**ლექცია 6 მასივები JavaScript-ში**

**ცვლადები**

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

* ცვლადის სახელის შემადგენლობაში შეიძლება შედიოდეს ლათინური ალფაბეტის დიდი და პატარა ასოები, ციფრები, ქვედა ხაზი და დოლარის სიმბოლო $;
* სახელიშეიძლება დაიწყოს ასოთი, ქვედა ხაზით ან დოლარის სიმბოლო $-ით.
* სახელი არ შეიძლება ციფრით იწყებოდეს;
* ენა განასხვავებს დიდსა და პატარა ასოებს. მაგალითად: Total და total ორი სხვადასხვა ცვლადია.
* სახელის სიგრძე ოფიციალურად შეზღუდული არ გახლავთ, თუმცა იგი კოდის სტრიქონზე გრძელი არ უნდა იყოს.

სწორად დაწერილი სახელების მაგალითებია: **total\_number\_of\_fish**  , **TotalNum** , **Totalnum** , **temp5** , **\_var94**

ცვლადებს JavaScript-ში, როგორც წესია წინასწარ აცხადებენ, თუმცა არგამოცხადებული ცვლადებიც დასაშვებია. ცვლადის წინასწარი გამოცხასება ხდება მისი გამოყენების წინ.

ცვლადების გამოცხადება შესაძლებელია შემდეგი სამოსამსახურო (საკვანძო) სახელებით: **var** , **let**, **const** . შესაძლებელია ცვლადი ჯერ მხოლოდ გამოვაცხადოთ და მერე მივანიჭოთ მას მნიშვნელობა (ანუ მოხდეს მათი ინიციალიზაცია), ან ეს მოვახდინოთ ერთდროულად.

ასე მაგალითად, ორივე სამართლიანია:

let bora;

bora=”სახელია“;

let bora=”სახელია“;

**გლობალური და ლოკალური (ფუნქციის მხედველობის არის) ცვლადები**

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

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

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

**var students** = 25;

**students** = 25;

ეს ოპერატორები ერთმანეთის ტოლფასია, თუმცა, კოდის უკეთ აღქმის თვალსაზრისით, უპირატესობას პირველ ხერხს ვანიჭებთ. რაც შეეხება ლოკალური ცვლადების გამოცხა­დებას, როგორც აღვნიშნეთ, ეს ხდება ფუნქციაში და, როგორც წესი, ვირჩევთ პირველ ხერხს. ლოკალური ცვლადის var საკვანძო სიტყვით გამოცხადება თავიდან გვაცილებს კონფლიქ­ტური სიტუაციის წარმოქმნის საშიშროებას (მხედველობაში გვაქვს ისეთი შემთხვევები, როცა მოცემული სახელის ცვლადი ადრე უკვე გამოცხადებული იყო, როგორც გლობალური).

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

**ლისტინგი 1**

<HTML>

<HEAD>

<TITLE>ფუნქციების გამოყენების სხვა მაგალითი </title>

<STYLE>

H2, P {font-family: LitNusx}

</style>

<SCRIPT LANGUAGE="JavaScript">

<!--ვინიღბებით!

var name1="გიგი";

var name2="გაგა";

function Greet(who) {

alert("დღეს პროგრამაშია " + who );

var name2="გურამი";

}

</script>

</head>

<BODY>

<H2>ვისწავლოთ ფუნციებთან მუშაობა!</h2>

<P>შეტყობინება გამოდის ორჯერად</p>

<SCRIPT LANGUAGE="JavaScript">

Greet(name1);

Greet(name2);

//ვიხსნით ნიღაბს-->

</script>

</body>

</html>

მივაქციოთ ყურადღება - **Greet**() ფუნქციაში ხელმეორედ იქმნება **name**2 ცვლადი (ეს ხდება

var ბრძანებით), ახალი **name**2 ცვლადი ლოკალური ტიპისაა და მისთვის მნიშვნელობის მი­ნიჭება არ ცვლის **name**2 გლობალური ცვლადის მნიშვნელობას.

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

გლობალური ცვლადების სათაურშივე გამოცხადება დაუწერელი წესია. თუმცა კი დასაშ­ვებია, ასეთი ცვლადები პროგრამის ნებისმიერ ადგილას გამოცხადდეს. აღვნიშნოთ, რომ თუ ცვლადს ვიყენებთ “ოფიციალურად” გამოცხადებამდე (ან ღია სახით მნიშვნელობის მინიჭე­ბის გარეშე), მას განესაზღვრება ნულოვანი მნიშვნელობა.

**var** საკვანძო სიტყვა JavaScript-ში გამოიყენება 1995 წლიდან, ხოლო 2015 წლიდან ცვლადების გამოცხადებისთვის მას დაემატა **let, const** და ამის შემდეგ **var** საკვანძო სიტყვა შედარებით იშვიათად გამოიყენება.

**let**-ი ძალიან გავს **var** საკვანძო სიტყვას, მაგრამ მისი მხედველობის არე განისაზღვრება იმ ბლოგში (კოდის ფიგურული ფრჩხილებით შემოსაზღვრეულ არეში) სადაც ის შემოტანილია, ხოლო var-ით განსაზღვრული ცვლადის მხედველობიის არე ფუნქციაა.

const -ს გააჩნია ის თვისება, რომ მისი საშუალებით გამოცხადებული ცვლადის მნიშველობა აღარ უნდა შეიცვალოს, ამასთან const -ით დეკლარირებული ცვლადი იქვე უნდა იყოს ინიციალიზებური. ცვლადის, დეკლარირებული const-ით, ისევე როგორც let -ის, მხედველობის არე არის ბლოკი, სადაც ის არის დეკლარირებული.

JavaScript-ში ცვლადებისთვის მნიშვნელობების მისანიჭებლად დასაშვებია შემდეგი ოპე­რატორების გამოყენებაც:

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

**lines** –= 1; **lines** – –;

ისინი ეკვივალენტურია **lines = lines+1** და **lines=lines–1** ოპერაციების. ++ და – – ოპერ­ატორები შეიძლება ცვლადის სახელის წინაც დავსვათ. აღვნიშნოთ, რომ სიტუაციიდან გამომდინარე, მათ შეიძლება რამდენადმე განსხვავებული როლიც დაეკისროთ. დავუშვათ, **lines** ცვლადის მნიშვნელობაა 40. alert (lines++) და alert (++lines) გამოსახულების შესრუ­ლება სხვადასხვა შედეგებს მოგვცემს:

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

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

**მონაცემთა ტიპები**

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

ყველა დასაშვები შემთხვევისთვის JavaScript ავტომატურად ახორციელებს მონაცემთა ერთი ტიპის სხვაში გარდაქმნას. ზემოთ უკვე განვიხილეთ ერთი ასეთი შემთხვევა **total** ცვლადის მაგალითზე. მოვიყვანოთ სხვებიც: ვთქვათ, **total** ცვლადის მნიშვნელობა გახლავთ 40.

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

ჯამის სიდიდეა 40

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

სტრიქონული ტიპისაა, მაშინ **average**  = **total** / 3 ოპერატორი ვერ შესრულდება. ასეთ შემთხვევებში მიმართავენ შემდეგ გარდაქმნის ფუნქციებს:

**parseInt()** - ტექსტური ტიპი გადაჰყავს მთელრიცხოვან ტიპში;

**parseFloat()** - ტექსტური ტიპი გადაჰყავს მცურავ წერტილიან რიცხვით ტიპში.

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

**stringvar**  = ”30 დათვი”;

**numvar** = **parseInt** (stringvar);

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

**ცვლადებისათვის მომხმარებლების მიერ**

**მნიშვნელობების მინიჭება**

აღნიშნული მიზნით გამოიყენება **prompt**  ფუნქცია. ამ მეთოდის სინტაქსისია

**prompt**( [param 1][, param 2]) : სადაც param 1, param 2 არააუცილებელი ატრიბუტები არიან. **prompt** -ის შესრულებისასგამოდის მონიტორზე მიმთითებელი ფანჯარა; მიმთითებელ ფანჯარაში აისახება შეტყობინება არააუცილებელი param 1 ატრიბუტი­სა; მიმთითებელი ფანჯარის შეყვანის ველში აისახება ჩუმათობის პრინციპით შესაყვანი შეტყობინება არააუცილებელი param 2 ატრიბუტისა; მიმთითებელ ფანჯა­რას აქვს ორი ღილაკი OK და CANCEL. თუ ავირჩევთ CANCEL-ს მაშინ **prompt** აბრუ­ნებს მნიშვნელობას null -ს. წინააღდეგ შემთხვევაში იმ ტექსტს, რაც შეყვანილ იქნა შეყვანის ველიდან. ამდენად გვექნება სტრიქონული ტიპის - string მნიშვნელობა.

ვაჩვენოთ ამ ფუნქციის გამოყენება შემდეგი კოდის მაგალითზე:

ლისტინგი 2

<HTML>

<HEAD>

<TITLE> მომხმარებელთან დიალოგის მაგალითი </title>

<STYLE>

H2, H3, P {font-family: LitNusx}

</style>

</head>

<BODY>

<H2> ვისწავლოთ ფურცლის აწყობა! </h2>

<P> მიღებულია ინფორმაცია: </p>

<SCRIPT LANGUAGE="JavaScript">

<!-- ვინიღბებით

saxeli=**prompt**("შეიტანეთ თქვენი სახელი ");

gvari= **prompt**("შეიტანეთ თქვენი გვარი ");

furclis\_satauri= **prompt**("შეიტანეთ ფურცლის სათაური ");

document.write("<H2>" + furclis\_satauri + "</h2>");

document.write("<H3>" + saxeli + " " + gvari + "</h3>");

//ვიხსნით ნიღაბს-->

</script>

<P> ფურცელი იმყოფება აწყობის სტადიაში.</p>

</body>

</html>

**სტრიქონული მონაცემები (ტექსტი)**

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

**test** = ”ეს ტესტია”;

**test** = **new String** (“ეს ტესტია”);

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

**test** = ”ეს ტესტია.”;

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

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

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

**toUpperCase()** - ტექსტი გადაჰყავს ასომთავრულ რეგისტრში;

**toLower Case()** - ტექსტი გადაჰყავს სტრიქონულ რეგისტრში.

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

ლისტინგი 3

<html>

<head>

<title> რეგისტრების ცვლილება</title>

<script language="javascript">

<!--ვინიღბებით

var Shakespeare="To be or NOT to be";

Shakespeare1=Shakespeare.toUpperCase();

Shakespeare2=Shakespeare.toLowerCase();

alert(Shakespeare1);

document.write(Shakespeare2);

// ვიხსნით ნიღაბს. -->

</script>

</head>

<body>

</body>

</html>

გავეცნოთ სხვა მეთოდების დანიშნულებასაც:

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

**alpha** = ” ABCDEFGHIJKLMNOPQRSTUVWXYZ ”;

**alpha**.**substring** (0,4) დაგვიბრუნებს ABCD მნიშვნელობას,

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

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

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

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

ლისტინგი 4

<html>

<head>

<title> ტექსტის ნაწილის ამოღება</title>

</head>

<body>

<h1>შინაარსიანი გვარი</h1>

<script language="JavaScript">

k="კვარაცხელია";

alert(k.substring(0,11)); k1=k.substring(0,11); document.write(k1+" ");

alert(k.substring(1,11)); k1=k.substring(1,11); document.write(k1+" ");

alert(k.substring(2,11)); k1=k.substring(2,11); document.write(k1+" ");

alert(k.substring(3,11));k1=k.substring(3,11); document.write(k1+" " );

alert(k.substring(4,11)); k1=k.substring(4,11); document.write(k1+" ");

alert(k.substring(5,11)); k1=k.substring(5,11); document.write(k1+" ");

alert(k.substring(6,11)); k1=k.substring(6,11); document.write(k1+" ");

alert(k.substring(7,11)); k1=k.substring(7,11); document.write(k1+" ");

alert(k.substring(8,11)); k1=k.substring(8,11); document.write(k1+" ");

alert(k.substring(9,11)); k1=k.substring(9,11); document.write(k1+" ");

alert(k.substring(10,11)); k1=k.substring(10,11); document.write(k1+" ");

</script>

</body>

</html>

**indexOf()** მოგვიძებნის მოცემულ ტექსტში მითითებული სიმბოლოების ერთობლიობის (საძებნი სიტყვა ან სიმბოლო) დამთხვევის პირველივე პოზიციას. თუ ეს საძებნი სიტყვა ტექსტში რამდენჯერმე გვხვდება, მეორე პარამეტრში შეიძლება მივუთითოთ, მერამდენე სიმბოლოდან უნდა და­იწყოს ძებნის პროცესი. თუ კი დამთხვევა არ აღმოჩნდა გვიბრუნებს მნიშვნელობას -1, წინააღმდეგ შემთხვევაში პოზიციის მდებარეობას მთელი რიცხვის სახით.

მაგალითად, შემდეგი გამოსახულება სიტყვა “მთას” **temp** სტრიქონულ ობიექტში მოგვიძებ­ნის მე-20 სიმბოლოდან:

**loc=temp.indexOf** (“მთა”, 19);

აქვე მივუთითოთ, რომ ძებნისას გაითვალისწინება სიმბოლოების რეგისტრი.

მაგალითი - ფრაზაში სიტყვის ძებნისა

ლისტინგი 4 სიტყვის ძებნა თავიდან

<html>

<body>

<script type='text/javascript'>

//ჩავწეროთ სტრიქონი

var str = 'ქარი ქრის, ქარი ქრის, ქარი ქრის...!';

/\* ვეძებთ 'ქრის' ტექსტის სტრიქონში და ავსახოთ

პირველივე თანხვედრის პოზიცია \*/

document.write('”ქრის”-ს პირველი დამთხვევა არის '+str.indexOf('ქრის') + ' პოზიციიდან'+'<br />');

//ეხლა ვეძებოთ არ არსებული ნაწილი ტექსტში - ქარიშხალი

document.write('”ქარიშხალი”-ს პირველი დამთხვევა არის '+str.indexOf('ქარიშხალი') + ' პოზიციიდან'+ '<br />');

//და ბოლოს ვეძებთ 'ქარი' 25 პოზიციიდან

document.write('”ქარი”-ს პირველი დამთხვევა 25 პოზიციიდან არის '+str.indexOf('ქარი',24));

</script>

</body>

</html>

ამ კოდის შესრულებისას მივიღებთ

”ქრის”-ს პირველი დამთხვევა არის 5 პოზიციიდან  
”ქარიშხალი”-ს პირველი დამთხვევა არის -1 პოზიციიდან  
”ქარი”-ს პირველი დამთხვევა 25 პოზიციიდან არის -1

**lastIndexOf()** მეთოდი წინასაგან იმით განსხვავდება, რომ საჭირო ფრაგმენტების ძიებას ტექსტში იგი ბოლოდან იწყებს და დააბრუნებს რიცხვს- პოზიციის ნომერს საძებნი სიტყვის პირველივე თანხვედრისა მოცემული ტექსტის ბოლოდან. მაგალითად

ლისტინგი 5. სიტყვის ძებნა ბოლოდან

<html>

<body>

<script type='text/javascript'>

//ჩავწეროთ სტრიქონი

var str = 'ქარი ქრის, ქარი ქრის, ქარი ქრის...!';

/\* ვეძებთ 'ქრის' ტექსტის სტრიქონში და ავსახოთ

პირველივე თანხვედრის პოზიცია \*/

document.write('”ქრის”-ს პირველი დამთხვევა არის '+str.lastIndexOf('ქრის') + '-ე პოზიციიდან'+'<br />');

//ეხლა ვეძებოთ არ არსებული ნაწილი ტექსტში - ქარიშხალი

document.write('”ქარიშხალი”-ს პირველი დამთხვევა არის '+str.lastIndexOf('ქარიშხალი') + ' პოზიციიდან'+ '<br />');

//და ბოლოს ვეძებთ 'ქარი' 25 პოზიციამდე ბოლოდან

document.write('”ქარი”-ს ბოლოდან პირველი დამთხვევა 25 პოზიციამდე არის '+str.lastIndexOf('ქარი',24)+'-ე პოზიციიდან');

</script>

</body>

</html>

ამ კოდის შესრულებისას მივიღებთ

”ქრის”-ს პირველი დამთხვევა არის 27-ე პოზიციიდან  
”ქარიშხალი”-ს პირველი დამთხვევა არის -1 პოზიციიდან  
”ქარი”-ს ბოლოდან პირველი დამთხვევა 25 პოზიციამდე არის 22-ე პოზიციიდან

**მასივები**

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

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

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

**score** [0] = 5;

**score** [25] = 9;

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

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

**test**=”მაკრატელი”;

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

**parts[0]**=”მ”;

**parts[1**]=”კრ”;

**parts[2]**=”ტელი”;

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

**Fullname = parts.join**(“ა”);

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

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

**names [0]**=” Fred ”;

**names [1]**=” George ”;

**names [2**]=” Alex ”;

შემდეგი ოპერატორი შეგვიქმნის ახალ, ალფაბეტის მიხედვით მოწესრიგებულ მასივს:

**sortednames=names.sort();**

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

**spacer=** ”. . . . . .”;

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

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

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

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