**რა არის JavaScript?**

JavaScript შეიქმნა ვებ გვერდების “გასაცოცხლებლად” და სტატიკურ HTLM-ში ინტერაქტიულობის შემოსატანად. ამ ენაზე დაწერილ ფუნქციონას სკრიპტები ეწოდება. მათი ჩაწერა შესაძლებელია პირდაპირ ვებ გვერდის HTML-ში და ავტომატურად ამუშავება გვერდის ჩატვირთვისას. სკრიპტები შესაძლებელია ჩავწეროთ როგორც უბრალო ტექსტი და ასევე მარტივად ხდება მისი გაშვება.

რატომ ჰქვია მას Java Script? თავდაპირველად JavaScript-ს სხვა სახელი ჰქონდა: "LiveScript". ვინაიდან ჯავა იმ დროს ძალიან პოპულარული იყო გადაწყდა, რომ ახალი ენის პოზიციონირება, როგორც ჯავის "უმცროსი ძმა" მას დასაწყისში დაეხმარებოდა. თუმცა განვითარებასთან ერთად JavaScript გახდა სრულიად დამოუკიდებელი ენა თავისი სპეციფიკაციით, სახელწოდებით ECMAScript. თანამედროვე JavaScript საერთოდ არ აქვს კავშირი Java-სთან.

დღეს JavaScript-ს შეუძლება გაეშვას არა მხოლოდ ბრაუზერში, არამედ სერვერზე და ნებისმიერ მოწყობილობაზე, რომელსაც აქვს სპეციალური პროგრამა სახელწოდებით JavaScript engine .

**რისი გაკეთება შეუძლია ბრაუზერში JavaScript-ს?**

ბრაუზერში JavaScript-ს შეუძლია გააკეთოს ყველაფერი, რაც დაკავშირებულია ვებგვერდის მანიპულირებასთან, მომხმარებელთან და ვებ სერვერთან ურთიერთქმედებებთან.

მაგალითად, ბრაუზერში JavaScript-ს შეუძლია:

* დაამატოს ახალი HTML გვერდზე, შეცვალოს არსებული ქონთენთი და სტილები;
* რეაგირება მოახდინოს იუზერის ქმედებებზე(action), მაუსის დაწკაპუნებაზე/ მაუსის ივენთებზე, პოინტერის/კურსორის მოძრაობაზე და კლავიატურაზე დაჭერაზე;
* გაგზავნოს მოთხოვნები ქსელის საშუალებით დისტანციურ სერვერებზე, ჩამოტვირთოს და ატვირთოს ფაილები (ე.წ. AJAX ტექნოლოგია);
* მოახდინოს ქუქისების get და set, მოახდინოს ინტერაქცია იუზერთან (დაუსვას შეკითხვები) და აჩვენოს შეტყობინებები;
* დაიმახსოვროს მონაცემები კლიენტის მხარეს ("local storage").

**რა ხდის JavaScript-ს უნიკალურს?**

JavaScript-ის შესახებ მინიმუმ სამი კარგი რამ არსებობს:

* სრული ინტეგრაცია HTML/CSS-თან;
* მარტივი კეთდება მარტივად;
* მხარდაჭერილია ყველა ძირითადი ბრაუზერის მიერ და ჩართულია ნაგულისხმევად.

JavaScript არის ერთადერთი ბრაუზერის ტექნოლოგია, რომელიც აერთიანებს ამ სამ რამეს.

სწორედ ეს ხდის JavaScript-ს უნიკალურს. სწორედ ამიტომ არის ყველაზე გავრცელებული ინსტრუმენტი ბრაუზერის ინტერფეისების შესაქმნელად.

გარდა ამისა, JavaScript შეიძლება გამოყენებულ იქნას სერვერებთან საკომუნიკაციოდ და მობილური აპლიკაციების შესაქმნელად.

<script> ტეგი

JavaScript კოდი შეიძლება ჩასვათ თითქმის ყველგან HTML დოკუმენტში ტეგის გამოყენებით <script>.

Მაგალითად:

<!DOCTYPE HTML>

<html>

<head></head>

<body>

<p>test script...</p>

<script>

alert( 'Hello, world!' );

</script>

</body>

</html>

მოცემული ტეგი <script> შეიცავს JavaScript კოდს, რომელიც ავტომატურად სრულდება, როდესაც ბრაუზერი ამუშავებს ტეგს.

თუ დიდი JavaScript კოდი გვაქვს, შეგვიძლია ცალკე ფაილში გავიტანოთ. სკრიპტის ფაილები მიმაგრებულია HTML-ზე ატრიბუტით src-ს გამოყენებით:

<script src="js/script.js" ></script>

გაითვალისწინეთ:

როგორც წესი, მხოლოდ უმარტივესი სკრიპტები იწერება HTML-ში. უფრო რთული სკრიპტების ცალკე ფაილებში გატანაა რეკომენდირებული. ცალკე ფაილის უპირატესობა ის არის, რომ ბროუზერი ჩამოტვირთავს მას და შეინახავს თავის ქეშში. ხოლო გვერდები, რომლებიც მიუთითებენ იმავე სკრიპტზე, მას ქეშიდან გამოიყენებენ ჩამოტვირთვის ნაცვლად. ეს ამცირებს ტრაფიკს და აჩქარებს გვერდებს.

თუ src გვაქვს გამოყენებული, მაშინ სკრიპტის შინაარსი იგნორირებულია. ერთ <script> ტეგს არ შეიძლება ჰქონდეს ატრიბუტი src და კოდი შიგთავსში.

ქვემოთ მოცემული კოდი არ იმუშავებს:

<script src="script.js" >

alert(' ტესტ ‘); // ეს ქონთენთი ინგორირებულია, რადგან src გვაქვს

</script>

**კომენტარები**

პროგრამები უფრო და უფრო რთული ხდება. საჭირო ხდება კომენტარების დამატება, რომლებიც აღწერს რას აკეთებს კოდი და რატომ.

კომენტარები შეიძლება განთავსდეს სკრიპტის ნებისმიერ ადგილას. ისინი გავლენას არ ახდენენ მის შესრულებაზე, რადგან ძრავი კომპილაციის დროს მათ უგულებელყოფს.

ერთი ხაზის კომენტარები იწყება ორი წინ დახრილი სიმბოლოთი // და მის მარჯვნივ დანარჩენი ხაზი არის კომენტარი. თუმცა მან შეიძლება სრული ხაზი არც დაიკავოს და უბრალოდ მოსდევდეს კოდის ფრაგმენტს.

როგორც მოცემული გაქვს ქვემოთ მაგალითში:

// კომენტარი იკავებს სრულ ხაზს/სტრიქონს

alert('Hello');

alert('World'); // კომენტარი მოსდევსკოდის ფრაგმენტს

მრავალსტრიქონიანი კომენტარი იწყება წინ დახრილი ხაზით რომელსაც მოსდევს ფიფქი /\* და მთავრდება ფიფქით, რომელსაც მოსდევს წინ დახრილი ხაზი \*/.

როგორც მოცემული გაქვს ქვემოთ მაგალითში:

/\* მრავალსტრიქონიანი კომენტარი მაგალითი

მოცემულია ამ ორ სტრიქონში

\*/

alert('Hello');

alert('World');

**ცვლადი**

ცვლადი არის "სახელწოდებული საცავი" მონაცემისათვის. JavaScript-ში ცვლადის შესაქმნელად გამოიყენეთ let საკვანძო სიტყვა.

ქვემოთ მოცემულა ცვლადი სახელწოდებით „მესიჯი“:

let message;

message = 'Hello'; // ვინახავთ მესიჯი ცვლადში 'Hello' ტექსტს.

ძველ სკრიპტებში, თქვენ ასევე შეიძლება შეგხვდეთ ცვლადის გამოცხადება var-ით ნაცვლად let-ისა. var საკვანძო სიტყვა თითქმის იგივეა რაც let, თუმცა მათ შორის არის ფუნდამენტური განსხვავება. ცვლადები, რომლებიც გამოცხადებულია var-ით, არის ან ფუნქციის მასშტაბის(function-scoped) ან გლობალური მასშტაბის(global-scoped).

Მაგალითად:

if (true) {

var test = true; // "var" ცვლადის გამოცხადება

}

alert(test); // true

ვინაიდან var აიგნორების კოდის ბლოკებს, ჩვენ მივიღეთ გლობალური ცვლადი test.

თუ გამოვიყენებდით let test, var test-ის ნაცვლად მაშინ ცვლადი მხოლოდ if ბლოკის შიგნით იქნება ხილული.

გავიმეოროთ ზემოთ მოცემული მაგალითი:

if (true) {

let test = true; // "let" ცვლადის გამოცხადება

}

alert(test); // ReferenceError: test is not defined

ერთადერთი გამონაკლისი არი ფუნქციის შიგნით, მაშინ var ხდება ფუნქციის დონის ცვლადი.

JavaScript-ში ცვლადების სახელებზე ორი შეზღუდვა არსებობს:

1. სახელი უნდა შეიცავდეს მხოლოდ ასოებს, ციფრებს ან სიმბოლოებს $ და \_
2. პირველი სიმბოლო არ უნდა იყოს ციფრი

ასევე მნიშვნელოვანია ვიცოდეთ, რომ ცვლადები დასახელებულია web და WEB არის ორი განსხვავებული ცვლადი, ანუ JS-ი არის რეგისტრების მიმართ მძნობიარე/ ქეი სენსითივი.

მნიშვნელოვანია აღვნიშნოთ, რომ არსებობს რეზერვირებული სიტყვების სიაც. ისინი არ შეიძლება გამოყენებულ იქნას როგორც ცვლადის სახელები, რადგან მათ თავად ენა იყენებს.

მაგალითად: let, class, return და function რეზერვირებულია.

მუდმივი (უცვლელი) ცვლადის გამოსაცხადებლად const გამოიყენება let-ის ნაცვლად. const semesterWeeks = 15; ახლა რომ ვცადოთ semesterWeeks ცვლადის ცვლილება მივიღებთ შეცდომას error, can't reassign the constant!. ზოგადად კონსტანტები გამოიყენება როდესაც პროგრამისტი დარწმუნებულია, რომ ცვლადი არასოდეს შეიცვლება. ამ სახით გამოცხადება ცვლადის გვაძლევს გარანტიას რომ ყველას ნათლად მიეწოდა მისი უცვლელობის ფაქტი.

არსებობს ფართოდ გავრცელებული პრაქტიკა მუდმივები დასახელებულია დიდი ასოებით და '\_' სიმბოლოთი. მაგალითად const COLOR\_BORDO = "#D10000"; ანუ კონსტანტები რომლეთა მნიშვნელობაც თავიდანვე ვიცით და რანთაიმის დროს არ ვიღებთ.

ცვლადების შესახებ ძირითადი დეტალებო რომ შევაჯამოთ:

* let– არის თანამედროვე ცვლადი დეკლარაცია.
* var– არის ძველი სკოლის ცვლადის დეკლარაცია. ჩვეულებრივ მას არ ვიყენებთ, მაგრამ აუცილებელია განსხვავების ცოდნა let და var შორის. რათა მზად ვიყოთ იმ შემთხვევისათვის, თუ ის დაგჭირდებათ.
* const– მსგავსია let-ისა, მაგრამ ამ უკანასკნელის შემთხვევაში ცვლადის მნიშვნელობა არ შეიძლება შეიცვალოს.

**მონაცემთა ტიპები**

JavaScript-ში ცვლადი ყოველთვის გარკვეული ტიპისაა. მაგალითად სტრიქონი ან რიცხვი. JavaScript-ში რვა ძირითადი მონაცემთა ტიპია.

მონაცემთა შვიდი პრიმიტიული ტიპი:

1. Number ნებისმიერი სახის რიცხვებისთვის: მთელი რიცხვი ან მცოცავი წერტილით(ათწილადი)
2. Bigint მთელი რიცხვებისთვის.
3. String სტრიქონი. შეიძლება ჰქონდეს ნული ან მეტი სიმბოლო
4. Boolean - true/ false.
5. Null – დამოუკიდებელი ტიპი, რომელსაც აქვს ერთი მნიშვნელობა null.
6. Undefined – დამოუკიდებელი ტიპი, რომელსაც აქვს ერთი მნიშვნელობა undefined.
7. Symbol უნიკალური იდენტიფიკატორებისთვის.

და ერთი არაპრიმიტიული მონაცემთა ტიპი:

1. Object მონაცემთა უფრო რთული სტრუქტურებისთვის.

ოპერატორი typeof საშუალებას გვაძლევს ვნახოთ რომელი ტიპია შენახული ცვლადში. ჩვეულებრივ გამოიყენება როგორც typeof x, მაგრამ ასევე შესაძლებელია typeof(x) ვარიანტის გამოყენება.

ცალკე მინდა გამოვყო JavaScript-ში სპეციფიური სტრინგის სახეობა.

JavaScript-ში არის სტრინგის/სტრიქონის 3 ტიპი.

1. ორმაგი ბრჭყალით: "Hello".
2. ერთმაგი ბრჭყალით: 'Hello'.
3. Backticks : `Hello`.

ორმაგი და ერთმაგ ბრჭყალებში JavaScript-ში პრაქტიკულად არანაირი განსხვავება არ არის.

განსხვავებული Backticks "გაფართოებული ფუნქციონირების" სტრინგები/ სტრიქონები. ისინი საშუალებას გვაძლევს ჩავრთოთ ცვლადები, არითმეტიკული ოპერაციები და დინამიური მონაცემი სტრიქონში, თუ მათ შემოვფარგლავთ ${ჩავრთოთ ცვლადები, არითმეტიკული ოპერაციები და დინამიური მონაცემი} მოგემული ფორმით.

მაგალითად:

let subject = 'JS';

// ცვლადის ჩაშენების მაგალითი

alert( `My first, ${subject}!` ); // შედეგი იქნება: My first, JS!

// არითმეტიკული ოპერაციის ჩაშენების მაგალითი

alert( `the result is ${1 + 2}` ); // the result is 3

**ფუნქციები მომხმარებელთან ურთიერთობისთვის: alert, prompt და confirm.**

რადგან ჩვენ გამოვიყენებთ ბრაუზერს, მოდით ვნახოთ რამდენიმე ფუნქცია მომხმარებელთან ურთიერთობისთვის: alert, prompt და confirm.

alert/გაფრთხილება ის უკვე რამოდენიმეჯერ გამოვიყენეთ. ის აჩვენებს შეტყობინებას და ელოდება მომხმარებლი როდის დააჭერს "OK"-ს. Მაგალითად: alert("Hello");

prompt/დროულად იღებს ორ არგუმენტს: result = prompt(title, [default]); ის აჩვენებს მოდალურ ფანჯარას ტექსტური შეტყობინებით, მომხმარებლისათვის შეყვანის ველით და ღილაკებით OK/Cancel. title არის ტექსტი იუზერისთის საჩვენებლად. default სურვილისამებრ მეორე პარამეტრი, საწყისი მნიშვნელობის მინიჭება შესაყვანი ველისათვის. კვადრატული ფრჩხილები სინტაქსში[...] მიუთითებს, რომ პარამეტრი არჩევითია და არ არის აუცილებელი.

Მაგალითად:

let age = prompt('რამდენი წლის ბრძანდებით?', 100);

alert(`თქვენ ბრძანდებით ${age} წლის!`); //შედეგი: თქვენ ბრძანდებით 100 წლის!

confirm/დაადასტურეთ. ფუნქცია confirm აჩვენებს მოდალურ ფანჯარას რომელშიც გვაქვს ტექსტი და ორი ღილაკებით: OK და Cancel. შედეგი არის true თუ დააჭერთ OK და false თუ დააჭერთ Cancel-ს.

Მაგალითად:

let isHappy = confirm("ბედნიერი ხარ?");

alert( isHappy ); // true თუ დავაჭირეთ OK ღილაკს

**ძირითადი არითმეტიკული ოპერატორები**

JS-ში მხარდაჭერილია ყველა მათემატიკური ოპერაცა:

* დამატება +,
* გამოკლება -,
* გამრავლება \*,
* მთელი გაყოგა /,
* წილადი ნაწილის აღება %,
* ახარისხება \*\*.

**შედარების ოპერატორები**

JS-ში მხარდაჭერილია არითმეტიკული შედარების ოპერატორი რომლებიც JavaScript-ში ასე იწერება:

* მეტია/ნაკლებია ვიდრე: a > b, a < b.
* მეტია/ნაკლები ანდა ტოლია: a >= b, a <= b.
* ტოლია მნიშვნელობა: a == b.
* მკაცრი ტოლობა, მონაცემის ტიპის ჩათვლით: a === b
* არ არის ტოლი: a != b.

**ლოგიკური ოპერატორები**

JS-ში მხარდაჭერილია შემდეგი ლოგიკური ოპერატორები:

|| (ან)

&& (და)

! (არა)

**პირობითი ოპერატორები: IF**

if(...) აფასებს მდგომარეობას ფრჩხილებში და თუ შედეგი არის true, ახორციელებს კოდის ბლოკს.

Მაგალითად:

let year = prompt('რომელ წელს გამოქვეყნდა ECMAScript-2015 სპეციფიკაცია?', '');

if (year == 2015) alert( 'პასუხი სწორია!' );

ისევე როგორც ნებისმიერი სხვა პროგრამირების ენაში if შეიძლება მოაცავდეს არასავალდებულო else if ან else ბლოკსაც.

**პირობითი ოპერატორი '?'**

ზოგჯერ გვჭირდება ცვლადის ინიციალიზება მდგომარეობიდან გამომდინარე.

let result = condition ? value1 : value2;

ფასდება condition: თუ სიმართლეა, მაშინ value1ბრუნდება, წინააღმდეგ შემთხვევაში – value2.

**ცილკის ოპერატორები while, do..while და for**

while– მდგომარეობა მოწმდება ყოველი გამეორების წინ.

do..while– მდგომარეობა მოწმდება ყოველი გამეორების შემდეგ.

for (საწყისი; პირობა; ბიჯი – მდგომარეობა მოწმდება ყოველი გამეორების წინ, ხელმისაწვდომია დამატებითი პარამეტრები.

switch ოპერატორი

switchშეუძლია შეცვალოს მრავალი if შემოწმება. ის იძლევა უფრო აღწერად ვარიანტს, რომ შევადაროთ ცვლადი მის მრავალ შესაძლო მნიშვნელობას.

switch(x) {

case 'value1': // if (x === 'value1')

...

[break]

case 'value2': // if (x === 'value2')

...

[break]

default:

...

[break]

}

ფუნქციები

JS-ში გაქვს Function, Function expression, Arrow function.

function showMessage() { // Function

alert( 'Hello everyone!' );

}

let sayHi = function() { //Function expression

alert( "Hello" );

};

let func = (arg1, arg2, ..., argN) => { //Arrow function

return expression;

};

**DOM tree / DOM ხე**

HTML დოკუმენტის ხერხემალი არის ტეგები. დოკუმენტის ობიექტის მოდელის (DOM) მიხედვით, ყოველი HTML ტეგი არის ობიექტი. ჩადგმული ტეგები არის „შვილები“. ტეგის შიგნით ტექსტი ასევე არის ობიექტი. ყველა ეს ობიექტი ხელმისაწვდომია JavaScript-ის გამოყენებით და ჩვენ შეგვიძლია მათი გამოყენება გვერდის შესაცვლელად.

მაგალითად: document.body წარმოადგენს <body> ტეგს. გავაწითლოთ body ტეგი წითლად js-ის გამოყენებით document.body.style.background = 'red';

ძიება: getElement\*, querySelector\*

ჩვენ შეგვიძლია ავითოღ ელემენტი js-ში ძიების მეთოდების გამოყენებით:

querySelectorAll, getElementById, getElementsByName, getElementsByTagName, getElementsByClassName

document.getElementById

<div id="elem">

<div id="elem-content">Element</div>

</div>

<script>

let elem = document.getElementById('elem');

elem.style.background = 'red';

</script>

document.querySelectorAll

<ul>

<li>The</li>

<li>test</li>

</ul>

<ul>

<li>has</li>

<li>passed</li>

</ul>

<script>

let elements = document.querySelectorAll('ul > li:last-child');

for (let elem of elements) {

alert(elem.innerHTML); // "test", "passed"

}

</script>

document.getElementsBy\*

* document.getElementsByTagName(tag) ეძებს ელემენტებს მოცემული ტეგით და აბრუნებს მათ კოლექციას/მასივს. პარამეტრი tag ასევე შეიძლება იყოს ვარსკვლავი "\*" "ნებისმიერი ტეგისთვის".
* document.getElementsByClassName(className) აბრუნებს ელემენტებს, რომლებსაც აქვთ მოცემული CSS კლასი.
* document.getElementsByName(name) აბრუნებს ელემენტებს მოცემული ატრიბუტით name, დოკუმენტის მასშტაბით. ძალიან იშვიათად გამოიყენება.

DOM მოდიფიკაცია არის "live" გვერდების შექმნის გასაღები.

**ელემენტის შექმნა**

DOM კვანძების შესაქმნელად, არსებობს ორი მეთოდი:

document.createElement(tag).

ქმნის ახალ ელემენტის კვანძს მოცემული ტეგით: let div = document.createElement('div');

document.createTextNode(text).

ქმნის ახალ ტექსტურ კვანძს მოცემული ტექსტით: let textNode = document.createTextNode('Here I am');

უმეტეს დროს ჩვენ გვჭირდება ელემენტების კვანძების შექმნა, როგორიცაა divშეტყობინების გაგზავნა.

განვიხილოთ შეტყობინების div-ის შექმნის მაგალი, რომელი 3 ნაბიჯისგან შედგება:

// 1. შევქმნათ <div> ელემენტი

let div = document.createElement('div');

// 2. დავამატორ კლასი "alert" ჩვენს მიერ შექმნილ ტეგს

div.className = "alert";

// 3. შევიტანოთ ქონთენთი ჩვენს მიერ შექმნილ ტეგში

div.innerHTML = "<strong>Hi there!</strong> You've read an important message.";

**ჩასმის მეთოდები**

მოცემული <div> ელემენტი, რომ ჩავსვათ დოკუმენტში უნდა გამოვიყენოთ შემდეგი კოდის მაგალითი:

<script>

let div = document.createElement('div');

div.className = "alert";

div.innerHTML = "<strong>Hi there!</strong> You've read an important message.";

document.body.append(div);

</script>

აქ ვიძახებთ append-ს document.body-ზე, მაგრამ შეგვიძლია გამოვიძახოთ append მეთოდი ნებისმიერ სხვა ელემენტზე. მაგალითად, შეგვიძლია დავამატოთ რაღაც <div> და მასში ჩავწეროთ სხვა ელემენტი div.append(anotherElement).

არსებობს ჩასმის მეტი მეთოდები, ისინი განსაზღვრავენ ადგილს სადაც ჩასვავენ ელემეტს:

* node.append(...nodes or strings)- დაამატე კვანძები ან სტრიქონები node-ის ბოლოში ,
* node.prepend(...nodes or strings)- ჩადეთ კვანძები ან სტრიქონები node-ის დასაწყისში,
* node.before(...nodes or strings)– კვანძების ან სტრიქონების ჩასმა node-მდე,
* node.after(...nodes or strings)– კვანძების ან სტრიქონების ჩასმა node-ის შემდეგ,
* node.replaceWith(...nodes or strings)–- ცვლის node მოცემული კვანძებით ან სტრიქონებით.

ამ მეთოდების გამოყენების მაგალითი:

<ol id="ol">

<li>0</li>

<li>1</li>

<li>2</li>

</ol>

<script>

ol.before('before'); // ჩასვი სტრინგი "before" <ol>-მდე

ol.after('after'); // ჩასვი სტრინგი "after" <ol> შემდეგ

let liFirst = document.createElement('li');

liFirst.innerHTML = 'prepend';

ol.prepend(liFirst); // ჩასვი liFirst <ol>-ის დასაწყისში

let liLast = document.createElement('li');

liLast.innerHTML = 'append';

ol.append(liLast); // ჩასვი liLast <ol>-ის ბოლოში

</script>

**ბრაუზერის ივენთები**

ივენთი არის სიგნალი იმისა, რომ რაღაც მოხდა. ყველა DOM ელემტი ქმნის ასეთ სიგნალებს.

ქვემოთ მოცემულია DOM-ის ყველაზე სასარგებლო ივენთების სია:

მაუსის მოვლენები:

* click– როდესაც მაუსი დააწკაპუნებს ელემენტზე.
* contextmenu– როდესაც მაუსი აწკაპუნებს ელემენტზე მარჯვენა ღილაკით.
* mouseover/ mouseout– როდესაც მაუსის კურსორი მოდის / ტოვებს ელემენტს.
* mousedown/ mouseup– როდესაც მაუსის ღილაკზე დაჭერით / აშვება ელემენტზე.
* mousemove- როდესაც მაუსი გადაადგილდება.

კლავიატურის მოვლენები:

* keydown და keyup– როდესაც კლავიატურის ღილაკზე დაჭერა და აშვება ხდება.

**ღონისძიების დამამუშავებლები / Event handlers**

მოვლენებზე რეაგირებისთვის შეგვიძლია გამოვიყენოთ დამმუშავებელი ფუნქციები, რომელიც მუშაობას იწყებს ივენთის დროს.

დამმუშავებლის დაყენება შესაძლებელია HTML-ში ატრიბუტით სახელად on<event>.

მაგალითად: <input value="Click me" onclick="alert('Click!')" type="button">

დამმუშავებელი DOM თვისების გამოყენებით on<event>.

<input id="elem" type="button" value="Click me">

<script>

elem.onclick = function() {

alert('Thank you');

};

</script>

დამმუშავებლის დამატების სინტაქსი:

element.addEventListener(event, handler);

მაგალითად: elem.addEventListener( "click" , () => alert('Thanks!'));