**ინტერნეტ დაპროგრამირება. ლექცია №5. ობიექტები, DOM**

JacaScripti ობიექტზე ორიენტირებული დაპროგრამების ენაა. მასში ყველაფერი არის ობიექტი. სტრიქონები, რიცხვები, ფუნქციები და მასივები - ყველანი ესენი წარმოადგენენ ობიექტებს თავისი *თვისებებით* და *მეთოდებით*. ყოველივ ე ეს უნდა კარგად გვესმოდეს და უნდა გავითვალისწინოთ თავიდანვე.

ობიექტის მეთოდები ( ანუ ობიექტზე რაღაც ქმედებების, ოპერაციის ჩატარება), ისევე როგორც თვისებები გამოიძახება წერტილის "." საშუალებით. მაგალითად

ლისტინგი 1 მეთოდის გამოყენება

<html><head><title>Vasia</title></head>

<body><h2 >გამარჯობა</h2><br>

<p>შეგვყავს Vasia</P> <br>

<P>მივიღეთ</p>

<script language="javascript">

name="Vasya";

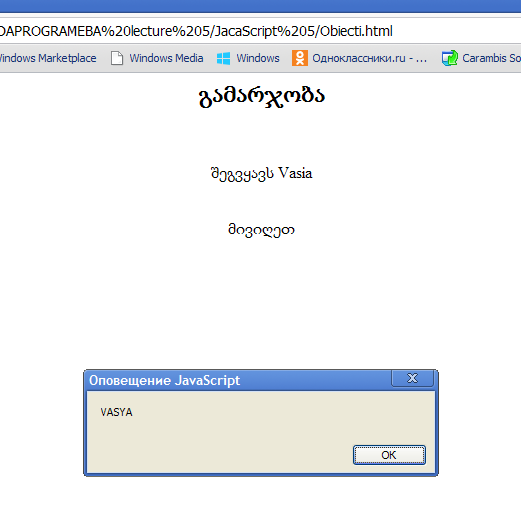
alert(name.toUpperCase());

</script>

</body>

</html>

ამ ლისტინგში მაგალითად ობიექტ name -ს მიესადაგა მეთოდი toUpperCase() - სტრიქონული ცვლადის მაღალ რეგისტრში გადაყვანა. ბრაუზერში გახსნისას მივიღებთ

****

ლისტინგი 2. ობიექტს მიესადაგება ჯერ მეთოდი, მერე თვისება

<html><head><title> ობიექტის თვისება და მეთოდი</title></head>

<body><center><h2 >გამარჯობა</h2><br>

<p><center>შეგვყავს მასივი</p> <br>

მივიღეთ<br>

<script language="javascript">

//შევქმნათ მასივი

var x=new Array();

//ჩავწეროთ მასივში ელემენტები

x[0]="მასივის I ელემენტი";

x[1]="მასივის II ელემენტი";

x[2]="მასივის III ელემენტი";

x[3]="მასივის IV ელემენტი";

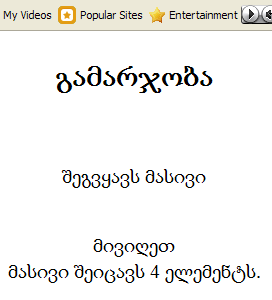
//დავთვალოთ რამდენი ელემენტია x მასივში და შედეგი გამოვიტანოთ გვერდზე

document.write("მასივი შეიცავს "+x.length+" ელემენტს.");

</script>

</body>

</html>

****

JacaScripti-ში ფაქტიურად ყველაფერი კეთდება სხვადასხვა ობიექტებისათვის მეთოდების გამოძახებით. ობიექტები იქმნებიან ფუნქცია-კონსტრუქტორების მიერ new დირექტივის საშუალებით. მაგრამ როგორც ლისტინგებიდანაც ჩანს, რიგ შემთხვევაში ობიექტი შეიქმნება მის გარეშეც.

ლისტინგი 3

<html><head><title> ობიექტის თვისება და მეთოდი</title></head>

<body><center><h2 >გამარჯობა</h2><br>

<p><center>შეგვყავს მასივი</p> <br>

მივიღეთ<br>

<script language="javascript">

//შევქმნათ მასივი

var x=new Array();

//ჩავწეროთ მასივში ელემენტები

x[0]="მასივის I ელემენტი";

x[1]="მასივის II ელემენტი";

x[2]="მასივის III ელემენტი";

x[3]="მასივის IV ელემენტი";

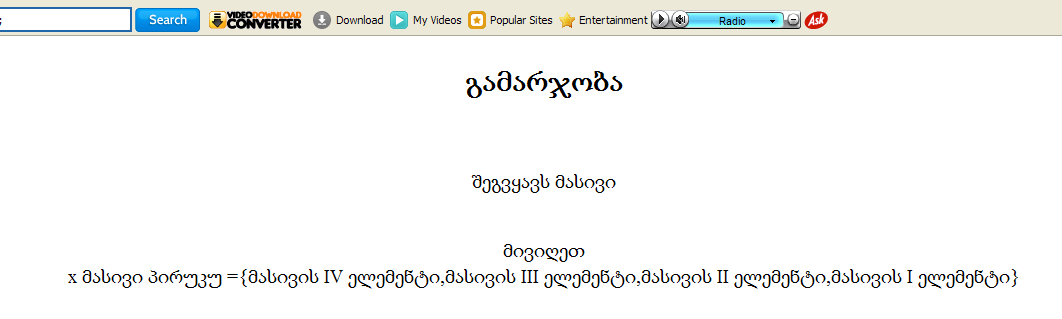
//გამოვიტანოთ x მასივის ელემენტი შებრუნებული რიგით

document.write("x მასივი პირუკუ ={"+x.reverse()+"}");

</script>

</body>

</html>

****

ობიექტი - სკრიპტული, ან პროგრამული ელემენტებია, რომელთაც აქვთ თვისებები, მოვ­ლენები და მეთოდები. Javascript-ი განიხილავს ვებ-გვერდის ყველა ელემენტს, მონიტორის თვალსაჩინო ნაწილს და ბრაუზერის ფანჯარას, როგორც პროგრამულ ობიექტს. რაც ნიშნავს იმას, რომ კოდის საშუალებით შეიძლება ვმართოთ ამ ობიექტის თვისებები, შევასრულოთ მეთოდები და დავამუშავოთ მათი მოვლენები.

ობიექტებს აქვთ თვისება, რომლებიც თავად შეიძლება იყვნენ ობიექტები. განვიხილოთ ადამიანის სხეული როგორც ობიექტი. თუკი ვიაზროვნებთ ობიექტზე ორიენტირებული დაპროგრამების ტერმინებით, მაშინ ობიექტს ”სხეული” (humanBody) აქვს თვისება ”ტორსი” (torso), რომელსაც თავის მხრივ აქვს თვისება ”გული” (heart), რომელსაც თავის მხრივ აქვს თვისება ”პარკუჭი” (ventricle). სხეულის ყველა ეს ნაწილი არის თვისებები, და ისინი ლოგიკური იერარქიით არიან დაკავშირებუნლ ობიექტთან ”სხეული” .

ანალოგიურად, ობიექტზე ორიენტირებული დაპროგრამების ნებისმიერი ელემენტი, ისეთის როგორიცაა Javascript-ი, შეიძლება იყოს ობიექტი. ყველა ობიექტი ერთმანეთთან დაკავშირე­ბულნი არიან იერარქიულად, ისევე როგორც სხეულის ნაწილები. მაგალითად, ბრაუზერის ფანჯარა -ობიექტია, რომელიც შეიცავს დოკუმენტების რამდენიმე ობიექტს, რომელთაც თავის მხრივ აქვთ ფორმების ობიექტები, მათ კი თავის მხრივ აქვთ სხვა ობიექტები და ა.შ. იერარქია შეიძლება იყოს საკმაოდ გრძელი.

რადგანაც Javascript-ი , ობიექტზე ორიენტირებული დაპროგრამების ენაა, მასში ობიექტები, მისი თვისებები და მოვლენები იმართება სკრიპტებით. და რამდენადაც Javascript-ი მხარდა­ჭერილია ბევრი ბრაუზერების მიერ, სკრიპტების ეს ენა გამოიყენება ვებ-გვერდების დიზ­აინისთვის კლიენტის მხარეზე.

ოპერატორი წერტილი (.)

Javascript-ში ოპერატორი წერტილი (.) იძლევა საშუალებას მივმართოთ ობიექტის თვისებებს და მეთოდებს. განვიხილოთ ისინი ადამიანის სხეულის მაგალითზე. ობიექტ ”humanBody”-ს აქვს ობიექტი ”torso”, რომელსაც თავის მხრივ აქვს ობიექტი ”heart”, რომელსაც თავის მხრივ აქვს ობიექტი ”ventricle”. თუ კი ისინი იქნებოდნენ რეალური იერარქიის ნაწილნი, მაშინ ისინი შეიძლება წარმოვიდგინოთ შემდეგნაირად:

humanBody+”.”+ torso+”.”+ heart+”.”+ ventricle”

humanBody.torso.heart.ventricle

როგორც ჩანს, პირველი წერტილი იძლევა საშუალებას მივმართავთ ობიექტს torso humanBody ობიექტის შიგნით. მეორე წერტილი იძლევა საშუალებას მივმართავთ ობიექტს heart torso ობიექტის შიგნით. მესამე წერტილი კი იძლევა საშუალებას მივმართავთ ობიექტს ventricle heart ობიექტის შიგნით. ეს არის სამი დონის იერარქიის მაგალითი. ოპერატორი წერტილი (.)იძლევა საშუალებას მივმართოთ იერარქიაში ყოველ ობიექტს, მის თვისებებს და მეთოდებს.

ჩვენ შეგვიძლია წარმოვადგინოთ ადამიანის ანატომიური ნაწილები, სხეულიდან მარჯვენა ხელის საჩვენებელი თითის ფრჩხილებამდე:

humanBody.torso.rigthHand.indexFinger.fingerNail

**DOM (Document Object Model)**

DOM (Document Object Model) იძლევა საშუალებას Javascript კოდში გამოვიყენოთ HTML-ის ელემენტები და მათი ატრიბუტები. არის ორი საშუალება Javascript კოდის საშუალებით მართო ელემენტები დოკუმენტში: ელემენტის სახელით ან კვანძების ხით. ეს ორი მეთოდი არა არის ურთიერთგამომრიცხავი და შეუძლიათ თანაარსებობა ერთსა და იმავე სკრიპტში.

როგორც ცნობილის ბრაუზერი კითხულობს HTML კოდს და აგებს მის გრაფიკულ გამოსახულებას შესაბამისი ტეგების გათვალისწინებით. ამისთვის ბრაუზერი წარმოადგენს HTML-ს როგორც დოკუმენტის სტრუქტურულ მოდელს.

*დოკუმენტის სტრუქტურა*

წარმოვადგინოთ ერთი კონკრეტული HTML კოდი სტრუქტურის სახით. ვთქვათ გვაქვს შემდეგი კოდი:

ლისტინგი 4. ს

<!doctype html>

<html>

<head>

<title> ჩემი პირადი გვერდი </title>

</head>

<body>

<h1> ჩემი პირადი გვერდი </h1>

<p>გამარჯობა, მე ვასილი, იგივე ვასო ვარ და ეს ჩემი პირადი გვერდია.</p>

<p>და კიდევ მე წიგნი დავწერე, წაიკითხეთ

<a href="http://eloquentjavascript.net">აქ</a>.</p>

</body>

</html>

დოკუმენტს, მოცემულს ლისტინგ 4-ში აქვს შემდეგი სახის სტრუქტურა ბრაუზერისთვის:

html

head

title

ჩემი პირადი გვერდი

body

h1

ჩემი პირადი გვერდი

p

გამარჯობა, მე ვასილი, იგივე ვასო ვარ და ეს ჩემი პირადი გვერდია.

p

და კიდევ მე წიგნი დავწერე, წაიკითხეთ

a

აქ

ნახატი 1. დოკუმენტის სტრუქტურული წარმოდგენა

ნახატ 1-ზე მოცემული სტრუქტურა ასახავს დოკუმენტის ფორმას. ყოველი უჯრისთვის არსებობს შესაბამისი ობიექტი, რომელთანაც შეგვიძლია ურთიერთობა , მისი შესახებ სხვადასხვა ინფორმაციის მოძიების საშუალებაა - რომელ ტეგს შეესაბამება, რომელ უჯრებს და ტექსტს შეიცავს. დოკუმენტის ასეთ წარმოდგენას Document Object Model (დოკუმენტის ობიექტური მოდელი), ან შემოკლებით DOM ჰქვია. დაწვრილებით ამას ქვემოთ შევეხებით. ამჯერად კი იმის ილუსტრირება მოვახდინოთ, თუ როგორ შეიძლება Javascript-მა შეიტანის ცვლილებები HTML დოკუმენტში. განვიხილოთ ცარიელი გვერდის HTML კოდი

ლისტინგი 5 ცარიელი გვერდის კოდი

<html>

<head>

<title> ცარიელი html </title>

</head>

<body>

</body>

</html>

ეხლა ვნახოთ, თუ რა Javascript სცენარი უნდა დავამატოდ ლისტინგ 5-ში, რომ ცარიელი გვერდი შეივსოს, ვთქვათ ყველაზე დიდი სათაურით. საბოლოოდ გვაქვს შემდეგი ლისტინგი

ლისტინგი 6. ცარიელი გვერდის შევსება Javascript სცენარით:

<html>

<head>

<title>სათაურის დამატება</title>

<script>

// ამ ფუნქციის გაშვება დოკუმენტის ჩატვირთვისას

window.onload = function() {

// რამდენიმე ელემენტის შექმნა

// ცარიელ HTML გვერდზე

heading = document.createElement("h1");

heading\_text = document.createTextNode("დიდი სათაური!");

heading.appendChild(heading\_text);

document.body.appendChild(heading);

}

</script>

</head>

<body>

</body>

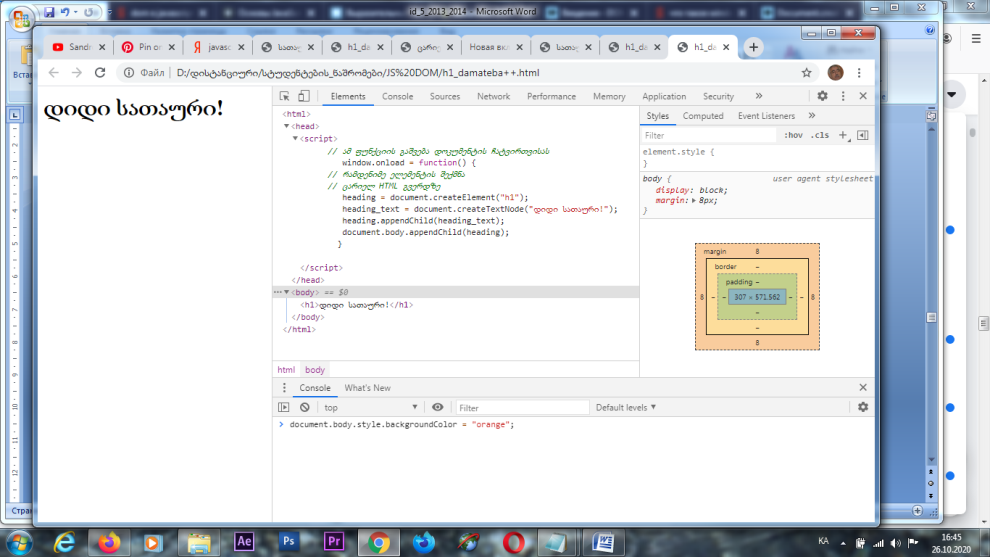
</html>

შედეგად მივიღებთ გვერდს რომელშიც წერია:

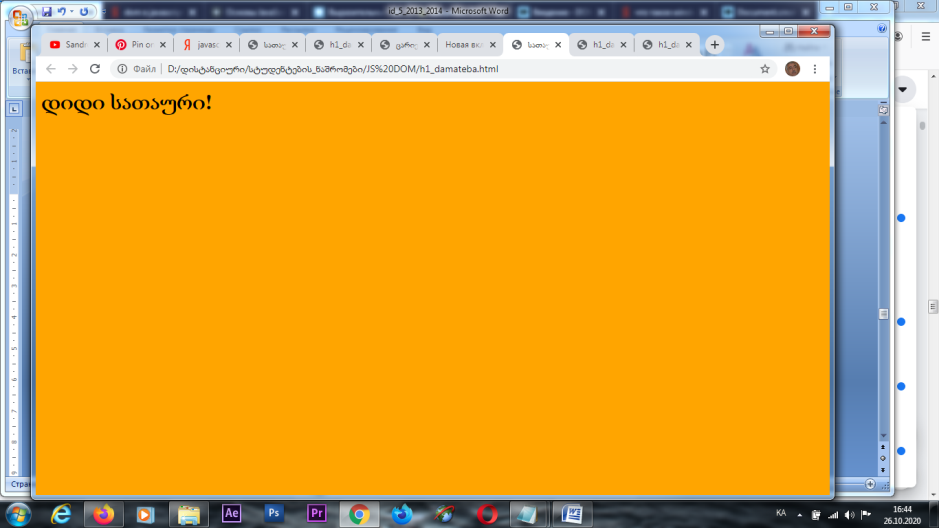
**დიდი სათაური!**

ჩვენ შეგვიძლი დოკუმენტის ობიექტებზე ზემოქმედება მოვახდინოთ სხვა მეთოდითაც. მაგალითად რეალიზაცია ვუყოთ ლისტინგ 6-ს ქრომ ბრაუზერში და გამოვიძახოთ შემდეგ კოდი (მაუსის მარჯვენა ღილაკის დაწკაპუნება და შესაბამისი კოდის არჩევა მენიუდან), შემდეგ ჩავწეროთ კონსოლში ეს ბრძანება

document.body.style.backgroundColor = "orange";



და გავუშვათ, შედეგად დაემატება ტანში ფერის არჩევის არიბუტი და მივიღებთ ასეთ გვერდს:



**ბრაუზერი და ობიექტები**

როგორც ვიცით, Javascript-ში ვებ-გვერდის ყველა ელემენტი (სტრიქონიც), მონიტორის თვალსაჩინო ნაწილი და ბრაუზერის ფანჯარა, შეიძლება იყოს პროგრამირებადი ობიექტი. მაგრამ პროგრამირება შეიძლება მხოლოდ იმ ობიექტისა, რომელიც გაგებულია ბრაუზერის მიერ. სამწუხაროდ, არა ყოველი ბრაუზერი კორექტულად ასახავს ნებისმიერ ობიექტს.

ობიექტების ის ერთობლიობა, რომელთაც ცალკეული ბრაუზერი აღითქვამს და მხარს უჭერს, ცნობილია როგორც **Document Object Model (DOM**). DOM -იც ასევე შესდგება ამ ობიექტებს, მათ თვისებებსა და მეთოდებს ურთიერთობისგან. როცა პროგრამა იწერება სპეციალურად რომელიმე ბრაუზერისთვის, გასათვალისწინებელია ბრაუზერის DOM. სასიხარულოდ უნდა აღინიშნოს, რომ განსხვავებები Internet Explorer, Netscape Navigator და World Wide Web Consortium (W3C - Internet -ის სტანდარტების საერთაშორისო ორგანიზა­ცია) -ს შორის არის უმნიშვნელო. ეს კი იძლევა საშუალებას გავერკვეთ ზოგად პროგრამი­რებად ობიექტების ერთობლიობაში ანუ ზოგად DOM -ში. თუკი გამოვიყენებთ ზოგად DOM -ს, შეიძლება შევადგინოთ Javascript -ის კოდი, რომელიც იქნება შეთავსებული მრავალ ბრაუზერთან.

ობიექტი window – მშობლიური ობიექტია ყველა სხვა ობიექტისთვის DOM-ში. შეიძლება ჩაითვალოს, რომ ობიექტი window თავად არის ბრაუზერი (Internet Explorer ან Netscape Navigator). ობიექტი window თავის მხრივ შეუძლია შეიცავდეს ობიექტს document-ს თავისი გამყოფილებებით <head> და <body> ან ელემენტს <frameset>. ყოველი ფრეიმი <frameset> ელემენტში, შეიძლება შეიცავდეს ობიექტ document-ს ან <frameset> -ის სხვა ელემენტებს.

ობიექტი window –ში აქტიურ ობიექტთან მისამართავად გამოიყენება გამოსახულება window.self . როცა ობიექტი window – შეიცავს <frameset>-ს, რომელიც თავის მხრივ შეიცავს კიდევ ობიექტ <frameset>-ებს, window.self შეიძლება გამოვიყენოთ იმისთვის რომ მივმარ­თოთ იმ დოკუმენტს, რომელიც არის აქტიურ ფრეიმში. window.parent -აქტიური ფრეიმის მშობლიურ ფრეიმში არსებულ დოკუმენტთან მისამართავად. window.top - მიმდინარე ფრეიმის ან ფანჯარის ყველაზე მაღალი ფრეიმის დოკუმენთთან მიმართვის­თვის. გვაქვს შემდეგი სინტაქსისი:

window.self // მიმართავს მიმდინარე აქტიურ ფრეიმს

window.parent // მიმართავს მშობლიურ აქტიურ ფრეიმს

window.top // მიმართავს ყველაზე ზემმდგომ ფრეიმს

თუკი ცალკეული დოკუმენტი, რომელსაც აპროგრამებთ განლაგებულია ცალკეულ ფრეიმში, მაშინ მასთან მიმართვა ხდება ფრეიმის დასახელებით:

Window.frameName

ყოველი *window* ობიექტი და ელემენტი <frame> შეიცავს ობიექტ document-ს, რომელიც შეიცავს გვერდის ყველა დანარჩენ ობიექტს. ობიექტ document-ზე მიუთითებენ ვებ-გვერდზე არსებული ყველა სხვა ობიექტები. როცა ხდება ობიექტ document-ის თვისებების და მეთოდების დაპროგრამება Javascript-ში, შეიძლება შეიქმნას შესრულებისას ვებ-გვერდის შიგთავსი კლიენტის მხარეზე.

**თვისებები**

ობიექტის თვისებები - არსებითად ეს არის ის მახასიათებლები, რომლებიც განსაზღვრავენ და ქმნიან ობიექტს. მაგალითად, ვთქათ, რომ ობიექტ ავტომობილს აქვს ორი მთავარი თვისე­ბა: *მარკა* და *მოდელი*. ეს ორი თვისება განსაზღვრავს, თუ როგორია ეს ავტომობილი.

დავუშვათ ნიკოს ჰყავს Ford Mustang-ი. ანუ მას აქვს ობიექტი - *ავტომობილი*, თვისება - *მარკას* აქვს მნიშვნელობა Ford, ხოლო თვისება მოდელი - აქვს მნიშვნელობა Mustang. თუკი ნიკოს უნდა იყიდოს ახალი Volvo C70, ობიექტს *ავტომობილი* შეეცვლება თვისება *მარკა* Volvo-თი, ხოლო თვისება *მოდელი* - C70-ით. თვისებების მნიშვნელობის მინიჭება, იძლევა საშუალებას შევცვალოთ ობიექტის მახასიათებლები.

თვისებების მნიშვნელობათა მინიჭებისას, შეიძლება მაგალითად, Javascript -ის კოდში შეიძლება გამოვაჩინოთ და შეიძლება არა, ჩარჩო ფრეიმის გარშემო:

window.document.frame[0].frameBorder=”yes”

ამ მაგალითში ობიექტ window -ს აქვს ობიექტი document -ი, რომელიც თავის მხრივ გააჩნია ფრეიმების მასივი (ერთობლიობა).ფრეიმერის მასივს ნულოვანი ინდექსით ( პირველი ელემენტი მასივ <frame>-ში) აქვს თვისება frameBorder, რომლის მნიშვნელობა იქნა yes, ეს ნიშნავს, რომ ფრეიმის გარშემო ჩარჩო არ იქნება თვალხილული.

თვისებები შეიძლება იყოს read-only (მხოლოდ წაკითხვა) ან read and write (წაკითხვა და ჩაწერა. read-only-ის თვისების ახალი მნიშვნელობა შეიძლება მივიღოთ ან არა მხოლოდ შესრულების დროს და არ შეიძლება შეცვლა. მნიშვნელობა read and write -ს მნიშვნელობა შეიძლება შეიცვალოს და მიღებულ იქნეს გამოთვლის შესრულებისას. წინა მაგალითში განხილული read and write-ის თვისების მნიშვნელობაა frameBorder . read-only-ის თვისების მაგალითია appName, რომელიც აბრუნებს ბრაუზერის დასახელებას:

Var myAppName = navigator. appName

ამ მაგალითში თვისება appName, navigator ობიექტისა შენახულია ცვლადში myAppName.

**მეთოდები**

თვისება - ეს ობიექტის მახასიათებელია. მეთოდი - ოპერაციებია ობიექტზე, რომლებიც აძლევენ მას მოქმედებების საშუალებას. მაგალითად თუ კი ავტომანქანის ბორბალი ლუჯია, მაშინ ობიექტი whell -ის თვისებას color –ს აქვს მნიშვნელობა blu:

Car.wheell.color=”blu”

მერეს მხრივ, ბორბალი შეიძლება იყოს რეგულირებადი, რაც ნიშნავს იმას, რომ იგი შეიცავს მეთოდს streer() :

Car.wheel.steer()

მეთოდების ჩაწერა, თვისებებისგან განსხვავებით ყოველთვის სრულდება მრგვალი ფრჩხილებით.

**კოლექციები**

კოლექციები - ეს იმ ობიექტების სიმრავლეა, რომლებიც არ არიან დაკავშირებულნი ობიექტების იერარქიით. ოჯახში მშობლები და შვილები იერარქიულად დაკავშირებულები არიან, ხოლო ღვიძლი ძმები არა, რამდენადაც იმყოფებიან იერარქიის ერთ დონეზე.

მაგალითად ნინო და ალი დაქორწილებულები არიან და ჰყავთ 4 შვილი: აიდა,ბარბარე, გიორგი, გოჩა. აი სინტაქსისი რათა მიუთითოთ ნინო და ალის შვილებს:

ალი.აიდა

ალი.გოჩა

ნინო.ბარბარე

ნინო.გიორგი

ხოლო ეს იერარქიული ჯაჭვი არასწორია

ალი.გიორგი.ბარბარე

ნინო.გოჩა.აიდა

მაშინ როცა აიდა,ბარბარე, გიორგი, გოჩა ეკუთვნიან ნინო და ალის, ბარბარე არ ეკუთვნის გიორგის და აიდა-ც არ ეკუთვნის გოჩას. რადგანაც ძმები და დები არიან იერარქიის ერთ დონეზე, ისინი ქმნიამ კოლექციას ან ღვიძლი ძმებისა და დების მასივს.

მასივში ყოველი ობიექტი არის ამ მასივის ობიექტი. იმისათვის, რომ პროგრამიდან გვქონდე წვდომა მასივის ელემენტებზე, უნდა ვიცოდეთ მასივი სინტაქსისი და ელემენტის ინდექსი მასივში. მასივში ელემენტის ინდექსი განისაზღვრება მისი რიგითობით, რომელიც აითვლება ნულიდა. მაგალითად:

window.document.images[0]

ეს მაგალითი მიმართავს პირველ ნახატს images[0] ნახატების მასივში. images[0] განსაზღვრავს პირველ სურათს. images[1] -მეორეს და ა.შ.

**კვანძები**

კვანძები - ეს არის კიდევ ერთი საშუალება HTML -ის ელემენტების დაპროგრამების. არის ორი ტიპის კვანძი: ელემენტის კვანძი და ტექსტური კვანძი. ელემენტის კვანძი შეიცავს ელემენტს და ტექსტს ელემენტის ტეგის შიგნით, მაშინ როცა ტექსტური კვანძი შეიცავს მხოლოდ ტექსტს.

<p>This is a paragraph</p>

ამ კოდში არის ორი კვანძი: <p>This is a paragraph</p> კვანძი რომელიც შეიცავს ტექსტურ კვანძს This is a paragraph .

სხვა მაგალითი,

<p><b>This is a phrase</b></p>

ამ მაგალითში ნაჩვენებია უკვე სამი კვანძი: <p><b>This is a phrase</b></p> კვანძია, იგი შეიცავს კვანძს <b>This is a phrase</b>, რომელიც თავის მხრივ შეიცავს ტექსტურ კვანძს This is a phrase

კვანძი რომელიც მეორე კვანძს შეიცავს ეწოდება *მშობლის* კვანძი, ამ უკანასკნელის მიმართ. ხოლო კვანძი, რომელიც შედის მეორეში - არის *კვანძი-შთამომავალი*. კვანძი-შთამომავლები, რომელთაც აქვთ მშობლების ერთიდაიგივე ატრიბუტები, ეწოდემა ძმები. ვებ-გვერდზე ყველა კვანძს აქვს მშობლის კვანძი, გამონაკლის წარმოადგენს კვანძი document, რომელსაც მხოლოდ კვანძები-შთამომავლები აქვს. მივიღოთ მხედველობაში, რომ DOM-ში ყველაზე მაღალი რანგის იერარქია ობიექტი window, ყველაზე ზედა ობიექტი კვანძთა ხეზეა - ობიექტი document . კვანძი document შეიძლება შეიცავდეს სხვა კვანძებს, მაგალითად <!DOCTYPE> და <html>. თავის მხრივ <html> შეიძლება შეიცავდეს სხვა კვანძები, <head> და <body> -ს ტიპისა, რომელიც ასევე შეიძლება შეიცავდეს სხვა კვანძებს.

ამისგან განსხვავებით, რომელიც იძლევა საშუალებას დავაპროგრამოთ ობიექტის ყოველი ელემენტი, ისე რომ არ მივმართოთ სხვა ელემენტს, კვანძების ხე გვაძლებს საშუალებას დავაპროგრამოთ HTML-ის ელემენტები, ელემენტებს შორის დამოკიდებულების საფუძველზე. მაგალითად ასე შეიძლება დავაპროგრამოთ მომდევნო ელემენტი, DOM -ის და ხეთა კვანძების გამოყენებით:

<div id=”myDiv”>first sample text

<p id=”myParag”>second sample text

</p>

</div>

გამოვიყენებთ რა DOM -ს, შეიძლება დავაპროგრამოთ ელემენტი <div> უშუალოდ:

Window.document.myDiv

გამოვიყენოთ რა ხეთა კვანძები, შეიძლება დავაპროგრამოთ <P> ელემენტი, როგორც კვანძი-მემკვიდრე <div> ელემენტისა, ხოლო ელემენტ <div> - როგორც მშობელი ელემენტისა:

window.document.myDiv.firstChild

window.document.myParag.parentNode