამბდა“ გამოსახულებები |

ფუნქციები

ფუნქცია თავისი არსით ავტონომიურად მოქმედ კოდს წარმოადგენს, რომლის ფუნქციონალის გაწერა ერთჯერადად ხდება, ხოლო გამოყენება შესაძლებელია მრავალჯერ.მაგ. საჭიროა ეკრანზე რაიმე რიცხვის კვადრატის გამოტანა. როგორც თქვენთვისაა ცნობილი რიცხვის კვადრატში ასაყვანად საჭიროა არსებული რიცხვის თავის თავზე გამრავლება. ე.ი პროცესი უცვლელია და მისი ყოველ ეტაპზე თავიდან დაწერა კოდის დუბლირებას წარმოადგენს, რაც შესაბამისად არც ისე კარგ ტონად მიიჩნევა.

Javascript\_ში ფუნქციები გვევლინებიან ობიექტებად და სახელი რომლის მინიჭებაც ხდება, წარმოადგენს მის იდენთიფიკატორს, შესაბამისად ის რეგისტრზე დამოკიდებულია. მაშასადამე დაუშვებელია ფუნქციის სახელებად გამოყენებულ იქნას Javascript\_ის რეზერვირებული სიტყვები (იხ.ცხრილი I), ასევე სხვა გლობალური ობიექტების სახელები. მაგ.: უკვე არსებული ცვლადების დასახელებები და ობიექტის იდენთიფიკატორები. თუ კოდში ერთიდაიგივე სახელით აღწერილია ორი ფუნქცია მაშინ არსებული ფუნქციის გამოძახებისას ხდება წვდომა არსებული სახელით შექმნილ ბოლო ფუნქციასთან. ფუნქცია შედგება ორი ნაწილისაგან ეს არის ფუნქციის სახელი და ტანი, სადაც ხდება სკრიპტის მობილიზება. სტანდარტული (სახელდებული) ფუნქციის დეკლარირება ხორციელდება შემდეგნაირად

functionshowMessage(){ //showMessage - სახელი

... ფუნქციის ტანი

}

ხოლო ანონიმური ტიპის ფუნქცია

var showMessage = function(){ //showMessage - სახელი

... ფუნქციის ტანი

}

ფუნქციის დეკლარირება მის ავტომატურ გამოყენებას არ გულისხმობს და არც ითვალისწინებას. როგორც ითქვა ერთგვაროვანი მოქმედების მრავალჯერ გამოყენებაა საჭირო ამისთვის სასურველ ადგილზე საჭიროა ფუქციის გამოძახება რომელიც მდგომარეობს შემდეგში

|  |
| --- |
| functionshowMessage(){ alert("შეტყობინება");  }  //ფუნქციის გამოძახება შესაძლებელია მრავალჯერადად. არსებული მაგალითზე დაყრდნობით ეკრანზე გამოვა alert\_დიალოგული ფანჯარა ორჯერ ტექსტით ‘შეტყობინება’    showMessage() //ფუნქციის გამოძახება showMessage() //ფუნქციის გამოძახება |

სტანდარტული და ანონიმუტი ტიპის ფუნქციები ერთმანეთისაგან ფუნქციის წვდომით განსხვავდებიან. თუ სტანდარტულ ფუნქციასთან წვდომა ნებისმიერი ადილიდან არის შესაძლებელი, ანონიმურთან მხოლოდ მისი დეკლარირების შემდგომ.

|  |  |
| --- | --- |
| //სტანდარტული ფუნქცია showMessage() //შედეგი ეკრანზე ‘შეტყობინება’    functionshowMessage(){ alert("შეტყობინება");  }    showMessage() //შედეგი ეკრანზე ‘შეტყობინება’    //ანონიმური ფუნქცია showMessage() //ფიქსირდება შეცდომა    var showMessage = function(){ alert("შეტყობინება");  } | |
| showMessage() | //ამ ეტაპზე მუსაობს --- შედეგი ეკრანზე ‘შეტყობინება’ |

ფუნქციას, როგორც მნიშვნელობის დაბეჭვდვა ( რომელიმე გამოტანის ოპერატორის alert(), console.log(), document.write() ) , ასევე მისი დაბრუნება შეუძლია. ფუნქციიდან ინფორმაციის დაბრუნება საკმაოდ მნიშვნელოვანი დეტალია, გამომდინარე იქიდან, რომ შესაძლოა მიღებული მნიშვნელობა გამოყენებულ იქნას სხვა დანიშნულებით. მაგ.: დავალება მდგომარეობს შემდეგში ერთი ფუნქციის მეშვეობით უნდა მოხდეს რიცხვის კვადრატის დადგენა, მეორე ფუნქცია ადგენს ამ რიცხვის უდიდეს გამყოფს და შედეგად უნდა მივიღოთ მათი სხვაობა. ამისათვის საჭიროა ფუნქციები რომლების უზრუნველყოფენ: პირველი - კვადრატის დადგენას, მეორე უდუდესი გამყოფის პოვნას. როგორც ატყობთ დავალება არ მდგომარეობს მათ დაბეჭვდაში. შესაბამისად გამოტანის ოპერატორების ფუნქციაში გამოუყენების არავითარი საჭიროება არ არსებობს. პროგრამისტისთვის მნიშვნელოვანია მიიღოს ინფორმაცია, რომელზეც შემდგომ იმოქმედებს. ასეთ შემთვევაში ფუნქციაში გამოიყენება დაბრუნების ოპერატორი return

functionshowMessage(){

return"შეტყობინება";

}

არსებულ შემთხევაში ფუნქციის გამოძახება არ მოგვცემს ვიზუალურ შედეგს ეკრანზე, მაგრამ იძლევა საშუალებას არსებულ მნიშვნელობაზე მოვახდინოთ მანიპულირება.

|  |  |
| --- | --- |
| functionshowMessage(){  return"შეტყობინება";  }    alert ( "საგანგაშო" + showMessage()) | //შედეგი ეკრანზე - საგანგაშო შეტყობინება |

ფუნქციიდან return მეთოდით დაბრუნებული შედეგის მინიჭება შესაძლებელია ასევე რაიმე ცვლადზე. მაგ.:

functionshowMessage(){

return"შეტყობინება";

}

var sms = showMessage( ) alert ( "სასარგებლო" + sms) //შედეგი ეკრანზე - სასარგებლო შეტყობინება

გაითვალისწინეთ ფუნქციაში შეუძლებელია რამოდენიმე მნიშვნელობის რამოდენიმე return მეთოდით მიღება, რადგანაც პირველივე return \_ის გამოზახება იწვევს ფუნქციის მოქმედების შეწყვეტას და მის შემდგომ არსებული კოდის შესაბამისად აღარ აღიქმება. უხეშად რომ შევადაროთ, რა გავლენასაც ახდენდა break ოპერატორი switch კონსტრუქციასა და ციკლებში, ფაქტობრივად იგივე ფუნქციონალის შეთავსება უხდება return\_ს გარდა ძირითადი ფუნქციისა, რომელიც აბრუნებს მნიშვნელობას

|  |  |
| --- | --- |
| functionshowMessage(){  return"სასიამოვნო"; return"სიახლე";  }  var sms = showMessage( ) alert (sms) //შედეგი ეკრანზე მხოლოდ - სასიამოვნო სიახლე | და არავითარ შემთხვევაში სასიამოვნო |

ფუნქციების მნიშნველოვან დანამატს წარმოადგენს არგუმენტები. არგუმენტები ეს ის მნიშვნელობებია რომლის გადაცემაც და დამუშავებაც შესაძლებელია ფუნქციის მიერ. შესაძლებელია ფუქნციისათვის, როგორც ერთი ასევე რამოდენიმე არგუმენტის ერთდროულად გადაცემა. მაგ.: დავალების შესაბამისად დავწეროთ რიცხვის კვადრატში აყვანის ფუნქცია.

|  |
| --- |
| functionsquare(x ){ //xწარმოადგენს არგუმენტეს return x \* x ;  }    alert ( square( 5) ) //შედეგი ეკრანზე 25 alert ( square( 10) ) //შედეგი ეკრანზე 100 alert ( square( 7) ) //შედეგი ეკრანზე 49 |

შესაძლოა არგუმენტად გადაეცემოდეს ესა თუ ის კონკრეტული ცვლადი

functionsquare( x ){ //xწარმოადგენს არგუმენტეს return x \* x ;

}

var n = 11;

alert ( square( n) ) //შედეგი ეკრანზე 121

დაიმახსოვრეთ არგუმენტის სახელი (მოცემულ შემთხვევაში - x) არ წარმოადგენს მნიშვნელობას რომელიც ხილულია ფუნქციის გარეთ. ის მხოლოდ ფუნქციის შგნით აღიქმება. ფუნქციის გარათ x\_ის დაბეჭვდა არ მოხერხდება , შედეგი ეკრანზე არ გამოისახება.

functionsquare( x ){ //xწარმოადგენს არგუმენტეს

return x \* x ;

}

alert (x) //Uncaught ReferenceError: x is not defined - შეგიძლიათ გადაამოწმოთ consol\_ში

არგუმენტების განსაზღვისას გასათვალისწინებელია, რომ ფუნქციამ უნდა მიიღოს იგივე რაოდენობის არგუმენტების გამოძახებისას.მაგ:

functiontest( x, y , z ){

return x + y + z ;

}

alert (5, 7, 3) //15

არგუმენტების არასრულად გადაცემის შემთხვევაში არსებული არგუმენტები მიენჭება საწყის ცვლადებს სანამ ისინი იარსებებს, დანარჩენი არგუმენტის მნიშვნელობები გახდება undefined. მაგ.: მხოლოდ 2 პარამეტრის მიწოდების შემთხვევაში ისინი იქნებიან x,y არგუმენტები და არა x, z ან z,y შესაბამისად z არგუმენტი იქნება undefined.

functiontest( x, y , z ){

return x + y + z ; // 5+3+undefined

}

alert (5, 3) //NaN

alert (5, 3, 8) //16

დამატებითი არგუმენტების გადაცემის შემთხვევაცი, როცა ისინი არ იყო წინასწარ განსაზღვრული ფუნქცია არ იღებს მათ მხედველობაში, ფაქტობრივად უფულველყოფს მათ და არ წარმოაჩენს. ისინი არ აღიემქებიან არც შეცდომებად. მათი არსებობა საერთოდ არსად ჩანს

functiontest( x, y , z ){

return x + y + z ; // 5+3+2

}

alert (test(5, 3, 2, 8, 9, 4)) // შედეგი 10 ---- 8,9,4 უგულველყოფილია, არც შეცდომა ,არც ყურადღების გამახვილება

არსებობს გზა თავის დასაცავად, თუ რომელიმე კონკრეტული არგუმენტი არ იქნა გადაცემული მოხდეს ამ არგუმენტის ჩანაცვლება რაიმე მნიშვნელობით რათა ფუნქციამ არ გამოიტანოს შეცდომა. მაგ.:

functiontest( x, y , z ){

|  |
| --- |
| x = x || 1; // თუ x არგუმენტი არ გამოეცია მნიშვნელობა x გახდება 1\_ის ტოლი y = y || 10; // თუ y არგუმენტი არ გამოეცია მნიშვნელობა y გახდება 10\_ის ტოლი z = z || 20; // თუ z არგუმენტი არ გამოეცია მნიშვნელობა z გახდება 20\_ის ტოლი    return x + y + z ;  }    alert( test(5, 3, 2)) //10 alert( test(5, 3)) // 28 - - - - x = 5, y = 3, z = 20რადანაც z არ გადაეცა მნიშვნელობად აიღო 20 alert( test(5)) // 35 - - - - x = 5, y = 10, z = 20 alert( test()) // 31 - - - - x = 5, y = 10, z = 20 |

Javascript\_ს გააჩნია arguments ობიექტი, რომლის საშუალებით შესაძლებელია ყველა არგუმენტის მიღება წინასწარ განუსაზღვრელი რაოდენობის მიუხედავად. arguments ობიექტი იგივე მასივია, შესაბამისად მასივის მეთოდებისა და ფუნქციები მიესადაგება

functiontest(){

return arguments.length;

}

alert( test(5, 3, 2)) //3

თითოეულ ელემენტთან წვდომა თქვენთვის ასევე ცნობილია

|  |
| --- |
| functiontest(){  return arguments[0]+arguments[2];  } alert( test(5, 3, 2) ) //7 |

მაგ. საჭიროა დაიწეროს ფუნქცია რომელიც მოახდენს რიცხვთა ჯამის წარმოჩენას, მაგრამ საწყის ეტაპზე არ არის გარკვეული თუ რამდენი რიცხვის დაჯამება იქნება საჭირო. ეს შეიძლება იყოს 4,10, და ა.შ. ამისათვის გამოიყენება ობიექტი arguments რომლის მეშვეობითაც შესაძლებელია არგუმენტების რაოდენობის დადგენა და მათზე მანიპულირება შეცდომების გარეშე ( როგორც იყო წინა შემთხვევაში განუსაზღვრელი არგუმენტის მნშვნელობა undefined ).

functiontest(){

var sum = 0;

for(var i =0; i <arr.length; i++){

sum+= arguments[i];

}

|  |  |
| --- | --- |
| return sum;  }    alert( test(5, 3, 2)) // 10 alert( test(2,11,25, 4, 3)) alert( test(10, 23)) // 33 | // 45 |

ფუნქციის თავისებურებას წარმოადგენს „ურთიერთობა ცვლადებთან“. რაც შემდეგში მოიაზრება. ნებისმიერი ცვლადი რომელიც აღწერილია სკრიპტში ფუნქციისათვის წარმოადგენს გლობალური ტიპის ცვლადს. ფუნქციიდან შესაძლებელია მის გარეთ მდებარე ცვლადთან წვდომა და მისი ცვლილება.მაგ:

var k = 25;

functiontest(){ k += 11 // k = k + 11

}

alert( k ); // 36

ფუნქციას გააჩნია ასევე ლოკალური ტიპის ცვლადები. რომელიც არ სცდება არსებული ფუნქციიის ხილვადობის არეალს - მის ტანს.არსებული ცვლადის დეკლარირება ხდება უშუალოდ ფუნქციის ტანში და ის ხილვადია არსებული ფუქნციის შიგთავსში ასევე სხვა ფუნქციებში რომლებიც გამოძახებულია მიმდინარე ფუნქციის ბლოკში {...}, მაგრამ არამც და არამც ძირითადი ფუნქციის გარეთ.

functiontest(){ var k = 25; k += 11 // k = k + 11

}

alert( k ); // k is not defined

შემთხვევა როცა ერთი და იგივე სახელით დეკლარირებულია ცვლადი ფუნქციის გარეთ და ფუნქციის შიგნით განიხილება შემდეგნაირად. გლობალური ცვლადი k რჩება გლობალურად და ის ფუნქციიდან არ განიცდის ცვლილებას გამომდინარე იქიდან, რომ არსებულ ფუნქციას თავისი ლოკალური k ვლადი გააჩნია, სწორედ მასი ცვლილება ხდება ფუნქციის შიდა გამოთვლის დროს.

var k = 25;

functiontest(){ var k = 11; k += 11 // k = k + 11

|  |  |
| --- | --- |
| alert( k );  }  test()    alert( k ); // 25 | // 22 |

არსებობს ფუნქციის შექმნის კიდევ ერთი გზა, რომელიც ძალიან იშვიათად გამოიყენება მაგრამ ფუნქციის დეკლალირების სრულყოფილი წარმოდგენისთვის მასაც შევეხოთ.

var showMessage = new Function()

არსებული სახით ფუნქცია იღებს და ამუშავებს მონაცემებს string\_ის სახით, ძირითადად გამოიყენება მოკლე კოდისთვის

|  |
| --- |
| var sum =newFunction('a,b',' return a+b; ');    var result =sum(1,2); alert( result ); // 3 |

Javascript Events

ვებ გვერდების დინამიურობა ძირითადად რაიმე ტიპი ხდომილებებზეა/ქმედებაზეა დამოკიდებული. ხშირია, როდესაც დოკუმენტი ან მისი რომელიმე ელემენტი გარკვეულ ქმედებაზე განიცდის რაღაცა სახის ცვლილებას. მაგალითისათვის ვებ ბრაუზერი ახორციელებეს ქმედებას, როდესაც ტვირთავს დოკუმენტს, მომხამრებელი მიერ მაუსის მიტანისას მოქმედებს ჰიპერბმულზე, ასევე ფორმის ღილაკზე ახორციელებს კლიკს. Javascript\_ით შესაძლებელია წინასწარ განისაზღვროს კონკრეტული ქმედება, რაიმე ფუნქციის ან სკრიპტის ფრაგმენტის მეშვეობით, რომელიც ამუშავდება მხოლოდ მომხმარებლის მიერ გამოხატული აქტოვობის შემდეგ.

ზოგადად სხვადასხვა ტიპის მოქმედება უზრუნველყოფს სხვადასხვა ტიპის ქმედების/ხდომილების წარმოქმნას. მაგ.: ბმულზე ხელის მიტანა და მასზე უშუალო ზემოქმედება (კლიკი) ერთმანეთისაგან განსხვავებული პროცესებია, ისევე როგორც თუნდაც ფორმის ორი ერთნაირი ღილაკი Submit და Reset აზრობრივად ერთმანეთისაგან რადიკალურად განსხვადება.

ბრაუზერების დახვეწამ დროთა განმავლობაში განავრცო Javascript ხდომილებები და დღეს დღეისობით დეველოპერებს საკმაოდ დიდი არჩევანი აქვთ . ცხრილ XIV მოცემულია იმ ძირითადი ხდომილებების ჩამონათვალი, რომელიც აქტიურად გამოყენებადია. ესენია მაუსის, კლავიატურის, ფორმის და ობიეტის ქმედებები. სრული სია შეგიძლიათ იხილოთ მითითებულ ბმულზე - <http://www.w3schools.com/jsref/dom_obj_event.asp>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ცხრილი XIV | | | | | |
| ხდომილება | | | აღწერა | | ტეგები, რომელსაც მიესადაგება |
| [onclick](http://www.w3schools.com/jsref/event_onclick.asp) | | | მაუსი | ერჯერადი კლიკი ( დაჭერა - აშვების პროცესი) | ფაქტობრივად ყველა html ელემენტი |
| [ondblclick](http://www.w3schools.com/jsref/event_ondblclick.asp) | | | ორმაგი კლიკი | ფაქტობრივად ყველა html ელემენტი |
| [onmousedow](http://www.w3schools.com/jsref/event_onmousedown.asp)  [n](http://www.w3schools.com/jsref/event_onmousedown.asp) | | | მაუსის ღილაკზე მხოლოდ დაწოლა ( აშვება არა ) | ფაქტობრივად ყველა html ელემენტი |
| onmouseup | | | მაუსის ღილაკზე ხელის აშვება | ფაქტობრივად ყველა html ელემენტი |
| [onmouseout](http://www.w3schools.com/jsref/event_onmouseout.asp) | | | მაუსის კურსორი ცდება ელემენტის საზღვრებს | ფაქტობრივად ყველა html ელემენტი |
| [onmouseover](http://www.w3schools.com/jsref/event_onmouseover.asp) | | | მაუსის კურსორი შედის ელემენტის საზღვრებში | ფაქტობრივად ყველა html ელემენტი |
| [onmousemov](http://www.w3schools.com/jsref/event_onmousemove.asp)  [e](http://www.w3schools.com/jsref/event_onmousemove.asp) | | | მაუსის კურსორი გადაადგილდება ტეგის არეალში | ფაქტობრივად ყველა html ელემენტი |
| [onkeydown](http://www.w3schools.com/jsref/event_onmousemove.asp) | | | კლავიატურა | კლავიატურის კლავიშზე ხელის დაწოლა | ფაქტობრივად ყველა html ელემენტი |
| [onkeyup](http://www.w3schools.com/jsref/event_onmousemove.asp) | | | კლავიატურის კლავიშზე ხელის აღება | ფაქტობრივად ყველა html ელემენტი |
| [onkeypress](http://www.w3schools.com/jsref/event_onmousemove.asp) | | | დაწოლიქნა და აშვებულია კლავიშზე ხელი | ფაქტობრივად ყველა html ელემენტი |
| [onload](http://www.w3schools.com/jsref/event_onload.asp) | | | ფრეიმები / ობიექტები | დოკუმენტის ჩატვირთვა დასრულებულია | BODY , IMG |
| [onerror](http://www.w3schools.com/jsref/event_onerror.asp) | | | შეფერხება წარმოქმნა სცენარის შესრულებისას | IMG |
| [onabort](http://www.w3schools.com/jsref/event_onabort.asp) | | | სურათის ჩატვირთვის შეფერხება | IMG |
| [onresize](http://www.w3schools.com/jsref/event_onresize.asp) | | | დოკუმენტის ზომის ცვლილება | WINDOW |
| [onscroll](http://www.w3schools.com/jsref/event_onscroll.asp) | | | ელემენტის დასკროლვისას | უმრავლესობა ტეგებზე |
| [onunload](http://www.w3schools.com/jsref/event_onunload.asp) | |  |  | ელემენტი ჩატვირთvის პროცესშია | ძირითადად BODY |
| [onfocus](http://www.w3schools.com/jsref/event_onfocus.asp) | |  | ფორმები | ფორმის მონიშვნა / კურსორის ჩაყენება |  |
| BUTTON, INPUT, LABEL,  SELECT, TEXTAREA |
| [onblur](http://www.w3schools.com/jsref/event_onblur.asp) | |  | ფორმის ელემენტიდან ამოსვლა |
| [onchange](http://www.w3schools.com/jsref/event_onchange.asp) | |  | ფორმის ველი არჩეულია | SELECT |
| [onreset](http://www.w3schools.com/jsref/event_onreset.asp) | |  | ფორმის გაბულება / ინფორმაციის ჩამოყრა | FORM |
|  | [onsubmit](http://www.w3schools.com/jsref/event_onsubmit.asp) |  | ფორმის მონაცემების გაგზავნა | FORM |
|  |
| [onselect](http://www.w3schools.com/jsref/event_onselect.asp) | |  | ტექსტის არჩევა | INPUT,  SELECT |

ხდომილების გამოყენება შესაძლებელია უშუალოდ ელემენტში მისი გაწერით ან სკრიპტიდან ელემენტზე მიმართვიანობით.

დავალება ღილაკზე მაუსის დაკლიკვით ბლოკ id="txt" შეეცვალოს ფონი და გახდეს ის წითელი

<divid = "txt">ტექსტი 1 </div>

<buttononclick ="setBg()">ღილაკი </button>

<script> function setBg(){ document.getElementById("txt").style.backgroundColor="red";

}

</script>

შესაძლებელი იყო არსებული ფუნქცია ცალკე გატანის გარეშე უშუალოდ მეთდ onclick\_ში გაგვეწერა. ეს გზა შედარებით არაკომფურტულია დიდ სკრიპტთან მიმართებაში და ნაკლებად გამოიყენება, მაგრამ მისი უგულველყოფა არიქნება სამართლიანი, რადგანაც მცირე კოდებში საკმაო გამოყენება აქვს. კოდის მინიმიზაცია თვალსაჩინოა.

<divid = "txt">ტექსტი 1 </div>

<buttononclick ="document.getElementById('txt').style.backgroundColor='red'">ღილაკი </button>

არსებობს კიდევ ერთი გზა, როცა event\_ის მიბმა უშუალოდ სკრიპტიდან ხორცილედება

<div id = "txt">ტექსტი 1 </div>

<button>ღილაკი </button>

<script> document.getElementsByTagName("button")[0].onclick = function(){ document.getElementById("txt").style.backgroundColor="red";

}

</script>

ქმედებებთან / ხდომილებებთან ურთიერთბისას მნიშვნელოვანია thisმნიშვნელობის არსის ცოდნა. ის გამოიყენება მიმდინარე ელემენტის გამოსახატად სა კომფორტულია ბევრ შემთხვევაში. რამოდენიმე ელემენტის შემთხვევაში სათითაოდ მოგვიწევს მათი წვდომის (document.getElement…..) განსაზღვრა, თავის მარტივად დასაძვრენად გამოვიყენოთ this

დავალება:

ბლოკებზე ხელის დაწოლისასგამოსახოს ეკრანზე არსებულ ბლოკში არსებული ინფორმაცია

|  |
| --- |
| <divonclick ="getTxt(this)">ტექსტი 1 </div>  <divonclick ="getTxt(this)">ტექსტი 2</div>  <divonclick ="getTxt(this)">ტექსტი 3</div>    <script> function getTxt(t){  alert(t.innerHTML);  }  </script> |

მოცემულ კოდში this არგუმენტის არცოდნის შემთხვევაში მოგწევდათ კოდის შემდეგნაირად გაშლა

|  |
| --- |
| <divonclick ="getTxt1()" id ="dv1">ტექსტი 1 </div>  <divonclick ="getTxt2()" id ="dv2">ტექსტი 2</div>  <divonclick ="getTxt3()" id ="dv3">ტექსტი 3</div>    <script> function getTxt1(){  alert(document.getElementById("dv1").innerHTML);  }  function getTxt(2){  alert(document.getElementById("dv2").innerHTML);  }  function getTxt(3){  alert(document.getElementById("dv3").innerHTML);  } |

</script>

ან შესაძლოა სხვა ინტერპრეტაცია ჰქონოდა სკრიპტის ვრცელ ვარიანტს