JAVLON ABDULLO

MUKAMMAL DASTURLASH

2

JavaScript asosida dasturlashni o'rganish
va bilimni mustahkamlash uchun
mukammal qo'llanma

JavaScript

Javlon ABDULLO

MUKAMMAL DASTURLASH

2-kitob:

JavaScript

3-nashr

Toshkent
«Akademnashr»
2022

UO‘K: 004.42
KBK: 32.973-018
A 15

A 15 **Abdullo, Javlon**
JavaScript [Matn]: ilmiy-ommabop / Abdullo Javlon. – Toshkent: Akademnashr, 2022. – 300 b.

ISBN 978-9943-6501-6-9

UO‘K: 004.42
KBK: 32.973

Ushbu kitob zamonaviy dasturlashni o‘rganmoqchi bo‘lganlar, dasturlash bilan shug‘ullanadiganlar va hatto dasturiy ta’minot yaratish bo‘yicha yetuk mutaxassislarga ham mo‘ljallangan bo‘lib, ilm olish va uni mustahkamlash uchun eng kerakli bo‘ladigan zaruriy bilimlarni o‘zida jamlagan hamda sodda tushunarli tilda bayon etilgan. Ortiqcha ma’lumotlar o‘rniga o‘n yillab yig‘ilgan tajribalardan foydalanib aniq ko‘rsatmalar keltirilgan. Kitobdagi ma’lumotlarni o‘zlashtirgan o‘quvchi malakali veb-dizayner sifatida keng faoliyat yuritshdan tashqari, nafaqat «front-end» (timusti) dasturlovchi bo‘lishi mumkin, balki to‘laqonli veb-dasturiy ta’minot yaratish borasida yetuk bilimlarga ham ega bo‘ladi.

Kitobda yoritilgan mavzular bo‘yicha fikr-mulohazalar, takliflar, savollar hamda o‘z amaliy dasturlash faoliyatingizda muammolar yuzaga kelsa, ***ilm.yurt.uz*** *sayti orqali murojaat qilishingiz mumkin.*

ISBN 978-9943-6501-6-9

1-BO‘LIM
KIRISH

1.1. Veb-dasturlash tili: yaratilish tarixiga bir nazar

JavaScript – dunyoda eng ommabop va juda keng tatbiq etiladigan veb-dasturlash tilidir. Hozirda barcha veb-brauzerlar ushbu tilni qo‘llagani bois uning joriy etilish qamrovi nihoyatda keng, u har bir sohaga faol kirib borgan.

Ushbu til haqida mukammal tasavvurga ega bo‘lish uchun, avvalo, veb-texnologiyaning ishlash uslubi bilan tanishib chiqish lozim bo‘ladi. Sodda qilib aytganda, brauzerda ishorat yozib, unga murojaat qilinganda, ko‘rsatilgan manzildagi serverga so‘rov yuboradi. U yerdagi tegishli dasturiy ta’minot mos ravishda javob qaytaradi. Brauzer ham, o‘z navbatida, kelgan javob bo‘yicha natijani aks ettiradi. Umuman olganda, brauzer bajaradigan barcha xatti-harakatlar foydalanuvchining o‘z kompyuterida (yoki maxsus qurilmasida) amalga oshiriladi, buni foydalanuvchi tomoni sifatida qaraladi.

Juda keng ommalashayotgan HTMLga qo‘shimcha imkoniyatlar berish uchun u bilan ishlaydigan dastur-

lash tili yaratishga ehtiyoj sezilgan edi. Bunday dasturlash tili shunday qulay bo'lishi kerak ediki, toki dasturlash bo'yicha chuqur bilimga ega bo'lmaganlar ham, faqat joylashtiruv tilini o'zlashtirganlar ham uni qiyinchiliksiz qo'llay olishsin. JavaScriptgacha mavjud dasturlash tillari operatsion tizimda ishlashga mo'ljallangan bo'lib, hech biri ochiq o'giriluvchan emasdi.

1995-yil bunday talablarga javob beradigan sodda, ochiq kodli dasturiy ta'minotni qo'llaydigan, o'girib-ishlatuvchi vositani mavjud «Netscape»ga yangi 2.0-talqini (versiyasi) sifatida biriktirib, tez vaqt ichida muomalaga chiqarish zarurati tug'ildi. Agar biroz kechikilsa, «Microsoft» kompaniyasi o'z mahsuloti bilan axborot texnologiyalari bozorini zabt etishi turgan gap edi. JavaScript yaratuvchisi Brendan Eyx xotiralarida yozishicha, u o'n kun ichida tinimsiz izlanib yangi tilning yadrosini yaratadi.

«Netscape Communications» kompaniyasi ta'sischilaridan Mark Andressen va «Sun Microsystems» ta'sischilaridan Bill Djoy, Brendan Eyx boshchiligida Java dasturlash tilining sodda qismini o'z ichiga olgan va sintaksisi jihatidan «C» dasturlash tiliga o'xshatilgan ochiq kodli til yaratishdi. Mazkur til tarkibi jihatidan «Self», «Smalltalk» va «Lisp» dasturlash tillariga yaqin ko'rinishga keltirildi. Albatta, ushbu o'xshashliklar va sodda sintaksisli ekani tufayli ushbu til ilk namoyishdayoq axborot texnologiyalari sohasidagi yetuk 28 ta kompaniya tomonidan tan olindi. Ular o'z mahsulotlarida shu tildan foydalanishni, uni amaliyotga tatbiq

etishni lozim topishdi. Lekin shoshilinch tayyorlangan til o'ziga yarasha nuqsonlarga ham ega edi. Ko'p o'tmay «Microsoft»ning undan ko'ra mukammal hisoblangan va uning o'rnini bosa oladigan «Visual Basic» asosidagi «JScript» hamda «VBScript» tillari yaratildi.

Ammo «Netscape»da mavjud kamchiliklar vaqtida, tez bartaraf etilganligi yuqoridagilarning ommalashishiga yo'l bermadi. Shu bilan birga, «Netscape» o'z tilini «ECMAScript» (European Computer Manufacturer's Association) nomi ostida standartlashtirishga erishdi.

2000-yillargacha deyarli barcha hisob-kitoblar, bajariladigan ishlar server tarafida amalga oshirilgani tufayli JavaScript mukammal til sifatida dasturlovchilar tomonidan u qadar keng qo'llanilmadi. Veb-texnologiya rivojlanib borgani sari ming-million foydalanuvchilarga ma'lumot uzatadigan serverlarda so'rovlarni kamaytirishga zarurat tug'ildi. Bundan tashqari, foydalanuvchi (mijoz) tomonida dasturning bajarilishi, faqatgina yagona foydalanuvchi uchun mo'ljallangani bois tez amalga oshishi bilan birga transportga ketadigan vaqtdan ham yutishga imkon beradi. Chunki server nafaqat butunlay boshqa joyda, balki ummon ortida, bir necha ming kilometr olisda joylashishi mumkin.

2005-yilga kelib «Google» kompaniyasi JavaScript yordamida veb-xaritani taqdim etdi va uning juda katta imkoniyatlarga egaligini butun jahonga namoyon qildi. Faqat JavaScript bilan ishlovchi «Google»ning katta jamoasi o'z sohasini kengaytirish maqsadida serverda ham aynan shu til orqali ishlovchi «**node.js**»ni taqdim

etishdi. Hozirgi kunda JavaScript nafaqat foydalanuvchi tomonida, balki serverda ham ishlaydigan mukammal dasturlash tili sifatida keng taqdim etilmoqda.

«Google» va axborot texnologiyalari sohasida ishlovchi boshqa yetakchi kompaniyalar hamda keng dasturlovchilarning talab va takliflarini inobatga olgan holda 2010-yilga kelib JavaScriptning 1.8.5 talqini yaratildi. Albatta, unda 15 yil oldingisiga nisbatan ulkan imkoniyatlar qo‘shilgan (masalan, JQuery imkoniyatlarini deyarli qamrab olgan), katta o‘zgarishlarga ega va xatarli nuqsonlardan xoli, shu bilan birga, ommabop brauzerlarda bir xilda natija beradigan yagona tarkibga keltirilgan.

1.2. Qo‘llanish sohasi

Dastavval JavaScript faqat veb-brauzerlar uchun foydalanuvchining qurilmasida ishlaydigan dasturiy ta’minot sifatida yaratildi. So‘ngra esa, yuqorida qayd etilgani kabi, serverda faoliyat yuritadigan talqinlari ham ixtiro etildi. Ushbu kitobda asosan foydalanuvchi (mijoz) tomonida — veb-brauzerda ishlaydigan asl JavaScript imkoniyatlari haqida atroflicha so‘z yuritiladi. Serverning o‘zida JavaScript asosida ishlaydigan dasturiy ta’minotlar (**node.js**, **nitro**, **rhino**, **sling** kabi) bayon etilmaydi. Umuman olganda, serverda ishlaydigan boshqa dasturiy ta’minotlar ko‘p. Shuni nazarda

tutsak, asosini JavaScript tashkil etadigan ta’minotlar ommaviy qo‘llaniladi deb bo‘lmaydi. Ammo dasturlovchilarning ko‘pi ommaviy tarzda faqat JavaScriptdan foydalanishadi. Barcha brauzerlar uni qo‘llaydi. Xuddi HTML singari oddiy matn tahrirlovchi muhit orqali yozilgan kodni ixtiyoriy veb-brauzer orqali aks etishini ko‘rish mumkin. Amalda esa faqat kompyuterdagi veb-brauzerlardan tashqari smartfonlarda, planshetlarda, veb-orazli (interfeysli) barcha oynali qurilmalarda va hattoki televizor va soatlarda ham ma’lum doiradagi dasturlar tuzish va natijasidan bahramand bo‘lish mumkin.

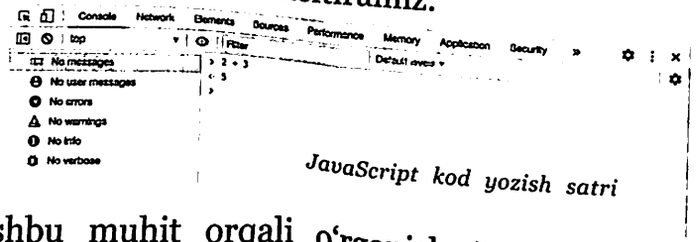
Texnik jihatdan HTML namoyon bo‘lgan joyda JavaScript ham ishlaydi, faqat maxsus tarzda brauzerning sozlamasida o‘chirib qo‘yilmagan bo‘lishi kerak. Ushbu kitobda hozirda eng so‘nggi hisoblangan (aslida barcha imkoniyatlari haligacha to‘liq tatbiq etib bo‘linmagan) **ECMAScript**ning **10**-talqinidagi (ECMAScript 2019) to‘ldirishlar bilan JavaScript imkoniyatlari xususidagi ma’lumotlar boyitilib, asoslantirilib, misollar bilan batafsil yoritilgan. Keltirilgan ilmlar barcha (desktop va mobile) brauzerlarda bir xil ishlaydi, ba’zi maxsus istisnolar o‘z o‘rnida bayon etilgan.

Biz misol tariqasida ilk mukammal brauzer bo‘lgan «Netscape» o‘rniga JavaScript ixtirochilari tomonidan taqdim etilgan **FireFox** va undan qanday foydalanilishi xususida batafsil so‘z yuritamiz. Negaki, u ochiq kodli va keng ommaviy tarzda takomillashtirilib boriladigan, dasturlovchi uchun qulay imkoniyatlarga ega brauzer

hisoblanadi. Albatta, hozirda **Google Chrome** ham undagi deyarli barcha imkoniyatlarni amalga oshira oladi, shuning uchun ham ko'pchilik eng ommaviy brauzer sifatida **Chrome**ni tanlashi mumkin. Tanlovni o'rganuvchining o'ziga qoldiramiz va har ikkisida bir xil bo'lgan dasturlovchilar uchun mo'ljallangan muhit haqida qisqacha to'xtalib o'tamiz.

Birinchi qadamda «Mozilla Firefox» yoki «Google Chrome» brauzerlaridan birini operatsion tizimga o'rnatamiz. So'ngra «F12» tugmasini (Ctrl+Alt+I yoki operatsion tizim va brauzer talqiniga qarab boshqa tugmalar birikmasi bo'lishi mumkin) bosish orqali maxsus muhitni chaqiramiz. Unda dasturlashni o'rganmoqchi bo'lganlar uchun murakkablik kasb etmaydigan, sodda interfeys (dasturiy ta'minot orazi) mavjud. Bizga birinchi o'rinda kerakli bo'ladigani — «konsol» bo'limidir.

Konsol — alohida (veb-sahifa namoyon bo'ladigan maydondan tashqarida, odatda, yashirin tarzda bo'ladigan) bajarilayotgan amallardagi xatoliklarni kuzatib borish yoki qo'shimcha JavaScript kodini ishga tushirish uchun muloqot darchasi. Buni to'liq tasavvur qilish maqsadida quyida dasturlovchi uchun mo'ljallangan muhitning ko'rinishini keltiramiz:



JavaScript kod yozish satri

Ushbu muhit orqali o'rganish jarayonida yozilgan kodlarning natijasini tez ko'rishdan tashqari, ularning

bajarilishidagi o'zgarishlarni qadam-baqadam kuzatib borish imkoniyati ham mavjud. Konsol imkoniyatlari va undan unumli foydalanish bo'yicha keyingi bo'limlarda [3.4.3.] amaliy mashg'ulotlar yordamida batafsil tanishtirib o'tamiz.

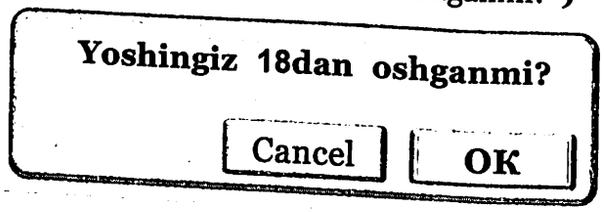
Kiritilgan qiymatlarni qanday o'zlashtirilib, ularning ustida amallar bajarish keyingi mavzularda batafsil, misollar bilan keng yoritilgan.

2.1.3. Tasdiqlash (confirm)

Tasdiqlash buyrug'i aynan ogohlantirish singari ishlaydi. Bunda foydalanuvchiga nafaqat xabar beriladi, balki undan oynada namoyon etilgan jumlaga munosabati so'raladi. Agar fikrga qo'shilsa tasdiqlash, aks holda bekor etish tugmasi tanlanadi. Dastur kodi bosilgan tugmaga qarab «rost» yoki «yolg'on» ma'nolaridan birini bildiruvchi mantiqiy qiymatni qabul qiladi.

Misol: **confirm**("Yoshingiz 18dan oshganmi?")

Natija:



2.1.4. Konsol yozuvi

Yuqorida keltirilgan uchta usuldan tashqari foydalanuvchiga ko'rinishi uchun konsol [1.2.] orqali ham ma'lumot chiqarish mumkin. Ammo oddiy foydalanuvchilar, odatda, bunday yozuvlarga ahamiyat bermaydi. Aslida konsoldagi chiqarilgan yozuv orqali muloqot

ham o'rnatilmaydi. Bu asosan, dasturlovchilar uchundir, chunki yozilgan kodlarning to'g'ri ishlashini tekshirish jarayonida bunday yozuvlar keng qo'llaniladi. Unda qiymatlarni batafsil yoyib ko'rish imkoniyati ham mavjud.

Konsolga qiymatlarni chiqarish «**console.log**» va «**console.dir**» buyruqlari yordamida amalga oshiriladi. Ular bilan «console» obyektiga bag'ishlangan mavzuda [3.4.3.] batafsil tanishib o'tamiz.

2.2. Javascriptni HTMLga bog'lash

Ushbu kitob JavaScriptga bag'ishlangani bois, avvalo uning kodlari qanday tatbiq etilishi bilan tanishib o'tish lozim. Keyingi mavzularda dasturlash bo'yicha umumiy tushunchalar JavaScript misolida yoritib boriladi. Keltirilayotgan nazariya va amaliy misollarning to'laqonli namoyon bo'lishi uchun yuqorida ko'rib o'tilgan — dasturlovchiga mo'ljallangan muhit yetarli emas [1.2.].

HTML — joylashtirish tili, JavaScript — esa dasturlash tili. Bir-biriga chambarchas bog'liq ushbu tillarning qo'llanilish chegaralarini aniqlab olish zarur.

JavaScript ham HTMLga xuddi CSS singari bog'lanadi, ya'ni maxsus teg ichida dasturlash kodini yozish orqali yoki alohida fayl sifatida e'lon qilingan kodni biriktirish ham mumkin.

Teg ichida kod yozish uchun maxsus **SCRIPT** tegi ishlatiladi. Uning quyidagi «alomatlar»i mavjud:

▪ **src** — dasturlash kodlari yozilgan fayl manzili ko'rsatiladi. Aslida alohida faylda emas, ushbu tegning ichida kod yozish ham mumkin, lekin HTML va JavaScript kodlar chalkashib ketmasligi uchun bunday yozish tavsiya etilmaydi. Doim maxsus faylni shu alomat orqali ko'rsatish ma'qul hisoblanadi.

▪ **async** — birgalikda bajarishni ta'minlaydi. Odatda, HTML sahifa yuklanib bo'lgandan keyingina unda yozilgan JavaScript kodlar (mavjud bo'lsa) ishga tushadi. Zarurmas qiymatga ega ushbu alomat esa, to'liq yuklanishni kutmay, o'qilgan kodlarni darhol amalga oshirilishi lozimligini ko'rsatadi. Bunda dasturlash kodi bajarilishi bilan birga sahifaning qurilishi ham davom etadi. Odatda, ushbu alomat o'chirilgan bo'ladi.

▪ **defer** — HTML sahifa to'liq yuklanib bo'lgandan keyingina «scr»da ko'rsatilgan fayldagi kodlar ishlashni boshlaydi. Bu juda nozik alomat hisoblanadi. Yetarli tajribaga ega bo'lmagan dasturlovchilar ushbu alomat kodni sahifa yuklanib bo'lgandan keyin bajarilishini ta'minlaydi deb o'ylab xatoga yo'l qo'yishadi. Agar kod alohida faylda yozilib, «scr»da ko'rsatilishi o'rniga, tegning ichida keltirilgan bo'lsa, albatta kutilgan natijani bermaydi. Shuni inobatga olish lozimki, HTML sahifa hali to'la yuklanib bo'lmasdan «scr»ni ko'rishi bilan undagi faylni ham biryo'la ko'chirib olishni boshlaydi. Agar ko'rsatilgan JavaScript fayl oldin yuklanib olsa, undagi kodlar darhol ishga tushadi va HTML sahifada keltirilgan elementlar ustida amal bajarish lozim bo'lsa,

hali yuklanib bo'lmagani uchun «topilmadi» degan xato ma'lumot chiqaradi. Dastur tuzish mobaynidagi sinovda JavaScript fayl doim (ehtimol hajmi kattaligi uchun) keyin yuklanadigan bo'lsa, katta xatolar o'tkazib yuborilishi mumkin. Haqiqiy foydalanuvchi ishlayotgan paytda JavaScript fayl «kesh»ga saqlanganligi uchun, tez yuklanib, keyinchalik xato oshkor bo'lishi tayin. Odatda, ushbu alomat o'chirilgan bo'ladi, biz uni doim qo'llashni tavsiya etamiz. Chunki dastavval HTML sahifa yuklanib o'z elementlarini namoyish etadi, so'ngra ular ustida dasturiy amallar bajarilishi maqsadga muvofiqdir.

▪ **language** — dasturlash kodlari qaysi tilda yozilganligini ko'rsatadi, u quyidagi qiymatlardan birini qabul qiladi:

◊ **JavaScript**: Odatiy qiymat, brauzer uchun aniqlik kiritish maqsadida talqin raqamini ham qo'shib qo'yish mumkin. Aslida brauzer (o'zining talqiniga mos) JavaScriptning eng oxirgi talqini bo'yicha kodlarning bajarilishini ta'minlaydi.

◊ **JScript**: «Microsoft» kompaniyasi tomonidan taqdim etilgan boshqacharoq yondashuvdagi dasturlash tili.

◊ **VBS** yoki **VBScript**: Microsoft kompaniyasining «Visual Basic» asosidagi Internet Explorer uchun chiqargan dasturlash tili.

▪ **type** — bajariladigan faylning tavsif-turi. HTML5 tilida uni keltirish shart emas, avvalgi talqinlarida esa «text/javascript», «text/vbscript» singari ko'rsatilar edi.

Quyida JavaScript kodlari alohida faylda yozilganda uni HTMLda qanday yuklanishiga misol keltiramiz:

```
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset="utf-8">
        <title> «JavaScript»ni õrganamiz. </title>
        <script defer src="value.js"></script>
    </head>
    <body>
        . . . . .
    </body>
</html>
```

Ushbu mavzuni yoritish davomida JavaScript va HTML kodlari qanday tarzda bir-biridan ajratilishini aniq belgilab oldik. Albatta, ularni imkon qadar boshqa-boshqa faylda alohida yozish mukammal dasturlash talablariga mos keladi. Shu bilan birga, bu tarzda yozish amaliyotda ko‘pchilik birgalikda bitta loyiha ustida ishlanayotgan vaziyatlarda vazifalarni aniq bo‘lishib olish jarayonida ham juda qo‘l keladi. Hozircha HTML teglarining ichida JavaScript kodi qanday o‘rin olishi mumkinligiga sodda misol keltiramiz:

```
<script> alert("Sahifa yuklanmoqda"); </script>
```

Agarda HTML va JavaScript kodlari alohida-alohida faylda yozish talabiga bo‘ysunmaslikka to‘g‘ri kelsa, bitta faylda umumlashtirilganda, «**script**» tegini «HTML» kodlarining ixtiyoriy qismiga joylashtirish mumkin, ammo uni faqat «head» tegi ichida keltirish tavsiya etiladi.

Mabodo, kod sahifa yuklangandan keyin ishlashi lozim bo‘lsa, faqat «body» tegining eng quyi qismida, undagi boshqa ichki teglarning tarkibiga kiritmasdan alohida yozish tavsiya etiladi. Aks holda uni ba’zi brauzerlar bajariluvchi kod emas, oddiy element yasovchi teg sifatida e’tiborsiz qoldirishi mumkin:

```
<body>
    . . . . .
    <div>
        <script>  <!-- xato joylashtirilgan -->
            confirm("Dasturlashni o‘rganasizmi?");
        </script>
    </div>
        . . . . .
    <script>      <!-- to‘g‘ri joylashtirilgan -->
        alert("Sahifa yuklanishi yakunlandi.");
    </script>
</body>
```

Yuqorida ta’kidlab o‘tganimizdek, imkon qadar JavaScript kodlarini alohida faylda yozib keltiramiz. Unda faqat «**script**» tegi ichidagi kodlar yoziladi, xolos. Ushbu misol bo‘yicha qaraydigan bo‘lsak, «**confirm**» yoki «**alert**» buyruqlarigina keltiriladi. JavaScript kodi bitiladigan fayl har qanday ortiqcha yozuvdan xoli bo‘ladi.

Birinchi kitobimizda[1] JavaScript kodlari HTML elementlarining hodisani bildiruvchi alomatlariga qiymat sifatida berilishi mumkinligini ham alohida ta’kidlab

[1] «Mukammal dasturlash. 1-kitob: HTML va CSS», Javlon Abdullo. «*Musiqa*» nashriyoti, T., 2017.

o‘tgan edik. Ushbu kitobda bunday vaziyatlarda qanday yo‘l tutilishi lozimligiga batafsil to‘xtalib o‘tamiz. Uzundan-uzoq dasturlash kodlarini satr ko‘rinishida alomatga qiymat sifatida berish aslida xato hisoblanmasa-da, bunday yo‘l tutish aslo tavsiya etilmaydi.

2.3. Dasturlash tili tushunchalari

Dastur tuzishni boshlashdan avval, uni tuzishda qo‘llaniladigan muhim tushunchalar bilan tanishib o‘tish lozim. Ta’kidlash joizki, bu tushunchalar dasturlash tilining asoslari sifatida xizmat qiladi.

2.3.1. Obyekt

HTMLni o‘rganayotganimizda obyekt deganda, teglarning brauzerda namoyon bo‘ladigan shakllarini nazarda tutgan edik. Aslida dasturlash tilida obyekt butunlay boshqacha tushuncha sanaladi. Har qanday dasturlash tillari insonlar tomonidan o‘ylab topilgan va albatta ular hayotdagi mavjud narsalarning soddalashtirilgan ko‘rinishi hisoblanadi. Misol tariqasida bitta ko‘p qavatli binoni olaylik, aytaylik, u tibbiyot muassasasi bo‘lsin. Binoga xodimlar ishlash uchun, bemorlar davolanish uchun kirishadi. O‘z o‘rnida, barcha tashrif buyuruvchilar qandaydir muolajada ishtirok etib, kirgan

holatdagiga nisbatan qandaydir o‘zgarish bilan chiqishadi. Bino ichida o‘z ko‘rinishini o‘zgartirmaydigan uskunalar ham bor, ma’lum vazifani bajarish uchun ishlatiladigan dori-darmonlar ham mavjud. Qaysidir xonalarda muolaja amaliyotlari bajarilib, natijada bemorlarning organizmida o‘zgarishlar sodir bo‘ladi.

Qiyosan, shu tibbiyot muassasasini dasturlash tilida **obyekt** deya talqin etsa bo‘ladi. Undagi qurilmalarni — **o‘zgarmas**, dori-darmonlarni — **o‘zgaruvchi**, muolaja amaliyotlarini — **funksiya**, tashrif buyuruvchilarni — **kiruvchilar**, binoni tark etayotganlarni — **chiquvchilar** sifatida qarash mumkin. Demak, unga quyidagicha ta’rif bersa bo‘ladi.

Obyekt — o‘zida ma’lum qiymatlarni saqlaydigan, berilganlarni qayta ishlab natija qaytara oladigan, ma’lum tushunchani anglatadigan majmuadir.

Yuqoridagi misolimizda obyekt tibbiyot sohasiga qaratilgan edi. Umuman olganda, obyektni boshqa yo‘nalishlarda ham shu tarzda talqin etish yoki sohaning bitta bo‘lagi sifatida ham (masalan, stomatologiya) qarash mumkin.

Dasturlash tilida obyektni saqlash va unga murojaat qilish uchun obyektga nom beriladi. Bunday nom esa **identifikator** (*bu haqda 1-kitobda yoritilgan — J.A.*) bo‘lishi talab etiladi. Obyekt gulli qavs bilan ifodalanadi [2.4.6.], uning ichida qiymatlar boshqa identifikatorlar orqali beriladi. Faqat obyekt ichida qiymat berilganda tenglik (=) belgisi o‘rniga ikki nuqta (:) ishlatiladi:

```
õzgaruvchi = { nom: "qiymat" }
```

Obyektning ichki qiymatiga murojaat qilishning ikki xil usuli mavjud. Birinchisi — nuqta, ikkinchisi — to'g'ri burchakli qavs. Ushbu misoldagi qiymatni quyidagi yo'llar bilan ko'rish mumkin:

```
alert(õzgaruvchi.nom)

console.log(õzgaruvchi["nom"])
```

2.3.2. Kod yozish qoidalari

Dasturlashda kodlar chapdan o'ngga hamda yuqoridan pastga qarab yoziladi va aynan shu tartibda bajariladi. Ketma-ketlik juda muhim, oddiy amallar bajarilayotganda bunga ahamiyat qaratilmasligi mumkin, ammo aynan JavaScriptda «**berkilish**» degan tushuncha mavjudki, aynan bitta kod ba'zi hollarda boshqa-boshqa natija qaytarishi mumkin. Albatta, bu dastur kodlari bajarilishi jarayonida qiymatlarining o'zgarishiga hamda qiymat aynan qachon qabul qilinishiga bog'liq. Kelgusi mavzularda [3.4.4.] ushbu tushunchani murakkabroq misollar bilan batafsil yoritiladi va uni bartaraf etish usullari keltiriladi.

Amallar bir-biridan nuqta-vergul bilan ajratiladi. Agar ular ketma-ket uchrasa, orasida bo'sh amal bor deb hisoblanadi va albatta u hech nima bajarilishini ta'minlamaydi. JavaScriptga xos bo'lgan kod yozishning ikkinchi qoidasi shundan iboratki, qatorning oxirida nuqta-vergul (;) qo'yish shart emas. Aslida, har bir

amal boshqasi bilan nuqta-vergul yordamida ajratiladi. Agar ikkita amal boshqa-boshqa qatorda yozilgan bo'lsa, birinchi qator oxirida nuqtali vergul (;) qo'yilmasa ham kod to'g'ri bajarilaveradi. Ammo bunday kod yozish maqsadga muvofiq emas. JavaScript ochiq kod bo'lganligi bois uni istagan foydalanuvchi brauzer yordamida ko'rishi mumkin. Tajribali dasturlovchi yoki «xaker» kod qanday yozilganligiga qarab, uning «nozik joyini» topa oladi, bu esa tajovuzkor hujumlarni qanday uyushtirishda ko'rsatma sifatida xizmat qiladi. Yoki oddiy havaskor dasturchi ham o'rganish maqsadida kodni ochib, sodda qilib aytganda, «o'g'irlik» qilishi mumkin. Bunday holatlarning oldini olish maqsadida maxsus dasturiy vositalar yordamida JavaScript kodini «siqib», «chalkashtirib» so'ngra veb-saytlarda qo'llash an'anaga aylangan. Agar qator oxiriga nuqta-vergul qo'yilmasa, bunday «siqilgan» kodlar brauzer to'g'ri tushunmaydigan darajada «chalkashib» qolishi tabiiy.

Uchinchi qoida ham faqat JavaScriptga xos bo'lib, u «UTF-8» kodlamasini qo'llaydi va identifikatorlarni nomlashda ham lotin yozuvidagilardan tashqari boshqa harflardan ham foydalanish mumkin [2.3.1.]. Ammo bunday yozish maqbul emas, boshqa deyarli barcha dasturlash tillarida o'zgaruvchilarga nom berishda faqat lotin harflari ishlatiladi. Mabodo, JavaScriptda yozilgan kodni boshqa tilga o'girish zarurati paydo bo'lsa, bunda jiddiy muammo keltirib chiqarishi mumkin.

To'rtinchi qoida, katta-kichik harflar farqlanadi. Aksariyat dasturlash tillarida (Paskaldan tashqari) shun-

day qoida mavjud, ammo JavaScript asosan HTML bilan birga qo'llangani bois, ba'zan chalkashlik yuzaga kelishi mumkin. HTMLda «BR», «br», «Br», «bR» bir xil tushuniladi, JavaScriptda esa bular alohida-alohida to'rt xil identifikator hisoblanadi.

Beshinchi — asosiy qoida shuki, JavaScriptda har bir narsa obyekt sifatida namoyon bo'ladi [2.3.1.]. Hatto oddiy sonlarni ham obyekt sifatida qarab, ular ustida amal bajarish mumkin. Arifmetikadagi sonlarning o'nli yozuvi, dasturlash talabiga binoan nuqta bilan yoziladi. Biz bir butun o'ndan ikkini «1,2» ko'rinishida yozib o'rgangan bo'lsak, kod sifatida u «1.2» tarzida yoziladi. Ushbu o'nli yozuvli son «bir» degan obyektning «ikki» degan elementiga murojaati ekanligini anglatmaydi. Shuning uchun ham identifikatorlarni nomlashda birinchi belgisi raqam bo'lmasligi shart deb talab qo'yilgan. «1.2» – o'nli yozuvdagi alohida haqiqiy son. Lekin unga ham obyekt sifatida murojaat qilish mumkin. Masalan:

```
1.2.toString();
```

Bu sonni satrga aylantirib beradi. Natural sonni satrga aylantirish lozim bo'lsa, «`12.toString();`» kabi yozish xatolik hisoblanadi. Demak, son obyekt sifatida qaralmas ekan-da, degan xulosa qilish to'g'ri emas.

JavaScriptda butun son ham, haqiqiy son ham yagona tur sifatida qaraladi. Shuni inobatga olgan holda kasr qismi nolga teng haqiqiy sonlar yoki natural sonlar uchun «`12..toString();`» yoki bo'sh joy (probel) bilan «`12 .toString();`» tarzda yozish talab etiladi.

2.3.3. Had va ajratuvchilar

Dasturlash tilida yakka o'zi alohida ma'noga ega bo'lgan yozuv «**had**» (xuddi matematikada o'rganganimiz singari — *J.A.)* deb yuritiladi. Har bir sonni yoki identifikatorni bitta had sifatida qarash mumkin. Masalan, yuqorida ko'rib o'tilgan misolda keltirilgan «`1.2.toString();`» yozuvida bir butun o'ndan ikki soni alohida had, «`toString`» buyrug'i esa alohida had hisoblanadi. Hadlar maxsus ajratuvchilar yordamida bir-biridan farqlanadi. Ushbu yozuvda ajratuvchi sifatida ikkinchi nuqta xizmat qilyapti. Birinchi nuqta esa, ajratuvchi emas, sonning o'nli yozuvini anglatyapti. Agar «1.2»dagi «bir»ni alohida, «ikki»ni alohida qarasak, bir butun o'ndan ikki ma'nosini bermaydi. Shuning uchun haqiqiy son «1.2» yagona had hisoblanadi.

Ajratuvchi sifatida esa nuqta, bo'sh joylar, matematik ishora(amal)lar va (turli) qavslar xizmat qiladi. Bo'sh joylar bir necha xil ko'rinishda bo'lib, enli-ensiz va hatto ko'rinmas bo'shliq sifatida namoyon bo'lishi mumkin. Tabulyatsiya va qatorlarni bo'luvchilar ham bo'sh joy sifatida qaraladi. O'quvchiga ko'rinmas yoki bir xilday tuyuluvchi, quyida kodlari keltirilgan ajratuvchilar o'rniga oddiy bo'sh joy ishlatish tavsiya etiladi, chunki ularning vazifasi bir xildir:

`\u0020` — oddiy bo'sh joy (probel)
`\u0009` — tabulyatsiya
`\u000B` — tik tabulyatsiya
`\u000C` — (boshqa shaklga) o'tkazuvchi
`\u00A0` — ajralmasligini ta'minlovchi

`\u200F` — o'ngdan ulovchi
`\u200E` — chapdan ulovchi
`\uFEFF` — tartibni o'zgartiruvchi

JavaScript tili UTF-8 kodlamasini qo'llagani bois aslida bo'sh joy hisoblanuvchi ayrim belgilar so'zlarni jipslashtirgan holatda ko'rinishini ta'minlagani uchun oddiy harfday qo'llaniladi. Ushbu ikki harf (**`\u200D`** va **`\u200C`**) ilk belgi sifatida ishlatilmasa, identifikator nomlanishida ham foydalanish mumkin. Bu JavaScriptning ko'pchilikka notanish bo'lgan nozik jihati bo'lib, uni qo'llash tavsiya etilmaydi.

Qo'shtirnoqlar ajratuvchi sifatida qaralmaydi. Qo'shtirnoq ichidagi bir necha so'zdan iborat satr ham yagona had sifatida qaraladi. Umuman olganda, qo'shtirnoq va qavslarning o'ziga xos mohiyati mavjud bo'lib, ular bilan kelgusi mavzularda batafsil tanishtirib boramiz [2.4.1.].

Ajratuvchi sifatida nuqta, arifmetik ishora, bo'sh joy va qatorlarni bo'luvchi belgidan tashqari asosan nuqtali vergul ishlatiladi. U buyruq yoki amalning yakunlanganini anglatadi. Yaxlit amal yoki arifmetik ifoda ichida bir necha had qatnashishi mumkin, lekin aslo nuqtali vergul ishtirok etmaydi. Masalan:

```
b * b - 4 * a * c
```

Bunda «`b`», «`4`», «`a`», «`c`»larning har biri alohida had. (Matematikada esa **`4ac`** bitta had hisoblanadi.) Ko'paytiruv va ayiruv ishoralari ajratuvchi bo'lish bilan birga matematik amalni ham ifodalayapti. Hadlar va amallar orasiga istalgancha bo'sh joy qo'yish mumkin,

bu faqat yozilayotgan kod chiroyiga ta'sir qiladi, xolos, bajarilishda esa uning ahamiyati yo'q.

Agar kodning qaysidir qismi bajarilishi lozim bo'lmasa yoki dasturlovchi o'zi uchun eslatma qoldirishni lozim topsa, bunday yozuvlar izoh sifatida ifodalanadi.

Bir qator izohni ifodalash uchun ikkita ketma-ket taqsim belgisi, ya'ni «`//`» dan keyin yoziladi. Qatordagi ushbu belgilardan oldin yozilgan kodlar bajariladi, undan keyingilari e'tiborsiz qoldiriladi. Izoh bir necha qatordan iborat bo'lsa, ularni, CSSda ta'riflanganday, boshlang'ich joyiga «`/*`» va yakuniy qismiga, aksincha, «`*/`» birikmalarini qo'yish kerak bo'ladi:

```
// Bir qatorli izoh, eslatma
/* Bir necha qatorli izoh ...
Dasturning shu qismi o'chib ketmasin ...
lekin bajarilmay ham tursin */
```

2.3.4. O'zgaruvchi va o'zgarmaslar

Dasturlashda qiymat yoki xotiradagi manzilni saqlash uchun maxsus nomlanishdan foydalaniladi. Ushbu nomni **o'zgaruvchi** deb yuritiladi, u identifikator talablariga javob berishi kerak:

```
qiymat = 12;
alert(qiymat);
```

Bunday yozish hech qanday xatolik keltirib chiqarmaydi, ammo u mutlaqo to'g'ri ham hisoblanmaydi. O'zgaruvchilarga qiymat berishdan avval ularni e'lon qi-

lish lozim. E'lon qilinayotganda esa u qaysi maydonda faoliyat yuritishini oldindan aniq belgilash joiz bo'ladi. Shu o'rinda «**maydon**» tushunchasi qanday aniqlanishi xususida ham batafsil to'xtalib o'tsak.

Dastur kodlari yozilgan umumiy veb-sahifa global maydon sifatida qaraladi. Uning ichida qo'llanilgan maxsus buyruq yoki amallarning o'z kichik maydoni (*keyingi o'rinlarda uni «**bo'lak**» deb ham yuritamiz*) mavjud. Agar o'zgaruvchi o'sha kichik maydonda e'lon qilinsa, faqat o'sha bo'lakdagina faoliyat ko'rsatadi, tashqi maydonda qo'llanilgan ayni shu nomdagi o'zgaruvchi o'zining boshqa qiymati bilan ishtirok etadi va nomi bir xil bo'lsa ham ular boshqa-boshqa o'zgaruvchilar hisoblanadi. Agar global maydonda o'zgaruvchi e'lon qilinsa, u har bir ichki kichik maydonchalarda ham o'z qiymati bilan ishtirok etish imkoniyatiga ega bo'ladi.

O'zgaruvchilarni e'lon qilish uchun «**var**» kalit so'zidan foydalaniladi. Umuman olganda, JavaScriptda shu kabi zaxiraga olingan, lekin unchalik ko'p bo'lmagan kalit so'zlar mavjud bo'lib, ular ma'lum maqsadlarda, nimanidir anglatish yoki qandaydir buyruqni bajarish uchun ishlatiladi. Ularning nomlaridan identifikator sifatida foydalanish mumkin emas.

Ko'pchilik identifikatorlarni nomlash uchun inglizcha so'zlardan foydalaniladi. Bu ingliz tilini biroz biladigan boshqa dasturlovchilar uchun kodni o'qishda tushunarli bo'lishini ta'minlaydi. Ammo nom kalit so'z bilan bir xil bo'lib qolmasligiga e'tibor berish lozim. Lotin yozuvida o'zbekcha so'zlar ishlatilsa, bunday muammo

deyarli bo'lmaydi. Ammo hamma ham ushbu fikrga qo'shilavermasligi mumkin. Chunonchi, o'zbek tilini bilmaydigan boshqa millat vakillari bilan ba'zida bitta loyiha ustida ishlashga to'g'ri keladi. Shuni nazarda tutib, dasturlovchilarimiz uchun zaxiradagi kalit so'zlar ro'yxatini keltiramiz. Identifikatorlarning ushbu so'zlardan farq qilishi kifoya. Yana e'tibordan chetda qoldirmaslik kerakki, *ushbu ro'yxat JavaScriptning keyingi talqinlarida kengayib borishi mumkin*:

abstract	false	private
arguments	final	protected
Array	finally	public
boolean	float	RangeError
Boolean	for	ReferenceError
break	function	RegExp
byte	Function	return
case	goto	short
catch	if	static
char	implements	String
class	import	super
clear	in	switch
const	Infinity	synchronized
continue	instanceof	SyntaxError
Date	int	this
debugger	interface	throw
decodeURI	isFinite	throws
encodeURI	isNaN	transient
decodeURIComponent	JSON	true
encodeURIComponent	let	try
default	long	TypeError
delete	Math	typeof
do	NaN	undefined

double	native	URIError
else	new	var
enum	null	void
Error	Number	volatile
eval	Object	while
EvalError	package	with
export	parseFloat	yield
extends	parseInt	

O'zgaruvchi qaysi maydonda «**var**» bilan e'lon qilingan bo'lsa, unda va uning ichki maydonlarida faoliyat ko'rsatadi. Ba'zi buyruqlar o'zining ichki maydonchasiga ham ega. Masalan, bitta vazifani ketma-ket yangi qiymatlar asosida takrorlaydigan buyruq bunga misol bo'la oladi. Unda qo'llangan o'zgaruvchi ichki qismda «var» bilan e'lon qilingan bo'lsa ham nafaqat ichki maydonchaga, balki buyruq joylashgan kichik bo'lakka ham taalluqli bo'ladi. Bu bir tarafdan har bir maydonchaning boshida o'zgaruvchilarni e'lon qilish shart degan noqulay qonuniyatga bo'ysunishdan forig' qilib, ishimizni osonlashtirsa-da, ikkinchi tarafdan aynan buyruq ichidagina ishlashi shart bo'lgan o'zgaruvchi lozim bo'lgan holatlarda chalkashlik keltirib chiqaradi. Buning oldini olish maqsadida «var»ning o'rniga «**let**» kalit so'zi qo'llaniladi. «**let**» kalit so'zi faqat aynan e'lon qilingan eng ichki «bo'lakcha»dagina faoliyat qilishini anglatadi. Ya'ni, ichki bo'lakda e'lon qilingan «var» maydondagi o'zgaruvchini yangilaydi, «**let**» esa tashqi maydondagini o'zgartirmaydi.

Quyida «var» bilan «let»ning farqini yaqqol ko'rsatuvchi misol keltiramiz:


```
var x = 12;
console.log(x);    // 12
{
   var x = 2;
   console.log(x); // 2
}
console.log(x);    // 2
```

```
var x = 12;
console.log(x);      // 12
{
   let x = 2;
   console.log(x);   // 2
}
console.log(x);      // 12
```

O'zgaruvchining qiymati dastur kodi bajarilishi davomida almashib boradi. Lekin shunday qiymatlar ham zarur bo'ladiki, ular o'zgartirilmasligi talab etiladi. Aytaylik, orqatasvir rangi. Bitta veb-sayt uchun u o'zgarmas, yagona bo'lishi shart. Ushbu koddan boshqa sayt uchun foydalanilganda esa boshqa qiymatga almashtirilishi mumkin. Dasturlovchi beixtiyor «o'zgarmas» hisoblangan qiymatni o'zgartirib yuborishga yo'l qo'ymasligi lozim. Bu qat'iy talab bo'lishi kerak. Bunday keskin chorani qo'llash uchun «**const**» kalit so'zidan foydalaniladi. Ayni bir o'zgarmasning nomidan o'zgaruvchi sifatida foydalanish yoki dastur davomida qayta e'lon qilish mumkin emas:

```
const color = 'yellow'; // o'zgarmasni e'lon qilish
var color = 'green'; // qiymatini o'zgartirish xato.
const color = 'red'; // qayta aniqlash mumkin emas.
```

Ammo o'zgarmas sifatida obyekt [2.3.1.] [2.4.6.] aniqlansa, uning ichki elementlarini o'zgartirsa bo'ladi.

2.4. Turlar

Yuqorida ta'kidlab o'tganimizday JavaScriptda barcha o'zgaruvchilar aslida obyekt sifatida ifodalanadi. Lekin dasturlash tilini o'rganishda o'zgaruvchilarni ma'lum turlarga ajratib, ularga munosib ravishda amallar bajarish maqsadga muvofiq. Shuning uchun biz mantiqiy nuqtayi nazardan bir necha tur kiritib, ular bilan tanishamiz. Kelgusi mavzularda o'zgarmas va o'zgaruvchilarga qiymat berishda hamda ular ustida amallar bajarishda ushbu tushunchalar juda qo'l keladi.

2.4.1. Satr

Satr — bir qator matndan iborat qiymat bo'lib, u «"» yoki «'» belgilarining orasida yoziladi. Har ikki qo'shtirnoq bir xil mavqega ega, satr boshida qay biri qo'yilsa, oxirida ham shu ishlatilishi kerak. Birining ichida ikkinchisini qo'llash mumkin, faqat bunda ichkaridagi belgi satr tugaganini anglatmaydi. Agar satr ichida u chegaralab turgan ayni qo'shtirnoq ishlatilishi shart bo'lsa, «\» belgisi bilan birga ishlatiladi. Misol:

```
const nom = "Mukammal dasturlash";
var satr1 = "\"2\"-kitob";
let satr2 = 'JavaScript';
```

Satr «**string**» so'zi bilan belgilanadi va unda ixtiyoriy uzunlikdagi yagona qatordan iborat matn beriladi. Bo'sh satr esa qo'shtirnoqlarni ketma-ket orasida hech

qanday belgi qo'ymasdan juftlashtirish «''» yoki «""» sifatida yozish bilan ifodalanadi. Agar bir necha qatorli matnni qabul qilish lozim bo'lsa, keyingi xatboshini anglatuvchi «\n» belgidan foydalanib, istalgancha qator hosil qilish mumkin.

To'g'ri bo'lish:
```
var nom = "Mukammal\n dasturlash";
```
Xato bo'lish:
```
var nom = "Mukammal
 dasturlash";
```

Yuqoridagi ikki xil qo'shtirnoq singari «`» dan ham foydalanish mumkin. Uning o'ziga xos imkoniyati mavjud bo'lib, satr ichida o'zgaruvchi «${...}» yordamida ifoda etilsa, uning qiymati inobatga olinadi:

```
var yosh = 40;
alert(`Yoshim ${yosh}da.`);
```
Yoshim 40da.

JavaScriptda yagona belgini anglatuvchi **char** turi mavjud emas, boshqa tillarda mavjud ushbu tur ham satr orqali ifodalanadi.

2.4.2. Son

Biz matematika fanidan «son» tushunchasi bilan yaxshi tanishmiz. Dasturlash tilida esa ulardan butun hamda haqiqiy songa taalluqli bo'lganlarinigina qo'llaymiz. JavaScriptda ularni umumiy qilib «**number**» deb yuritiladi va o'z navbatida, quyidagi ikki xil sonlarni o'z ichiga oladi:

▪ **Butun son** — oddiy matematikada qo'llanilgani singari, ya'ni natural (sanoqda ishlatiladigan) sonlar, ularga qarama-qarshi sonlar va nol. Shartli ravishda bunday sonlarni «**integer**» yoki qisqa qilib, «**int**» deb yuritiladi. Odatda, butun sonlar «$-(2^{53}-1)$»dan «$2^{53}-1$»-gacha oraliqda deb qaraladi, ammo aslida obyekt sifatida ifodalangani bois bu chegarani yanada kengaytirish (tavsiya etilmasa ham) mumkin.

▪ **Haqiqiy son** — matematikadagi haqiqiy sonlarning chekli o'nli kasr ko'rinishida ifodalash mumkin bo'lgan qismi. U **float** so'zi bilan belgilanadi. Albatta, kompyuter xotirasi ham, uning imkoniyati ham cheklangan, shuning uchun dasturlashda cheksiz degan tushunchaning o'zi yo'q. Shu bois har qanday son yoki ma'lumot ma'lum me'yorda, cheklangan holda ifodalanadi.

▪ **Cheksiz** — elementar matematikada sonlarni nolga bo'lish mumkin emas deb o'rgatiladi. Limitlar haqida tushuncha berilganda esa nolga intilgan (unga juda ham yaqin) songa bo'lish mumkin, bunda natija benihoyat katta son chiqadi deb tushuntiriladi. Boshqa dasturlash tillarida «nolga bo'lish xatoligi uchradi» deb bajarilish jarayoni to'xtab qoladi, JavaScriptda esa shunchaki «**Infinity**» qiymat qabul qildi deb qaraladi.

Misol:	`2 / 0`
Natija:	`Infinity`

▪ **Sonmas** — bu arifmetik amal bajarishda sodir bo'ladigan xatolik, «**NaN**» degan so'z bilan ifodalanadi. U

aniq bir qiymatni ifodalamaydi. Masalan, son emas, satr ustida bo'lish, ko'paytirish yoki ayirish amali bajarilganda yuzaga keladi. Satrga sonni qo'shilganda ular shunchaki matn sifatida birlashtirib qo'yiladi:

Misol:	`"Men" / 2`	Misol:	`"Sen" + 2`
Natija:	`NaN`	Natija:	`"Sen2"`

▪ **O'n darajali** ko'rinishda ham yoziladigan sonlarning maxsus shakli mavjud bo'lib, matematikadagi «$a * 10^n$» tarzidagi amalni ifodalaydi. Faqat undagi «*** 10**»ning o'rniga «**e**» yoki «**E**» harfi qo'llaniladi:

Misol:	`1.2e3`	Misol:	`8E-3`
Natija:	`1200`	Natija:	`0.008`

O'nning darajasidagi son manfiy bo'lsa, ayonki, u bo'lishni ifodalaydi.

▪ **O'n oltilik** sanoq tizimida ham sonlarni yozish mumkin. Bu ranglarning kodini yozishda keng qo'llaniladi, boshqa hollarda ham «**0x**» yoki «**0X**» birikmasi bilan boshlangan sonlarni 16 lik sanoq tizimida deb tushuniladi:

Misol:	`0xF0`	Misol:	`0X0000FF`
Natija:	`240`	Natija:	`255`

JavaScriptning ko'p talqinlari, ayniqsa, amalga oxirgi tatbiq qilinganlari sakkizlik sanoq tizimini qo'llamaganligi bois, unga to'xtalib o'tishni lozim topmadik.

2.4.3. Maxsus qiymatlar

JavaScriptda hech bir turga mansub bo'lmagan maxsus ikkita qiymat mavjud:

- **`null`** — «hech narsa» ma'nosini beruvchi yagona qiymat. Boshqa dasturlash tillaridagi singari, u mavjud bo'lmagan obyektga yoki o'lchami mavjud bo'lmagan maydonga ko'rsatkichni anglatmaydi. Ammo ma'no jihatdan qiymat mavjud emasligini bildiradi. Matematik amallarda qiymat **0** sifatida qabul qilinadi, faqat satrga qo'shilganda esa «**null**» so'zi olinadi. Albatta, u satr ham, son ham emas. Shuning uchun unga alohida yondashish kerak. Chunki satr va sonlar ustidagi amallarning barchasini uning ustida qo'llab bo'lmaydi.

Misol:	`null + 12`	Misol:	`null + '12'`
Natija:	`12`	Natija:	`"null12"`

- **`undefined`** — o'zgaruvchi e'lon qilingan, lekin hali biror narsaga tenglashtirilmagan holda «qiymat berilmagan» ma'nosini anglatuvchi kattalik.

```
var x;   alert( x );
                          undefined
```

Nima uchun bor-yo'g'i bittadan qiymatga ega ushbu kattaliklarni alohida tur sifatida qarash kerak?!

Sababi, JavaScriptda ham boshqa ayrim dasturlash tillari singari o'zgaruvchining turi unga qiymat berilganda aniqlanadi va kod bajarilish davomida nafaqat qiymati, balki, uning turi ham o'zgarib borishi mumkin. Kezi kelganda quyidagicha qiymatlar bersa ham bo'ladi:

```
a = null;
                          x = undefined;
```


2.4.4. Mantiqiy tur

Faqat «rost» yoki «yolg'on» qiymatlaridan birini qabul qila oladigan tur — mantiqiy tur hisoblanadi va «**`boolean`**» so'zi bilan ataladi. Uning qiymatlari «**`true`**» va «**`false`**»lardan iborat deb cheklash noto'g'ri. Chunki mantiqan mavjud emas degan ma'noni beradigan boshqa turdagi qiymatlar ham «false» sifatida namoyon bo'ladi, aksinchalari esa o'z navbatida, «true» o'rnida qo'llaniladi. Masalan:

Yolg'on:	**`false, undefined, null, 0, ''`**
Rost:	**`true,`** *noldan farqli sonlar*, *bo'shmas satrlar*

Bu yerda tushunmovchilik bo'lmasligi uchun oydinlik kiritib o'tish joiz. Boshqa turdagi qiymatlar mantiqiy tur sifatida qaralayotgani yo'q, faqat qo'llanishda mantiqiy tur o'rnida boshqa turdagi kattaliklar kelgan holda, qanday qiymatlar rost yoki yolg'onga ajratilishi ko'rsatilmoqda. Yana qo'shimcha shartlar yoki oddiygina matematik taqqoslash amallari yordamida ham mantiqiy qiymat hosil qilish mumkin.

Taqqoslash amalida tenglik belgisi (=) yakka ishlatilmaydi, u yakka uchragan holda qiymat berishni anglatadi. Ma'noviy tenglik amali bajarilayotganda ikkita ketma-ket, ular orasida hech qanday ajratuvchilarsiz qo'llaniladi. Ma'noviy tenglikda «2» soni bilan «`'2'`» satri bir xil qiymatga ega hisoblanadi.

Misol:	`2 * 2 > 5`	`'2' == 2`	`a = 2 * 2 == 4`
Natija:	`false`	`true`	`true`

JavaScript asosan boshqa tillar yordamida berilgan ma'lumotlar asosida ishlaydi. Unda sonlar ba'zan satr sifatida kelishi mumkin, mantiqiy turdagi ushbu satr va sonlarning aynan qiymati bilan taqqoslash imkoniyati bunday hollarda juda asqotadi. Ammo bu ko'pincha chalkashliklar keltirib chiqarishi tabiiy. Quyidagi mantiqiy qiymatlarning to'rtinchisi, uchinchiga qiyoslaganda aynan nima asosida kelib chiqqanini tushuntirish ham mushkul:

```
Misol:  true == 2 > 1   1 == 2 > 1   '1' == 2 > 1   '2' == 2 > 1
Natija: true            true         true           false
```

Shu kabi tushunmovchiliklar [2.5.] kelib chiqmasligi uchun «**aynan taqqoslash**» amali kiritilgan. U uchta tenglik belgisini ketma-ket qo'yish bilan ifodalanadi.

Maxsus qiymatlarni o'zaro taqqoslash mumkin, lekin aniq bir qiymatni ifodalamaydigan «sonmas» [2.4.2.] hatto o'ziga ham teng bo'la olmaydi:

```
null == null        undefined == undefined        NaN == NaN
true                true                          false
```

2.4.5. Funksiya

Avvalgi kitobimizda tanishganimizday, funksiya — ma'lum amallar bajaradigan dasturning bir bo'lagi. Dasturda aynan bir ma'nodagi hisob-kitoblar bir necha ma-rotaba qayta-qayta yozilmasligi uchun funksiyadan foydalaniladi. Uni bir marta e'lon qilib, istalgancha

murojaat qilish mumkin. Qandaydir qiymatlar asosida kodlar bajarilishi lozim bo'lsa, kiruvchi omillar berilishi mumkin. Chiquvchi natija va amalga oshiriladigan jarayon albatta unga mutanosib bo'ladi.

Funksiyani tur sifatida qarash o'zi bir tarafdan juda g'ayrioddiy, negaki, boshqa aksariyat dasturlash tillarida unga qiymat sifatida munosabat bildirilmaydi. Lekin aynan JavaScriptda alohida istisno tariqasida funksiyani o'ziga xos tur sifatida baholash mumkin. Sababi, uni o'zgaruvchiga to'g'ridan-to'g'ri qiymat sifatida bersa bo'ladi. Yana bir bor eslatib o'tamiz, JavaScriptda turlar nisbiy tushuncha bo'lib, aslida hamma narsa obyekt sifatida namoyon bo'ladi.

Funksiyani yaratish, ya'ni uni e'lon qilish uchun «**function**» so'zidan foydalaniladi, uning qisqacha sintaksisi quyidagicha:

```
function nom([omil [, omil]]) {
    ... turli amallar ...
    [ return qiymat; ]
}
```

Aslida funksiyada faqatgina keltirilgan amallarni bajarish bilan kifoyalanib, natija sifatida biror qiymat ko'rsatilishi shart bo'lmasa, «**return**» so'zi qo'llanmasligi ham mumkin. Agar qo'llanadigan bo'lsa, undan keyin o'zi joylashgan maydondagi yozuvlar bajarilmasdan, e'tiborsiz qoldiriladi. Shunday ekan, «return»dan keyin o'sha maydonga kod yozishning foydasi yo'q. Misol tariqasida berilgan qiymatni ikkiga ko'paytirib qaytaradigan kichik funksiya yaratamiz:

```
function x2(a) { var b = a * 2;  return b; }
```

Misol: `x2( 12 );` Natija: `24`

Yuqorida biz maydonlarga alohida urg'u berib o'tdik, endi ularni qanday aniqlanilishi bilan ushbu misol yordamida batafsil tanishamiz. Bunda funksiyaning tanasi gulli qavs ichiga olingan maydon hisoblanadi va u yerda e'lon qilingan o'zgaruvchilardan boshqa joyda foydalanish mumkin emas. Misol:

```
var c = x2( 12 );
alert(b);
```

```
24
b is not defined
```

Funksiya ichki maydonida e'lon qilingan o'zgaruvchi «**b**» tashqarida qo'llangani uchun xatolik kelib chiqyapti. Tashqarida ham «**b**» e'lon qilinsa, shuni unutmangki, uning funksiya ichida e'lon qilingan adashiga hech qanday aloqasi yo'q, tashqaridagi adash «**b**» boshqa qiymatga ega bo'lgan boshqa o'zgaruvchi hisoblanadi.

Yana bir holat. Aytaylik, funksiyadan tashqarida o'zgaruvchi e'lon qilindi, albatta u global hisoblanadi va o'zi yaratilgan maydonda hamda shu maydondagi barcha ichki maydonchalarda qo'llanilishiga imkon berilgan. Funksiya ichida shu nomdagi yangi o'zgaruvchi e'lon qilinganda «var» kalit so'zi ishlatilmasa, bunday holda global o'zgaruvchining qiymati almashib ketadi.

Amaliyotda sanoqni belgilash uchun ko'pincha «**i**» o'zgaruvchisi deyarli hamma joyda ishlatiladi, funksiya ichida yangidan aniqlanmagani uchun global qiymat o'zgartirilib yuboriladi va to'g'ri algoritmda tuzilgan dastur ham xato ishlashiga sababchi bo'ladi. Misol:


```
function x2(a) {                    funksiya
  i = a * a;                        e'lon
  return i;                         qilindi
}
var i = 10;                         10
var c = x2(12);                     144
alert(i);                           144
```

Eng yomoni, ba'zan amallar bajarilishida cheksiz takrorlanuvchan aylanaga tushib qoladi. Agar funksiya butunlay boshqa joyda (faylda) e'lon qilingan bo'lsa va dastur kodi juda katta bo'lsa, lokal va global o'zgaruvchilar to'g'ri aniqlanmaganligi sababli kelib chiqadigan bu kabi xatolikni topish mushkul bo'ladi. Shuning uchun har bir lokal maydonchaning o'zida ishlaydigan o'zgaruvchilarni «**var**» yoki «**let**» bilan [2.3.4.] e'lon qilish shart. Agar funksiyani lokal e'lon qilish kerak bo'lsa, u qavsga olinishi lozim:

```
(                                   alohida
  function dskr(a, b, c) {          maydonda
    var d = b * b - 4 * a * c;      funksiya
    return d;                       e'lon
  }                                 qilindi
);
var d = dskr(1, 2, 0);              dskr is not defined
```

Agar alohida maydonda yaratilgan funksiyaga murojaat qilib bo'lmasa, undan qanday foydalanish mumkin degan tabiiy savol tug'iladi. Buning yo'li oson, funksiya qiymat sifatida global o'zgaruvchiga beriladi.

Aslida funksiyaning nomi o'zimiz bilgan oddiy identifikator bo'lib, u «**function**» turidagi qiymatni qabul qiladi. Shunday ekan, quyidagi ikki xil yozuv bir xil ma'noni anglatadi:

```
function f()   { return true; }
f = function () { return true; }
```

Quyidagi yozuvlar esa turlicha ma'noni bildiradi:

kod	natija
`alert(f);`	`function f() { return true; }`
`alert(f());`	`true`

Ikkala sintaksis umumlashtirilsa:

kod	natija

`fun = function f() {`
`  f = 'Ali Bobo';`
`  alert(f);`
`}`
`f();`
`fun();`
	`f is not a function`
	`fun = function f() {`
	`  f = 'Ali Bobo';`
	`  alert(f);`
	`}`

Ayon bo'ladiki, «function» so'zidan keyin turgan «nom» bo'yicha murojaat qilish mumkin emas. Undan faqatgina funksiyaning tanasida lokal o'zgarmas sifatida foydalanish mumkin. Ya'ni qiymatini o'zlashtirish mumkin-u, o'zgartirish mumkin emas.

2.4.6. Obyekt

▪ **Obyekt** — eng murakkab va mukammal tur bo'lib, o'zida bir necha turdagi qiymatlarni mujassamlashtira oladi [2.3.1.]. Obyekt «**object**» so'zi yordamida anglashiladi. Uning sintaksisi quyidagicha:

{ *nomlanish*: *qiymat*, [, *nomlanish*: *qiymat*] }

Bunda «nomlanish» identifikator talablariga javob berishi kerak, aslida u oddiy o'zgaruvchi bo'lib, qiymati «:» dan so'ng beriladi.

Aytaylik, bir insonning omillari quyidagi tarzda turli o'zgaruvchilarda saqlangan:

```
var ism = "Anvar";
var yosh = 39;
var erkak = true;
```

Ushbu uchala qiymatni yagona o'zgaruvchiga obyekt sifatida umumlashtirish mumkin:

```
var inson = {
   ism: "Anvar",
   yosh: 39,
   erkak: true
};
```

Bunda «**inson**» nomli obyekt yaratilib, mos qiymatlar berildi. Faqat obyektning ichki identifikatorlariga qiymat tenglik (=) belgisi bilan emas, ikki nuqta (:) orqali o'zlashtiriladi. Nuqta-vergul (;) hadlarni ajratuvchi belgi bo'lganligi uchun oddiy verguldan (,) foydalaniladi.

Obyekt tanasi gulli qavsga olinadi, uni funksiyaning tanasi bilan chalkashtirish kerak emas, ular turli vazifani bajaradi. Albatta, obyekt ichki elementining qiymati sifatida yuqorida keltirilgan barcha turdagi qiymatlar, xususan, funksiya va obyekt berilishi mumkin.

Obyektning ichki elementiga murojaat qilish uchun ikki xil sintaksisning biridan foydalanish mumkin [2.3.1.]:

```
alert(inson.ism);
alert(inson['ism']);
```

Ikkinchi usul shunisi bilan qulayki, uning elementiga o'zgaruvchi yordamida murojaat qilsa bo'ladi:

```
var elem = 'ism';
alert(inson[elem]);
```

Obyekt haqida yana bir nozik jihatni qayd etib o'tish joizki, u o'zgarmas sifatida e'lon qilingan bo'lsa ham omillarini (elementlarining qiymatlarini) dastur davomida almashtirish mumkin [2.3.4.]:

```
const obj = { jinsi: "Erkak" };
obj = 39;
obj['jinsi'] = "Ayol";
```

{ jinsi: "Erkak" }

xato

{ jinsi: "Ayol" }

Jadval — obyektning maxsus ko'rinishi bo'lib, uning elementlari tartibli indekslangan bo'ladi va identifikatorlarni emas, faqatgina qiymatlarni saqlaydi. U gulli emas, to'g'ri burchakli qavs yordamida yaratiladi. Indeks doim noldan boshlanib, natural sonlar bilan o'sish tartibida davom etadigan bo'ladi. Jadval «**array**» so'zi yordamida nomlanadi.

Yuqoridagi misolni jadval tarzida quyidagicha yozish mumkin:

```
var inson = [ "Anvar", 39, true ];
```

Bunda «Anvar» — nolinchi, 39 — birinchi va nihoyat «true» — ikkinchi element hisoblanadi:

```
alert(inson[0]);
```

Anvar

Agar jadvalning **n**-elementiga qiymat berilsa, u mavjud bo'lgan taqdirda, yangilanadi. Yo'q bo'lsa, yaratiladi va ungacha bo'lgan elementlarni «undefined» qiymat bilan to'ldiradi:

```
var jadval = [];
jadval[2]  = 12;
```

[]

[undefined, undefined, 12]

Agar jadval yaratilayotganda uning elementlarini ajratuvchi vergul ishlatilsa-yu, qiymat ko'rsatilmasa, mos elementni «undefined» deb hisoblaydi. Faqat oxirgi vergul e'tiborsiz qoldiriladi:

```
var vektor = [, 246];
var v = [37, , "bet",];
```

[undefined, 246]

[37, undefined, "bet"]

2.4.7. Turni aniqlash

O'zgaruvchi qaysi turga mansubligini aniqlovchi maxsus **typeof** buyrug'i mavjud. Albatta, u o'zgarmaslarning turini ham ko'rsata oladi, lekin aniq qiymat turi o'zi ma'lum bo'lgan vaziyatda undan foydalanish samarasiz. Aynan bitta o'zgaruvchi ustida amallar baja-

rilishi jarayonida yoki unga yangi qiymat berilganda o'z turini o'zgartirib yuborishi mumkin. Ayni paytda u qanday turdaligini bilish lozim bo'lsa, «**typeof**» buyrug'i yordam beradi. Bu zamonaviy dasturlashda keng qo'llaniladigan uslub. Masalan, ayni bir o'zgaruvchi ba'zan yagona (number), ba'zan esa bir necha qiymat (array) qabul qila oladigan bo'lsin. Misol uchun, kimningdir bankdagi hisobi bitta, kimnikidir ko'p. Bunday vaziyatlarda avvalo turni tekshirib olish lozim, «number» bo'lsa, boshqacharoq kod yoziladi, jadval bo'lsa, o'ziga mos kod bajarilishi talab etiladi. Umuman olganda, o'zgaruvchini qo'llashdan oldin doim uning mavjudligini va qiymatini tekshirib olish – xatolikka kamroq yo'l qo'yilishiga sabab bo'luvchi muhim omildir.

JavaScriptning ilk talqinlarida «**typeof**» satr qaytaruvchi funksiya shaklida yaratilgan, keyinchalik buyruq sifatida mukammallashtirilgan. Shuning uchun ham uning ikki xil sintaksisi saqlanib qolgan:

```
typeof (o'zgaruvchi)
typeof o'zgaruvchi
```

Albatta, zamonaviyroq hisoblangan ikkinchi ko'rinishdan foydalanishni tavsiya etamiz.

Ammo afsuski, u doimo kutilganday aniq natija beravermaydi. Eng yomoni, «null» qiymatni ham obyekt sifatida aniqlaydi. Bu esa bo'sh obyekt «null»ga teng ekan degan noto'g'ri xulosa chiqarishga sabab bo'lishi mumkin, aslida ikkisi butkul boshqa-boshqa qiymatdir:

```
n = {};
```

```
n = null;
```


Quyida «x» o'zgaruvchining bir necha qiymatlarida qanday turga mansub bo'lishini «typeof» buyrug'i bilan aniqlaymiz. E'tibor qaratish kerakki, u nafaqat «null»ni, balki jadvalni ham obyekt sifatida ko'radi. Bunday yondashuv, aslida JavaScriptning o'zgaruvchilarga munosabati nuqtayi nazaridan olingan:

x	typeof x
0	number
0.1	number
NaN	number
Infinity	number
2E2	number
null	object
number	undefined
undefined	undefined
{}	object
[]	object
""	string
function () {}	function
true	boolean
false	boolean

Yuqorida turlar bilan tanishganimizda har birining qanday nom bilan atalishiga alohida urg'u berdik. Sababi, o'sha nom (bosh harfdan boshlanadi) va «new» kalit so'zi bilan o'zgaruvchini obyekt sifatida yaratish mumkin. Quyidagi ikki xil usulda e'lon qilishda ikkalasi ham teng kuchli (ahamiyatli) hisoblanadi:

```
var m = [];
var m = new Array();
```

E'tibor qaratish kerakki, «new»dan keyin kelayotgan atama, ism singari birinchi harfi katta, qolganlari kichik tarzda yoziladi. Barcha turlarni «new» orqali e'lon qilsa bo'ladi, ammo bunda o'zgaruvchining turi faqat obyekt sifatida shakllanadi:

```
var s = "A";                  satr: "A"
var s = new String("A");      obyekt: {0: "A", length: 1}
```

Obyektlar bilan ishlashda ular o'ziga xos qo'shimcha imkoniyatlar beradi [2.6.8.]. Ularni qo'llash shart bo'lmagan holda o'zgaruvchini «new» orqali e'lon qilish maqsadga muvofiq emas. Ammo shu tariqa yaratilgan o'zgaruvchilarning turini «typeof» faqat obyekt deb ko'rsataversa, asl turini qanday aniqlasa bo'ladi degan tabiiy savol tug'iladi. Bunga ham, juda mukammal bo'lmasa-da, ma'lum yechim mavjud. Buning uchun maxsus «**instanceof**» buyrug'idan foydalaniladi. U faqatgina obyektlar uchun qo'llaniladi, to'g'ridan-to'g'ri qiymat berilgan o'zgaruvchilar uchun qo'llansa, xato natija beradi. Misol keltiramiz:

```
s = "A";                  s instanceof String    false
s = new String("A");      s instanceof String    true
s = new String("A");      s instanceof Object    true
s = new String("A");      s instanceof Array     false
a = new Array(2,3,5);     a instanceof Array     true
```


Sonlar ustida ishlaganda «`var n = 3.14;`» singari qiymat berish, aslida «`var n = new Number(3.14);`» kabi e'lon qilingandan ko'ra sodda, tushunarli va qulayroq hisoblansa, «**Number**» obyekti nima uchun kerak degan o'rinli savol tug'ilishi tabiiy. Albatta, ushbu obyektning ham asqotib qolishi mumkin bo'lgan o'ziga xos xususiyatlari mavjud:

`Number.MAX_VALUE`	eng katta qiymati
`Number.MIN_VALUE`	nolga eng yaqin qiymati
`Number.POSITIVE_INFINITY`	musbat cheksiz son
`Number.NEGATIVE_INFINITY`	manfiy cheksiz son

«**Number**» obyektining boshqa xususiyatlariga keyingi mavzularimizda batafsil to'xtalamiz [2.6.1.].

Yuqorida ta'kidlab o'tganimiz kabi «**typeof**» ham, «**instanceof**» ham doimo biz kutgan natijani beravermaydi. Turni aniq bilish uchun o'zimiz maxsus funksiya yaratishimizga to'g'ri keladi:

```
function typeOf(v) {
  return Object.prototype.toString.call(v).slice(8,-1);
}
```

Ushbu funksiya turlarni aniqroq farqlashda yordam beradi. Unda yozilgan kodlar obyekt va funksiyalar ustida amallar bajarishga bag'ishlangan mavzular bilan tanishganimizdan so'ng tushunarliroq bo'ladi. Hozircha faqat uni «**typeof**» buyrug'i bilan taqqoslab bir necha misol ko'rib o'tishimiz mumkin:

x	typeof x	typeOf(x)
[]	object	Array
null	object	Null
new Number	object	Number

2.5. Amallar

Ifoda deganda, qiymati aniqlangan va aniqlanmagan o'zgaruvchi va o'zgarmaslar (ishtirokchilar) hamda ular vositasida bajariladigan amallar ketma-ket keltirilgan yozuvni tushunamiz. Demak, ifoda faqat ikki xil narsadan tarkib topadi. Birinchisi — ishtirokchilar, ikkinchisi — ular ustida bajariladigan amallar. Ishtirokchilar ma'lum turdagi o'zgaruvchi yoki identifikator yordamida berilsa, amallar nafaqat arifmetik va mantiqiy bo'ladi, balki funksiya va buyruqlar yordamida shakllanadi. Buni aniq tasavvur qilish uchun bir necha misollarni ko'rib o'tamiz:

ifoda	ishtirokchilar	amallar
12 * 5	12 va 5	*ko'paytirish*
11 / (x + 2)	x, 2, 11	*bo'lish, qo'shish, guruhlash*
2.4 - dot.x	2.4, dot, x	*ayirish, obyekt elementi*

Matematikadan bizga ma'lumki, ifodada amallar qanday ketma-ketlikda yozilmasin, ularning bajarilish ketma-ketligi qat'iy tartib asosida amalga oshiriladi. Har bir amalning o'z mavqei mavjud, u qanchalik yuqo-

ri bo'lsa, o'z qatoridagi amallarga nisbatan avvalroq bajarilishi lozimligini bildiradi. Masalan: qo'shishdan ko'ra ko'paytirish ustun turadi, ya'ni oldin bajariladi. Ko'paytirishdan ko'ra darajaga oshirish, undan ham qavs yordamida guruhlash yuqori mavqeli hisoblanadi.

Albatta, matematikada o'rganganimizga nisbatan JavaScriptdagi amallar turfa xil ekanligi, o'ziga xosligi bilan ajralib turadi. Quyida har bir amalning mavqeiga qarab, tartib raqami bo'yicha batafsil tanishib o'tamiz:

- **0 — umumlashtirish**: eng quyi mavqedagi amal bu – sanashni anglatuvchi, ifodalarni bir-biridan ajratib turuvchi vergul bilan yoziladi. Aslida ifodalar nuqtali vergul (;) bilan tugallanadi. Lekin yozuvni qisqartirish yoki bir ifoda o'rniga bir nechasini yozish maqsadida ular vergul (,) bilan ajratilishi ham mumkin. Quyidagi ikki xil yozuv aynan bir xil ahamiyatga ega:

```
var x = 5; var y = 2;        var x = 5, y = 2;
```

Bunda e'lon qiluvchi kalit so'z «var» qayta-qayta yozilmasligi uchun ifodalar vergul bilan umumlashtirilyapti. Bajarilish tartibi chapdan o'ngga, ya'ni avval verguldan chap tarafdagi ifoda hisoblanadi, keyin o'ng tarafdagisiga «o'tiladi». E'lon qilishdan tashqari, bir paytning o'zida bir necha o'zgaruvchiga turli xil qiymatlar berishda ham verguldan foydalanish qulay hisoblanadi. Bunday yo'lni «tarkiblanmagan qiymat berish usuli» deb ham yuritiladi. E'tibor qaratish kerakki, aslida bunda jadval imkoniyatlaridan foydalaniladi. Misol:

```
x = 5; y = 2;        [x, y] = [5, 2];
```

▪ **1 — kengaytirish** amali. Jadval va obyekt elementlarini yoki funksiyadagi kiruvchi qiymatlarning ro'yxatini kengaytirish uchun qo'llaniladi. Ya'ni mavjud obyekt elementlariga qo'shimcha kiritiladi. Masalan:

```
a = [9, 82, 17]; b = [ 13, ...a ];    [ 13, 9, 82, 17 ]
o = {min: 18}; j = { ...o, max: 90 }; { min: 18, max: 90 }
```

Unutmaslik kerakki, ushbu amal jadval, obyekt va funksiya omillaridan boshqa joylarda ishlamaydi. Bundan tashqari, boshqa turdagi qiymatni kengaytirish sifatida qo'llash ham xatodir. Masalan, obyekt ichida jadval kengaytirgich vazifasida ishlamaydi.

▪ **2 — `yield`** amali juda nozik tushuncha bo'lib, bir qarashda funksiyadagi «return»ga o'xshab ketadi. Faqat ko'rsatilgan qiymatning o'zini emas, «value» va «done» elementlariga ega bo'lgan obyektni qaytaradi. Buning uchun «navbatdagi» ma'nosini anglatuvchi ichki funksiyani chaqirish orqali murojaat qilishga to'g'ri keladi. Muhim jihati shundaki, ichki maydonda e'lon qilingan bo'lsa ham, qaytarilayotgan o'zgaruvchining oxirgi qiymati saqlab qolinadi. Ushbu amal faqat maxsus funksiyalarning ichidagina qo'llaniladi hamda unga o'ziga xos uslubda yondashish talab etiladi. Bu haqda to'la tasavvur paydo bo'lishi uchun yanada kengroq misollardan birini ko'rib o'tamiz:

```
function* f1() {
  var index = 0;
  yield index++;
}
```

oddiy funksiyadan farq qilish uchun yulduzcha (*) bilan e'lon qilish shart.


```
var fn = f1();
fn.next();          {value: 0, done: false}
fn.next();          {value: undefined, done: true}
f1().next();        {value: 0, done: false}
```

Yuqoridagi misoldan ko'rinadiki, funksiyani o'zgaruvchiga o'zlashtirib olmasa, «**yield**»da ko'rsatilgan o'zgaruvchining qiymati saqlanib qolmaydi. Ushbu g'ayritabiiy funksiya alohida munosabatni talab qiladi.

▪ **3 — qiymat berish** oddiy tenglik ishorasi bilan amalga oshiriladi. Faqat yozuvni qisqartirish maqsadida, shartli ravishda bir necha qo'shaloq ishoralar ham kiritilgan. Ular bilan batafsil tanishishdan avval arifmetik amallarni eslab o'tamiz: qo'shish (+), ayirish (-), ko'paytirish (*), bo'lish (/), (*bo'lingandagi*) qoldiq (%).

Ma'lumki, kompyuterda tub hisob-kitoblar ikkilik sanoq tizimida amalga oshiriladi. Undagi mavjud ikkitagina son **0** va **1**, «rost» va «yolg'on» sifatida baholanib, ular ustidagi bajariladigan amallarni «mantiqiy» deb yuritiladi. Mantiqiy amallar arifmetik amallardan farq qiladi. Ular qanday bajarilishi bilan quyidagi jadvalda tanishib o'tamiz:

X o'zgaruvchi qiymati	Y o'zgaruvchi qiymati	X & Y mantiqiy ko'paytirish	X ^ Y mantiqiy farqlash	X | Y mantiqiy qo'shish
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	0	1

Mantiqiy farqlashda ishlatilgan belgi (^) «BASIC» tilida darajaga oshirish uchun qo'llaniladi, buni chalkashtirmaslik kerak. JavaScriptda esa darajaga oshirish uchun ikkita ketma-ket ko'paytirish belgisi (**) qo'llaniladi: `4 ** 3 = 64`.

Dasturlash tilida surish degan tushuncha ham mavjud bo'lib, u oddiy matematik amalni anglatmaydi. Odatda, u ma'lumotlarni shifrlashda (maxfiylashtirish) foydalaniladi. U haqida biroz tushuncha hosil bo'lishi uchun quyidagi oddiy misolni ko'rib o'tamiz:

```
5 << 2                                                    20
```

Bu qanday hosil bo'ldi? Aytaylik, besh sonini saqlash uchun 32 bit joy ajratilgan bo'lsin (aslida ajratiladigan joy ko'pgina omillarga bog'liq). Ikkilik sanoq tizimida u nol va birlar bilan quyidagicha aks etadi hamda ikki xona chapga surilganda boshqa son hosil bo'ladi:

```
00000000000000000000000000000101    5
00000000000000000000000000010100    20
```

$1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 4 + 0 + 1 = 5$

$1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 0 * 2^0 = 16 + 4 = 20$

Dasturlashga qiziqqan o'quvchining o'nlik sanoq tizimidagi sonni ikkilikka va aksincha o'girish uquvi borligini inobatga olib, besh va yigirmani qanday kelib chiqqanligiga qisqacha to'xtaldik xolos. Umuman olganda, ushbu surish amali aslida juda kam qo'llaniladi.

Endi qiymat berish amali soddalashtirilgan ko'rinishda qanday yozilishini ko'rib chiqamiz:

`x = y`	`x = y`	***qiymat berish*** ...
`x += y`	`x = x + y`	... *qo'shish bilan*
`x -= y`	`x = x - y`	... *ayirish bilan*
`x *= y`	`x = x * y`	... *ko'paytirish bilan*
`x **= y`	`x = x ** y`	... *darajaga oshirish bilan*
`x /= y`	`x = x / y`	... *bo'lish bilan*
`x %= y`	`x = x % y`	... *qoldiqli bo'lish bilan*
`x <<= y`	`x = x << y`	... *chapga surish bilan*
`x >>= y`	`x = x >> y`	... *o'ngga surish bilan*
`x >>>= y`	`x = x >>> y`	... *faqat musbat surish bilan*
`x &= y`	`x = x & y`	... *mantiqiy ko'paytirish bilan*
`x ^= y`	`x = x ^ y`	... *mantiqiy farqlash bilan*
`x |= y`	`x = x | y`	... *mantiqiy qo'shish bilan*

Qo'shish amalida ishtirokchilardan biri satr bo'lgan taqdirda natija satr sifatida namoyon bo'ladi. Qolgan amallar aslida sonlar uchun mo'ljallangan.

▪ **4 — talab amali** ma'lum mantiqiy shart asosida ikki ifodadan birining bajarilishini ta'minlaydi.

```
shart ? rost bo'lsa bajariladi : yolg'on bo'lsa ishlaydi
```

Bunda mantiqiy qiymat qabul qiladigan shartdan so'ng albatta ikki xil vaziyatda amalga oshiriluvchi ifoda yozilishi talab etiladi, ya'ni qisqartma sifatida (boshqa tillarda mavjud bo'lgani singari «**?:**» tarzida) bittasini tushirib qoldirish mumkin emas.

Misol:

```
yosh > 18 ? ruxsat = true : alert('Mumkin emas');
```

▪ **5 — «yoki» amali** mantiqiy qo'shish bilan bir xil hisoblanadi, faqat nol va bir ishtirokida emas, mantiqiy qiymat qabul qiladigan shartlar ustida bajariladi. Natija faqatgina «true» va «false» bo'lib qolmasdan, bor bo'lsa, birinchi uchragan «rost» turkumga mansub qiymatni qaytaradi. Bu bir necha qiymatlardan mavjudini tanlab olib qo'llash uchun juda qulay. Albatta, hamma ifoda «yolg'on» qiymat qabul qiladigan bo'lsa, noiloj birinchi uchragan ifoda qiymatini qabul qiladi. Ushbu «yoki» amali «||» belgilari bilan ifodalanadi:

```
"a" / 3 || 2 * 10 || 2 > 1          20
```

▪ **6 — «va» amali** (&&) mantiqiy ko'paytirish singari hisoblansa ham bir necha ifodalar ichidan qiymati yolg'on bo'ladigan birinchisini tanlab olishga imkon beradi. Agar barcha ifoda rost qiymat qabul qiladigan bo'lsa, oxirgisini qaytaradi:

```
200 / 3 && 13 > 3 && 2 / "a" && 12 * 4      NaN
200 / 3 && 13 > 3 && 12 * 4                 48
```

▪ **7 — mantiqiy qo'shish** (|).

▪ **8 — mantiqiy farqlash** (^).

▪ **9 — mantiqiy ko'paytirish** (&).

▪ **10 — tenglik** amallari. Tenglik amali yuqorida tanishganimiz singari «==» belgilari ketma-ketligi bilan ifodalanadi [2.4.4.]. Lekin bu aslida qat'iy bo'lmagan tenglik hisoblanadi. Unda satr sifatida kelgan son oddiy son yoki ifoda natijasi bilan bir xil hisoblanadi. Bu esa

ba'zida qandaydir xatolikka yo'l qo'yishga sabab bo'ladi. Shuning uchun ham qat'iy tenglik kiritilgan, u «===» kabi qo'llaniladi. Bunda taqqoslanayotgan ifodalarning qiymati qaysi turga mansubligi ham inobatga olinadi.

Tenglikka teskari bo'lgan «teng emas» (!=) va «qat'iy teng emas» (!==) amallari ham 10-mavqeli sanaladi:

Misol:	`2 * '2' == 4`	`2 + '2' == 22`	`2 + '2' === 22`
Natija:	`true`	`true`	`false`

▪ **11 — taqqoslash** amallari. Biz uchun «kichik» (<), «kichik yoki teng» (<=), «katta» (>), «katta yoki teng» (>=) taqqoslash amallari juda yaxshi tanish. Ular bilan bir qatorda «**instanceof**» [2.4.7.] ham 11-mavqeli sanaladi.

Yana bir ajoyib shartni tekshiruvchi amal bor-ki, u berilgan qiymat ko'rsatilgan obyekt yoki jadval indeksida mavjudligini aniqlab beradi. E'tibor qaratish kerakki, u qiymat bo'yicha emas, indeks bo'yicha tekshiradi. Ushbu amal «**in**» kalit so'zi bilan ifodalanadi. Misol:

```
var a = ["ali", "vali", "hali"];          jadval
var o = {ali: "aka", vali: "amaki"};      obyekt
   2  in a;                               true
   3  in a;                               false
  '2' in a;                               true
'ali' in a;                               false
'ali' in o;                               true
'aka' in o;                               false
```

▪ **12 — surish** amallari yuqorida tanishganimizday chapga (<<), o'ngga (>>) va faqat musbat o'ngga (>>>) ikkilik sanoq tizimiga o'tkazib siljitish orqali bajariladi.

▪ **13 — qo‘shish** (+) va **ayirish** (-) amallari.

▪ **14 — ko‘paytirish** (*), **bo‘lish** (/) va **qoldiq** (%) amallari bajarilayotganda ishtirokchilar imkon qadar songa aylantirib olinadi, buning iloji bo‘lmagan taqdirda natija «NaN» bo‘ladi.

▪ **15 — darajaga oshirish** (**). $3^3 = 27$ matematik yozuv «3 ** 3» singari ifodalanadi. Ushbu amalning nozik jihati shundaki, uning bajarilish tartibi biz ko‘nikkanday chapdan o‘ngga emas, aksincha, o‘ngdan chapga qarab amalga oshiriladi. Ya’ni, «3 ** 2 ** 3» ifodaning qiymati 729 emas, 6561.

```
3 ** 2 ** 3 = (3 ** 2) ** 3 = 9 ** 3 = 9 * 9 * 9 = 729    xato
3 ** 2 ** 3 = 3 ** (2 ** 3) = 3 ** 8 = 6561              to‘g‘ri
```

▪ **16** — ifodaning chap tarafidan yoziladigan quyidagi amallar ushbu 16-mavqega kiradi:

◊ **Mantiqiy inkor.** Ifodaning qiymati «rost»ga teng ahamiyatga ega bo‘lsa «yolg‘on» va «yolg‘on»ga teng kuchli bo‘lsa, «rost»ga o‘zgartiradi. Ifoda oldiga undov belgisi qo‘yish bilan aniqlanadi:

```
!ifoda
```

◊ **Bitli inkor**. Qiymatni ikkilik sanoq tizimiga o‘tkazishdan hosil bo‘lgan sondagi nollar o‘rniga bir va birlar o‘rniga nol qo‘yib almashtirishdan hosil bo‘lgan yangi son. Bunday bitli amallar hayotiy masalalarni hal etishda keng omma tomonidan amaliyotda ko‘p qo‘llanilmasligi mumkin, ammo kompyuterning ichki hisob-kitobida bu juda qulay va mumtoz (ilk)

usul hisoblangani uchun tatbiq etishdan butkul voz kechilmagan. Umuman olganda, ixtiyoriy sonning bitli inkori modul bo‘yicha o‘zidan bitta katta teskari ishorali songa teng bo‘ladi, bu ayrim matematik hisoblashlarda asqotishi mumkin. U «**to‘lqin**» (~) belgisi (tilda) bilan ifodalanadi:

```
~x                    -(x + 1)
```

◊ **Ishora qo‘yish**. Matematikadan ma’lumki, manfiy sonlar minus (-) ishorasi bilan yoziladi, musbat sonlar oldida esa plyus (+) bor deb qaraladi va uni yozish talab etilmaydi. Xuddi shu qoida JavaScriptda ham o‘zgarishsiz tatbiq etiladi. Agar sonning oldiga minus qo‘yilsa, u qarama-qarshi songa aylanadi: «y = -x» tarzda yozish to‘g‘ri hisoblanadi, ayrim boshqa dasturlash tillaridagidek, «y = -1 * x» sifatida yozish shart deya talab etilmaydi. Agar bunda «x» son bo‘lmasa, imkon qadar uni songa aylantirishga harakat qiladi, uddalay olmasa, «NaN» javobini qaytaradi.

◊ **Olddan oshirish** amali ikkita plyus (++) hamda **olddan kamaytirish** amali ikkita minus bilan (--) belgilanib, ifodaning oldi tarafiga qo‘yiladi, bunda uning qiymati ifodani bajarish jarayonida mos ravishda birga orttirilib yoki kamaytirilib hisoblanadi:

```
var a = 32, b = 12;          (a = 32, b = 12)
24 + ++a;                    57 (a = 33)
24 - --b;                    13 (b = 11)
```

◊ **typeof** — turni aniqlovchi amal [2.4.7.].

◊ **void** — berilgan ifodani bajaradi, ammo natija sifatida «undefined» qiymat qaytaradi. Natija emas, hisoblanishi lozim bo'lgan ifodalarning qiymatini aniqlash muhim bo'lgan hollarda qo'llaniladi:

```
var a = 32, b = 12;                    (b = 12)
void(b = a ** 2);          undefined   (b = 1024)
```

◊ **delete** — dasturlovchi tomonidan yaratilgan obyektning elementini o'chirish uchun qo'llaniladi. JavaScript tomonidan taqdim etilgan standart obyekt elementlarini yoki boshqa turdagi o'zgaruvchilarni o'chirish imkoniyatiga ega emas. O'chirish muvaffaqiyatli kechsa «true», aks holda «false» qaytaradi:

```
var a = { boyi: 12, eni: 28};
delete a.boyi                          {eni: 28}
```

▪ **17 — ortdan oshirish** yoki **ortdan kamaytirish** amallarida avvaldan berilgan qiymat bo'yicha hisoblash amalga oshirilgandan so'ng, o'zgaruvchining o'z qiymati bittaga oshiriladi yoki kamaytiriladi:

```
var a = 32, b = 12;                    (a = 32, b = 12)
24 + a++;                  56          (a = 33)
24 - b--;                  12          (b = 11)
```

▪ **18** — funksiyaga murojaat qilish [2.4.5.] yoki omilsiz obyekt yaratish [2.4.7.].

▪ **19** — obyekt elementiga murojaat va omil bilan obyekt yaratish ushbu mavqega mansub amallardir:


```
var a = { boyi: 12, eni: 28 };
a.boyi                                 12
a['eni']                               28
new String('Ha')                       { 0: "H", 1: "a", length: 2 }
```

▪ **20 — guruhlash** qavsga olish orqali amalga oshiriladi va u barcha amaldan yuqori mavqega ega bo'lib, birinchi bajarilishi ta'minlanadi:

```
3 * (4 + 8) ** 2                       432
```

Yuqorida keltirilgan ro'yxat bo'yicha barchasining bajarilish ketma-ketligini aniq yodda tutish qiyin, albatta. Shuning uchun ifoda yozish jarayonida qaysi amal birinchi bajarilishiga ishonch komil bo'lmasa, qavsdan foydalanish tavsiya etiladi.

2.6. Biriktirilgan obyektlar

Unutmasligimiz kerakki, JavaScriptda hamma narsa obyekt sifatida aniqlanadi. Oddiy o'zgaruvchiga bitta son qiymat bersak ham aslida u global «**window**» obyektining elementi sifatida aks ettiriladi. Agar «a» o'zgaruvchini «`var a = 10;`» deb e'lon qilsak, unga quyidagi ikki usuldagi murojaat ham birday qabul qilinadi:

```
alert(a);                  alert(window.a);
```

Ushbu bo'limda JavaScriptning o'zida mavjud bo'lgan standart ichki (biriktirilgan) obyektlar bilan tani-

shib o'tamiz. Aslida ular juda ham ko'p va aksariyati boshqalarini qo'llash hamda ayrim vazifalarinigina bajarib berish uchun kiritilib, asosan brauzer yaratuvchilariga mo'ljallangan. Biz faqat asosiy, shu bilan birga, mukammal dasturlash uchun yetarli bo'lgan muhim obyektlar bilangina tanishib o'tamiz.

Hamma narsa obyekt va uning elementlari sifatida qaralsa ham, biz o'rganish qulay bo'lishi uchun boshqa dasturlash tillariga mutanosib holda, tushunish oson bo'ladigan ravishda maxsus turlarga bo'lib, kelishilgan atamalar kiritib o'tishimizga to'g'ri keladi. Biriktirilgan obyektlarning nomlanishiga bog'langan va qat'iy o'z obyekti «ism»i bilan birga murojaat qilinadigan funksiyani «**tamoyil**», aniq qiymat qaytaradigan nomlanishni «**xossa**» deb yuritamiz. Obyekt qiymatiga va uni o'zlashtirgan o'zgaruvchi yoki o'zgarmasga birlashtirib yoziladigan hamda uning ustida ma'lum amallarni bajarib natija qaytaradigan funksiyani — «**uslub**», identifikatorni esa — «**xususiyat**» deb ataymiz.

Bitta nomdagi xususiyat va uslublar turli obyektlar uchun turlicha hisoblanib, o'ziga xos natija qaytarishi mumkin.

2.6.1. Son (Number)

Matematika fanidan «son» tushunchasi yaxshi tanish. Bilamizki, o'nli yozuvda verguldan keyin qancha nol qo'yilsa ham qiymat o'zgarmaydi. Dasturlash tilida sonning oldiga yozilgan nollar ham xuddi shunday e'tiborsiz qoldiriladi. Bunday raqamlar «vaznsiz» deb qaraladi. Son oldiga yozilgan minus manfiyligini bildirgani uchun u «vaznli», plyus esa «vaznsiz» belgi hisoblanadi.

Sonni o'zgaruvchiga qiymat berib oddiygina yaratishdan tashqari quyidagicha sintaksisda obyekt sifatida aniqlash ham mumkin:

```
new Number(qiymat)
```

Agar bunda «qiymat»ni son sifatida aniqlash imkoni bo'lmasa, «NaN» qabul qiladi. Umuman olganda, faqat raqamlar (va yagona nuqta) qatnashgan satrni «new»ni qo'llamasdan, «Number» funksiyasi orqali songa aylantirish mumkin.

`Number` obyekti quyidagi xossalarga ega:

- `Number.EPSILON` — ikkita haqiqiy sonning bir-biridan farqi eng kamida qancha bo'lishini aniqlaydi.
- `Number.MAX_SAFE_INTEGER` — JavaScriptda qo'llash tavsiya etiladigan eng katta butun son: $(2^{53}-1)$.
- `Number.MAX_VALUE` — eng katta musbat son.
- `Number.MIN_SAFE_INTEGER` — JavaScriptda qo'llash tavsiya etiladigan eng kichik butun son: $-(2^{53}-1)$.
- `Number.MIN_VALUE` — eng kichik musbat haqiqiy son. Ya'ni, haqiqiy sonlar dasturlash tilida uzluksiz hisoblanmagani uchun, noldan keyin aynan shu son keladi.
- `Number.NaN` — «sonmas» (son bo'lmagan) qiymat [2.4.2.]. Aynan shu qiymat ko'p qo'llanilgani bois global

tarzda ishlatish imkoniyati yaratilgan. Ya'ni, «NaN» va «Number.NaN» bir xil ma'noni anglatadi.

- **Number.NEGATIVE_INFINITY** — manfiy cheksizlikni anglatadi. Ya'ni «-Infinity» [2.4.2.].

- **Number.POSITIVE_INFINITY** — musbat cheksizlikni anglatuvchi (katta) son. (Aslida 2^{53} ga teng)

Number obyekti quyidagi tamoyillarga (xos funksiyalarga) ega:

- **Number.isNaN(*qiymat*)** — «qiymat»ning sonligini tekshiradi, mantiqiy tur qaytaradi. Sonmas o'ziga ham teng bo'lmagani uchun hisoblash natijasini taqqoslash amali bilan tekshirib bo'lmaydi [2.4.4.], faqat ushbu funksiyadan foydalaniladi. Uni global tarzda qo'llash ham mumkin.

- **Number.isFinite(*qiymat*)** — «qiymat»ning asl sonligini tekshiradi. Sondan boshqa qiymatlar yoki «NaN», «Infinity» kabilar berilgan holda «yolg'on» qaytaradi. Global «**isFinite**» funksiyadan farqli o'laroq, satrni songa o'zi aylantirishga urinmaydi, ya'ni «**Number.isFinite('0')**»ning qiymati «**false**», ammo global «**isFinite('0')**»ning qiymati esa «**true**».

- **Number.isInteger(*qiymat*)** — berilgan «qiymat» aniq butun son ekanligini tekshiradi.

- **Number.isSafeInteger(*qiymat*)** — «qiymat»ning xavfsiz oraliqdagi butun son ekanini tekshiradi. Ya'ni, $-(2^{53}-1)$ dan $(2^{53}-1)$ gacha oraliqdagi butun son.

- **Number.parseFloat(*satr*)** — berilgan «satr»ni imkon qadar haqiqiy songa o'girib beradi, uddalay olmasa, «NaN» qaytaradi. Ya'ni satrda plyus, minus, raqamlar va nuqta ko'rsa va ularning ketma-ketligi o'nli yozuvdagi haqiqiy sonni ifodalaydigan bo'lsa, to boshqa belgi uchraguncha yoki satr tugaguncha o'shanday tartibdagi belgilar bilan yagona sonni qaytaradi. Ushbu tamoyilni global tarzda ham qo'llash imkoniyati yaratilgan. Misol:

```
Number.parseFloat("1212.31w");      1212.31
```

- **Number.parseInt(*satr*)** — «parseFloat»ning faqat butun son uchun tatbiq etiladigan ko'rinishi.

Yuqoridagilardan tashqari Number obyekti o'ziga xos uslublar (biriktirilgan funksiyalar)ga ham ega:

- **toExponential(*aniqlik darajasi*)** — sonning «eksponensial» shakli, ya'ni o'n darajasi bilan ifodalangan ko'rinishini satr sifatida qaytaradi. Bunda birinchi raqamdan keyin butun qismning tugaganini anglatuvchi nuqta qo'yiladi va o'nning darajasiga ko'paytirish yordamida «nuqta suriladi»:

```
123456                          1.23456·10^5
0.0123456                       1.23456·10^-2
123456 .toExponential()         "1.23456e+5"
123.456.toExponential()         "1.23456e+2"
0.0123456.toExponential()       "1.23456e-2"
123.456.toExponential(2)        "1.23e+2"
12.3456.toExponential(3)        "1.235e+1"
```

E'tibor berish kerakki, butun sondan so'ng bo'sh joy qo'yilgan, aslida uslublarga murojaat qilinayotganda doim nuqtani bo'sh joylar bilan ajratib yozish ham mumkin. Butun sonning o'z tarkibida nuqta bo'lmagani uchun uslub nomidan avval qo'yiladigan nuqtadan oldin bo'sh joy bo'lishi shart yoki son qavs ichida keltirilishi kerak, aks holda sintaksis xato hisoblanadi.

▪ `toFixed(`*aniqlik darajasi*`)` — sonni berilgan aniqlikda yaxlitlab ko'rsatadi. Agar son manfiy bo'lmasa, natijani satr sifatida qaytaradi. Misol:

```
-2.345.toFixed(7);          -2.345
2.345.toFixed(7);           "2.3450000"
```

▪ `toLocaleString([`*mahalliylash*`[, `*siymo*`]])` — sonni mahalliy ommaga tushunarli tarzda ifodalovchi universal funksiya. Bunda zarurmas «mahalliylash» omili qaysi tildagi raqamlar va ularning tartiblash qonuniyatini tatbiq etishni aniqlaydi. Odatda, arab raqamlari deb yuritiladigan sonlar qonuniyatlari tatbiq etiladi. «Siymo» sifatida quyidagi elementlarga ega bo'lishi mumkin bo'lgan yagona obyekt beriladi:

◊ **localeMatcher** — mahalliy qonuniyatlarni tatbiq etish algoritmi, u «`best fit`» (*eng ma'quli*) yoki «`lookup`» (*izlash*) qiymatlaridan birini qabul qiladi.

◊ **style** — aks ettirish uslubi, aniqlangan tur: «`decimal`» (*son uchun, odatiy qiymat*), «`currency`» (*valyuta, pul uchun*), «`percent`» (*foiz uchun*).

◊ **currency** — qaysi valyuta birligida ko'rinishini aniqlaydi (`style = "currency"` holda ishlaydi).

◊ **currencyDisplay** — valyuta belgisi qanday namoyon bo'lishini aniqlaydi, aniqlangan tur: «`code`» (*ISOdagi kodi bilan*), «`symbol`» (*maxsus belgi bilan*), «`name`» (*atalishi singari yozish orqali*).

◊ **useGrouping** — guruhlab yozishni ta'minlash uchun qo'llaniladi. Mantiqiy tur qabul qiladi, odatiy qiymati «true».

◊ **minimumIntegerDigits** — butun qismida kamida nechta raqam bo'lishi (1 dan 21 gacha orasida).

◊ **minimumFractionDigits** — kasr qismida kamida nechta raqam bo'lishi (0 dan 20 gacha).

◊ **maximumFractionDigits** — kasr qismida ko'pi bilan nechta raqam bo'lishi (0 dan 20 gacha). Odatiy qiymati foiz uchun 0, pul uchun 2, qolgan hollarda 3 raqami qo'llaniladi.

◊ **minimumSignificantDigits** — vaznli raqamlarning eng kam miqdori (1 dan 21 gacha).

◊ **maximumSignificantDigits** — vaznli raqamlar yuqori chegarasi (1 dan 21 gacha). Ko'rsatilmasa «minimumSignificantDigits»dagi qiymatni oladi:

```
123456.789.toLocaleString(      "123 456,79 $"
  'ru-RU', {
    style: 'currency',
    currency: 'USD' }
);
```

▪ `toPrecision(`*vaznli raqamlar soni*`)` — sonning vaznli raqamlarini berilgan miqdordagicha yaxlitlab ko'rsatadi. Vaznli raqamlar miqdori sanalayotganda haqiqiy sondagi nol butun va undan keyin keladigan boshlan-

g'ich nollar hisobga kirmaydi. O'n darajasi bilan ifodalangan sonlar uchun «**e**»gacha raqamlar inobatga olinadi:

```
0.000098765.toPrecision(4)      "0.00009876"
98765 .toPrecision(4)           "9.877e+4"
98.765.toPrecision(4)           "98.77"
-98765.toPrecision(4)           -98770
-987.65.toPrecision(4)          -987.6
```

- **toString(*sanoq tizimi*)** — sonni ko'rsatilgan sanoq tizimiga o'girib satr sifatida qaytaradi:

```
10 .toString(8)                 "12"
(-10).toString(2)               "-1010"
(-0xff).toString()              "-255"
```

- **valueOf** — obyekt sifatida aniqlangan sonning «number» sifatidagi qiymatini qaytaradi.

Ushbu funksiyalar uchun «aniqlik darajasi» sifatida haddan ziyod katta qiymat berilganda (bir necha yuz nazarda tutilyapti), «RangeError» xatosi yuzaga keladi, «TypeError» xatosi esa o'z nomidan ayonki, boshqa turlarga tatbiq etilganda sodir bo'ladi.

2.6.2. Satr (String)

Satr [2.4.1.] ko'rinishidagi obyekt yaratish va uning ustida amallar bajarishda «String»dan foydalaniladi:

new String(*qiymat*)	«qiymat»li obyekt yaratish
String(*qiymat*)	«qiymat»ni satrga aylantirish

Satr yagona «length» xususiyatiga ega bo'lib, u satrda nechta belgi borligini aniqlaydi:

```
"«Toshkent»".length             10
```

Bunda «UTF-8» kodlamasi qo'llaniladi, ba'zi ushbu kodlamani qo'llamaydigan dasturlash tillari kirill yozuvidagi harf va belgilarni ikkita yoki uchta belgi sifatida sanashi ham mumkin.

Avval ta'kidlab o'tganimiz kabi, satrni bir necha qatorlarga bo'lib yozish mumkin emas. Agar shu kabi maxsus ajratuvchilar qo'llanishi kerak bo'lsa, o'ziga xos kod bilan yozish taqozo etiladi:

\0	**null** belgi
\'	bir tirnoq
\"	qo'shtirnoq
****	akstaqsim
\n	yangi satr
\r	bo'lakcha yakuni
\v	tik tabulyatsiya
\t	yotiq tabulyatsiya
\b	ortga qaytarish (*olddan o'chirish — backspace*)
\f	sahifa ajratuvchisi
\uXXXX	yunikodda yozilgan belgi
\xXX	«Latin-1» kodlamasidagi belgi

E'tibor qaratish kerakki, NULL belgi mavjud bo'lmagan satrni anglatmaydi, ya'ni "**\0**" belgi "" (bo'sh) satrga teng emas. Shuningdek, namoyon etilayotganda bir necha qatorda ko'rinadigan yozuv ham qiymat sifatida o'zlashtirilayotganda «**\n**» belgisi bilan ifodalanishi shart. Ko'rinadigan barcha belgilar o'z holicha

yozilaveradi, lekin «Enter» tugmasining qiymati «\r» (Windows tizimida «\n\r», boshqalarida «\n» bo'lishi mumkin) olddan o'chirish (backspace) tugmasining qiymati «\b» sifatida qabul qilinadi. Albatta, yozuv namoyon etilganda «\r», «\v», «\b» kabi belgilar ko'rinmasligi mumkin, ammo ularning mavjudligini «length» xususiyati yordamida tekshirib olsa bo'ladi.

String obyekti quyidagi tamoyillarga ega:

- **String.fromCharCode**(num1[, ... [, numN]]) — berilgan sonlarga mos yunikoddagi belgilarni birlashtirib satr (obyekt emas!) sifatida qaytaradi:

```
String.fromCharCode(75, 73, 84, 79, 66);  "KITOB"
```

Ushbu uslubga muqobil bo'lgan «fromCodePoint» va satr ichida o'zgaruvchilarning qiymatini berish uchun mo'ljallangan «raw» (PHPdagi kabi) uslublari ko'p brauzerlarda qo'llanmagani bois ishlatish tavsiya etilmaydi.

Dasturlashda eng ko'p qo'llaniladigan amallar satrlarning ustida bajariladi. Biror turdagi qiymatni satrga aylantirib olish mumkin. Umuman olganda, bajarilishi lozim bo'lgan kodni ham satr sifatida qarash mumkin.

Ixtiyoriy satrni JavaScript kodi sifatida ishga tushirish uchun **eval** funksiyasidan foydalaniladi. Ushbu funksiya, ayniqsa, tajribali dasturchilar uchun ishlash jarayonida qiyinchiliklarni yengillashtiruvchi juda qulay vosita hisoblanadi. Lekin bunda bir nozik jihatga alohida e'tibor berish shart, oddiy satr bilan obyekt sifatida yaratilgan satrni **eval** orqali bajarishda biroz farq bor:


```
var s1 = '2 + 2';                  "2 + 2"
var s2 = new String('2 + 2');      String {Obyekt}
alert(s1);                         2 + 2
alert(s2);                         2 + 2
alert(eval(s1));                   4
alert(eval(s2));                   2 + 2
```

String obyektining quyidagi uslublari mavjud:

- **charAt**(*indeks*) — ko'rsatilgan «indeks»dagi belgini qaytaradi, u noldan satr uzunligidan bitta kam oraliqdagi butun son bo'lishi kerak, aks holda bo'sh satr qaytaradi. Sanoq dasturlash tillarida noldan boshlangani uchun, birinchi belgining indeksi 0 hisoblanadi. Umuman olganda, ushbu funksiyani ishlatishning o'rniga indeksiga to'g'ridan-to'g'ri murojaat qilish mumkin:

```
"«Toshkent»".charAt(7)  "n"   "«Toshkent»"[7]  "n"
```

- **charCodeAt**(*indeks*) — ko'rsatilgan «indeks»dagi belgining yunikod bo'yicha tartib raqamini qaytaradi, qiymati son. Agar berilgan indeks qiymati satr uzunligidan oshib ketsa, natija «NaN» bo'ladi.

- **concat**(*satr2*, *satr3*[, ..., *satrN*]) — satr ortidan berilgan satrlarni birlashtiradi.

- **includes**(*qidiriluvchi*[, *o'rni*]) — satr ichidan «qidiriluvchi» qism satrni ko'rsatilgan «o'rni»dan boshlab izlaydi. Izlashning boshlang'ich «o'rni» ko'rsatilmagan holda u nol qabul qiladi. Topilishiga qarab mantiqiy tur qaytaradi. Ushbu funksiya keyin qo'shilgan,

hozircha uni Internet Explorer va aksariyat mobil brauzerlar qo'llamasligi mumkin.

▪ **endsWith(*qidiriluvchi*[, *o'rni*])** — satr berilgan «qidiriluvchi» bilan yakunlanganligini tekshiradi, «o'rni»ga qiymat sifatida satr uzunligidan kichik son berish mumkin, agar ko'rsatilmagan bo'lsa, satr oxirigacha qaraladi (ushbu funksiya ham keyin qo'shilganligi bois hozircha ayrim mobil brauzerlarda ishlamasligi mumkin). Mantiqiy tur qaytaradi, misol:

```
var str = new String("Mukammal dasturlash. 2-kitob");
str.endsWith("kitob")                    true
str.endsWith("dasturlash")               false
str.endsWith("dasturlash", 19)           true
```

▪ **startsWith(*qidiriluvchi*[, *o'rni*])** — «endsWith» singari, faqat satr boshlang'ich qismini tekshiradi.

▪ **indexOf(*qidiriluvchi*[, *o'rni*])** — satr ichidan «o'rni»dan boshlab ilk uchragan «qidiriluvchi»ning joyini qaytaradi. Qiymati: butun son, agar topilmasa, **–1** qaytaradi. Agar qidiriluvchi bo'sh satr bo'lsa, satr uzunligini qaytaradi, faqat «o'rni» sifatida undan kichik natural son ko'rsatilgan taqdirdagina shu sonning o'zini qaytaradi. Ushbu funksiya JavaScriptning ilk talqinidayoq kiritilganligi bois «includes» va «endsWith»lardan farqli o'laroq barcha brauzerlarda bir xilda o'z vazifasini to'g'ri bajaradi.

▪ **lastIndexOf(*qidiriluvchi*[, *o'rni*])** — faqat so'nggi uchragan «qidiriluvchi»ning joyini qaytaradigan ushbu funksiya «indexOf» bilan bir xil uslubga ega.

▪ **localeCompare(*qiyosiy satr*)** — «qiyosiy satr» asos satrdan alifbo tartibida oldin kelishi kerak bo'lsa 1, aks holda –1 qiymat qaytaradi. Aslida ushbu funksiyaning «toLocaleString»nikiga o'xshash yana ikkita qo'shimcha omili mavjud bo'lib, ko'pgina brauzerlarda qo'llanmaganligi bois, ular haqida batafsil to'xtalmadik. Bu funksiya satrlarni taqqoslash uchun xizmat qiladi:

```
'HTML'.localeCompare('CSS')              1
'HTML'.localeCompare('JavaScript')       -1
```

▪ **padEnd(*uzunligi*[, *qo'shiluvchi*])** — satr ko'rsatilgan «uzunligi»ga yetguncha davomidan «qo'shiluvchi» (odatiy qiymati: bo'sh joy) qism satrni takrorlab ulab boradi. Ushbu yangi imkoniyat ba'zi brauzerlarda ishlamasligi mumkin. Misol:

```
'abc'.padEnd(6)                          "abc   "
'abc'.padEnd(10, "jas");                 "abcjasjasj"
```

▪ **padStart** — «padEnd» singari, boshidan qo'shadi.

▪ **repeat(*karra*)** — satrning ko'rsatilgan «karra» marotaba takrorlab «uzaytiradi». (Ba'zi brauzerlarda ishlamasligi mumkin.)

▪ **replace(*nimani*, *nimaga*)** — qism satrni almashtiradi, misol:

```
"JavaScript".replace('S', '$');          "Java$cript"
"JavaScript".replace('s', '$');          "JavaScript"
"JavaScript".replace('a', '@');          "J@vaScript"
```

Ushbu misollardan ko'rinadiki, ushbu funksiya faqat birinchi uchragan qism satrning katta-kichik harflarini

farqlagan holda almashtirar ekan. Unda barchasini almashtirish imkoniyati ham bor. Buning uchun «muntazam ifoda»lardan foydalaniladi [2.7.].

- **slice(*dan*[, *gacha*])** — qism satrni kesib oladi. Agar omil sifatida manfiy son berilsa, joyi satr orqasidan teskari tartibda sanaladi, misol:

```
"JavaScript".slice(-8, 4);          "va"
```

- **split(*ajratuvchi*[, *miqdori*])** — satrni berilgan «ajratuvchi» yordamida bo'laklarga bo'lib, jadval sifatida qaytaradi. Agar «miqdori» berilgan bo'lsa, jadvalning unda ko'rsatilgandagidan oshiq elementlarini tashlab yuboradi. Misol:

```
"png|gif|jpg".split('|');          ["png","gif","jpg"]
"HTML,XML,SVG".split(",", 2);      ["HTML", "XML"]
```

- **substr(*dan*[, *miqdori*])** — qism satr kesib oladi.

- **substring(*joy1*[, *joy2*])** — satrning «joy1» va «joy2» oralig'idagi qismidan nusxa oladi, bu «slice» va «substr»lardan farqli o'laroq, manfiy omil berilganda satr ortidan boshlab sanamaydi. Kiritiladigan ikkinchi qiymat birinchisidan katta bo'lishi ham talab etilmaydi:

```
"JavaScript".substring(4, 7);      "Scr"
"JavaScript".substring(7, 4);      "Scr"
```

- **toLowerCase(); toLocaleLowerCase()** — satrni faqat kichik harflarda ifodalaydi. Ikkinchi funksiyaning iyeroglifli yozuvdagi til boyligi ko'proq.

- **toUpperCase(); toLocaleUpperCase()** — satrdagi kichik harflarni bosh harflarga o'giradi.

- **toString()** — obyekt sifatida e'lon qilingan satrni oddiy satrga aylantiradi.

- **trim()** — satr boshi va oxiridagi barcha bo'sh joylarni, xatboshi, qatorlarni bo'luvchi belgi hamda tabulyatsiyalarni o'chiradi. Faqat o'ngdan yoki faqat chapdan o'chiruvchi **trimLeft()** va **trimRight()** funksiyalari hozircha «Chrome» va «Firefox»lardagina ishlaydi xolos.

- **valueOf()** — ushbu funksiya satr obyekti uchun «toString» bilan bir xil vazifani bajaradi.

JavaScript tili HTML elementlarini dinamik ravishda boshqarish uchun mo'ljallangani bois asosiy tur hisoblangan satrda teglarni yasash uchun imkoniyatlar qo'shilgan. Ammo amaliyot shuni ko'rsatdiki, bunday usulda teg yaratish samarali emas. Quyida misol bilan keltiriladigan uslublarning barchasi muomaladan chiqarilish jarayonida, shuning uchun ularni qo'llash mutlaqo tavsiya etilmaydi:

```
"JS".anchor("til")      "<a name="til">JS</a>"
"JS".big()              "<big>JS</big>"
"JS".blink()            "<blink>JS</blink>"
"JS".bold()             "<b>JS</b>"
"JS".fixed()            "<tt>JS</tt>"
"JS".fontcolor("red")   "<font color="red">JS</font>"
"JS".fontsize(6)        "<font size="6">JS</font>"
"JS".italics()          "<i>JS</i>"
```

```
"JS".link("yurt.uz")   "<a href="yurt.uz">JS</a>"
"JS".small()           "<small>JS</small>"
"JS".strike()          "<strike>JS</strike>"
"JS".sub()             "<sub>JS</sub>"
"JS".sup()             "<sup>JS</sup>"
```

2.6.3. Jadval (Array)

Jadval sifatidagi obyektni [2.4.6.] uch xil shaklda yaratish mumkin:

```
[element0[, element1, ..., elementN]]
new Array(element0[, element1, ... elementN])
new Array(elementlar soni)
```

Uchinchi usulda jadval qiymatlari «undefined» bo'lib, ko'rsatilgan miqdorda joy ajratiladi xolos.

Jadvalda nechta element mavjudligini «length» xususiyati yordamida bilish mumkin, indeksiga esa to'g'ri qavs bilan murojaat qilinadi:

```
var q = new Array(5); q[1] = 2.1; q[2] = "JavaScript";
q.length  5  [empty, 2.1, "JavaScript", empty × 2]
```

Jadval obyekti ma'lumotlarni saqlash va qayta ishlashda keng qo'llaniladi. Har biri o'zining muhim vazifasiga ega quyidagi tamoyillari mavjud:

- **Array.from(***jadvalsifat***[, *o'girish*])** — jadvalga aylantirish mumkin bo'lgan obyektlardan yangi jadval

yaratadi. Bunda «jadvalsifat» obyektning har bir elementiga «o'girish» qonuniyatini tatbiq etish mumkin. Oddiy satrni ham har bir belgisini mos element sifatida jadvalga o'giraveradi. (Ayrim mobil brauzerlarda ishlamasligi mumkin.) Misol:

```
Array.from("JS")                          ["J","S"]
Array.from([1, 2, 3], elm => 3 * elm)     [3, 6, 9]
Array.from({length: 3}, (elm, i) => i**3) [0, 1, 8]
```

- **Array.isArray(***obyekt***)** — Berilgan «obyekt»ning jadvalligini tekshiradi. Mantiqiy tur qaytaradi.

- **Array.of(***element0***[, ... *elementN*])** — berilgan elementlar asosida jadval yasaydi. Aslida bu tamoyilni «new»ni qo'llab jadval yasashdan farqi yo'q. Ehtimol, shuning uchun ham ba'zi brauzerlar o'ziga tatbiq etishni lozim topmagandir.

Jadvalning o'z qiymatini o'zgartiradigan quyidagi uslublari mavjud:

- **copyWithin(***joyi***[, *boshlang'ich*, *oxirgi*])** — jadvalning o'z elementlarini ko'rsatilgan «joyi» indeksidan boshlab «boshlang'ich» elementidan (*odatiy qiymati: 0*) to «oxirgi» elementigacha (*kirmaydi, ko'rsatilmasa jadval uzunligiga teng*) ketma-ket almashtiradi. (Ayrim mobil brauzerlarda ishlamasligi mumkin.) Misol:

```
[1, 2, 3, 4, 5].copyWithin(3, 1)  [1, 2, 3, 2, 3]
```

- **fill(***qiymat***[, *boshi* = 0[, *oxiri* = this.length]])** — jadval elementlarini «boshi»dan to «oxiri»gacha beril-

gan «qiymat»ga almashtirib chiqadi, «copyWithin» bilan tabiatan bir xil.

- **pop()** — oxirgi elementni qaytaradi va uni jadvaldan o'chiradi. Misol:

```
var raqam = [1, 2, 3, 4, 5, 6]; [1, 2, 3, 4, 5, 6]
raqam.pop();                    6
raqam                           [1, 2, 3, 4, 5]
```

- **push(*element0*[, *elementN*])** — jadval davomidan berilgan elementlarni qo'shib, hosil bo'lgan yangi jadval uzunligini qaytaradi. Misol:

```
var raqam = [1, 2, 3, 4, 5, 6]; [1, 2, 3, 4, 5, 6]
raqam.push(7, 9)                8
raqam                           [1, 2, 3, 4, 5, 6, 7, 9]
```

- **reverse()** — jadvalni teskari tartiblashtiradi. Birinchi elementi oxirgi bo'lib qoladi va aksincha.

- **shift()** — birinchi elementni qaytaradi va uni jadvaldan o'chiradi, «pop»ning aksi.

- **splice(*boshi*, *soni*[, *element0*, ... *elementN*])** — jadval elementlarini «boshi»dan boshlab «soni»ta (berilmagan holda, oxirigacha) o'chiradi va o'rniga keltirilgan (bo'lsa) yangi elementlarni qo'shadi. Bunda indekslar o'zgarishi mumkin, natija sifatida esa o'chirilgan elementlar jadvali qaytariladi. Bunday uslublarda jadvalning qiymatini qaytgan natija bilan chalkashtirmaslik zarur. Misol:

```
var raqam = [1, 2, 3, 4, 5, 6]; [1, 2, 3, 4, 5, 6]
raqam.splice(3, 2, "JS");       [4, 5]
raqam                           [1, 2, 3, "JS", 6]
```


- **sort(*taqqoslash funksiyasi*)** — jadval elementlarini o'sish tartibida joylashtiradi. Bunda elementlarni solishtirish uchun «taqqoslash funksiyasi»dan ham foydalanish mumkin:

```
['80', 9, '700', 40, 1, 5].sort();  [1,40,5,"700","80",9]
['80', 9, '700', 40, 1, 5].sort(    [1,5,9,40,"80","700"]
  function(a, b) {
    return parseInt(a) > parseInt(b);
  }
);
```

- **unshift(*element0*[, ... *elementN*])** — berilgan element(lar)ni boshiga biriktiradi va jadvalning «uzunligi»ni qaytaradi.

Quyidagi uslublar jadvalning asl qiymatini o'zgartirmasdan, uning ustida ishlab, natijasini qaytaradi:

- **concat(*element0*[, ... *elementN*])** — davomidan berilgan element(lar)ni biriktirib yangi jadval hosil qiladi.

- **includes(*izlanuvchi*[, *o'rin*])** — berilgan «o'rin»dan (berilmasa, nol) boshlab «izlanuvchi»ni topsa, «rost», aks holda «yolg'on» qiymat qaytaradi.

- **join(*ajratuvchi* = ",")** — jadval elementlarini bitta satr sifatida biriktiradi. Misol:

```
[1, 2, 3, 4, 5, 6].join()       "1,2,3,4,5,6"
[1, 2, 3, 4, 5].join(' + ')     "1 + 2 + 3 + 4 + 5"
["ECMA", "Script", 7].join("")  "ECMAScript7"
```

▪ **toString(); toLocaleString()** — satrga o'giradi.

▪ **indexOf(*element*[, *joyi* = 0])** — berilgan «joyi»-dan boshlab jadvalning qaysi elementi ko'rsatilganga teng bo'lsa, uning indeksini qaytaradi, topa olmasa, **–1**.

▪ **lastIndexOf(*element*[, *joyi* = *this.length*])** — jadvalda (xuddi indexOf singari) «element» mavjudligini aniqlaydi. Faqat qidirishni orqadan teskari tartibda amalga oshiradi.

Jadvallarning har bir elementiga tatbiq etiladigan amalni qo'llash uchun quyidagi uslublar ishlatiladi:

▪ **forEach(function(*qiymat*, *indeks*, *jadval*){...})**:

```
var j = [2, 5, , 9];                    // 2-elementi yo'q.
j.forEach(function(el, id, ar) {
  ar[id] = 3 * el;
});
j
                                        [6, 15, empty, 27]
```

▪ **entries()** — jadval elementlariga ketma-ket murojaat qilishni qulaylashtiruvchi «davomiylik» yaratadi, «next» uslubiga ega yangi obyekt hosil qiladi. Har safar uni chaqirilganda navbatdagi qiymatni indeksi bilan birga «value» elementi orqali qaytaradi. Oxirgi element qaytarilgandan so'ng «done» xususiyati «true» qiymat qabul qiladi. Misol:

```
e = ["a", "b"].entries();  // Array Iterator {}
e.next();                  {value: [0, "a"], done: false}
e.next();                  {value: [1, "b"], done: false}
e.next();                  {value: undefined, done: true}
```


Belgi (Symbol) obyektining imkoniyatlaridan foydalanib, ushbu tarzdagi murojaatni quyidagicha ta'minlash ham mumkin:

```
e = ["a", "b"].[Symbol.iterator]();  // Array Iterator {}
```

▪ **every(function(*qiymat*, *indeks*, *jadval*){...})** — jadvalning har bir elementi funksiyada keltirilgan shartni bajarayotganini tekshiradi. Agar biror element shartni buzsa, «yolg'on» qiymat qaytaradi. Elementlarining barchasini o'ndan kichiklikka tekshirishga misol:

```
[2, 4, 8, 7].every(el => el < 9)                    true
```

Amaliyotda ushbu ajoyib uslubni satrdagi har bir element uchun qo'llash ko'p zarur bo'ladi. Masalan, satrning har bir elementi raqamdan iboratligini tekshirish kerak bo'lsa (yuqorida tanishgan bilimlarga asoslangan holda), uni jadvalga aylantirib:

```
Array.from("2487").every(el => el <= 9 && el >= 0)
```

tarzda aniqlash mumkin. Ammo shart murakkab bo'lgan hollarda hisoblashni funksiya ichida bajarish ma'qul bo'lganligi sababli, quyidagicha imkoniyatdan foydalanish tavsiya etiladi:

```
Array.prototype.every.call("2487",
  function(ch) { return ch >= '0' && ch <= '9'; }
);
```

▪ **some(function(*qiymat*, *indeks*, *jadval*) {...})** — jadvalning, hech qursa, bitta elementi keltirilgan shartni bajarsa «rost» qiymat qaytaradi. Ishlashi «every» uslubi bilan aynan bir xil.

- **filter(function(*qiymat*, *indeks*, *jadval*) {...})** — funksiya qaytaradigan qiymatiga qarab yangi jadval hosil qiladi, uning barcha elementlari keltirilgan shartni qanoatlantiradigan bo'ladi. Misol:

```
[12,5,8,99].filter(function(v) { return v > 10 }); [12,99]
```

- **find(function(*qiymat*, *indeks*, *jadval*) {...})** — funksiyadagi shartni qanoatlantirgan birinchi elementni (topilmasa, «undefined») qaytaradi. Misol:

```
[12,5,8,99,43].find(function(v) { return v>40; }) 99
```

- **findIndex(function(*qiymat*, *indeks*, *jadval*){...})** — «find» singari ishlaydi, faqat topilgan elementnimas, uning indeksini qaytaradi, topa olmasa natija: **–1**.

- **keys()** — xuddi «entries» kabi ishlaydi, faqat «value»da jadval emas, indeksning o'zi qaytariladi.

- **map(function(*qiymat*, *indeks*, *jadval*) {...})** — funksiya qaytaradigan qiymatlar asosida yangi jadval hosil qiladi. Misol:

```
[1,2,3,4].map(function(v) { return v ** 2 }) [1,4,9,16]
```

- **reduce(function(*avval*, *hozir*, *indeks*, *jadval*) {...}, *bosh*)** — murakkab takrorlanuvchi hisob-kitoblarni bajarish uchun qo'llaniladi. Agar «bosh» qiymat berilgan bo'lsa, «jadval»ning har bir elementi uchun funksiyada yozilgan amallar bajariladi. Aks holda ikkinchi (indeks bo'yicha birinchi) elementdan boshlab ketma-ket davom etadi. Bunda ayni paytdagi indeks va qiymatdan tashqari, natijaning avvalgi qiymatni ham bevosita ifodada

ishtirok etishini ta'minlash mumkin. E'tibor berish kerakki, «avval» bu jadvaldagi «hozir»dan oldingi qiymat emas, funksiyaning avvalgi qaytargan natijasi, ilk bor «bosh» qiymatga teng bo'ladi. Farqni quyidagi misolda ko'rish mumkin:

```
[20, 1, 25, 3, 40, 11].reduce(                          200
   function(avval, ayni, index, jadval) {
      return avval + ayni;
   },
   100
);
[20, 1, 25, 3, 40, 11].reduce(                           51
   function(avval, ayni, index, jadval) {
      return jadval[index-1] + ayni;
   },
   100
);
```

- **reduceRight** — xuddi «reduce» kabi bo'lib, faqat jadvalning oxirgi elementidan boshiga qarab hisob-kitoblar bajariladi.

2.6.4. To'plam (Set)

To'plam qiymatlarning majmuasi. Bir qarashda jadval(array)ga juda o'xshab ketadi, lekin unda qiymatlar takroriy uchramaydi. Elementar matematikadan bizga ma'lum bo'lgan «natural sonlar to'plami»ni tasavvur etadigan bo'lsak, unda sanoqda ishlatiladigan sonlar

tushuniladi. Bir necha marta «bir» yoki «ikki» deb sanalmaydi, bunda mantiq yo'q. Shunday ekan, to'plamga ham har bir elementi takrorlanmas (unikal) bo'lgan jadval sifatida qarashimiz mumkin. Sintaksisi:

```
new Set([sanoqli]);
```

Bunda «sanoqli» obyekt sifatida jadval yoki satr berish mumkin. To'plam quyidagi uslublarga ega:

- **add(*qiymat*)** — «qiymat»li element qo'shish.
- **clear()** — barcha elementlarini o'chirish.
- **delete(*qiymat*)** — «qiymat»li elementi mavjud bo'lsa, o'chirib «rost» qaytaradi, aks holda «yolg'on».
- **entries()** — jadval [2.6.3.] yoki xaritadagi [2.6.9.] singari, «**keys**» va «**values**» uslublarining ham faoliyati aynan bir xil.
- **forEach()** — jadvaldagi singari [2.6.3.].
- **has(*qiymat*)** — to'plam tarkibida «qiymat» borligini tekshiradi, «rost» yoki «yolg'on» qaytaradi.

To'plamning jadvaldan keskin farqi shundaki, unda indeks mavjud emas. Takrorlanmaydigan elementlari shunday joylashtirib boriladiki, «has» uslubi jadvaldagi indeksga bog'langan izlashdan ancha tez ishlaydi.

To'plam nechta elementdan iboratligini aniqlash uchun «**size**» xususiyatidan foydalanish mumkin.

2.6.5. Mantiqiy tur (Boolean)

Mantiqiy turdagi obyekt quyidagicha yaratiladi:

```
new Boolean([qiymat])
```

Bunda «qiymat» berilmasa yoki `0`, `-0`, `null`, `NaN`, `false`, bo'sh satr (`""`) va `undefined`lardan biri berilsa, obyekt «yolg'on» natijaga ega bo'ladi, qolgan hollarda «rost» qiymat qabul qiladi.

Dasturlash tilida mantiqiy qiymatlar bilan ishlaydigan «**if**» operatori mavjud [3.1.]. Shu o'rinda uning sintaksisi bilan tanishib o'tamiz:

```
if (shart)  { shart bajarilganda ishlaydigan amallar }
else        { shart bajarilmasa ishlaydigan amallar }
```

Agar «amallar» yagona bo'lsa, uni gulli qavsga olish shart emas. Shart bajarilmasa ishlaydigan amallar mavjud bo'lmasa, «else» qismini (ikkinchi qatorni) yozmasa ham bo'ladi. Aksincha holatda, shart bajarilganda ishlashi lozim amallar bo'lmasa, bo'sh operatordan foydalanish mumkin (shunchaki nuqtali vergul qo'yiladi), unda kod quyidagicha ko'rinishda bo'ladi:

```
if (shart) ; else { shart yolg'on bo'lsa ishlaydigan amallar }
```

Aslida bu xato emas, lekin xunuk kod hisoblanadi, undan ko'ra **inkor**dan foydalanish ma'qul [2.5.(16)]:

```
if (!shart) { shart rost bo'lmasa ishlaydigan amallar }
```

Mantiqiy obyekt berilgan qiymatga qarab natijasini qanday aniqlasa, «shart»ning bajarilish-bajarilmasligi ham xuddi shu tarzda hisoblanadi. Ammo mantiqiy tur-

ning o'zi bilan u qabul qilgan qiymatni chalkashtirib yubormaslik kerak. Quyidagi ikki misolda ularning farqini yaqqol ko'rish mumkin:

```
var b = new Boolean(false);    // Boolean { false }
if (b) alert(true);            Ogohlantirish chiqadi
if (b.valueOf()) alert(true);  Ogohlantirish chiqmaydi
```

2.6.6. Sana (Date)

Sana-vaqt bilan ishlovchi obyektni quyidagi usullardan biri yordamida yaratish mumkin:

```
new Date()
new Date(son)
new Date(satr)
new Date(yil,oy[,kun[,soat[,daqiqa[,soniya[,millisoniya]]]]])
```

Agar omil berilmasa, joriy vaqtni qabul qiladi, «son» sifatida 1970-yil 1-yanvar 00:00:00 umumjahon muvofiqlashtirilgan vaqti — UTC (Universal Time Coordinated)dan keyin o'tgan millisoniya hisobidagi qiymat beriladi («**Unix davri**»). Vaqtni ko'rsatuvchi «satr» sifatida maxsus ko'rinishdagi matn berish mumkin. Asosan inglizcha so'zlar va andozalar hisobga olingan. Yoki sodda qilib, oxirgi qatorda keltirilgan tartib bo'yicha vaqtni aniqlovchi sonlarni quyidagicha tarzda berish ham mumkin: «2017-12-19 20:36:12.123».

Oxirgi sintaksisda «oy» sifatida tartib raqamini bildiruvchi 0 dan 11 gacha son beriladi. Agar «kun» kiritilmasa, odatiy qiymati 1 bo'ladi, qolgan beriluvchilar esa yo'q bo'lganda, 0 sifatida qabul qilinadi.

Bunday usulda e'lon qilishning yana bir qulay tarafi shundaki, masalan, 29-fevral sanasini yaratishda, u mavjud bo'lmasa, 1-mart sifatida namoyon bo'laveradi.

E'lon qilinayotganda «new» yozilmasa, vaqt emas, satr sifatida natija qaytariladi.

Date obyektining «length» xususiyati doim 7 qaytaradi, bu uning qabul qiladigan omillari sonini anglatadi.

Date quyidagi tamoyillarga ega:

- **Date.now()** — «Unix davri» ichida o'tgan millisoniyani (1970-01-01 00:00:00.000 UTCdan so'ng) qaytaradi.
- **Date.parse(*satr*)** — ISO 8601 standarti bo'yicha berilgan «satr»ni vaqtga aylantiradi, «new Date(*satr*)» bilan bir xil vazifani bajaradi.
- **Date.UTC(*yil*,*oy*[,*kun*[,*soat*[,*daqiqa*[,*soniya*[,*ms*]]]]])** — to'rtinchi usulda vaqt yaratish bilan teng kuchli.

Date obyekti quyidagi o'ziga xos uslublarga ega:

- **getTimezoneOffset()** — daqiqa hisobida GMTga nisbatan tafovut. Toshkent vaqti uchun: **–300**;
- **getTime()** — «Unix davri»dagi millisoniya, manfiy qiymat 1970-yilgacha vaqtni anglatadi (Universal uslub «valueOf» bilan bir xil natija qaytaradi);

Quyidagi uslublar mahalliy vaqtga nisbatan qiymatlarni qaytaradi:

- **getDate()** — kun qiymatini (1, 2, ..., 31);
- **getDay()** — hafta kuni (0, 1, 2, 3, 4, 5, 6);
- **getFullYear()** — yil (4 xonali son);
- **getHours()** — soat (0, 1, ..., 23);
- **getMilliseconds()** — millisoniya (0, 1, ..., 999);
- **getMinutes()** — daqiqa (0, 1, ..., 59);
- **getMonth()** — oy (0, 1, ..., 11);
- **getSeconds()** — soniya (0, 1, ..., 59);

Mahalliy vaqtga emas, UTCga nisbatan qiymat olish uchun yuqoridagi nomlardagi «get»dan so'ng «UTC» qo'shib yozish kifoya. Berilgan vaqtni ayni bir omili bo'yicha o'zgartirish uchun yuqoridagi kvadrat bilan belgilangan 8 ta uslubni va ularga «UTC» qo'shib hisoblaganda hosil bo'ladigan jami 16 ta funksiya nomlaridagi «get» so'zi o'rniga «set» yozib («getDay» o'rniga «setDay» nazarda tutilyapti, aniq sananing hafta kunini o'zgartirish mumkinmas!), qiymat berish kifoya:

```
var d = new Date();          // hozirgi vaqt
d.setTime(1585866983945); Fri Apr 03 2020 03:35:30 GMT+0500
```

Quyidagi uslublar Date obyekti qiymatini satr shaklida o'zgartirib qaytaradi:

d.toString()	"Fri Apr 03 2020 03:35:30 GMT+0500"
d.toDateString()	"Fri Apr 03 2020"
d.toISOString()	"2020-03-04T03:35:30.810Z"
d.toJSON()	"2020-03-04T03:35:30.810Z"
d.toGMTString()	"Fri, 03 Apr 2020 03:35:30 GMT"
d.toLocaleDateString()	"04.03.2020"
d.toLocaleString()	"04.03.2020, 03:35:30"
d.toLocaleTimeString()	"03:35:30"
d.toTimeString()	"03:35:30 GMT+0500 (+05)"
d.toUTCString()	"Tue, 19 Dec 2017 18:37:00 GMT"

2.6.7. Matematika (Math)

JavaScriptda Math funksional obyekt hisoblanmaydi. U faqat matematik funksiyalarni va o'zgarmaslar qiymatini saqlaydi. Shuning uchun «new» yordamida yangi o'zgaruvchi yaratilmaydi. Faqatgina aniq sonni yoki ma'lum argument bo'yicha mavjud funksiyaning qiymatini olish mumkin. Ya'ni faqatgina tamoyil va xossalarga ega, uslub va xususiyati esa mavjud emas.

- **Math.E** — natural logarifm asosidagi son. Uni Eyler yoki Neper nomi bilan ham ataladi. Qiymati taxminan ≈**2.718** ga teng. Matematikada uni «***e***» harfi bilan belgilanishi va aniq qiymati quyidagicha formula asosida topilishidan xabardormiz:

$$\lim_{x\to\infty}\left(1+\frac{1}{x}\right)^{x}$$

- **Math.exp(x)** — «***e***»ning darajasi: e^x .

▪ **Math.LN2** — ikki sonining natural logarifmi, qiymati taxminan **≈0.693** ga teng: $\ln 2 = \log_e 2$

▪ **Math.LN10** — o'n sonining natural logarifmi, qiymati: **ln10 ≈ 2.3**

▪ **Math.LOG2E** — ikki asosga ko'ra «Eyler» logarifmi, qiymati: $\log_2 e \approx 1.442$

▪ **Math.LOG10E** — o'n asosga ko'ra «Eyler» logarifmi: $\log_{10} e \approx 0.434$

▪ **Math.PI** — «pi» (π) soni. Ixtiyoriy aylana uzunligini uning diametriga nisbati: **≈3.14159**

▪ **Math.SQRT1_2** — yarimning (1/2) ildiz ostidan chiqqan qiymati: $\sqrt{0.5} \approx 0.7071$

▪ **Math.SQRT2** — ikkining ildiz ostidan chiqqan qiymati: $\sqrt{2} \approx 1.414$

Quyidagi tamoyillarning trigonometriyaga oidlari faqat radian qiymat bilan ishlaydi. Gradusni radianga aylantirish uchun «**Math.PI / 180**»ga ko'paytirish talab etiladi. Aksincha, javob gradusda lozim bo'lsa, bo'lishga to'g'ri keladi. Beriladigan qiymat ko'rsatilgan oraliqqa tegishli bo'lmasa, «NaN» qaytaradi.

▪ **Math.abs(*son*)** — «son»ning moduli, absolyut qiymati, ya'ni doim musbat holda qaytaradi. Agar son satr sifatida berilgan bo'lsa, imkon qadar o'girib oladi. Iloji bo'lmagan holda «NaN» qaytaradi.

▪ **Math.acos(x)** — «x»ning arkkosinusi, **x ∈ [−1; 1]**.

▪ **Math.acosh(x)** — «x»ning giperbolik arkkosinusi, **x ≥ 1**. Matematik ifodasi: $\operatorname{arcosh} x = \ln\left(x + \sqrt{x^2 - 1}\right)$. (∅)

▪ **Math.asin(x)**— «x»ning arksinusi, **x ∈ [−π/2; π/2]**.

▪ **Math.asinh(x)** — «x»ning giperbolik arksinusi. Matematik ifodasi: $\operatorname{arsinh} x = \ln\left(x + \sqrt{x^2 + 1}\right)$. (∅)

▪ **Math.atan(x)** — arktangens, **x ∈ [−π/2; π/2]**.

▪ **Math.atan2(y, x)** — berilgan (x, y) nuqta va x o'qining musbat qismi orasidagi soat miliga qarshi yo'nalishda olingan burchakning arktangensi.

▪ **Math.cbrt(x)** — «x»ning uchinchi ildiz ostidan chiqadigan qiymati. Matematik ifodasi: $\sqrt[3]{x}$. (∅)

▪ **Math.ceil(*son*)** — berilgan «son»dan katta yoki teng, eng kichik butun sonni qaytaradi (yuqoriga yaxlitlaydi).

▪ **Math.cos(*burchak*)** — «burchak»ning kosinusi.

▪ **Math.cosh(*son*)** — «son»ning giperbolik kosinusi. Matematik ifodasi: $\cosh x = (e^x + e^{-x})/2$. (∅)

▪ **Math.expm1(x)** — Matematik ifodasi: $e^x - 1$. (∅)

▪ **Math.floor(*son*)** — berilgan «son»dan kichik yoki teng eng katta butun son (quyiga yaxlitlaydi).

▪ **Math.hypot(x1[, x2[, …]])** — berilgan sonlar kvadratlari yig'indisining ildizdan chiqqan qiymati. (∅) Matematik ifodasi:

$$\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}\ .$$

- **Math.log(x)** — berilgan sonning ($x > 0$) natural logarifmi. Matematik ifodasi: $y = \log_e x = \ln x$. $e^y = x$.

- **Math.log10(x)** — sonning ($x > 0$) o'n asosga ko'ra logarifmi. Matematik ifodasi: $y = \log_{10} x$. $10^y = x$.

- **Math.log1p(x)** — birga orttirilgan sonning natural logarifmi, $x > -1$. Matematik ifodasi: $\ln(x + 1)$. (∅)

- **Math.log2(x)** — sonning ($x > 0$) ikki asosga ko'ra logarifmi. Matematik ifodasi: $y = \log_2 x$. $2^y = x$. (∅)

- **Math.max([x1, x2[, ...]])** — berilgan sonlardan eng kattasini qaytaradi. Berilmasa: «–Infinity».

- **Math.min([x1, x2[, ...]])** — berilgan sonlardan eng kichigini qaytaradi. Berilmasa, «Infinity» qaytaradi.

- **Math.pow(x, n)** — darajaga oshirish: x^n.

- **Math.random()** – har safar **[0; 1)** oraliqdagi har xil qiymatdagi tasodifiy haqiqiy sonni qaytaradi.

- **Math.round(*son*)** — «son»ning qiymati bo'yicha yaxlitlaydi. Agar kasr qismi yarimdan kichik bo'lsa, uni tushirib qoldiradi, aks holda butun qismidan aynan birga katta bo'lgan sonni qaytaradi. Misol:

```
Math.round(1.4)   1 Math.floor(1.9)   1 Math.ceil(1.9)   2
Math.round(-1.4) -1 Math.floor(-1.9) -2 Math.ceil(-1.9) -1
Math.round(-1.6) -2 Math.floor(-1.1) -2 Math.ceil(-1.1) -1
Math.round(1.5)   2 Math.floor(1.5)   1 Math.ceil(1.5)   2
```

- **Math.sin(x)** — «x»ning sinus qiymati: $\sin x$.

- **Math.sinh(x)** — «x»ning giperbolik sinusi. Matematik ifodasi: $\sinh x = (e^x - e^{-x})/2$. (∅)

- **Math.sign(*son*)** — «son»ning ishorasini aniqlaydi. Musbat son berilsa, **1**; manfiy son berilsa, **–1**; nol berilsa **0** qaytaradi. Agar «son»ni number turiga aylantirish mumkin bo'lmasa, **NaN** qabul qiladi. (∅)

- **Math.sqrt(x)** — berilgan musbat sonning ($x \geq 0$) ildiz ostidan chiqadigan qiymati: $\sqrt{x}$.

- **Math.tan(*son*)** — «son»ning tangensini qaytaradi.

- **Math.tanh(x)** — «x»ning giperbolik tangensini qaytaradi. (∅) Matematik ifodasi:

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

- **Math.trunc(*son*)** — «son»ning kasr qismini tashlab yuborib, faqat butun qismini qaytaradi, yaxlitlash amali bajarilmaydi. (∅)

Yuqorida keltirilgan uslublarning barchasi mos ravishda matematik funksiyani bajaradi. (O'quvchiga qaysidir funksiyalar tushunarsiz bo'lsa, trigonometriya mavzularini takroran o'qib olishni maslahat beramiz — *J.A.*) Uslub sharhida «(∅)» belgisi uchragan bo'lsa, bu Internet Explorer va ayrim mobil brauzerlarda qo'llanmasligi mumkinligini anglatadi.

2.6.8. Obyekt (Object)

Yuqorida bir nechta turlarga mansub obyektlar yaratish bilan tanishdik. Xuddi shularni yoki ularning qolipiga tushmaydigan, aniq standartga ega bo'lmagan ma'lumotlarni ham yagona obyekt sifatida yaratish mumkin. Buning uchun quyidagicha sintaksis qo'llaniladi:

```
new Object([qiymat])
```

Quyidagicha ikki xil e'lon qilish, bir xil natija beradi:

```
new Boolean(true)     new Object(Boolean(true))
```

Aslida obyektni «Object» kalit so'zni ishlatmasdan ham sodda ko'rinishda e'lon qilish mumkin [2.4.6.].

Har bir obyekt uchun tizim unga «__proto__» xususiyatini biriktiradi. JavaScriptning avvalgi talqinlarida foydalanuvchiga uni qo'llashga imkoniyat berilmagan. Hozirda esa ushbu xususiyatga qiymat berib, undan foydalanish mumkin. Obyektning elementiga murojaat qilinayotganda, avval o'zidagi nomlanishlarni tekshirib chiqadi. Topa olmasa, «__proto__» xususiyatiga o'tadi:

```
var a = { b:2, c:"U" };     { b:2, c:"U" }
a.d                         undefined
a.__proto__ = { d: 9 };     { d: 9 }
a                           { b:2, c:"U" }
a.d                         9
```

Aslida «a» obyektining «d» elementi yo'q bo'lsa ham, «__proto__»da mavjud bo'lgani uchun murojaat qilinganda kiritilgan qiymatni qaytarmoqda.

Ushbu yashirin element amaliyotda qanday naf keltirishi mumkin, oshkor ishlataverish ham mumkinku, zero uning maxfiyligi yo'q ekan, degan o'rinli savol tug'ilishi tabiiy. Aslida «__proto__» xususiyati o'zgaruvchi taalluqli bo'lgan obyektda mavjud bo'lgan «**prototype**» xos obyektiga murojaat qiladi. Uning yordamida esa ixtiyoriy mavjud biriktirilgan obyektlarga o'zida mavjud bo'lmagan istalgancha yangi xususiyat yoki uslublar qo'shish imkoniyatini yaratadi. Bundan foydalanib ixtiyoriy qiymat yoki o'zgaruvchiga istalgan nomdagi alomat biriktira olishga sazovor bo'lamiz.

Masalan, sana bilan ishlaydigan «Date» obyekti boshqa tillarda mavjud qulay funksiyalarga ega emas. Lekin lozim bo'lganda bunday imkoniyatni o'zimiz qo'shishimiz mumkin. O'zgarmas sifatida «asr» xususiyatini qo'shish uchun bir marta quyidagicha yozilsa:

```
Date.prototype.asr = 21;
```

barcha «**new Date()**» orqali yaratilgan o'zgaruvchilar uchun ushbu element qiymatidan foydalanish mumkin bo'ladi. Misol:

```
b = new Date();
b.asr                    21
```

Ammo aksariyat hollarda vaqtning qiymatiga qarab o'zgaruvchan uslub qo'shish talab etiladi. Aytaylik, o'sha vaqt bo'yicha sana. Bunday hollarda funksiya imkoniyatidan foydalaniladi. Misol:

```
Date.prototype.sana = function() {
   let m = 1 + this.getMonth();
```

```
    let y  = this.getFullYear();
    if (m < 10) m = "0" + m;
    return this.getDate() + "." + m + "." + y;
}
```

Bundan keyin ixtiyoriy Date obyektiga mansub o'zgaruvchilarning o'zimiz ko'nikkan shakldagi sanasini olishimiz mumkin. Misol:

```
b.sana()                "24.05.2018"
```

Mavjud obyektlar uchun qo'shimcha ravishda qulay xususiyatlarni ushbu usulda qo'shishni «**timsollash**» deb yuritamiz. Hozirda ommabop obyektlar uchun tayyor timsollar yaratilgan kutubxonalar juda ham ko'p.

Obyekt quyidagi o'ziga xos tamoyillarga ega:

▪ **Object.assign(*asos*, *qo'shiluvchi obyektlar*)** — obyektlarni umumlashtirish yoki ulardan nusxa olish uchun xizmat qiladi. Agar «asos» sifatida mavjud obyekt ko'rsatilsa, uning qiymati ham o'zgarib ketadi. O'zgarmasligi uchun bo'sh obyekt berilishi kerak:

```
var o1 = { a: 1 };                      { a: 1 }
var o2 = { b: 2 };                      { b: 2 }
var o3 = { c: 3 };                      { c: 3 }
Object.assign({}, o1, o2, o3);          { a: 1, b: 2, c: 3 }
o1                                      { a: 1 }
Object.assign(o1, o2, o3);              { a: 1, b: 2, c: 3 }
o1                                      { a: 1, b: 2, c: 3 }
```

Qo'shiluvchi obyektlar sifatida satr berilganda, uning har bir belgisi bo'yicha `String` obyekti shaklida qabul qiladi. Boshqa turdagi qiymatlar e'tiborsiz qoldiriladi.

▪ **Object.create(*timsol*, *omillar*)** — berilgan «timsol» asosida «omillar» bo'yicha yangi obyekt yaratadi. Bunda omilning har bir elementi **value**, **writable**, **enumerable** hamda **configurable** nomlanishlariga ega obyekt ko'rinishida e'lon qilinadi. Ya'ni omil sifatida, misol uchun **{d: 8}** shaklida qiymat berish xato, maxsus alomatlar yordamida **{d: {value: 8}}** ko'rinishida berilishi shart. Qolgan uchta alomatga mantiqiy qiymat beriladi, ko'rsatilmagan holda o'zi «yolg'on» (`false`) qiymat qabul qiladi. Misol:

```
o = { a: 1, b: 2};                          {a: 1, b: 2}
n = Object.create(o, { c:{value: 3} });     {c: 3}
n.b                                         2
n.__proto__                                 {a: 1, b: 2}
n.c                                         3
n.c = 40
n.c                                         3
```

Ushbu misolda yangi yaratilgan obyektga omil sifatida qo'shilgan elementga berilgan qiymatni qabul qilmadi, sababi «**writable**»ga «rost» qiymat berilmagan edi. Bunday imkoniyat faqat o'qish uchun qiymatlarga ega obyektlar bilan ishlashda qo'l keladi. Amaliyotda shunday obyektlarga ehtiyoj tug'iladiki, ulardagi ba'zi elementlarining qiymati dasturlash jarayonida o'zgaradi, ba'zilariniki esa o'zgarmasdan saqlanishi shart bo'ladi. Bunday hollarda obyektning aynan «create» xususiyati asqotadi. Qiymatini o'zgartirish mumkin emas, ammo uni o'chirib yuborish imkoniyati mavjud, agar uning «**configurable**» omiliga «rost» qiymat berilgan bo'lsa, albatta. Indekslarni tartiblab, obyektni takrorlash ope-

ratorlarida [3.2.] qo'llash imkoniyatini yaratish uchun «**enumerable**»ga «rost» qiymat berish kerak.

«Timsol» sifatida beriladigan obyekt mavjud bo'lmasa, shunchaki «null» yozish kifoya.

▪ **Object.defineProperty(***obyekt, alomat, omillar***)** — «obyekt»ga ko'rsatilgan «omillar» asosida berilgan «alomat» nomlanishidagi element qo'shadi. Masalan, «**o**» bo'sh obyektga «**q**» nomlanishli **9** qiymatga ega element qo'shish uchun oddiy ko'rinishda «**o.q = 9**» sifatida yoziladi. Lekin ushbu xususiyat quyidagicha ko'rinishda yozishni talab etadi:

```
Object.defineProperty(o, 'q', { value: 9 });
```

Bir qarashda bu ortiqcha murakkablik tug'dirayotganga o'xshaydi. Sababi, «value»dan tashqari yuqorida tanishib o'tganimizday **configurable, enumerable** va **writable** kabi omillarga ham ega. Aslida esa bular taqdim etadigan imkoniyatlardan tashqari «**set**» va «**get**» funksiyalari ham mavjud-ki, elementning qiymati o'zgartirilayotganda yoki unga murojaat bo'lganda ushbu qo'shimcha funksiyalar ishga tushiriladi. Masalan, mavjud obyektning «width» elementining qiymati o'zgarganda veb-sahifamizdagi ma'lum maydonning kengligi o'zgarsin yoki qiymati olinayotganda «BODY» tegining haqiqiy eni necha piksel ekanligini ko'rsatsin, deylik. Obyektning bunday qulayliklari veb dasturlashda JavaScriptning beqiyos imkoniyatlarini ochib beradi. Berilgan qiymatni tanlash operatori [3.1.] yordamida tekshirib, qabul qilish yoki inkor etish mumkin bo'lgan bir misolni keltiramiz:


```
var myAlign = 'center', joyi = {};
Object.defineProperty(joyi, 'align', {
  get: function() { return myAlign; },
  set: function(value) {
    if (value == 'right')
      alert('O'ngga tekislanmaydi!');
    else myAlign = value;
  }
});
joyi.align                          "center"
joyi.align = 'left'                 "left"
joyi.align = 'right'                // alert!
joyi.align                          "left"
```

▪ **Object.defineProperties(***obyekt,elementlar***)** — «obyekt»ga ko'rsatilgan shartlar asosidagi elementlarni biriktiradi. Bunda «elementlar» yagona obyekt sifatida beriladi. Undagi har bir nomlanish alohida element ko'rinishida qo'shiladi. Misol:

```
Object.defineProperties(o, {ye1:{...}, ye2:{...}});
```

▪ **Object.preventExtensions(***obyekt***)** — «obyekt»ga yangi element qo'shishni ta'qiqlaydi.

▪ **Object.freeze(***obyekt***)** — «obyekt»ni muzlatadi. Ya'ni uning barcha elementlarining qiymatini o'zgartirish yoki o'chirish imkoniyatidan mahrum etadi. Obyekt to'laligicha faqat o'qish tartibiga o'tadi.

▪ **Object.getOwnPropertyDescriptor(***obyekt, indeks***)** — «obyekt»ning «nomlanish»li elementining qanday omillarga egaligini ko'rsatadi. Misol:

```
var obj = { d: 7 };
Object.getOwnPropertyDescriptor(obj, 'd')
```
{ value: 7, writable: true, enumerable: true, configurable: true }

- **Object.getPrototypeOf(*obyekt*)** — agar «obyekt» boshqa obyektning timsoli sifatida yasalgan bo'lsa, namunasini bergan asl obyektni qaytaradi.

- **Object.seal(*obyekt*)** — «obyekt»ning elementlarini o'chirishni taqiqlaydi.

- **Object.is(*ko'rsatkich1*, *ko'rsatkich2*)** — ko'rsatkichlar aynan bitta obyektni anglatayotganligini tekshiradi. Ixtiyoriy obyekt, shu jumladan, qiymatga murojaat qilar ekanmiz, aslida xotirada u saqlangan manzilga ko'rsatkichni yozamiz. Bunda obyekt ayni o'shami yoki ikki ko'rsatkich aslida yagona obyektni anglatadimi degan nuqtayi nazar bilan taqqoslanadi. Bu yerda qiymat bo'yicha solishtirilmaydi. Ehtimol shuning uchun ham ko'pchilikka tushunarsiz tuyular, shuning uchun ham ayrim brauzerlar qo'llamasligi kuzatiladi. Lekin amaliy dasturlashda bir necha identifakator orqali e'lon qilingan obyektlar aslida xotirada yagona joyda saqlanayotganligini aniqlash juda muhim muammolarni hal etishda yordam beradi.

- **Object.isExtensible(*obyekt*)** — «obyekt»ga yangi element qo'shishga ruxsat berilganligini tekshiradi. Muzlatilgan yoki faqat o'qish uchun yaratilgan obyekt berilganda «yolg'on» qiymat qaytaradi.

- **Object.isFrozen(*obyekt*)** — «obyekt»ning faqat muzlatilganligini tekshiradi.

- **Object.isSealed(*obyekt*)** — «obyekt»ni muhrlanganligini tekshiradi. Element qo'shish yoki mavjud elementni o'chirish imkoniyati cheklangan obyektlar «muhrlangan» hisoblanadi, ammo elementlarining qiymatini o'zgartirish mumkin.

- **Object.getOwnPropertyNames(*obyekt*)** — «obyekt»-dagi barcha elementlarning indeks nomlanishlarini jadval sifatida qaytaradi.

- **Object.keys(*obyekt*)** — «obyekt»ning sanoqli indeks nomlanishlarini jadval sifatida qaytaradi. Bir qarashda uning «getOwnPropertyNames»dan farqi yo'qday, ammo u sanalmaydigan nomlanishlarni ham qaytaradi. Quyidagi misolda bu farq yaqqol ko'rsatilgan:

```
var a = ['a', 'b', 'c'];
Object.keys(a)                  // ["0","1","2"]
Object.getOwnPropertyNames(a)   // ["0","1","2","length"]
```

- **Object.setPrototypeOf(*obyekt*, *timsol*)** — «obyekt»ning yangi «timsol»ini yaratadi.

2.6.9. Xarita (Map)

Xarita aslida **uyushgan jadval**, ya'ni indekslari faqat 0, 1, 2, ... bo'lgan qiymatlar majmuasi emas, balki berilgan identifikatorlar bo'yicha aniqlanadigan obyektning bir ko'rinishini yaratadi. Sintaksisi:

new Map([*maxsus jadval*])

Bunda «maxsus jadval» sifatida «null» kelishi yoki [["*indeks1*","*qiymat1*"], ["*indeks2*","*qiymat2*"], ...] shaklidagi jadval berilishi mumkin.

Mavjud jadval asosida xarita yaratilganda u oddiy obyektdan farqli o'laroq quyidagi ko'rinishda bo'ladi:

{ "*indeks1*" => "*qiymat1*", "*indeks2*" => "*qiymat2*", ... }

Aniqroq tasavvur hosil qilish uchun ushbu misolni ko'rib chiqamiz:

```
var m = new Map([['a', 1], [10, 82], ['joy', '89']]);
m              {"a" => 1, 10 => 82, "joy" => "89"}
m['a']         undefined
m[10]          undefined
```

Bundan ko'rinadiki, xaritaning elementlariga xuddi obyektdagi singari indeksi bilan murojaat qilish mumkin emas. Elementining qiymatini olish uchun quyidagi uslub yordamida indeksi bo'yicha chaqiriladi:

```
m.get('a')   1
m.get(10)    82
m.get('10')  undefined
```

Indeks qanday turda berilgan bo'lsa, shunday murojaat qilish shart, unga satr sifatida qarash xatodir.

Xaritaga qo'shimcha qiymat biriktirish uchun quyidagi shakldagi maxsus uslubdan foydalaniladi:

set(*indeks***, *qiymat*)**

Xarita qiymat berish va olish imkoniyatlaridan tashqari quyidagi uslublarga ham ega:

- **clear()** — barcha elementlarini o'chiradi.

- **delete(***indeks***)** — xaritadan «indeks»li elementni o'chiradi. Muvaffaqiyatli bajarilsa, «rost», bunday elementi mavjud bo'lmasa, «yolg'on» qiymat qaytaradi.

- **has(***indeks***)** — xaritada «indeks»li element mavjudligini tekshirib, mantiqiy qiymat qaytaradi.

- **size** — elementlari sonini qaytaradi.

- **entrsies()** — xarita elementlarini sanoqli jadval sifatida tartiblaydi. Har bir elementini keyingisi-keyingisi deb, birma-bir takroriy ravishda chaqirish mumkin bo'ladi [2.5.2.]:

```
var m = new Map([['a', 1], [10, 82], ['joy', '89']]);
var k = m.entries();      MapIterator {…}
k.next().value            ["a", 1]
k.next().value            [10, 82]
k.next().value            ["joy", "89"]
k.next().value            undefined
```

- **forEach(***funksiya***[, *joriy obyekt*])** — xaritadagi har bir elementining ustida ko'rsatilgan «funksiya»ni bajaradi. Bunda ko'rsatilishi shart bo'lmagan «joriy obyekt» odatdagi «**this**»ning qiymatini qabul qiladi. Quyida xaritaning barcha elementini navbatma-navbat ko'rsatadigan ikki misol bilan tanishamiz:

```
var m = new Map([['a', 1], [10, 82], ['joy', '89']]);
m.forEach(alert);                1        82        89
function logMapElements(value, key, map) {
   console.log('"' + key + '" => "' + value + '"');
}
```

```
m.forEach(logMapElements);  "a" => "1"
                            "10" => "82"
                            "joy" => "89"
```

- **keys()** — xarita elementlarining indekslarini sanoqli tarzda tartiblaydi (**entries** singari).

- **values()** — xarita elementlarining qiymatlarini sanoqli tarzda tartiblaydi (**entries** singari).

2.6.10. JSON

JavaScript tilida obyekt qanday ko'rinishda yozilishi haqida to'liq tasavvurga ega bo'ldik. Umuman olganda, obyektni bir qancha ma'lumotlar jamlanmasi sifatida qarash mumkin. Shunday nuqtayi nazar bilan birinchi kitobda tanishgan XML shaklidagi ixtiyoriy ma'lumotlar majmuasini sodda, ixcham, qulay va tushunarli tarzda yagona obyekt sifatida ifodalash mumkinligini JavaScript tili rivojiga ulkan hissa qo'shgan dasturlovchi Duglas Krokford joriy etib, 1999-yilgi «ECMA-262» standartga kiritishga erishgan. 2002-yilda «json.org» saytini ochib, unda obyektning satr sifatida maxsus qanday ifodalanilishini yoritgan. Ma'lumotlarni bunday shaklda saqlanilishini o'ziga xos nom bilan **JSON** (JavaScript Object Notation — JavaScript obyektining ifodalanishi) deb atagan.

JSON ma'lumot saqlashning maxsus shakli ekan, unda albatta o'ziga xos qat'iy talablar mavjud. Ba'zilari JavaScript ijozat beradigan istisnolarni ham man etadi.

Masalan, indeksni anglatuvchi identifikator albatta satr sifatida faqat «"» ko'rinishdagi qo'shtirnoq ichida berilishi shart. JSONning umumiy sintaksisi:

obyekt {} | {*elementlar*}
elementlar *juftlik* | *juftlik*, *elementlar*
juftlik "*identifikator*" : *qiymat*
qiymat *satr* | *son* | *obyekt* | *jadval* | **true** | **false** | **null**

Bunda «satr» ham qat'iy tayinlanganday «"» ichida yoziladi. Sonlar va jadvallar yuqorida tanishganimiz [2.6.3.] kabi ifoda etilaveriladi. Satr ichida qo'shtirnoq yoki boshqa maxsus belgilar qatnashishi shart bo'lsa, akstaqsimdan (\) foydalaniladi [2.6.2.].

2006-yil ushbu ma'lumot uzatish shakli RFC 4627 raqam ostida internet kelishuvlarida ro'yxatdan o'tkazilgandan so'ng boshqa dasturlash tillari ham uni berilganlar turi sifatida tatbiq etishni boshladi. Hozirda deyarli barcha dasturlash tillarida **JSON** tur sifatida mavjud va uning ustida amallar bajarish uchun zaruriy funksiyalar kiritilgan. Albatta, ommalashgan sari JSONga qo'yiladigan talablar ham o'zgarib bordi, uning turli talqinlari amaliyotga tatbiq etilmoqda. Turli dasturlash tillari o'zlariga moslab turlicha shartlar va istisnolar kiritgan. Ammo biz barcha uchun umumiy bo'lgan JSONning asosiy qoidalari bilan tanishamiz.

JSON obyekti o'ziga xos quyidagicha muhim tamoyillarga ega:

- **JSON.parse(***matn***[, *funksiya*])** — JSON shaklidagi matnni obyektga aylantiradi. Agar matn sifatida

shunchaki qiymat berilgan bo'lsa, murakkab obyekt o'rniga oddiy qiymatni qaytaradi:

```
JSON.parse('{}');                              {}
JSON.parse('{"a": 1, "b": "do"}');             {a: 1, b: "do"}
JSON.parse('true');                            true
JSON.parse('"matn"');                          "matn"
JSON.parse('[1, 5, "false"]');                 [1, 5, "false"]
JSON.parse('null');                            null
JSON.parse('function a() { a = 1 }');          SyntaxError
```

Matnni obyektga aylantirish jarayonida har bir element uchun indeks va qiymatiga mutanosib ravishda amallar bajaradigan funksiya biriktirish mumkin:

```
JSON.parse('{"1":1,"2":2,"u":{"4":"o"}}',
  function(index, value) {                     1 ~> 1
    console.log(index + ' ~> ' + value);       2 ~> 2
    return value;                              4 ~> o
  }                                            u ~> {…}
);                                              ~> {…}
```

E'tibor berish kerakki, qiymat sifatida obyekt berilsa, uni o'girish ustuvor ahamiyatga ega hisoblanmoqda. Shuning uchun «2» indeksdan keyin «u» emas, «4» chiqyapti. Agar jadval berilganda barcha elementini «0»dan boshlab o'sish tartibida indeks bo'yicha chop etar edi. Oxirida esa qiymat sifatida obyekt borligini ko'rsatar ekan, indeksi yo'qligi anglashilyapti. Bu butun berilgan matn yagona obyektga aylanganini, aslida unga mos indeks mavjud emasligini bildiradi.

JSON matnidagi sintaksisda «:» va «,» yonlarida istagancha bo'sh joylar bo'lishiga ruxsat beradi. Ammo

andozada mutanosiblik buzilgan bo'lsa, o'girish jarayonida xatolik sodir bo'ladi va JavaScript kodlarining bajarilishidan to'xtashiga sabab bo'ladi.

▪ **JSON.stringify(*qiymat*[, *o'zgarish*[, *bo'lgich*]])** — berilgan qiymat yoki obyektni JSON shaklidagi matnga aylantiradi. Mohiyatan «JSON.parse»ga teskari.

Ma'lumki, JavaScript obyektida element qiymati sifatida funksiya kelishi ham mumkin. Lekin bu JSON standartiga to'g'ri kelmagani bois o'girish jarayonida funksiyali elementlarni tashlab yuboradi. Misol:

```
var o = { a: 1, f: function() { return 2;}, s: "a"};
JSON.stringify(o);                             '{"a":1,"s":"a"}'
```

Agar o'girilayotgan obyektning biror elementi avvaldan aniqlangan boshqa o'zgaruvchining qiymatini olgan bo'lsa, o'sha qiymat bilan aylantirilaveriladi. Aniqlanmagan qiymat «undefined» qabul qilgan elementlar ham tushirib qoldiriladi. O'girish jarayonida qandaydir o'zgartirish kiritish lozim bo'lsa, maxsus funksiyadan foydalaniladi. Deylik, qiymati «undefined» bo'lgan elementlarini «null»ga aylantirib o'girishni misol tariqasida ko'ramiz:

```
var ob = { a: 1, n: undefined, s: "a" };
JSON.stringify(ob,                             '{
  function(k, v) {                               "a": 1,
    if (v == undefined) return null;             "n": null,
    else return v;                               "s": "a"
  }, "  ");                                     }'
```

Tushunarliroq ko‘rinishi uchun har bir elementni alohida qatordan ikki bo‘sh joyli siljish bilan ko‘rsatadigan qilamiz.

Faqat keltirilgan indeksli elementlarni olishi uchun maxsus funksiya o‘rniga omil sifatida ularning ro‘yxati berilgan jadvalni ko‘rsatish kifoya:

```
JSON.stringify(ob, ["a", "s"] );   '{"a":1,"s":"a"}'
```

Obyekt juda keng qamrovli tushuncha. Amaliyotda qandaydir sinfga oid barcha obyektlarni matnga o‘girishga to‘g‘ri kelib qolsa, uning ichidagi ayrimlari ichma-ich joylashgan funksiya va obyektlardan tashkil topganligi bois matnga o‘girib berish uchun juda ko‘p ortiqcha ma’lumotlar qayta ishlanib biriktiriladi. Maxsus funksiyadan foydalanish samara bermasligi ham mumkin. Sababi, ular bir xil qolipga tushmasligi ehtimoli ko‘p, qolaversa, bunday funksiyalar o‘girishni ancha sekinlashtiradi. Obyektning barcha imkoniyatlarini ishga solib o‘ta murakkab holda yarataveramiz, lekin JSON matnga aylantirishda qat’iy cheklov o‘rnatish mumkin. Buning uchun obyektga «toJSON» indeksli, qiymati funksiyadan iborat element qo‘shiladi va unda qaytariladigan matngina ko‘rsatilishi ta’minlanadi. Misol:

```
o = { ..., toJSON: function() { return 'OBJECT'; } };
JSON.stringify(o);                          '"OBJECT"'
```

JSONning ham «Math» singari uslub va xususiyatlari mavjud emas.

2.7. Muntazam ifodalar

Muntazam ifodalar (regular expressions) matnlar bilan ishlash uchun barcha dasturlash tillarida beqiyos imkoniyatlar yaratadi. Dasturning kodi matndan iborat ekanligini inobatga olsak, muntazam ifodalar ustida bajariladigan amallarni dasturlashning asos ustunlaridan biri sifatida baholash mumkin.

Satrlar bilan tanishish jarayonida qism satrlarni topish imkoniyatlari bilan tanishib o‘tdik. Lekin ular doim ham yetarli emas. Ma’lum andoza asosida qism satr ustida ishlash uchun undagi imkoniyatlar ojizlik qiladi.

Matn ichidan berilgan andoza bo‘yicha qism satr qanday topiladi?!

Operatsion tizimlarda fayllarni izlash uchun ularning nomini to‘liq yozmasdan yulduzcha ishlatishni va natija qanday qaytarilishini yaxshi bilamiz. Bunda yulduzcha (*) istalgancha belgini anglatadi. Yagona ixtiyoriy belgi o‘rniga esa so‘roq (?) qo‘yiladi. Bularni «Joker» yoki «almashinuvchi» belgilar deb yuritiladi.

Ammo bunday cheklangan imkoniyat bilan fayl nomlari faqat raqamlardan iborat bo‘lganlarini topish kabi murakkabroq talablar qo‘yishning iloji yo‘q. Muntazam ifodalar esa istalgan murakkablikdagi talablar bo‘yicha qism satrlar izlash hamda natijalarni guruhlash imkoniyatiga ega. Buning uchun izlanuvchi qism satr andozasini aniqlovchi ma’lum kelishuvlar kiritilgan va ular barcha dasturlash tillari uchun bir xil:

- Nuqta – satrlarni bo‘luvchi `\n`, `\r`, `\u2028` yoki `\u2029` ishoralaridan tashqari ixtiyoriy belgi.

`\w` Asosiy lotin alifbosidagi harflar va tagchiziq (_) hamda raqamlar majmuasidagi bitta ixtiyoriy belgi.

`\W` Asosiy lotin alifbosiga kirmaydigan hamda raqam yoki pastki chiziq bo'lmagan belgilardan biri.

`\d` Ixtiyoriy raqamni bildiradi. (0dan 9gacha)

`\D` Asosiy lotin alifbosidagi harflarni bildiradi. Raqam emas degan ma'noda qo'llaniladi.

`\s` so'zlarni, qatorlarni, sahifalarni ajratuvchi belgi. Unga bo'sh joy, tabulyatsiya, «Enter» va shunga o'xshash quyidagi to'plamdagi belgilar kiradi:
`[\ \f\n\r\t\v\u00a0\u1680\u180e\u2000\u2001 \u2002\u2003\u2004\u2005\u2006\u2007\u2008 \u2009\u200a\u2028\u2029\u202f\u205f\u3000]`

`\S` Ajratuvchi bo'lmagan belgi. «`\s`» aniqlagan to'plamga kirmaydigan ixtiyoriy belgi.

`\t` tabulyatsiya

`\v` tik tabulyatsiya

`\b` so'zni ajratuvchi

`\r` bo'lakni ajratuvchi

`\n` satrni ajratuvchi

`\f` sahifani ajratuvchi

`\0` bo'sh belgi

`\cA` buyruqli tugmalar birikmasini anglatadi. Masalan, «`\cD`» – «`ctrl+D`» tasavvurini beradi.

`\` Asl belgilarni kelishuvdagisidan farqlash uchun oldiga akstaqsim qo'yiladi. Masalan, nuqta ixtiyoriy belgini anglatsa, «`\.`» esa aynan nuqtaning o'zidir.

`\xdd` Bunda «`dd`» o'n oltilik sanoq tizimidagi son, belgining maxsus kodini anglatadi.

`\udddd` Bunda «`dddd`» o'n oltilik sanoq tizimidagi son, belgining Yunikod kodlamasidagi maxsus kodi.

`[…]` Belgilar to'plamini anglatadi. Uning ichida chiziqcha «dan-gacha» ma'nosini bildiradi. Masalan, raqamlar `[0-9]` kabi ifodalanilishi «`\d`»ga teng.

`[^…]` To'plamdagi belgilar istisno qilinishini anglatadi. Misol, `[^0-9]` yozuvi «`\D`» bilan teng kuchli.

`[\b]` Oldga o'chirish (backSpace) belgisi.

`^` Matnning (izlanuvchi qismsatrning emas) boshi ekanligini anglatadi. Uni yuqoridagi to'plamning inkori bilan chalkashtirmaslik kerak. Agar berilgan matn qidirilayotgan qism satr bilan boshlangan bo'lsa, ijobiy natija qaytaradi.

`$` Matnning oxirini bildiradi.

`\b` So'zning birinchi yoki oxirgi belgisidan ajratuvchigacha bo'lgan chegarani anglatadi. Bu aslida belgi emas va uni yuqoridagi «oldga o'chirish» bilan chalg'itmaslik kerak. «So'z» deganda bunda faqat lotin harflari, tag chiziq va raqamlarning birlashmasi tushuniladi. Ammo kirill yozuvini qo'llamaydi. Masalan, «`Salom, Osmon`» jumlasidan «m»ning o'zini qidirilganda ikkita natija qaytsa, «m» bilan tugagan so'zni «**m\b**» tarzda izlanganda esa faqat bitta natijaga erishiladi.

`\V` «so'z» yoki «ajratuvchilar»ning ikkita belgi orasidagi belgilarni bildiradi, «**\b**»ning aksi.

`*` Oldindagi belgi(lar)ning bir necha marta takrorlanishini anglatadi. Takrorlanishlar soni nol, ya'ni mavjud bo'lmasligi ham mumkin.

`a(?=b)` «**a**»dan keyin faqat «**b**» kelganligini tekshiradi.

`a(?!b)` «**a**»dan keyin «**b**» kelmaganligini tekshiradi.

`+` Oldindagi belgi(lar)ning bir necha (bir yoki undan ko'p) marta takrorlanishini bildiradi.

`?` Oldindagi belgi(lar)ning 0 yoki 1 marta qaytarilganligini tekshiradi. Bittadan ko'p qaytarilgan bo'lsa, ikki va undan keyingilarini inobatga olmaydi.

`(...)` Guruhlash. Bunda bir necha belgilar ketma-ketligini yagona guruh sifatida ajratish mumkin. Topilgan natijalar maxsus jadvalda saqlanadi, keyingi safar ularga mos guruhlar tartibi bo'yicha `$1`, `$2`, ... sifatida murojaat qilsa bo'ladi.

`(?:...)` Faqat guruhlanadi, natijalar saqlanmaydi.

`a|b` «**a**» yoki «**b**» lardan biriligini anglatadi.

`...{n}` Oldingi belgi(lar) «**n**»marta takrorlanganini tekshiradi. (**n** — natural son)

`...{n,k}` Oldingi belgi(lar) «**n**»tadan «**k**»martagacha takrorlanganini tekshiradi. (**n** < **k** — natural sonlar)

`...{n,}` Oldingi belgi(lar) kamida «**n**»marta takrorlanganini tekshiradi. (**n** — natural son)

JavaScriptda muntazam ifodalar andozasini satr sifatida emas, shunchaki ikki egri chiziq (taqsim) orasida yozish mumkin. Haqiqiy son uchun andozaga misol:

```
/^\d+\.{1}\d+$/
```

Matn boshidan raqamlar (bitta yoki undan ko'p), keyin yagona nuqta va yana oxirigacha raqamlar ma'nosini beruvchi andoza keltirilgan.

Andozani chegaralovchi ikkinchi egri chiziqdan keyin qo'shimcha omillar berilishi mumkin. Ular bayroq deb yuritiladi. Bayroq sifatida quyidagi belgilar ishlatiladi:

`g` Global qidiruv. Bunday bayroqsiz, birinchi natijadan keyin izlash to'xtatiladi.

`i` Bosh va kichik harflar birday izlanadi.

`m` Bir necha qatorli qidiruvni ta'minlaydi. Bunda «^» va «$» belgilari odatdagidek butun matn uchun emas, har bir qator (xatboshi) uchun tatbiq etiladi.

Ushbu bayroqlar yordamida yuqoridagi andozani quyidagi ko'rinishda yozish mumkin:

```
/^\d+\.{1}\d+$/gm
```

Umuman olganda, muntazam ifodalar ustida amallar «RegExp» biriktirilgan obyekti yordamida bajariladi. Sintaksisi:

```
new RegExp(andoza satr[, bayroqlar])
```

Bunda omillar chegaraviy egri chiziqlarsiz qo'shtirnoq ichida satr sifatida keltiriladi. Quyidagi ikki xil e'lon qilish teng kuchli:

```
/^\d+\.{1}\d+$/gm
new RegExp("^\d+\.{1}\d+$", "gm")
```

Mavjud muntazam ifoda qanday bayroqlarga ega yoki umuman olganda, tarkibini bilish uchun quyidagi xususiyatlaridan foydalanish mumkin:

- **global** — global izlash bayrog'iga egaligini tekshiradi, mantiqiy qiymat qaytaradi:

```
var d = new RegExp('\d{2}\.\d{2}\.\d{4}', 'g');
d.global
```

```
true
```

```
var f = new RegExp('\d+\,*\d*', 'm');
f.global                                    false
```

▪ **ignoreCase** — katta-kichik harflarni farqlamasdan qidirishni ta'minlovchi bayrog'i qo'yilganligini tekshiradi, mantiqiy qiymat qaytaradi.

▪ **multiline** — bir necha qator bo'yicha izlash bayrog'i mavjudligini tekshiruvchi mantiqiy qiymat.

▪ **flags** — barcha o'rnatilgan bayroqlarni satr sifatida qaytaradi.

```
var f = new RegExp('\d+\,*\d*', 'mig');
f.multiline                                 true
f.ignoreCase                                true
f.flags                                     "gim"
```

▪ **source** — qidirish andozasi mantnini qaytaradi:

```
var f = new RegExp('\d+\,*\d*', 'g');
f.source                                    "d+,*d*"
```

▪ **lastIndex** — qidirish qaysi o'rindan boshlanilishini ko'rsatadi. Boshlang'ich qiymati: 0. Bitta o'xshashlik topgandan keyin, qiymati o'sha o'ringa teng bo'ladi va izlashni undan keyingi joydan boshlaydi. Uning qiymatini o'zgartirish ham mumkin.

RegExp obyekti quyidagicha ikkita uslubga ega:

▪ **exec(*satr*)** — berilgan «satr»dan muntazam ifoda bo'yicha izlaydi, topilgan natijalarni jadval tarzida, aks holda null qaytaradi:


```
var s = "Sayding qo'yaber, Sayyod, sayyora ekan mendek.";
var r = /Say/ig;
var top = r.exec(s);
   ["Say", index: 0,  input: ...]   (r.lastIndex = 3)
top = r.exec(s);
   ["Say", index: 16, input: ...]   (r.lastIndex = 19)
top = r.exec(s);
   ["Say", index: 23, input: ...]   (r.lastIndex = 26)
top = r.exec(s);
   null                             (r.lastIndex = 0)
```

▪ **test(*satr*)** — berilgan «satr»da muntazam ifoda bor-yo'qligini tekshiradi, mantiqiy tur qaytaradi:

```
var s = "Sayding qo'yaber, Sayyod, sayyora ekan mendek.";
var r = /Say/ig;
top = r.test(s);    true   (r.lastIndex = 3)
top = r.test(s);    true   (r.lastIndex = 19)
top = r.test(s);    true   (r.lastIndex = 26)
top = r.test(s);    false  (r.lastIndex = 0)
```

Agar global bayroq qo'yilmasa, «lastIndex» doim 0 qabul qiladi va «**test**»ga necha marta murojaat qilinmasin, «rost» qiymat qaytaraveradi.

Satr obyektidagi qismsatrni boshqasiga almashtiradigan **replace** [2.6.2.] uslubida ham muntazam ifoda qo'llaniladi. Uning yordamida matn ichidan ma'lum qismlarini ko'rsatilgan andoza bo'yicha boshqa satrga almashtirishimiz mumkin.

Misol tariqasida Cho'lponning quyidagi misralarini birinchi shaxs birlikdan ikkinchi shaxs birlikka o'tkazish muntazam ifoda yordamida nechog'li qulay ekanligini ko'rib o'tamiz:

```
var txt = "Men yo'qsil na bo'lib uni suyubman,\nUning-chun yonibman, yonib kuyibman,\nBu boshni zo'r ishga berib qo'yibman,\nMen suyub... men suyub... kimni suyubman?!!!"

txt.replace(/m([ea])n/ig, 'S$1n')

"Sen yo'qsil na bo'lib uni suyubSan,
Uning-chun yonibSan, yonib kuyibSan,
Bu boshni zo'r ishga berib qo'yibSan,
Sen suyub... Sen suyub... kimni suyubSan?!!!"
```

Qavs yordamida guruhlab olingan qismga «$» belgisi va tartib raqami yordamida murojaat qilinayotganiga e'tibor berish lozim.

Albatta, bu yerda biroz kamchilik bor. So'z o'rtasida kelayotgan «San» qo'shimchasi ham bosh harf bilan yozilib qolmoqda. Ushbu nuqsonni bartaraf etish uchun maxsus funksiyani va shartni tekshirish operatorini ishlatish lozim bo'ladi. Biroz kod yozib, kutilgan natijaga erishishimiz mumkin:

```
txt.replace(/(m)([ea])n/ig, function(top, t1) {
  if (t1 == 'M')  return top.replace(t1, 'S');
  else return top.replace(t1, 's');
} );

"Sen yo'qsil na bo'lib uni suyubsan,
Uning-chun yonibsan, yonib kuyibsan,
Bu boshni zo'r ishga berib qo'yibsan,
Sen suyub... sen suyub... kimni suyubsan?!!!"
```


Bunda matn ichidan andoza bo'yicha topilgan qism satr «top»dagi birinchi guruhlash bo'yicha katta «M» harfini katta «S»ga almashtirilmoqda. Aks holda andoza bo'yicha faqat kichik «m» bo'lishi mumkin, xolos. U kichik «s»ga o'zgartirilyapti. Bunda ichki «replace» funksiyalar muntazam ifoda uchun qo'llanilgani yo'q. Faqatgina asosiy tashqi «replace» uslubidagina muntazam ifodaga murojaat qilinyapti. Bundan ko'rinadiki, «replace» anchagina mukammal funksiya bo'lib, unda qism satr o'rniga muntazam ifoda qo'llash va hatto qo'shimcha funksiya ishlatish ham mumkin ekan.

3-BO'LIM

OPERATORLAR BILAN ISHLASH

Operator «amal bajaruvchi» degan ma'noni anglatgani uchun yuqorida tanishgan **amallar**ni [2.5.] ba'zi manbalarda «operatorlar» deb ham qo'llaniladi. Bunday qarashni inkor etmagan holda, ushbu kitobda ma'lum bir jarayonni amalga oshirish uchun vosita sifatida xizmat qiladigan maxsus buyruq va belgilar majmuasini «**operator**» deb yuritamiz. Undagi kalit so'z va maxsus belgilarni «**operant**» deb ataymiz.

Har bir operator asosiy **tana** qismiga ega, ba'zan tanasi bir nechta **bo'lak**lardan tashkil topishi mumkin. Bo'laklar, odatda, yagona **ifoda**dan [2.5.] tashkil topadi. Bir necha ifodani umumlashtirish lozim bo'lganda, ular gulli qavs ({}) ichida yoziladi (birlashtiriladi) va bunday **bo'lak**dan so'ng nuqtali vergul qo'yish talab etilmaydi.

3.1. Tanlash operatorlari

Tanlash operatorlari berilgan shart asosida bir necha amallar guruhidan faqat bittasi bajarilishini ta'minlaydi.

3.1.1. if

Asosiy tanlash operatori «`if`»ning sintaksisi:

`if (`*shart*`)` *bo'lak1* `[else` *bo'lak2*`]`

Bunda «shart» qanoatlantirilsa, «bo'lak1» ishga tushadi, aks holda faqat «bo'lak2» bajariladi. Sintaksisdan ko'rinib turibdiki, ikkinchi qismi, ya'ni «else» mavjud bo'lmasligi ham mumkin [2.6.5.].

Musbatligini tekshirish bilan «y»ga «x»ning modulini o'zlashtiradigan misol ko'ramiz:

```
if (x > 0) y = x; else y = -1*x;
```

E'tibor berilsa, bunday shart bilan qiymat berishda ortiqcha yozuv ko'p. Mukammal dasturlash talablariga ko'ra imkon qadar yozuvni optimallashtirish lozim.

3.1.2. ? ... : ...

Yozuvni optimallashtirish uchun «if»ning o'rniga uning muqobilini qo'llash tavsiya etiladi:

shart `?` *bo'lak1* `:` *bo'lak2*

Ya'ni yuqoridagi misolni soddagina yozish mumkin:

```
y = x > 0 ? x : -1*x;
```

Lekin bunda ikki xil qiymat berish shart bo'ladi. Musbat hollarda «x»ning qiymatini ol, aks holda hech nima qilma deb yozish imkonsiz. Bunday holatda to'liq bo'lmagan «if»ni qo'llash maqsadga muvofiqdir [2.6.5.].

3.1.3. switch

Agar ifodaning bir necha qiymati uchun o'ziga xos amallar bajarilishi lozim bo'lsa, bir nechta shart bilan «if»larni ketma-ket yozish noqulaylikni yuzaga keltiradi. Bunday vaziyatlarda «**switch**» operatori qo'llaniladi. Uning asosiy tanasi bir necha bo'lakdan tashkil topgan bo'lib, gulli qavs ichida yoziladi va unda **case**, **break**, **default** kabi operantlar ishlatilishi mumkin:

```
switch (ifoda) {
  case qiymat1:
    bo'lak1
  [break;]
  case qiymat2:
    bo'lak2
  [break;]
  ...
  case qiymatN:
    bo'lakN
  [break;]
  default:
    bo'lak
  [break;]
}
```

Bunda berilgan «ifoda» qabul qilgan qiymat bo'yicha mos «case»ga borib, «break» uchraguncha yoki asosiy tanasidagi barcha bo'lakchalardagi kodlar bajarilib bo'lguncha «switch» o'z faoliyatini olib boradi. Qiymat bo'yicha faqatgina bitta bo'lakchadagi amallar bajarilishi kerak bo'lsa, bo'lak so'ngida «break» qo'llash shart, aks

holda o'sha «case»dan boshlab operator oxirigacha barcha kodlar bajariladi. Agar «ifoda» «**case**»larda ko'rsatilgan birorta ham qiymatga teng bo'lmasa, unda «**default**»da yozilgan bo'lak ishlaydi. Berilgan sonni oltiga bo'lgandagi qoldiq bo'yicha ishlaydigan bir misol ko'rib o'taylik:

```
function bol6ga(x) {
  let txt = '';
  switch (x % 6) {
    case 1:
      txt += " 1 ga ortiq";
    case 3:
    case 5:
      txt += " toq son.";
    break;
    case 2:
    case 4:
      txt += " juft son, lekin 6 ga bo'linmaydi";
    break;
    default:
      txt += " 6 ga bo'linadi";
  }
  return txt;
}
bol6ga(1221);   " toq son."
bol6ga(1219);   " 1 ga ortiq toq son."
bol6ga(1212);   " 6 ga bo'linadi"
bol6ga(2020);   " juft son, lekin 6 ga bo'linmaydi"
```

3.2. Takrorlash operatorlari

O'zgaruvchilarning ma'lum qiymatlari asnosida ifodalarning qayta-qayta bajarilishini ta'minlash uchun takrorlash operatorlari qo'llaniladi.

3.2.1. for

Eng ko'p qo'llaniladigan **for** takrorlash operatorining sintaksisi quyidagicha:

for ([*ilk qiymat*]; [*shart*]; [*ifoda*]) *bo'lak*

Ishlash uslubi to'liq tushunarli bo'lishi uchun quyida bir misol keltiramiz:

```
for (let i = 1; i < 4; i++)
    console.log(i + '-qadam');
```

Yuqoridagi sintaksisdan ayonki, uchta omildan ixtiyoriysini tushirib qoldirish mumkin ekan. Ammo nuqtali vergul joyida turishi shart. Bunday hollarda qo'shimcha shart «**for**»ning tana qismida beriladi:

```
for (let i = 1; ; i++) {
  console.log(i + '-qadam');
  if (i > 10) break;
}
```

Faqat bitta o'zgaruvchining o'ziga bog'lanib qolmasdan bir nechtasidan ham foydalanish mumkin [2.5.(0)]:

```
for (let i = 1, j = 10; i * j < 100; i++, j += 10)
    console.log(i + '/' + j +  '-joy');
```


3.2.2. for ... in

Berilgan obyektning har bir indeksi bo'yicha takrorlashni ta'minlash uchun quyidagicha sintaksis bo'yicha murojaat qilinadi:

for (*o'zgaruvchi* **in** *obyekt*) *bo'lak*

Misol:

```
var i, a = [88, 'ali', false],
    o = {"a":1, "b":2, "c":true};
for (i in a) console.log(i, ' => ', a[i]);
for (i in o) console.log(i, ' => ', o[i]);
```

3.2.3. for ... of

Berilgan obyektning indeksi bo'yicha emas, har bir elementining qiymati bo'yicha takrorlashni ta'minlash uchun quyidagicha sintaksis qo'llaniladi:

for (*o'zgaruvchi* **of** *obyekt*) *bo'lak*

Yuqorida obyekt elementining qiymatini olish uchun «for...in»ni qo'llash bilan tanishdik, bir qarashda ushbu «for...of» ortiqchadek tuyuladi. Lekin ba'zi maxsus jadval uchun uning farqi muhimligi seziladi. Misol:

```
var i, v, a = [88, 'ali', false];
a.kalit = "qiymat";

for (i in a)
    console.log(a[i]);
```

```
[ 88, "ali", false,
  kalit: "qiymat" ]
88
ali
false
qiymat
```

```
for (v of a) console.log(v);
```

```
88
ali
false
```

3.2.4. while

Odatda, **for** necha marta takrorlash aniq bo'lgan hollarda qo'llanishga mo'ljallangan. Lekin yuqoridagi misoldan ko'rinadiki, takrorlashni to'xtatuvchi shartni keyin tana qismida ko'rsatish ham mumkin ekan.

Albatta, bu biroz noqulaylikni yuzaga keltiradi. Ma'lum shart bajarilguncha takrorlashni ta'minlash uchun **while** operatori qo'llaniladi, sintaksisi:

while (*shart*) *bo'lak*

Bunda «shart» bajarilsa, «bo'lak»dagi kodlar takror va takror ishga tushaveradi.

3.2.5. do ... while

Shartni tekshirishdan avval biror ifoda bajarilishi kerak bo'lsa, shartli takrorlash operatori quyidagicha qo'llaniladi:

do *bo'lak* **while** (*shart*);

Misol:

```
var i = 1;
do console.log(i++ + '-takror')
while (i < 5);
```


Takrorlash operatorlarida eng muhim talab — shartni to'g'ri qo'yishdir. Doimo cheksiz davriy takrorlanish paydo bo'lib qolmasligini, har bir qatnashayotgan o'zgaruvchilarning qabul qilishi mumkin bo'lgan qiymatlarini alohida e'tibor bilan inobatga olish shart. Aks holda, samarasiz hisob-kitoblarni tinimsiz bajarayotgan brauzer xato berishi yoki qotib qolishi tayin.

3.2.6. Takrorlanishni tark etish

Yuqorida [3.1.3.] [3.2.1.] «**break**»ni qo'llash bilan tanishib o'tdik. U o'zi joylashgan bo'lakdan chiqib ketishni, ya'ni operator faoliyatini yakunlashni anglatadi. Umuman olganda, ixtiyoriy takrorlash operatori tanasida «break»ni ishlatish mumkin. Ko'rsatilgan shart bajarilganda o'zidan keyingi amallarni bajarmasdan operator tanasini butkul tark etadi va qaytib ortidagi ifodalarga murojaat etmaydi. Lekin amaliyotda faqat ayrim shartlardagina ifodalarni e'tiborsiz qoldirib, boshqa hollarda barcha amallar qayta bajarilishi talab etilishi mumkin. Bunday vaziyatlarda «**continue**» qo'llaniladi. Masalan, nolga bo'lish mumkin emas degan matematik talabni inobatga oluvchi misolni ko'rib chiqaylik:

```
var k = -4;
while (true) { // cheksiz takrorlash
  k++;
  if (k == 0) continue;
  console.log(10 / k);
  if (k > 3) break;
}
```

```
-3.3333333333333335
-5
-10
10
5
3.3333333333333335
2.5
```

Ko‘p hollarda takrorlash operatorini ichma-ich qo‘llashga to‘g‘ri keladi. Ayniqsa, ko‘p o‘lchamli jadvallar bilan ishlashda bunga juda jiddiy ehtiyoj seziladi. Qandaydir shartda barcha takrorlashni tugatish uchun yoki tashlab o‘tish bilan bajarish uchun birgina «break» yoki «continue» ojizlik qiladi. Masalan, «o‘nta ichma-ich takrorlanishning uchinchisidan chiqib ket», deyilgan shartni yozish juda mushkul. Bunday vazifani bajarish uchun qo‘shimcha «**label**» kiritilgan. (*U aslida identifikator bo‘lib, faqatgina takrorlash operatorlari oldidan ikki nuqta (:) bilan ajratib yoziladi!*) Takrorlash operatori tanasida ma’lum shartdan keyin qo‘yilgan «break» yoki «continue»dan so‘ng o‘sha identifikator nomi yoziladi. Misol sifatida quyidagi jadvalni keltiramiz:

12.8	23.98	90.6	66.9	26.91
56.6	81.01	—	44.2	64.27
13.1	93.14	71.3	—	43.18

Ushbu jadvaldagi barcha sonlarning yig‘indisini hisoblovchi kod yozish lozim bo‘lsa, u yerdagi minuslarni alohida e’tiborga olishga to‘g‘ri keladi. Avvalo, ikki o‘lchamli jadval yaratib, unga yuqoridagi ayni qiymatlarni beramiz:

```
var a = [
  [12.8, 23.98, 90.6, 66.9, 26.91],
  [56.6, 81.08, "—",  44.2, 64.27],
  [13.1, 93.14, 71.3, "—",  43.18],
];
```

Barcha elementlarni bir-biriga qo‘shib boradigan kod yozsak, kutilgan natija chiqmasligi aniq:


```
var r, e, s = 0; "358.87—44.264.2713.193.1471.3—43.18"
for (r of a)
  for (e of r)
    s += e;
```

Demak, har bir elementni qo‘shishdan oldin uning satr emasligini tekshirib, uni tashlab ketish kerak:

```
for (r of a)                                    688.06
  for (e of r) {
    if (typeof e == 'string')
      continue;
    s += e;
  }
```

Aytaylik, qatorda minus uchrasa, undan keyingi hamma sonlarning ishonchliligi shubha ostiga olinib, ularning barchasini tashlab ketish lozim bo‘lsin. Keyingi qatordagi sonlarni minus uchraguncha qiymatlari qo‘shilishi talab etilsin.

Buning uchun maxsus nishon qo‘yib, davom ettirishni u yerdan boshlash kerakligi ko‘rsatilishi lozim:

```
                                                536.41
nishon:
for (r of a)
  for (e of r) {
    if (typeof e == 'string')
      continue nishon;
    s += e;
  }
```

Takrorlash operatorlari ichida juda uzoq muddat ishlaydigan kod yozishdan avval, albatta, yuzaga kelish mumkin bo‘lgan xatoliklarni inobatga olish lozim.

3.3. Talab operatorlari

Ushbu bo'limda noodatiy, qandaydir talablar asosida bajariladigan operatorlar bilan tanishamiz.

3.3.1. try

Har doim ham ayni bir kod kutilganidek benuqson bajarilavermaydi. O'zgaruvchilar qatnashgan ifodalarda ularning qiymatlariga qarab, gohida xatolik yuz berishi mumkin. Mukammal dasturlashning talablaridan yana biri, sodir bo'lishi mumkin bo'lgan barcha kamchiliklarning oldini olish chorasini ko'rishdir.

JavaScriptda biror xatolik yuz bersa, kodlarning bajarilishi o'sha joyda to'xtab, konsolga mos xabar chiqariladi. Oddiy foydalanuvchi uni ko'rmasligi mumkin. Masalan, sahifadagi bitta tugma bosildi va u o'z vazifasini bajarayotgan paytda qandaydir xatolik yuzaga kelib, amallar bajarilishi to'xtadi. Bu haqida foydalanuvchiga tegishli axborot berilmasa, u yanglish xulosaga kelishi mumkin. Misol sifatida ixtiyoriy o'zgaruvchini satrga o'giradigan «**toString()**» uslubini [2.3.2.] ko'raylik. U to'rt xil vaziyatda xatolik keltirib chiqaradi:

`x.toString();`	Noma'lum o'zgaruvchi
`x = undefined; x.toString();`	Qiymat aniqlanmagan
`x = null; x.toString();`	Qiymat «null» ga teng
`x = {toString:1}; x.toString();`	toString funksiyamas

Uslubning bajarilish mavqei indeksnikidan pastroq bo'lgani uchun to'rtinchi vaziyatda «toString» obyektning alomati sifatida qaralmoqda.

Aytaylik, shu to'rtala vaziyatni inobatga olmagan holda HTMLdagi mavjud darchaga kiritilgan qiymatni yosh sifatida qabul qilib, u haqida ma'lumot chiqarish uchun «`toString()`» yordamida sonni satrga aylantirilsa, xato yuz berishiga zamin hozirlangan bo'ladi. Bu kabi qo'pol kamchiliklarga yo'l qo'ymaslik maqsadida «**try**» operatoridan foydalaniladi. Sintaksisi:

```
try { asosiy bo'lim }
catch (xato) { xatolik bo'limi }
finally { yakuniy bo'limi }
```

Bunda «asosiy bo'lim»da yozilgan kodda xatolik yuz bersa, «xatolik bo'limi» ishga tushadi, «yakuniy bo'lim» esa har qanday vaziyatda ishlaydi. Ya'ni odat bo'yicha JavaScriptning bajarish faoliyati to'xtab qolishiga yo'l qo'yilmaydi. Ushbu operatorda «asosiy bo'lim»dan so'ng «xatolik» yoki «yakuniy» bo'limlarning hech qursa bittasi qatnashishi shart. Aynan qaysi turdagi xatolik yuz berganini [3.4.2.] aniqlash uchun «instanceof»dan foydalaniladi. [2.4.7.] Misol:

```
try {
   x.toString();
} catch (xato) {
   if (xato instanceof ReferenceError)
    alert(`${xato.name}: E'lon qilinmagan!`);
   else alert(xato.message);
}
finally { console.log('Satrga o'girildi.'); }
```

Ushbu misoldan ko'rinib turibdiki, xatolikni aniqlovchi obyekti «name» va «message» xususiyatlariga ega ekan. Ular atalishi bilan nima maqsadda qo'llanishi tushunarli. Bulardan tashqari «stack» xususiyati ham mavjud bo'lib, u quyidagi ma'lumotlarni o'zida ko'rsatilgan ko'rinishda jamlaydi:

`${xato.name}: ${xato.message} at funksiya(fayl):qator:o'rin`

Ushbu to'laqonli ma'lumotni sinchiklab ko'rganda qaysi faylning, qaysi qatorida, nechanchi o'rinda, qaysi funksiya ichida, qanday (nomli) xatolik, qanday vaziyatda sodir bo'lganini aniqlash mumkin.

Yozilgan talab bajarilmaganda maxsus istisnolarni qanday amalga oshirilishini aniqlovchi «**try**» operatori mukammal dasturlashda keng qamrovda qo'llaniladi.

3.3.2. throw

Kod bajarilar ekan, JavaScriptda standartlashtirilmagan xatoliklar ham sodir bo'lishi ehtimoli mavjud. Ba'zi turdagi xatoliklarni dasturlovchi o'zi aniqlashiga imkon yaratilgan [3.4.2.]. Ammo ulardan tashqari ham o'ziga xos nuqsonlarni ko'rsatib borishga to'g'ri keluvchi holatlar uchrab turadi. Bunday hollarda maxsus talab operatori «**throw**»dan foydalaniladi. U uchragan joyda dastur bajarilishdan to'xtab, berilgan xatolik ko'rsatiladi. Sintaksisi:

throw *xatolik*;

Albatta, «xatolik» sifatida ixtiyoriy qiymat, xususan, oddiy xabar yozilgan satr berilishi mumkin. Lekin umumiy xatolik qoidalariga tushishi uchun maxsus obyekt kiritish tavsiya etiladi. Misol:

```
var nuqson = new Error("nozik kamchilik yuz berdi");
nuqson.name = "qaltisNuqson";
try {
   // if (qaltisShart)
    throw(nuqson);
} catch (xato) {
   console.log(xato);
   alert(xato.name);
}
```

Agar «try» o'rniga «if»ning o'zi qo'llanilsa, dastur bajarilishdan to'xtab qoladi.

3.3.3. «async/await»

JavaScriptda o'ziga xos yana bir obyekt mavjud bo'lib, uning odatdagidan farqli jihati shundaki, qiymat o'rniga «**va'da**» qaytaradi. O'z navbatida, u ortga surilgan yoki birvarakayiga bajarilmaydigan amallar ustida ish boshlaydi va hisoblash yoki kutish nihoyasiga yetgach, va'da qilingan qiymatni o'zlashtiradi. Kod ishga tushishni boshlagandan va'da bajarilguncha u uch xil holatdan birida bo'lishi mumkin:

- **kutish (pending)** — boshlang'ich vaziyat, hali bajarilish nihoyasiga yetmadi va bekor ham qilinmadi;

- **bajarildi(fulfilled)** — kod muvafaqqiyatli amalga oshirildi;
- **rad etildi (rejected)** — bekor qilindi;

Va'da obyekti quyidagi sintaksis bo'yicha qo'llaniladi:

```
new Promise(function(buyruq, bekor) {
    ...
    // buyruq(qiymati);
    ...
    // bekor(qiymati);
});
```

O'zgaruvchiga «va'da» emas, uning natijaviy qiymatini o'zlashtirish uchun «**await**» kutish operatori qo'llaniladi. Funksiya natija sifatida «va'da» qaytaradigan bo'lsa, e'lon qilishda «funtion» so'zidan oldin «**async**» kalit so'zi ishlatiladi. Uni «birvarakay ishlamaydigan funksiya» deb ham yuritiladi. Ya'ni «**birvarakaymas**» funksiyaga murojaat qilingandan keyin yozilgan kodlar, u natija qaytarishini kutmay bajarilaveradi. Kutilgan natija olingandan so'ng «buyruq», xatolik sodir bo'lgan vaziyatda esa «bekor» o'rniga keltirilgan shartli funksiyalar ko'rsatilgan qiymatni qaytaradi. Misol:

```
function birvarakaymas() {
  return new Promise((buyruq, bekor) => {
    const val = Math.round(Math.random());
    val ? buyruq('Buyruq bajarildi.')
        : bekor('Bekor qilindi!!!');
  });
}
```



```
async function bajar() {
  try {
    const b = await birvarakaymas();
    console.log(b);
  } catch(x) {
    console.log(x);
  }
}
bajar();                Buyruq bajarildi.
bajar();                Buyruq bajarildi.
bajar();                Bekor qilindi!!!
```

Odatda, va'da obyekti serverga murojaat qilib natijani kutish jarayonida yoki ma'lum vaqtdan so'ng ishga tushadigan funksiyalar bilan ishlashda keng qo'llaniladi. Kiritilgan millisoniyalardan so'ng, ko'rsatilgan qiymatni qaytarish uchun «**setTimeout**» funksiyasidan foydalanamiz. Bu va'da obyektining faoliyatini to'laqonli tushunishga yordam beradi. Sintaksisi:

setTimeout(*funksiya*, *millisoniya*[, ...*omillar*]);

Birinchi omil sifatida mavjud funksiyaning nomi beriladi yoki shu yerning o'zida e'lon qilinadi. Ikkinchi o'rinda necha millisoniyadan so'ng ishga tushishi ko'rsatiladi. Keyin funksiyaga omillar berilishi lozim bo'lsa, kiritiladi. Ushbu funksiya o'zidan keyingi yozilgan kodlar bajarilishini kuttirmaydi. Berilgan vaqtdan so'ng alohida oqim sifatida ishga tushadi.

Misol:

```
setTimeout(alert, 1000, 22);
console.log(11);
```

Konsolga darhol «11» qiymati chiqariladi, bir soniya o'tgandan so'ng «alert» xabari «22»ni ko'rsatadi.

Va'da obyekti quyidagi tamoyillarga ega:

- **Promise.resolve(*qiymat*)** — berilgan «qiymat»ni va'daga aylantiradi. Albatta, qiymat sifatida «va'da»ning o'zi ham berilishi mumkin.
- **Promise.reject(*sabab*)** — keltirilgan «sabab» bo'yicha bekor qilingan «va'da»ni qaytaradi.
- **Promise.rase([*va'dalar*])** — ko'rsatilgan «va'dalar»dan qaysi birida «kutish» holati birinchi yakunlansa, o'shaning o'zini qaytaradi. Misol:

```
var vada1 = new Promise(function(resolve, reject) {
    setTimeout(resolve, 500, 'Ilk va'da');
});
var vada2 = new Promise(function(resolve, reject) {
    setTimeout(resolve, 100, 'Keyingi va'da');
});
Promise.race([vada1, vada2]); // 2-va'da qaytadi
```

- **Promise.all([*va'dalar*])** — barcha «va'da»lar bajarilib bo'lgandan keyin natijalari ko'rsatilgan tartibda jadval sifatida qaytariladi.

Yuqorida [3.3.1.] «**catch**»ni «try» talab operatorining davomi sifatida o'rgandik. Lekin JavaScriptda aynan shu nomdagi uslub ham mavjud bo'lib, u «va'da» uchinchi holatga tushib qolganda chaqiriluvchi funksiya hisoblanadi. Birinchi holat yakuniga yetganda, ya'ni ikkinchi yoki uchinchi holatda «**then**» uslubi qo'llaniladi.

Ushbu ikki uslub «timsollash» [2.6.8.] yordamida yaratilgan bo'lib, ixtiyoriy va'dalarda yoki birvarakaymas funksiyalarda tatbiq etilishi mumkin. Omil sifatida ularga kutish holati yakunlangandan keyin bajariluvchi mos funksiya beriladi. Ushbu juftlik qo'llanilishini bitta misol orqali ko'rib o'tamiz:

```
function vada(vaqt, qiymat) {
    return new Promise((bajar, bekor) => {
        if (qiymat > 1000) bekor();
        setTimeout(bajar, vaqt, qiymat);
    });
}
var v = vada(700, 1060);
v.catch(
    function(e) {
        console.log(e);
        return false;
    }
).then(
    function(e) {
        if (e) alert(e);
    }
);
```

Ushbu misolda qiymat sifatida 1000 dan katta bo'lmagan son kiritilsagina, berilgan vaqtdan so'ng «alert»da u ko'rsatiladi. Aks holda bekor qiluvchi «catch» uslubi ishga tushadi va konsolga xatolikni chiqarib, «yolg'on» qiymat qaytaradi. So'ng «then» uslubidagi funksiya omili o'sha «yolg'on» qiymatni qabul qiladi va tekshiruvga ko'ra «alert» chiqarilmaydi.

Agar «then» uslubini qo'llash qiyin tuyulsa, kutish operatoridan foydalanish kifoya. Bunda va'da birinchi holatdan chiqquncha boshqa kodlar bajarilmaydi. Ya'ni natija qaytguncha «qotib turish»iga majbur bo'linadi:

```
var t = await vada(3000, 100);
```

Ammo xatolik yuz berishi mumkinligini inobatga olib, ayniqsa, «catch» o'rniga qo'llanilayotganda «try» talab operatoridan foydalanish maqsadga muvofiqdir.

3.4. Kod tahlili (debug)

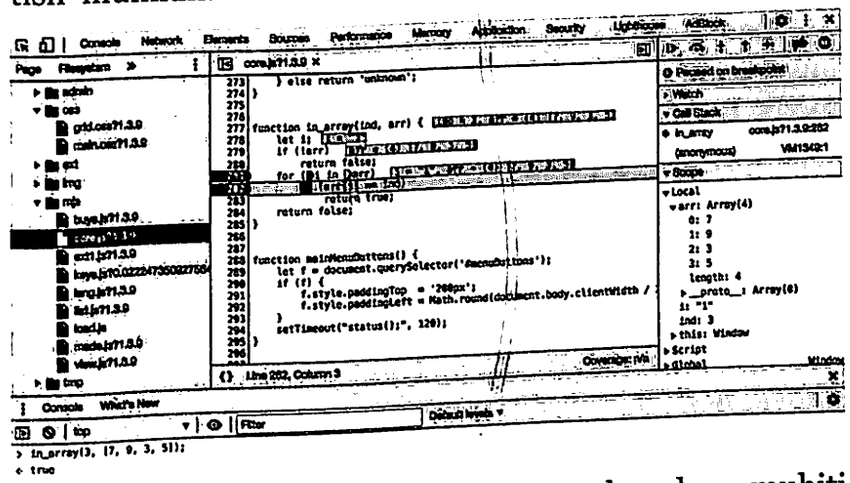
Murakkab vazifalarni bajaruvchi dastur tuzilar ekan, undagi kodlarning hajmi ham anchagina katta bo'lishi tabiiy. Avvalo kod yozishda tartibga qat'iy rioya qilish kerak, bu bilan uning dasturlovchiga tushunarli bo'lishi ta'minlanadi hamda kerakli joyga tuzatish kiritishni osonlashtiradi.

3.4.1. Qadam-baqadam tekshirish

Aytish joizki, mukammal dastur hech qachon bir marta yozilish bilan paydo bo'lib qolmaydi. Shuning uchun ham ommaviy dasturiy ta'minotlarning deyarli barchasining vaqt o'tgan sayin yuqori raqamlangan yangi talqinlari paydo bo'lib, yana ko'tarilib boraveradi.

Dasturda aksariyat hollarda xatolik hamma holatlarni yuz foiz inobatga olmasligimizdan kelib chiqadi. Muhimi, uni to'g'ri aniqlab, tez bartaraf etishda. Lekin bir necha yuz fayllar ichidagi bir necha ming satrlar orasidan birgina holdagi qandaydir xatoni topish, albatta, oson ish emas.

Bunda bizga brauzerning o'zi katta yordam beradi. hozirda barcha ommaviy brauzerlarda «dasturlovchining uskunalari» mazmunida nomlangan alohida muhit mavjud. Unda yuklanayotgan fayllar, qanday oldi-berdi qilinayotganidan tortib, qanday ishlayotganigacha kuzatish mumkin.



Ushbu rasmda **Chrome**ning uskunalar muhiti tasvirlangan. Boshqa brauzerlarning oxirgi talqinlarida ham xuddi shunday ko'rinishda imkoniyatlar mavjud. Unda «core.js» faylining 282-qatorida «in_array» funksiyasinig maydonida ayni paytda «arr», «i» va «ind» o'zgaruvchilarining qanday qiymat qabul qilganligini o'ng tarafdagi oynachadan ko'rish mumkin. Bu qadam-

baqadam tekshirish uchun juda qulay imkoniyatni yaratadi. Navbatdagi funksiya ichiga kirish kerak yoki shart emasligiga qarab, yuqoridagi tugmalardan biri bosiladi va keyingi bosqichda o'zgaruvchilar qanday qiymatlar qabul qilayotgani kuzatib boriladi. Eng ko'p xatolik hisoblashda o'zgaruvchining kutilgan qiymatda kelmasligi yoki mavjud emasligidan kelib chiqadi. Har bir brauzer xato yuz berganda uni qaysi faylnng qaysi qatoridaligini ko'rsatadi. Chap tarafdagi «daraxt» yordamida faylni tez topish mumkin. Hatto ko'rsatilgan qatorda bir-biriga o'xshash bir qancha ifodalar yozilgan bo'lsa ham, aynan nimaning hisobiga xatolik yuz berayotganining aniq o'rnini ko'rish mumkin.

Yuqoridagidek tekshiruvni bajarish uchun, brauzer imkoniyatlaridan kelib chiqib, fayl va undagi qatorni topib, batafsil tahlil qilish joyini ko'rsatish kerak. Kod yozib tekshirilayotgan paytda bu jarayon dasturlovchiga noqulaylik tug'dirishi mumkin. Ayniqsa, bir jamlanma kodda bir necha fayllarga murojaat bo'lsa.

Dasturlovchi uchun yanada qulaylik yaratish maqsadida, «**debugger;**» buyrug'i kiritilgan bo'lib, kodni tahlil qilinish talab etilsa, kerakli qatorga uni yozish kifoya. Kod bajarilishi jarayonida ushbu buyruqqa duch kelinganda amallar bajarilishidan to'xtab, brauzerning yuqorida bayon etilganidek tahlil muhiti ochiladi. Faqat tekshirish yakuniga yetgandan so'ng bu buyruqni o'chirishni unutmaslik kerak! (Aks holda oddiy foydalanuvchi uchun tushunarsiz yozuvlar ko'rinib qoladi.)

Ko'p hollarda JavaScript kodini bo'laklarga yaqqol ajratilmagan holda bitta qatorga siqib yozilganiga guvoh bo'lamiz. Albatta, boshqalar yozilgan kodni anglab, uni

o'zlashtirish imkoniyatini qiyinlashtirish bilan birga fayl hajmini ham sezilarli darajada kamaytiradi. Bunday «chalkashtirib ixchamlash»ni «obfuscation» deb yuritiladi. JavaScript fayllarini obfuskatsiya qiluvchi dasturlar juda ko'p, ularning mukammallari hatto har bir o'zgaruvchi va funksiya nomlarini ham almashtirib beradi. Shunday ekan, «JavaScript ochiq kod, birov algoritmimni o'zlashtirmasin», degan xavotir bilan keyinchalik o'zingiz ham tushunmaydigan darajada betartib kod yozish kerak emas.

Ko'p qirrali mukammal dasturiy ta'minot yaratish qoidalariga keyingi mavzularda yana batafsil to'xtalamiz. Shu o'rinda fayllar soni ko'payib ketishidan qo'rqmasdan, har bir kichik vazifani bajarish uchun alohida fayl yaratib, unda kodlar majmuasini hosil qilish, xatoni topishni qulaylashtirish bilan birga uning mohiyatini anglashda ham katta yordam berishini ta'kidlab o'tish joiz.

Yuqoridagi rasmda tasvirlangan muhitda ishlash yo'riqnomasi ushbu kitobda bayon etilmaydi. Negaki, dasturlashni o'rganmoqchi bo'lgan kishiga unday muhitlardan foydalanish ortiqcha qiyinchiliklarni keltirib chiqarmaydi. Kompyuterga buyruq berayotgan odam uni qanday bajarish kerakligini yaxshi biladi. Bajarilish jarayonini kuzatib borish uchun esa bu juda qulay vosita hisoblanadi.

Albatta, u kamchiliklarga ham ega. Masalan, ming marta takrorlash lozim bo'lgan kodda xatolik to'qqiz yuzinchi qadamdan keyin sodir bo'lsa, bir necha yuz marotaba qayta-qayta bir joyni behuda kuzatib borish ma'qul emas. Bunday holatlarda takrorlash operatori

ichiga qo'shimcha tekshiruvchi shart kiritib, dastur bajarilishini shu yerda to'xtatgin degan mazmunda qatorni belgilash lozim.

Brauzerlarning ilk talqinlarida bunday imkoniyatlar bo'lmagan, xatolikni topish uchun har qadamda «alert» yoki «console.log» ishlatishga to'g'ri kelgan. Ehtimol, bunday noqulaylik ko'pgina dasturchilarni oddiy hisobkitoblarni ham serverga yuborib, unda bajarib natija olishga undagandir. Lekin hozir JavaScriptda mukammal dasturiy ta'minot yaratish uchun barcha imkoniyatlar mavjud.

3.4.2. Error obyekti

Dastur bajarilish jarayonida xatolik yuzaga kelsa, u o'z omillari bilan Error obyektiga o'zlashtiriladi. Bundan tashqari, dasturlovchi o'z xohishiga qarab, quyidagi sintaksis asosida qo'shimcha xatolik obyekti ham yaratishi mumkin:

```
new Error([izoh[, fayl nomi[, satr raqami]]]);
```

Obyekt yaratilayotganda qaysi faylning nechanchi qatorida nima sababdan xatolik yuzaga kelganining izohi bilan ko'rsatish mumkin. JavaScriptning oxirgi talqinlarida «fayl nomi» va «satr raqami»ni kiritish tavsiya etilmaydi, baribir o'zi aniqlaganini inobatga oladi.

Error obyekti bilan ishlashdan avval, xatolar JavaScriptda necha xil turda bo'lishi bilan tanishib o'taylik:

- `EvalError` — Satrni bajarishdagi `eval` funksiya [2.6.2.] xatoligi. Bu funksiya ixtiyoriy satrni JavaScript kodi sifatida bajarilishini ta'minlaydi. Misol:

`eval("1 + 2");`	3
`eval("1 # 2");`	Kutilmagan sintaksis xato

- `InternalError` — ichki xato aniqlanmaganda tashqi joyda ko'rsatiladigan xato. Masalan, bitta funksiya chaqirilganda, o'sha joy **tashqi** maydon, funksiyaning tanasi esa **ichki** maydon hisoblanadi. Funksiyada o'ziga-o'zi davriy ravishda takroriy murojaat mavjud bo'lsa, «too much recursion» degan xatolik yuzaga keladi. Odatda, bu xato juda katta ma'lumotlar bilan ishlaganda yoki tinimsiz qaytarilaverganda sodir bo'ladi.

- `RangeError` — o'zgaruvchining qiymati nazarda tutilgan chegaradan chiqib ketgan hollarda qo'llaniladi.

- `ReferenceError` — e'lon qilinmagan o'zgaruvchiga murojaat bo'lganligini anglatadi.

- `SyntaxError` — kodda sintaksis xato uchraganda qo'llaniladi.

- `TypeError` — o'zgaruvchining turi xato bo'lganda.

- `URIError` — Berilgan ishorat noto'g'riligini bildirish uchun ishlatiladi. Odatda, lotin harflari, raqamlar va ruxsat etilgan ayrim ishoralardan tashqari belgilar yordamida ifodalangan ishoratni faqat shular yordamida kodlovchi `encodeURI` hamda uning aksincha yoyuvchi `decodeURI` funksiyalari bilan ishlaganda foydalaniladi.

Yuqorida keltirilgan xato turlari hammasi alohida obyekt bo'lib, xuddi «Error» singari yaratiladi. Dasturlovchi o'zi istagan izoh, fayl nomi va qator tartib raqamini berishi mumkin.

3.4.3. console obyekti

Konsol nimaligi haqida oldingi mavzularimizda tanishdik [1.2.] [2.1.4.]. Endi u bilan ishlaydigan maxsus biriktirilgan obyektga alohida to'xtalamiz. Uning muhim tamoyillari quyidagilardan iborat:

- **console.assert(***shart***[, *qiymatlar*])** — agar «shart» bajarilsa e'tiborsiz qoldiriladi, aks holda konsolda keltirilgan qiymatlarni ko'rsatgan holda xatolik haqida xabar chiqaradi. Aslida bu tamoyilni «timosti» (backend) yo'nalishida ishlovchi Web Workers boshqa maqsadda kiritgan bo'lsa ham, kodni tahlil qilishda undan samarali foydalanish mumkin.

- **console.clear()** — konsolni tozalaydi.

- **console.count([***nomlanish***])** — kodning ayni shu joyiga nechanchi marta murojaat etilayotganligini konsolga «nomlanish»ni ko'rsatgan holda chiqarib boradi. Bu takrorlanuvchi operatorlar qo'llanilganda aynan qachon xatolikka sabab bo'layotganini aniqlashda juda foydali hisoblanadi.

- **console.countReset([***nomlanish***])** — ko'rsatilgan «nomlanish»li sanagichning joriy qiymatini 0 ga o'zgartiradi. Agar «nomlanish» berilmagan bo'lsa, «nomlanish» berilmagan «count»lar uchun sanagichlarning qiymatlarini nolga tenglashtiradi.

- **console.debug(***qiymat***[, *qiymat*])** — «console.log» singari ishlaydi (uning muqobil shakli).

- **console.dir(***obyekt***)** — berilgan obyektning tarkibi bilan batafsil konsolga chiqaradi.

- **console.dirxml(***element***)** — berilgan HTML yoki XML elementning teglarini ichma-ich tartibda tushunarli tarzda chop etadi. Agar «element» veb-sahifadagi html/xml obyekt bo'lmasa, JavaScript obyekt sifatida «**console.dir**» singari namoyon etadi. Bu imkoniyat dasturiy yo'l bilan dinamik yaratilgan HTML elementlarini tahlil qilishda keng qo'llaniladi.

- **console.error(***qiymat***[, *qiymat*])** — berilgan qiymatlarni ko'rsatgan holda xatolik haqida xabar beradi. (Ba'zi brauzerlarda «**console.exception**» sifatida ham yozish mumkin, lekin qo'llash tavsiya etilmaydi.)

- **console.group([***nomlanish***])** — chiqayotgan xabarlarni («nomlanish» bo'yicha) guruhlab, ichma-ich tartib bilan o'qishga qulay joylashishini ta'minlaydi. Guruhlash boshlanayotganini anglatadi. Kodning turli joyida bir necha xil xabarlar chiqarish lozim bo'lsa, bir-biridan farqlash uchun bu juda qulay vositadir.

- **console.groupEnd([***nomlanish***])** — «nomlanish»li guruhlashning yakunini bildiradi. Bitta guruhni shunday yopmasdan boshqasini ochish tavsiya etilmaydi.

▪ **console.groupCollapsed([*nomlanish*])** — konsolga juda ko‘p xabarlar chiqarilayotgan paytda, kerakli xatolikni aniqlash uchun ba’zi guruhlarni yig‘ib qo‘yishga to‘g‘ri keladi. Har gal ayni paytda muhim bo‘lmagan bir qancha guruhlarni qo‘lda yig‘ishtirish ancha vaqt va diqqatni talab etadi. Shunday hollarda qaysi guruh avvaldan yig‘ilgan holda chop etilishi lozim bo‘lsa, «console.group» o‘rniga ushbu tamoyil ishlatiladi.

▪ **console.info(*qiymat*[, *qiymat*])** — «console.log» singari berilgan qiymatlarni konsolga chiqaradi, lekin xatboshida axborot ma’nosini bildiruvchi «**i**» shaklidagi nishoncha qo‘yilishi kerak. Ammo hozircha ba’zi brauzerlar buni qo‘llamasdan qiymatlarning o‘zini chiqarishi mumkin.

▪ **console.profile([*nomlanish*])** — yuqorida bayon etilganidek yig‘iladigan guruh emas, shunchaki izoh sifatida ajratilishi lozim bo‘lgan xabarlarning boshlanishini bildirish uchun qo‘llaniladi. Berilgan «nomlanish»dagi kesim (bo‘lak) boshida yozuv chiqaradi.

▪ **console.profileEnd([*nomlanish*])** — «nomlanish»-li kesim tugagani haqida xabar chiqaradi.

▪ **console.table(*obyekt* [, *ustunNomi*])** — berilgan obyektni jadval sifatida namoyon qiladi. Misol:

```
var jadval = [
  {"Ism": "Ali", "So‘z": "Olgan"},
  {"Ism": "Vali", "So‘z": "qarzing"},
  {"Ism": "G‘ani", "So‘z": "qani"}
];
console.table(jadval, "Ism");
```

(index) ▲	Ism
0	"Ali"
1	"Vali"
2	"G‘ani"

Ushbu funksiyaning qulayligi, obyektning indeksi bo‘yicha faqatgina kerakli qiymatlarini chiqara olish qobiliyati mavjudligidan tashqari, hosil bo‘lgan jadvalning ixtiyoriy ustunini o‘sish yoki kamayish tartibida saralash (▲) imkoniyati ham mavjudligidadir.

▪ **console.time([*nomlanish*])** — qandaydir amallarning bajarilish vaqtini kuzatishga to‘g‘ri kelsa, «nomlanish»li vaqt o‘lchagichdan foydalanish mumkin. Bitta veb-sahifada bir paytda o‘n mingtagacha turli vaqt o‘lchagichlar qo‘llash mumkin.

▪ **console.timeEnd([*nomlanish*])** — avvalgi qatordagi aynan shu «nomlanish»li «**console.time**» bajarilgan ondan to ushbu tamoyil uchraguncha o‘tgan vaqt millisoniyada konsolga chiqariladi. Serverlarga ma’lumot yuborib, undan qaytgan natija bo‘yicha amal bajaradigan JavaScript uchun so‘rovlarni tahlil qilishda bu juda muhim funksiya hisoblanadi.

▪ **console.trace()** — chaqirilgan funksiyalarni ketma-ket nomi va unga ko‘rsatkich bilan e’lon qiladi. Aytaylik, qaysidir fayldagi «a» nomli funksiya ichida ba’zida xatolik ko‘rsatyapti. Sinov jarayonida muammosiz to‘g‘ri ishlar edi. Shunday vaziyatda «a» funksiya ichiga «console.trace();» deb yoziladi va ushbu funksiyaga murojaat qilinguncha qaysi tashqi funksiyalar ishlaganligi ko‘riladi. Ko‘p hollarda brauzer xato deb ko‘rsatgan ichki funksiya to‘g‘ri algoritm bo‘yicha ishlaydi, lekin unga murojaat qiluvchi funksiya gohida kutilmagan qiymatlar berib yuboradi. Bunday hollarda chaqiruvchi funksiyalarning barchasini qulay ravishda

tartiblab, xatoni tezroq aniqlashda ushbu tamoyil katta xizmat qiladi.

- **console.warn(***qiymat*[, *qiymat*]) — «console.log» singari berilgan qiymatlarni konsolga chiqaradi, faqat boshiga ogohlantiruvchi nishoncha (⚠) qo'yadi.

Konsolga ma'lumot chiqarilar ekan, odatda, o'zgaruvchilarning qandaydir qiymatlarini kuzatib borish asosiy maqsad sanaladi. Yuqorida keltirilgan tamoyillarning aksariyati obyekt yoki satrning o'zini chiqarishga mo'ljallangan. Albatta, guruhlash va izoh yozish imkoniyatlari ham mavjud, lekin yetarli emas. O'zimizga xos bo'lgan izohlarni qiymatlar bilan uyg'unlashtirishimizda ayrim kutilmagan muammolarga duch kelishimiz mumkin. Masalan, songa satrni qo'shish bilan yuqorida tanishib o'tdik. O'zgaruvchini son sifatida kutilsa-yu, u ba'zan qo'shtirnoq ichida serverdan kelsa, doim qat'iy o'girish inobatga olinmagan bo'lsa, xatoliklar keltirib chiqarishi tabiiy. Bir nechta misollar ko'rib chiqaylik:

```
var a = 12, b = 10, c = '0';
console.log('A + B = ' + a + b);      A + B = 1210
console.log('A + B = ' , a + b);      A + B = 22
console.log('A + C = ' , a + c);      A + C = 120
console.log('A * C = ' + a * c);      A * C = 0
```

Ushbu misollardan ko'rinyaptiki, «c»ni qo'shganda satr sifatida biriktiryapti, ko'paytirishda esa son sifatida matematik amal bajaryapti.

Shu o'rinda ta'kidlab o'tish joizki, JavaScriptning va brauzerlarning eski talqinlarida songa satr sifatida

kelgan sonni qo'shganda — matematik amal, satrga sonni qo'shganda esa — qator ko'rinishida birlashtirish bajarilar edi. Shuning uchun doim matematik amallar bajarilayotganda o'zgaruvchini (parseFloat yoki parseInt yordamida [2.6.1.]) songa aylantirib olish tavsiya etiladi.

Konsolga satr chiqarilar ekan, odatda, u faqat dasturlovchi uchun, nega ortiqcha funksiyalar ishlatish kerak ekan degan o'rinli savol tug'ilishi tabiiy. Aslida, turlarni o'girmasdan «**o'zgartiriluvchi satr**»dan foydalanish mumkin. Bunda satr ichida o'zgaruvchining qiymati chiqishi kerak joyga «%» belgisi qo'yiladi va har biriga mos o'zgaruvchini vergul bilan ajratib, alohida omil sifatida berib boriladi. Faqat «%» yolg'iz qo'llanilmaydi, u bilan birlashtirilib quyidagi belgilar ishlatiladi:

- **%o** yoki **%O** — obyekt (yoyib ko'rish imkoniyati bor)
- **%s** — satr
- **%d** yoki **%i** — butun son
- **%f** — haqiqiy son

almashtirib qo'yish uchun joy ajratadi. Misol:

```
var k = 10, n = 3, m = {a: k, b: false};
console.log("«K»ning qiymati %d bo'lganda", k);
console.log("«M»ning qiymati %O bo'ldi.", m);
console.info("%d / %i = %f", k, n, k/n);
```

```
«K»ning qiymati 10 bo'lganda
«M»ning qiymati ▶{a: 10, b: false} bo'ldi.
10 / 3 = 3.3333333333333335
```

Umuman olganda, yaxlitlab chiqarish imkoniyati ham mavjud, lekin u ba'zi brauzerlarda ishlamagani uchun bu imkoniyat xususida batafsil to'xtalmaymiz.

3.4.4. BDD usulida avtomatik sinash

Katta hajmdagi murakkab dastur tuzilar ekan, qanday hollarda kutilmagan xatoliklar yuz berish mumkinligini oldindan aniqroq taxmin qilish ham mushkul. Kod yozilganda uning ishlashi berilgan ma'lum qiymatlar asosida sinab boriladi. Lekin unga qo'shimchalar kiritilgandan so'ng, ba'zi vaziyatlarda kutilmagan kamchiliklar yuzaga kelishi mumkin. Bunday nuqsonlar sababini aniq topish ba'zan juda ko'p vaqt va sarmoya yo'qotishga sabab bo'ladi. Shu kabi noxush holatlarning oldini olish maqsadida, o'zi tekshiradigan mexanizmni qo'llashga zaruriyat seziladi.

Hozirda «**mocha**», «**chai**» va «**sinon**» singari ko'plab xatolikka tekshiradigan kutubxonalar mavjud. Ular **BDD** (*Behavior Driven Development*) yo'nalishida ishlaydi. Ya'ni shunchaki kamchilikni topish bilan cheklanmasdan, uchta to'laqonli vazifani bajaradi:

1. **Izohlash** — qaysi kod yoki funksiyani tekshirayotganini anglatadi. Sintaksisi:

```
describe(«Izoh», function() { ... })
```

2. **Tavsiflash** — kod nima maqsadda yozilganini va qanday foydalanish kerakligi haqida ma'lumot

beradi. Buning uchun qiyidagi sintaksisda «batafsil ma'lumot» kiritiladi:

```
it(«batafsil ma'lumot», function() { ... })
```

3. **Sinash** — berilgan qiymatlar bo'yicha dastur kodi yoki funksiyaning to'g'ri natija qaytarishini tekshiradi. Unda qiymatlarni mantiqiy qiyoslovchi «**assert**»ning bir nechta funksiyalaridan foydalanish mumkin:

Funksiya sintaksisi	Tekshirish sharti
`assert(q)`	`q == true`
`assert.equal(q1, q2)`	`q1 == q2`
`assert.strictEqual(q1, q2)`	`q1 === q2`
`assert.notEqual(q1, q2)`	`q1 != q2`
`assert.notStrictEqual(q1,q2)`	`q1 !== q2`
`assert.isTrue(q)`	`q === true`
`assert.isFalse(q)`	`q === false`

Keltirilgan shartlarning birortasi bajarilmasa, mos xatolik haqida xabar chiqariladi.

JavaScriptda berkilish [2.3.2.] (**closure**) tushunchasi mavjud bo'lib, uni qo'llashda ko'pchilik dasturlovchilar ba'zi kamchiliklarga yo'l qo'yishi natijasida kutilmagan xatoliklarga duch kelishadi. Shu o'rinda ushbu atamani batafsil bayon etib, undagi xatolikni BDD yo'nalishida tekshirishga doir bir misol keltirish maqsadga muvofiq.

Berkilish — bitta dastur bo'lagida funksiya yaratib, unga murojaat qilib, natijasidan foydalanish.

Bu o'z-o'ziga qayta-qayta murojaat qiladigan funksiya emas. Undan anchagina sodda. Odatda, bitta funksiya ichida boshqa ichki funksiya yaratiladi va uni

chaqiriladi. Berkilish qo'llanilgan, ichma-ich joylashgan funksiyaga misol keltiramiz:

```
function ayir(a) {
   return function(b) {
      return a - b;
   }
}
ayir(120)(20)
```

100

Serverga ma'lumot yuborib, qaytgan natija bilan amal bajarish jarayonida ikkinchi qismni alohida ichki funksiyaga «berkitib» qo'llash JavaScriptda keng tatbiq etiladi, odatda, bunday funksiya «callback» deb yuritiladi. Berkilishning asl mohiyati shundan iboratki, funksiya chaqirilganda, u o'z faoliyatini tugatgandan so'ng, uning tanasida e'lon qilingan ichki o'zgaruvchilardan foydalanib bo'lmaydi. Aslida ular xotiradan o'chirilishi kerak, ammo ichki funksiya yaratilib, unga murojaat bo'lganda, tashqi funksiya o'z ishini tugatgandan keyin ham ushbu ichki ishlatilgan funksiyani qo'llash imkoniyati saqlanib qoladi. Berkilishga sabab bo'lgan bu ichki funksiya tanasida tashqi maydonda e'lon qilingan barcha o'zgaruvchilarni qo'llash imkoniyati ham mavjud. Shunday qilib, faoliyati tugatilgan funksiyadagi barcha ichki e'lon qilingan o'zgaruvchilar xotiradan o'chmay saqlanib turaveradi. Tashqaridan esa ularni qo'llash mumkin bo'lmaydi. Shuning uchun ham ba'zilar berkilishdan foydalanishni tavsiya etishmaydi.

Endi xatolik keltirib chiqarish mumkin bo'lgan berkitilgan funksiya yozamiz:


```
function marotaba(son) {
      // ...
      function orttir(ilk) {
            son *= ilk;
            return son;
      }
      return orttir;
}
marotaba(100)(12)                    1200
marotaba(100)(13)                    1300
var karra = marotaba(100);
karra(12)                            1200
karra(13)                            15600
```

Ushbu misolda «karra» o'zgaruvchisiga boshlang'ich qiymat bilan funksiyani o'zlashtirib olingandan so'ng, ikkinchi qiymat berilgan paytda, **berkilish** yuzaga kelyapti va saqlanib qolgan ichki o'zgaruvchi qiymati natijaga ta'sir etyapti. Ikkala ko'paytiriluvchini birdaniga berilgan holda bunday xatolik yuz bermaydi.

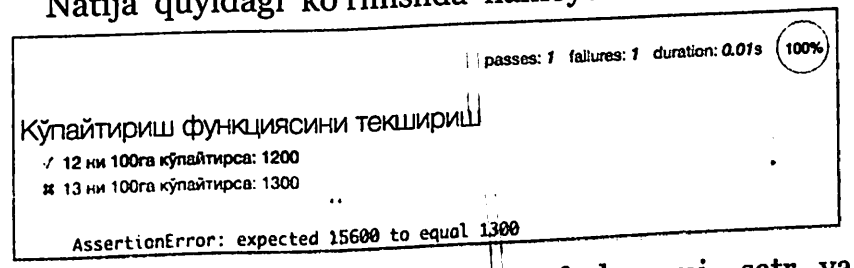
Endi BDD usulida o'zi tekshiradigan kod yozib «test.js» nomi bilan fayl sifatida saqlaymiz:

```
describe("Ko'paytirish funksiyasini sinash", function(){
   function sinash(x) {
      var qiymat = x * 100;
      it(x + " ni 100ga ko'paytirsa: " + qiymat,
         function() {
            assert.equal(karra(x), qiymat);
         }
      );
   }
   for (var x = 12; x < 14; x++) { sinash(x); }
});
```

Sinovni amalga oshirish uchun alohida veb-sahifa yaratamiz va unga shu «test.js»ni ulaymiz:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <!-- Mocha shamoyillarini yuklovchi fayl bog'lanadi -->
  <link rel="stylesheet"
    href="https://cdnjs.cloudflare.com/ajax/libs/mocha/2.1.0/mocha.css">
  <!-- Mocha kutubxonasi yuklanishi uchun -->
  <script
    src="https://cdnjs.cloudflare.com/ajax/libs/mocha/2.1.0/mocha.js">
  </script>
  <!-- BDD sinovga sozlash -->
  <script> mocha.setup('bdd'); </script>
  <script <!-- chai yordamchi kutubxonasini yuklash -->
    src="https://cdnjs.cloudflare.com/ajax/libs/chai/2.0.0/chai.js">
  </script>
  <!-- chaidagi kerakli assertni global aniqlab olamiz -->
  <script>
    var assert = chai.assert;
  </script>
</head>
<body>
  <script>
    function marotaba(son) {
      // ...
      function orttir(ilk) {
        son *= ilk;
        return son;
      }
      return orttir;
    }
    var ort = marotaba(100);
  </script>
  <!-- test.js ni biriktiramiz -->
  <script src="test.js"></script>
  <!-- natijalar quyidagi maydonga chiqadi -->
  <div id="mocha"></div>
  <!-- tahlilni ishga tushirish -->
  <script> mocha.run(); </script>
</body>
</html>
```

Natija quyidagi ko'rinishda namoyon bo'ladi:



Davomida xatolik yuz bergan fayl nomi, satr va undagi belgi o'rni ko'rsatiladi.

Amaliyotda aniqki, dasturning kamchiliklarini to'g'irlab, benuqson ishlashini ta'minlash uchun uni tuzishdan ko'ra ancha ko'p vaqt sarflanadi. Ushbu bo'limda keltirilgan imkoniyatlardan to'g'ri foydalanish dasturlovchiga vaqtini tejash uchun o'ta zarurdir.

4-BO'LIM

OBYEKTGA YO'NALTIRILGAN DASTURLASH

4.1. OYD haqida tushuncha

Avvalgi mavzular orqali obyekt tushunchasi bilan batafsil tanishdik. Tibbiyot muassasasi misolida hayotiy hamda bir qancha dasturda qanday qo'llanilishiga oid misollar ko'rib o'tdik. Endi butun dasturlash jarayonini shunday obyektlar asosida amalga oshirishga o'tamiz.

Obyektga yo'naltirilgan dasturlash (OYD) — dasturni muayyan andoza-uslublarga solib, boshqarish nuqtayi nazaridan qulay, keng ma'lumotlarni qamraydigan va qayta ishlaydigan obyektlar asosida qurishdir, u o'z navbatiga, to'rtta talabga javob berishi kerak: tarkiblash, qobiqlash, meroslash va turlichalash.

Mukammal dasturlash kitobimizning birinchi qismida keltirilgan veb-sahifaga element qo'yishni esga oladigan bo'lsak, masalan, hujjat ichiga DIV elementini qo'yish uchun maxsus teg yozilar edi. Endi shuni dasturiy usulda, berilgan alomatlarning qiymatlari bo'yicha ko'rsatilgan jiloda va kerakli o'rinda namoyon qilish algoritmini ko'rib chiqaylik.

Mukammal dasturlashning asosiy talablaridan biri — ayni maqsadni bajaruvchi kodni takror yozmaslik. Bitta

teg chiqaruvchi funksiya yozib, unga kerakli alomatlar berib foydalanishning o'zi yetarli emasmi degan savol tug'ilishi tabiiy. Bu faqat yagona maqsadga erishish uchun ehtimol kifoya qilar, lekin blokli DIV tegi o'rniga undan anchagina farq qiluvchi qatorli SPAN tegini akslantirish lozim bo'lsa, funksiyani qaytadan yozib chiqishga to'g'ri keladimi?!

Javob sifatida o'sha funksiyada hamma narsani inobatga olgan holda yozish tavsiya etilishi mumkin. Ammo dasturning asosiy qismida faqat blokli teg yaratilsa, funksiyani chaqirganda qancha ortiqcha tekshiruv bo'ladi? Hajmi katta funksiyani har gal brauzer tahlil qilib bajarishiga yana qancha vaqt va quvvat sarflanadi?!

Shuning uchun ham bitta aniq maqsadga yo'naltirilgan ixcham obyekt yaratish va kerak bo'lganda undan nusxa olib, funksionallarini kengaytirib borish maqsadga muvofiq. Ushbu usul nafaqat tez ishlashga sabab bo'ladi, balki xotiradan ham anchagina tejashga yordam beradi.

4.2. Obyekt yaratish usullari

JavaScriptda obyektlar qanday yaratilishini yana bir eslab o'taylik. Buning uch xil usuli mavjud.

1. **Qiymat berish orqali** [2.4.6.]. Avval tanishganimiz kabi, o'zgaruvchiga qiymat berish yo'li bilan oddiy obyekt yaratish mumkin:

```
var div = {
    border: "1px solid gray",
    color: "maroon",
    display: "flex",
    height: 100,
    margin: "4px 10px 6px",
    padding: "8px 12px",
    position: "static",
    width: 300
};
```

2. **«new» kalit so‘zi bilan** [2.4.7.] [2.6.8.]. Yuqoridagi obyektni quyidagicha hosil qilsa bo‘ladi:

```
var div = new Object();
div.border   = "1px solid gray";
div.color    = "maroon";
div.display  = "flex";
div.height   = 100;
div.margin   = "4px 10px 6px";
div.padding  = "8px 12px";
div.position = "static";
div.width    = 300;
```

Ikkinchi usul birinchisiga nisbatan noqulayroq tuyulsa ham aslida u qanday farqlanishini batafsil tanishib o‘tganmiz. Endi yanada murakkabroq, biroq o‘ziga xos imkoniyatlarga ega bo‘lgan boshqa usullarni ham bayon etamiz.

3. **Tarkiblash usuli yordamida**. Avvalgi mavzularimizda funksiyani tur sifatida ham tanishib o‘tdik [2.4.5.]. Umuman olganda, JavaScriptda funksiya alohida bir obyekt sifatida namoyon

bo‘ladi, «new» kalit so‘zi bilan obyekt yaratar ekanmiz, aslida tegishli funksiyani chaqirgan hisoblanamiz. Funksiyaning tana bo‘lagida **this** aynan shu obyektni anglatadi va unga tegishli alomatlarni biriktirish mumkin:

```
function divYasa(border, width, height) {
    this.border   = border;
    this.color    = "maroon";
    this.display  = "flex";
    this.height   = height;
    this.margin   = "4px 10px 6px";
    this.padding  = "8px 12px";
    this.position = "static";
    this.width    = width;
};
var div = new divYasa("1px solid gray", 300, 100);
```

Ushbu uchinchi usul yordamida endi obyektga yo‘naltirilgan dasturlashni tatbiq etishimiz mumkin. Obyekt ichida e’lon qilingan o‘zgaruvchilarni «**xususiyat**», funksiyani esa «**uslub**» deb ataymiz.

Umuman olganda, OYD turli manbalarda turfa xil bayon etiladi. Avvalgi adabiyotlarda uning faqatgina uchta talabi bor deb hisoblab, tarkiblash bilan qobiqlashni yagona sifatida qaralgan. Yangi manbalarda esa asl OYDning tub mohiyatini bilmagan holda uning talablari beshta va hattoki undan ham ko‘p deya hisoblashadi. Biz esa eng to‘g‘ri talablarni inobatga olamiz va ular aslida nima maqsadda qo‘llanilishini mavhum atamalardan xoli tarzda, sodda misollar yordamida tushuntirib o‘tamiz.

4.3. Tarkiblash (abstraction)

Biz yuqorida div elementini yaratish uchun zaruriy alomatlarga ega obyekt qanday yaratilishi bilan tanishib chiqdik. Birinchi kitobimizda yoritilgani kabi, unda barcha kerakli alomatlar uchun qiymatlar aniqlanmagan. Hozircha, biz faqat o'zgartirishimiz mumkin bo'lgan qiymatlarnigina kiritib, qolganlarini ahamiyatsiz deb qaraylik.

Oldingi mavzuda faqatgina o'z ehtiyojimizdan kelib chiqib, obektni aniqladik. Endi uning ba'zi qiymatlarini qandaydir hisob-kitob natijasida olishga to'g'ri kelsa, mos ichki funksiyalar yaratamiz. Masalan, eni va bo'yi tashqi element o'lchamlaridan kelib chiqadigan bo'lsa, «getWidth» va «getHeight» nomlanishda ichki funksiya yaratishimiz mumkin. HTML element yaratilib bo'lgandan keyin, uning o'lchamini o'zgartirishga to'g'ri kelsa, yana qo'shimcha «setWidth» va «setHeight» funksiya kiritamiz. Aslida obyekt ichida «width» va «height»lar mavjud edi, ortiqcha funksiyalarga biror ehtiyoj yo'q.

Agar ularning odatiy qiymati maxsus hisob-kitoblar bilan o'zgartirilgan bo'lsa, bizning obyektdan foydalanayotgan dasturlovchi uning boshlang'ich qiymatini ko'rishi ham, o'zgartira olishi ham kerak emas. Kalit so'z «this» bilan xususiyati aniqlangan bo'lsa [4.2.3.], obyekt o'zlashtirilgandan keyin; ularning qiymatini ko'rish ham, o'zgartirish ham mumkin. Bunday imkoniyatni cheklash uchun «var» yoki «let» yordamida oddiy e'lon qilish kerak. (Boshqa dasturlash tillarida «public» va «private» kalit so'zlaridan foydalaniladi.)

Faqat obyekt ichidagina ko'rinadigan xususiyat va uslublarni «**yashirin**», tashqaridan ham undan foydalanish mumkin bo'lganlarini «**oshkor**» alomatlar deb yuritamiz.

Tarkiblash (abstraction) – obyektni yaratilish maqsadi nuqtayi nazaridan undagi barcha yashirin va oshkor xususiyat va uslublarni e'lon qilish. Bunda uslublarni aniqlovchi funksiyalarning faqat nomi va omillarini to'la bayon etish nazarda tutiladi, tana qismiga e'tibor qaratilmaydi.

Blokli tegni eni va bo'yi o'lchamlari bo'yicha aniqlaydigan obyektga misol keltiramiz:

```
function maydonYarat(eni, boyi) {
  var border  = "1px solid gray",
      display = "flex",
      margin  = "4px 10px 6px",
      padding = "8px 12px",
      bgColor = "lightGray"
      color   = "black";
  this.width  = eni;
  this.height = boyi;

  var getParentWidth  = function()  { }

  var getParentHeight = function() { }

  this.maydonKorsat = function () {
    var s, w = getParentWidth(),
      h = getParentHeight(),
      b = Math.round((h-parseInt(this.height))/2),
      e = Math.round((w-parseInt(this.width))/2),
```

```
        s = '<div style="'
          + 'border:   ' + border  + '; '
          + 'margin:   ' + b + 'px ' + e + 'px; '
          + 'display:  ' + display + '; '
          + 'padding:  ' + padding + '; '
          + 'color:    ' + color   + '; '
          + 'background: ' + bgColor + '; '
          + 'width: ' + (
            typeof this.width == 'number'
              ? this.width += 'px'
              : this.width) + '; '
          + 'height: ' + (
            typeof this.height == 'number'
              ? this.height += 'px'
              : this.height) + '; '
          + '"></div>';
      document.write(s); // tegni namoyon etish
    }
};
var div = new maydonYarat(400, 200);
div.maydonKorsat();
```

Ushbu misolda «maydonKorsat» uslubi to'la bayon etildi, aslida tarkiblash jarayonida bu shart emas, xuddi «getParentWidth» va «getParentHeight» uslublari singari yoritilishi kifoya. Ularning ichiga mos ravishda «return document.body.clientWidth;» bilan birga «return document.body.clientHeight;» yozib, OYD qanday ishlashini tekshirish mumkin. Keyingi mavzularimizda «**document**» obyekti bilan batafsil tanishganimizda bu misol to'laqonli tushunarli bo'ladi. Hozircha

uning qanday ishlashi ahamiyatga ega emas, nima ish bajarishga mo'ljallanganligi muhim.

Tashqi element o'lchami butunlay boshqacha hisob-kitoblar orqali ham aniqlanishi mumkin. Bu faqat ushbu obyekt yaratuvchisining shaxsiy kodi sanaladi, undan foydalanuvchi boshqa dasturchilar uchun uning ahamiyati bo'lmasligi lozim. Bu yondashuv bir ishni yana qaytadan behuda yozib chiqishning oldi olinishi bilan birga maxfiy saqlanishi lozim kodlarni yashirishga ham imkon beradi. JavaScript ochiq kodli bo'lgani uchun xufyonalik foyda bermaydi, ammo «**qurama**» (kompilyatsiya) qilinuvchi dasturiy ta'minotlarda buni inobatga olish lozim, albatta.

Misol oxirida obyektning «maydonKorsat» uslubini chaqirdik, u veb-sahifada tashqi element o'rtasiga berilgan o'lchamdagi to'g'ri to'rtburchak hosil qildi. U funksiya sifatida yozilgan. Funksiya sifatida yozilgan boshqa uslub «getParentWidth» yoki «getParentHeight»ga murojaat qilsak, xato yuzaga keladi. Chunki ular yashirin uslub hisoblanib, obyekt tashqarisida ularni qo'llash mumkin emas. Xuddi shuningdek «div.height» va «div.width» singari yozib foyladanish mumkin, lekin «div.border» va «div.margin» kabilar esa aniqlanmagan qiymat sifatida qaraladi.

Demak, tarkiblash — obyektning ma'lum reja asosida qolipini yaratish hisoblanar ekan.

4.4. Qobiqlash (encapsulation)

Juda oddiy bir misol ko'raylik, unda chegara qalinligi aniqlanishi lozim bo'lsin. Berilgan son 0 bo'lsa, chegara ko'rinmasligi tushunarli, lekin manfiy qiymat kiritiladigan bo'lsa, mantiqsizlik kelib chiqadi. Tarkiblash jarayonida obyektdan foydalanuvchi dasturchiga ba'zi xususiyatlarga murojaat qilishni butunlay cheklab qo'yish mumkinligini o'rgandik, lekin unda qiymatiga qarab tekshiruvdan o'tkazish imkoniyati mavjud emas. Xususiyatni aniqlash paytida shart qo'yib bo'lmaydi, buning uchun uslub sifatida maxsus funksiya yozishga to'g'ri keladi. Bu kabi to'laqonli dasturlash qobiqlash jarayonida amalga oshiriladi.

Qobiqlash (encapsulation) — berilgan qiymatlar, o'zgaruvchilar va ular ustida amal bajaradigan funksiyalarni o'zaro mutanosib ishlashini ta'minlab, zaruriy ma'lumotlar va hisob-kitoblarni o'zida jamlab, kerakli cheklovlarni inobatga oladigan yaxlit dasturiy ta'minot sifatida taqdim etish.

Tarkiblash jarayonida uslublarning tana qismiga e'tibor qaratilmagan edi, qobiqlashda u to'liq (va imkon qadar to'g'ri) yozilgan bo'lishi kerak. Mukammal dasturlashda bitta vazifa bitta obyekt yoki uning bir uslubida amalga oshiriladi va u ko'p marta foydalanishga mo'ljallangan bo'ladi. Agar beriladigan turli xil qiymatlar uchun obyektning ichki kodini o'zgartiraverishga ehtiyoj bo'laversa, demak OYD talablari bajarilmagan hisoblanadi.

Yagona qiymatga ega, shu qiymatni ma'lum chegarada o'zgartirish va o'qib olish mumkin bo'lgan kichik obyekt yaratamiz:

```
function qalin() {
  var bw = 1;

  this.olQalin = function () {
    return bw;
  }
  this.berQalin = function(n) {
    if (parseInt(n) === n) {
      if (n < 0) alert('Musbat son kiriting.');
      else
        if (n >= 20) alert('20dan kichik bo'lsin.');
        else bw = n;
    } else alert('Butun son kiriting.');
  }
};
var q = new qalin();
alert('Qalinligi: ' + q.olQalin()); // Qalinligi: 1
q.berQalin(12);
alert('Qalinligi: ' + q.olQalin()); // Qalinligi: 12
```

Keltirilgan shartlardan tashqaridagi qiymat berilganda ogohlantiruv yozuvi chiqadi.

Obyektning o'zi ham, undagi ichki uslublar ham «function» yordamida e'lon qilinilishi biroz tushunishni qiyinlashtirishi mumkin. Kalit so'z «this» qaysi «function»ning bo'lagida yozilgan bo'lsa, unga nisbatan emas, qaysi obyekt tanasida yozilgan bo'lsa, shu yaratilayotgan obyektga ko'rsatgich hisoblanadi. Masalan, yuqoridagi misolda «berQalin» funksiyasi-

ning ichiga «olQalin();» deb yozilsa, «topilmadi» degan xatolik yuzaga keladi, «this.olQalin();» singari yozilsa, kutilgan qiymatni oladi.

JavaScript «**constructor**» xususiyatini har bir yaratilgan obyektga biriktiradi. Uning yordamida obyekt nomini (name), qaysi funksiya uni chaqirganini (caller), qanday omillar qabul qilganini (arguments) va ular sonini (length) aniqlash imkoniyati mavjud. Obyekt ichida esa «this.constructor.name» singari murojaat qilish mumkin, tashqarida esa «this» o'rniga u o'zlashtirilgan o'zgaruvchi yoziladi. Bu kengayib boruvchi turli-tuman obyekt yaratib, ularni boshqarishda juda qo'l keladi.

Obyekt alomatlarini yashirin yoki oshkor qilish tarkiblash jarayonida, ularning qiymati bo'yicha cheklash esa qobiqlash jarayonida amalga oshiriladi. Avvalgi adabiyotlarda, yuqorida ta'kidlaganimiz kabi, bu ikkisi bir-biri bilan umumlashtirib qaraladi. Yuqorida esa ular o'rtasidagi aniq chegara belgilab berildi. Albatta, sohalar rivojlanish jarayonida alohida qismlarga bo'linib qaralishi tabiiy hol.

4.5. Meroslash (inheritance)

Tarkiblash va qobiqlash jarayonida tayyor dasturiy ta'minotning kerakli qismi yaratilar ekan, yana ortiqcha talabga hojat bormi degan savol tug'iladi. Oddiy funksional dasturlashda bu yetarli bo'lishi mumkin, ammo

OYD uchun bu kamlik qiladi. Yana mavjud dastur bo'lagidan mohiyatini o'zgartirmagan holda boshqa qonuniyatlar bilan kerakli joylarda qo'llash imkoniyati mavjud bo'lishi lozim.

Meroslash (inheritance) — mavjud obyekt asosida, uning barcha xususiyat va uslublarini o'zlashtirgan holda, yangi qo'shimcha imkoniyatlarga ega obyekt yaratish.

Ya'ni mavjud obyektning nusxasi yaratiladi va unga kerakli xususiyat va uslublar qo'shiladi yoki mavjudlari o'zgartiriladi. Mavjudlarini o'zgartirish uchun aynan o'sha nomdagisini yana e'lon qilish kifoya. Obyektda bitta nomdagi alomatlar takror uchrasa, oxirgi yozilgani inobatga olinadi. Aslida nom — bu xotiradagi ma'lum ko'rsatkich, o'zgaruvchi xotiraga o'zlashtirilayotganda ayni nomdagisi eskisining o'rniga yoziladi.

Misol tariqasida «maydon» nomli obyekt yaratamiz hamda undan eni va bo'yini meros olib, chegarasini o'zgartirib, yangi rangda boshqa obyekt hosil qilamiz:

```
function maydon() {
   this.width  = 400;                    // obyekt
   this.height = 200;                    // yaratildi
   this.border = "1px solid gray";
};
var m = new maydon();                    // o'zlashtirildi
m.border                                 "1px solid gray"

function shamoyil(rang, foni) {
   maydon.call(this);
   this.color   = rang;
   this.bgColor = foni;
```

```
    this.border = "1px dashed blue";
}
var r = new shamoyil("red", "aqua");
r.border                                    "1px dashed blue"
r.height                                    200
```

Ushbu misoldagi «shamoyil» ichida «width» va «height»lar e'lon qilinmagan bo'lsa ham «maydon» obyektidan olinmoqda, «border» xususiyati esa o'zgartirilmoqda. Buning uchun «maydon» to'laligicha «call» uslubi yordamida ikkinchi obyektga o'zlashtirildi.

Avvalgi mavzularda «timsollash» (**prototype**) [2.6.8.] yordamida ixtiyoriy obyektga istagancha xususiyat yoki uslub qo'shish mumkinligini ko'rgan edik. JavaScriptning shu imkoniyati orqali meroslashni amalga oshirish mumkin emasmi?!

Yuqorida keltirilgan «maydon» obyekti misolida tatbiq etib ko'raylik:

```
function maydon() {
    this.width  = 400;
    this.height = 200;
    this.border = "1px solid gray";
};
maydon.prototype.color = "red";
maydon.prototype.width = 1024;
var m = new maydon();
m.color                                     "red"
m.height                                    200
m.width                                     400
```

Obyekt keyin kiritilgan rangni oldi, lekin kengligi o'zgarmadi. Bu esa meroslash talabiga ziddir. Demak,

prototype yordamida mavjud bo'lmagan xususiyat va uslublarni qo'shishda foydalanish mumkin, o'zgartirish uchun esa birmuncha boshqacharoq yo'l tutiladi.

4.6. Turlichalash (polymorphism)

Dasturlashdagi qo'shish amali matematikadagidan farq qiladi. Masalan: satrlarni qo'shish, sonlarni qo'shish, vaqtlarni qo'shish bir-biridan butkul boshqacha algoritmlar asosida amalga oshiriladi. Umuman olganda, mantiq bir xil, jarayon turlicha. Ikkita qiymatni berib qo'shish talab qilinsa, uning o'z xususiyatiga qarab tegishli amal bajarib natija qaytarish jarayonini OYDda turlichalash deb tushunish mumkin.

Turlichalash (polymorphism) — mohiyati jihatdan bir-biriga yaqin ma'noga ega, yagona nomdagi masalalarni har xil amallar bajarish orqali umumlashgan natijani hosil qilish jarayoni.

OYDda bitta masalani yechish uchun yagona obyektning tegishli uslubiga murojaat qilish kifoya, aks holda bir necha alohida-alohida funksiyalar yozib chiqishga to'g'ri keladi. Chunki kiritilayotgan omillar ham har xil turda bo'lishi mumkin, aksariyat funksional dasturlashda turlarning aniq ko'rsatilishi talab etiladi.

Birinchi kitobdan bizga yaxshi tanishki, CSS yordamida elementning chegaralarini aniqlovchi ushbu «border-color», «border-width» va «border-style»

qiymatlarini umumlashtirib yagona «border» xususiyatiga istalganlarini, ixtiyoriy ketma-ketlikda berish mumkin. Qiymatlari qat'iy belgilangan tartibda berilishi shart mavjud obyekt asosida yangisini erkin ravishda son yoki satr qabul qilib ishlay oladigan tarzda meros-lab-turlichalab yaratishga misol keltiramiz:

```
function chegara(qalin, turi, rang) {
  this.width = qalin ? qalin : '1px';
  this.style = turi  ? turi  : 'solid';
  this.color = rang  ? rang  : 'gray';
  this.borderStyle = function () {
    console.log('border: %s %s %s',
      this.width, this.style, this.color);
  }
};

function yoyliChegara(chiziq, yoy) {
  var q, r, t, c, i, e, b
   // chegara qalinligini anglatuvchi so'zlar
  eni = ["inherit", "initial", "medium", "thick",
         "thin", "unset"];
   // chiziq ko'rinishini anglatuvchi so'zlar
  tur = ["dashed","dotted","inset","none","ridge",
     "hidden","outset","groove","solid","double"];
  if (typeof chiziq == 'number') q = chiziq;
  else {   // satrni bo'laklab mos qiymatlarini olish
    c = chiziq.trim().split(/\s+/);
    for (i of c) {
      b = false;
      if (parseInt(i) >= 0) { q=i; b=1; }
      if (parseInt(i) == i) { q=i+"px"; b=1; }
      for (e of eni) if (e == i) { q=i; b=1; }
      for (e of tur) if (e == i) { t=i; b=1; }
      if (!b)  r=i;
    }
  }
  chegara.call(this, q, t, r);  // meroslash
  this.radius = parseInt(yoy) == yoy ? yoy + "px" : yoy;
  if (yoy) // «yoy» berilganda, qaytadan uslubni aniqlash
    this.borderStyle = function () {
      console.log('border: %s %s %s; border-radius: %s',
        this.width, this.style,
        this.color, this.radius);
    }
}
var b = new yoyliChegara(2);  // asl obyekt bo'yicha
var r = new yoyliChegara("3 red","7pt"); //merosi
b.borderStyle(); // border: 2 solid gray;
r.borderStyle(); // border: 3px solid red; border-radius: 7pt
```

Ushbu misolda berilgan qiymatlarga qarab birinchisida asl obyektdagi, ikkinchisida esa merosxo'ridagi uslub ishlayapti.

Turlichalashning ikkita muhim jihati mavjud. Avvalo berilgan qiymatlarning soni va turi o'zgarsa ham, unga moslashib ishlash. Ikkinchisi — bir nomdagi uslublardan keraklisini bajarish.

Foydalanib kelinayotgan, sinovlardan o'tgan mavjud obyektni faqat birgina o'zgacha qiymat uchun takomillashtirishga to'g'ri kelsa, unga umuman o'zgartirish kiritilmaydi, merosxo'rini yaratib, ortiqcha kod yozmasdan faqat kerakli uslubigina qayta aniqlanadi.

Amaliyotda bitta dasturning ma'lum talqini foydalanishga joriy etilgan bo'lsa, ba'zi o'zgartirishlar bilan

yangisini ommaga havola etilganda, odatda, shunday yo'l tutiladi. Bu bilan eski talqinida ishlovchilar ham, yangi talqinini o'rnatib olganlar ham muammoga duch kelishmaydi. Aks holda hayotga ko'p uchrab turuvchi vaziyat – dasturning yangi talqini chiqqandan so'ng, eskisining xatolik berish holati boshlanadi. Hamma ham bir paytda yuklab olmasligi va umuman yangilashni xohlamasligi mumkinligini inobatga olish lozim. Boshqa fayl, kutubxona yoki freymvorkdagi mavjud obyektni takomillashtirib, o'zingizga moslashtirish uchun shunday yo'l tutish maqsadga mavofiq.

4.7. OYDning «class ... extends» shakli

Obyekt yaratish jarayonida funksiyaning ichida yana funksiyalar yozish ko'pchilikka g'alati tuyulib, ba'zi tushunmovchiliklarga sabab bo'lishi mumkin. Yana «this» kalit so'zi obyektga emas, undagi ichki funksiyaga ko'rsatkich, chunki uning bo'lagida yozilgan, degan yanglishishlar ham keltirib chiqarish ehtimoli bor. Shunday vaziyatda chalkashlik yuzaga kelmasligi uchun «this»ni qo'llamasdan JavaScriptda qanday OYD tuzish mumkinligini keltirib o'tishni lozim topdik. Kelgusi mavzularimizda «this»ni boshqa hollarda ham qo'llanilishi bilan tanishib o'tamiz. Quyidagi ikki obyekt aynan bir xil vazifani bajaradi:


```
function myValue() {
   var value = 12;
   this.setValue = function (v) { value = v; };
   this.getValue = function () { return value; };
}
function myValue() {
   var value = 12;
   return {
      setValue: function (v) { value = v; } ,
      getValue: function () { return value; }
   }
}
```

JavaScriptdagi OYDda oshkor alomatlarni «this»ga biriktirish o'rniga obyekt sifatida qaytarsa bo'ladi. Ammo obyektni funksiya so'zi bilan e'lon qilish boshqa dasturlash tillaridan xabari bor odam uchun noodatiy tuyuladi. Sababi, boshqalarda «class» kalit so'zidan foydalaniladi. Nega JavaScriptda butkul boshqacha bo'lishi kerak?!

ECMAScript 6 (2015) dan boshlab yana bir qancha boshqa tillardagi qulayliklar JavaScriptda ham keng tatbiq etila boshladi. Masalan, funksiya omiliga odatiy qiymat berish mumkin bo'ldi:

```
function myValue(v = 12) {
   console.log(v);
}
```

Funksiyani «function» kalit so'zisiz o'zgarmas sifatida e'lon qilish imkoniyati ham yaratildi. Agar bunda tana qismida «**this**» ishlatilsa, joriy obyekt emas, global obyekt (window) tushuniladi:

```
const sin = (x) => { return Math.sin(x); };
```

Faqat bitta operatordan foydalanib natija qaytariladigan bo'lsa, gulli qavs qo'llash ham shart emas:

```
salom = (ism) => "Assalomu alaykum! " + ism;
```

Agar faqatgina bitta kiritiluvchi omil beriladigan bo'lsa, qavs qo'ymaslik ham mumkin:

```
salom = ism => "Assalomu alaykum! " + ism;
```

Funksiyalarni bu kabi e'lon qilishda «=>» qo'llanilganligi bois «**nayza**» nomini olgan. Nayza funksiyalarini qo'llash yozuvda juda qulaylik keltiradi.

Eng muhim o'zgarishlardan biri OYDda «class» kalit so'zidan foydalanish mumkin bo'ldi. Unda «function» ishlatishdan butkul voz kechilgan. Sodda sintaksisi bilan tanishish uchun nuqtaning koordinatalarini aniqlovchi misol keltiramiz:

```
class nuqta { x = 41.3; y = 69.7; }
var n = new nuqta();
n.x
```
41.3

Avval tanishgan sintaksis bo'yicha ushbu misol quyidagicha ko'rinishda bo'lar edi:

```
function nuqta() { this.x = 41.3; this.y = 69.7; }
var n = new nuqta();
n.x
```
41.3

Solishtirilsa, bir qarashda u qadar katta o'zgarish sezilmaydi, qisqartirish faqatgina qavs va «this» kalit so'zida ko'zga tashlanadi. Aslida «this» chiqarib tashlanmagan, aksincha, tarkiblash jarayonida oshkor xusu-

siyatni e'lon qilishda qo'llaniladi. Shu o'rinda sinflash tizimi bo'yicha OYD tuzishning o'ziga xos jihatlari bilan tanishib chiqaylik:

1. **Turg'un xususiyat va uslublarni e'lon qilish.** Yaratilgan obyektni bitta o'zgaruvchiga avval o'zlashtirib olib keyin alomatlariga murojaat qilish doim ham qulay hisoblanmaydi. To'g'ridan-to'g'ri kerakli uslub yoki xususiyatini ishlatish ko'pincha qo'l keladi. Buning uchun alomatlarni «**static**» kalit so'zi yordamida turg'un deb e'lon qilinadi:

```
class nuqta { static x = 4.3; static y = 6.1; }
nuqta.y
```
6.1

2. **Loyihalash bo'limini alohida tuzish.** Kiritiluvchi qiymatlarni yuklash hamda oshkor va yashirin xususiyatlarni aniqlash dastavval shu bo'limda yoziladi. Oshkor alomatlar «this» kalit so'zi orqali berilsa, yashirin o'zgaruvchilarning birinchi belgisi «#» (*panjara*) bo'lishi shart (*hozircha ba'zi brauzerlar yashirin alomatni qo'llamasligi mumkin*). Loyihalar bo'limi «**constructor**» kalit so'zi bilan funksiya sifatida yoziladi va undagi barcha amallar obyektga murojaat bo'lgan zahoti ishga tushadi. Albatta, buning uchun «new» yordamida o'zlashtirib olishga to'g'ri keladi. Misol:

```
class ortaQiymat {
   constructor(x, y) {
      this.value1 = x;
      this.value2 = y;
```

```
        console.log('Yuklandi');
    }
}
var o = new ortaQiymat(12, 6);          Yuklandi
```

3. **Natija olish** uchun oddiygina «**get**» kalit so'zidan foydalaniladi. Uslub ko'rinishida e'lon qilinsa ham, xususiyat sifatida qiymatni oladi. Unda qiymat «return» yordamida qaytarilishi nazarda tutilishi shart:

```
class qiymatlar {
  static get pi() {
    return Math.round(Math.PI * 100)/100;
  }
}
qiymatlar.pi()                          Xato
qiymatlar.pi                            3.14
```

4. **Qiymat berish** uslubi «**set**» kalit so'zi orqali e'lon qilinib, unda yagona qiymat berish mumkin:

```
class Harorat {
  Selsiy;
  static set Farengeyt(t) {
    this.Selsiy = Math.round((t-32)/0.18) / 10;
    console.log(`${t}°F = ${this.Selsiy}°C`);
  }
}
Harorat.Farengeyt = 80;          // 80°F = 26.7°C
```

Ushbu misolda o'zgaruvchilarni UTF–8 kodlamasini qo'llashi hamda konsolga o'zgaruvchilarni matn ichida qanday chop etilishi ham namoyish etildi.

5. **Uslub nomining o'zi yoziladi**, «function» so'zisiz, kiritiladigan omillari bilan qo'llaniladi:

```
class hisob {
   static ayir(x, y) { return x - y; }
}
hisob.ayir(12, 2);                      10
```

6. **Meros olish** uchun «**extends**» kalit so'zi maxsus sintaksis bilan qo'llaniladi:

```
class <merosObject> extends <aslObject> {…}
```

7. **Asl obyektga murojaat** «**super**» kalit so'zi yordamida amalga oshiriladi.

Endi yuqorida bayon etilgan imkoniyatlarning ayrimlaridan foydalanib, turlichalashni qo'llab, mavjud «hisoblash» nomli obyektdan uning yangi merosxo'rini yaratamiz. Asl obyekt ichida o'rta kvadratik qiymat faqat ikkitagina son uchun aniqlangan bo'lsin. Biz uni ixtiyoriy miqdorda berilgan sonlar uchun umumlashtiramiz. Matematik formulasi quyidagicha:

$$S = \sqrt{\frac{a_1^2 + a_2^2 + \cdots + a_n^2}{n}}$$

Meros olgan obyekt nomi «hisob» bo'lsin, kodi:

```
class hisoblash { // qo'shimcha kodlar bo'lishi mumkin
  static ortaKvadrat(a, b) {
    console.log("Asl obyekt bajardi.");
    return Math.sqrt( (a ** 2 + b ** 2) / 2 );
  }
}
```

```
class hisob extends hisoblash { // qo'shimcha kodlar
  static ortaKvadrat(...a) {
    var s = 0, e, m = [...a];
    if (m.length == 2)
      return super.ortaKvadrat(m[0], m[1]);
    else {
      for (e of m) s += e ** 2;
      console.log("Merosxo'r obyekt ishladi.");
      return Math.sqrt(s / m.length);
    }
  }
}
hisob.ortaKvadrat(3,4);          Asl obyekt bajardi.
                                 3.5355339059327378
hisob.ortaKvadrat(3,4,7,9); Merosxo'r obyekt ishladi.
                                 6.224949798994366
```

Yuqorida bayon etilgan bilimlar asosida JavaScript dasturlash tilida to'laqonli OYD yaratish mumkin.

5-BO'LIM

VEB-SAHIFA OBYEKTLARI

5.1. window obyekti

JavaScriptda window obyekti global umumlashtiruvchi vazifani bajarishini [2.6.] ta'kidlab o'tgan edik. Aslida xabarlarni chiqaruvchi funksiyalar [2.1.] ham shu obyektning uslubi, biriktirilgan obyektlar [2.6.] esa global «**window**»ning alomatlari hisoblanadi:

```
window.alert(window.Math.PI);  3.141592653589793
```

Ushbu window obyekti «faqat o'qish uchun» yaratilgan bo'lib, qiymatini o'zgartirish aslo mumkin emas, ya'ni unga yangi qiymat berish aslida xatolik keltirib chiqarmasa ham befoyda urinish, xolos:

```
window = 1;                      1
window.Math.E                    2.718281828459045
```

5.1.1. window xususiyatlari

Quyida window obyektiga tegishli asosiy xususiyatlar bilan tanishib o'tamiz. Qiymatini o'zgartirish mumkin bo'lganlari kvadrat (■), faqat o'qish uchun mo'ljallanganlari esa aylana (○) shakli bilan belgilanadi. Lokal

o‘zgaruvchilar bilan chalkashtirmaslik uchun «window» so‘ziga biriktirib qo‘llash tavsiya etiladi.

- **window.applicationCache** – tarmoqdan uzilgan paytlarda ba’zi kerakli qiymatlarni yodda saqlab turish uchun foydalaniladigan obyekt.
- **window.crypto** – brauzerning o‘zida mavjud shifrlash algoritmini obyektga tatbiq etuvchi majmua.
- **window.devicePixelRatio** – birinchi kitobda CSSda bayon etilgan **px** o‘lchami ekrandagi nechta asl nuqtani ifodalashini anglatadi (*HTML va CSS haqidagi birinchi kitobda bayon etilgan – J.A.*).
- **window.frameElement** – agar kod iframe yoki unga muqobil obyekt ichida yozilgan bo‘lsa, shu `<iframe>` yoki `<object>`ni qaytaradi, aks holda `null` qabul qiladi.
- **window.frames** – joriy sahifani va undagi mavjud freymlarni jadval sifatida qaytaradi. Umuman olganda:

```
window.frames === window                    true
```

- **window.fullScreen** – sahifa to‘liq ekranda namoyish etilayotganligini aniqlovchi mantiqiy qiymat qaytaradi. Afsuski, bu xususiyat «Chrome» va «Internet Explorer»larda ishlamaydi.
- **window.innerHeight** – brauzerdagi veb-sahifa namoyon bo‘ladigan qismining balandligi. Unda yotiq surgich (scroll) uchun joy ham inobatga olinadi. Yangi qiymat beriladigan bo‘lsa, uni o‘zida saqlaydi, ammo sahifaning hajmini o‘zgartira olishiga kafolat yo‘q (o‘zgartirishi yoki o‘zgartira olmasligi ham mumkin).

- **window.innerWidth** – Xuddi «innerHeight» singari, faqat enining o‘lchamini aniqlaydi va tik surgich (scroll) hajmini ham o‘z ichiga oladi.
- **window.name** – oyna nomi.
- **window.isSecureContext** – dastur kodi xavfsizligini ta’minlovchi funksionallar mavjudligini tekshiradi.
- **window.length** – sahifada nechta freym mavjudligini ko‘rsatadi. Aniq mavjud bo‘lgan ichki freymga uning tartib raqami bo‘yicha murojaat qilish mumkin: `window[0]`, `window[1]`, ...
- **window.locationbar** – asosiy vazifasi ishorat yoziladigan darcha ko‘rinayotganligini tekshiradigan obyektni qaytarishdan iborat.
- **window.localStorage** – veb-sayt uchun berilgan qiymatlarni saqlovchi obyekt. Undagi ma’lumotlar brauzer qayta yuklanganda ham, sahifa yangilanganda ham o‘chmaydi. Qadam-baqadam ishlovchi sahifalarda server bilan aloqa uzilib qolish xavfi bo‘lganda, joriy bosqichdagi qiymatlarni saqlab borish uchun juda qulay vosita hisoblanadi. Qurilma qayta yuklangandan keyin va hatto uzoq muddat o‘chiq bo‘lgandan so‘ng ham ishni to‘xtalgan vaziyatdan davom ettirishni ta’minlay oladi. Quyida uning imkoniyatlari bilan misollar yordamida tanishamiz:

```
window.localStorage.length // o‘zgaruvchilar soni   0
window.localStorage.setItem('son',123);
window.localStorage.length // qo‘shgandan so‘ng     1
window.localStorage.son                             123
window.localStorage.setItem('qiymat', 'OK');        "OK"
```

```
window.localStorage.length                          2
window.localStorage.removeItem('son');
window.localStorage.length // o'chirgandan so'ng    1
window.localStorage.clear(); // (barini) tozalash
window.localStorage.length                          0
```

- **window.menubar** – asosiy vazifasi brauzer menyusi ko'rinayotganini tekshiruvchi obyektni qaytarishdan iborat.

- **window.opener** – agar joriy sahifa avvalgisida JavaScript kodi yozish orqali ochilgan bo'lsa va ikkisi ham yagona saytga tegishli bo'lsa (*bu boshqa saytni buzishga yo'l qo'ymaslik uchun*), yangi sahifada ochuvchining «window» obyektiga murojaatni ta'minlaydi.

- **window.outerHeight** – brauzerning to'liq tashqi o'lchamini aniqlovchi to'g'ri to'rtburchak balandligi.

- **window.outerWidth** – brauzerning to'liq tashqi chegaralari bilan aniqlanuvchi yuzasining eni.

- **window.pageXOffset** yoki **window.scrollX** – surgich yotiq yo'nalishda nechta nuqta birligida siljiganligini aniqlaydi.

- **window.pageYOffset** yoki **window.scrollY** – surgich tik yo'nalishda nechta nuqta birligida siljiganligini ko'rsatadi.

- **window.parent** – freym yoki unga muqobil obyekt ichidan turib, tashqi sahifa «window»iga murojaatni ta'minlaydi.

- **window.personalbar** – asosiy vazifasi xos tugmalar paneli ko'rinayotganini tekshiradigan obyektni aniqlashdan iborat (`window.personalbar.visible`).

- **window.screen** – ekran o'lchamlarini, egriligi hamda undagi ranglar turfa xilligini aniqlovchi qiymatlarga ega obyekt.

- **window.scrollbars** – aylantirib surgichlar (tik va yotig'i) ko'rinishi mumkinligini tekshiradigan obyekt (`window.scrollbars.visible`).

- **window.scrollMaxX** va **window.scrollMaxY** – surgichning yotiq va tik yo'nalishlardagi eng so'nggi nuqtasini aniqlaydi. Afsuski, bu soddagina xususiyatlar xrom, safari kabi ba'zi brauzerlarda ishlamasligi mumkin. Bu sahifani oxirigacha surib ko'rishni dasturiy yo'l bilan amalga oshirishni ta'minlaydi, bunday imkoniyatlar hujjat obyektida [5.2.] ham bor.

- **window.scrollX** – yotiq ravishda nechta nuqta birligida surgich o'ngga siljiganligi.

- **window.scrollY** – tik ravishda nechta nuqta birligida pastga surgichning siljiganligi.

- **window.self** – o'ziga ko'rsatkich, o'z-o'ziga boshqa nom bilan bunday murojaatning ta'minlanishi ba'zan (muqobilsiz) zaruriy vosita hisoblanadi [7.4.].

- **window.sessionStorage** – «localStorage» singari qiymatlarni saqlaydigan obyekt, faqat brauzer qayta ishga tushganda barcha ma'lumotlar yo'qoladi.

- **window.status** – brauzerning quyida turuvchi holat satriga chiqadigan matn. Shu o'rinda ta'kidlab o'tish joizki, zamonaviy brauzerlar holat satrni doimiy ko'rinib turadigan vaziyatda tutishdan voz kechishgan, faqat ishoratlar faollashganda asl manzilini quyida ko'rsatadi,

xolos, boshqa paytda u yashirin bo'ladi. Shuning uchun ushbu xususiyatdan endilikda foydalanish tavsiya etilmaydi.

- **window.statusbar** — asosiy vazifasi holat satri ko'rinayotganini tekshiradigan obyektni aniqlashdan iborat (`window.statusbar.visible`). Lekin yuqorida bayon etilganidek, u doim ko'rinib turmasa ham «rost» qiymat qabul qiladi, bu ishoratlar faollashganda quyi satrda uning ko'rinishini anglatadi.
- **window.toolbar** — sozlash paneli ko'rinayotganini tekshiruvchi obyekt (`toolbar.visible`).
- **window.top** — eng tashqi «window» obyektiga ko'rsatkich. Agar bir necha ichma-ich freymlar ishlatilgan bo'lsa, nechanchi qatlamdagi freym ichida bo'lishidan qat'iy nazar, (veb) sahifaning asosiy «window» obyektini aniqlashda qo'llaniladi.
- **window.window** — o'z-o'ziga murojaat. O'ziga qayta ko'rsatkich yaratish bir nechta amallarni bitta ifoda sifatida ketma-ket yozish uchun qulaylik ham yaratadi. Masalan, freym mavjud bo'lmagan sahifada quyidagi misolda «name»dan oldingi nuqta bilan ajratilgan ixtiyoriy qism olib tashlansa ham ayni qiymatni aniqlayveradi:

```
window.parent.top.window.name
```


5.1.2. window uslublari

O'ziga xos uslublarini «window» kalit so'zi ishlatmasdan, oddiy funksiya sifatida qo'llash ham mumkin. Quyida oldingi mavzularimizda yoritilmagan asosiy uslublarini sanab o'tamiz.

- **btoa(*matn*)** — berilgan «matn»ni «**base64**» texnologiyasini qo'llagan holda ajratuvchi belgilardan xoli shifrlangan satrga o'giradi. (Afsuski, bu standart funksiya faqat Latin1 kodlamasini qo'llaydi.)
- **atob(*satr*)** — aksincha, «**base64**» yordamida shifrlangan «satr»ni asl holiga qaytaradi. Misol:

```
btoa('JavaScript kitobi.')      "SmF2YVNjcmlwdCBraXRvYmku"
atob('SmF2YVNjcmlwdCBraXRvYmku') "JavaScript kitobi."
```

- **open(*ishorat*, *oynaNomi*, [*qiymatlar*])** — berilgan ma'lum «qiymatlar» asosida yangi oyna yaratishni ta'minlaydi. Odatda, brauzer uchun alohida sahifa (tab) ochadi. Agar ayni nom qayta berilsa, ochiq turgan sahifa yangilanadi. Misol:

```
if (typeof wo == "undefined" || !wo || wo.closed)
   wo = window.open("http://yurt.uz/", "sahifa");
```

- **close()** — agar sahifa «open» orqali JavaScript yordamida ochilgan bo'lsa, uni yopadi. Dasturiy ta'minotning bir qismini alohida sahifa sifatida ochib, zaruriy paytda yopish uchun qo'llaniladi. Foydalanuvchi brauzerda o'zi ochgan boshqa sahifalarni yopib yuborish xavfi bartaraf etilgan.

▪ **stop()** – sahifa yuklanishini to'xtatadi.

▪ **setInterval(***satr, millisoniya***)** – berilgan satrni har ko'rsatilgan millisoniya o'tgandan so'ng **eval** singari [2.6.2.] JavaScript kodi sifatida takroriy ishga tushiraveradi. Har berilgan vaqt o'tganda ayni kodni qayta bajartiraverish uchun bu juda ham ajoyib imkoniyat. Lekin nozik jihati shundaki, u alohida oqim sifatida faoliyat qilganligi bois funksiya ichida chaqirilganda uning ichki lokal o'zgaruvchilarini o'ziga yuklab olmaydi. Shuning uchun global o'zgaruvchilar yoki OYD imkoniyatlaridan foydalanish tavsiya etiladi [7.3.]. Bu albatta noqulaylik paydo qiladi. Ushbu noqulaylikni bartaraf etish uchun «satr» o'rniga funksiya yozish ham mumkin. Funksiyadagi omillar qiymatlari millisoniyadan so'ng beriladi:

```
var intervalID = setInterval(
  function(a, b) {
    var t = new Date();
    console.log(a + t.toLocaleTimeString() + b);
  },
  1000,      // har bir soniyada takrorlash uchun
  'Vaqt: ', // a, birinchi omil qiymati
  ' soniya' // b, funksiyada ko'rsatilgancha qiymat beriladi
); // «Vaqt: 05:24:34 soniya» kabi har soniyada chiqadi
```

▪ **clearInterval(***tartibRaqam***)** – berilgan «tartib raqam»li «setInterval» faoliyatini bekor qiladi.

▪ **setTimeout(***satr***[, *millisoniya*])** – faoliyati jihatdan «setInterval» bilan bir xil, ammo ko'rsatilgan vaqt

o'tgandan keyin shunda faqat bir marta ishga tushirilishini ta'minlaydi. Aslida kompyuterda uzluksiz va cheksiz degan tushunchalar yo'q. Kutish jarayonida hodisalarni har 3500 microsoniyasida tekshiradi. Bunga yana o'z protsessoridagi oqimlarni yo'naltirish faoliyati ham ta'sir qiladi. Shuning uchun aynan ko'rsatilgan millisoniyada emas, biroz kechroq chaqirilishi mumkin. Umuman olganda, inson soniyasiga 24 ta tasvir ko'ra olishini inobatga olinsa, bu farq sezilarli darajada bo'lmaydi. Toq va hattoki o'nlik sondagi aniqlikda millisoniya kiritish maqsadga muvofiq hisoblanmaydi.

▪ **clearTimeout(***tartibRaqam***)** – kiritilgan «tartib raqam»li «setTimeout»ni bekor qiladi.

▪ **find(***satr, sezgir, ortda, qator, to'liq, freym, darcha***)** – sahifadan berilgan «satr»ni izlaydi. Undan keyin faqat mantiqiy qiymatlar beriladi. Katta-kichik harflar qat'iy «sezgir»ligini ta'minlash, joriy o'rindan «ortda»gi matnni ham inobatga olish, satr to'liq faqat bitta «qator»da yozilganini tekshirish, so'zni «to'liq» (ma'lum qismini emas) izlanishi, hatto «freym»da bo'lsa ham topish imkonini qo'shish, brauzer ruxsat bersa, «CTRL+F» bosilganda chiqadigan «darcha»ni chaqirish ham mumkin. Agar topa olmasa, «yolg'on» qiymat qaytaradi. Topsa, «rost» qiymat qabul qilish bilan birga satrni belgilab ko'rsatadi. Ammo ushbu uslub hali to'liq amaliyotga tatbiq etilmagani uchun ayrim brauzerlarda topilgan satr belgilanmasligi yoki darcha ko'rinmasligi ehtimoli (hozircha) mavjud.

▫ **focus()** – joriy sahifani faollashtiradi. (tab tugmasini yoki sichqonchani bosib tanlangani singari)

▫ **blur()** – nofaol tarzga o'tkazadi, «focus»ning aksi.

▫ **getComputedStyle(*element*[, *holat*])** – ko'rsatilgan HTML elementining jamlangan CSS tarkibini yuklab oladi. «Jamlangan tarkib» shundan iboratki, unda dasturchi tomonidan ko'rsatilan elementning shamoyillari bilan birgalikda brauzer odatiy holda o'zi aniqlaydigan alomatlar ham qiymati bilan umumlashtiriladi. Birinchi kitobimizda CSS bilan batafsil tanishtirib o'tilgan. Undagi bilimlarni inobatga olgan holda bitta oddiy misol keltiramiz:

```
getComputedStyle(h2, 'after').color "rgb(51,51,51)"
```

Bunda «h2» tegning nomi emas, HTML element yuklangan o'zgaruvchi [5.2.]. Faqat FireFox uchun dasturchi tomonidan ko'rsatilgan shamoyillarni e'tiborsiz qoldirib, brauzerning odatiy qiymatlari bo'yicha elementning umumlashtirilgan CSSini yuklab oladigan «getDefaultComputedStyle()» funksiyasi ham bor.

▪ **getSelection()** – sahifada foydalanuvchi tomonidan belgilangan matnni (uni o'zida saqlovchi HTML elementi bilan) mukammal obyekt sifatida qaytaradi. HTML elementi JavaScript obyektida qanday ifodalanishi bilan keyingi mavzularimizda batafsil tanishamiz [5.2.], hozircha esa belgilangan matnning o'zini olish uchun quyidagicha kod yozish kifoya:

```
getSelection().toString()     "belgilangan matn"
```


▫ **print()** – sahifani chop etish oynasini chaqiradi. Odatda, chop etilayotganda varaqning quyisida ishorat ham qo'shilib qoladi. Uni olib tashlash uchun, birinchi kitobda yoritilgani kabi, sahifaning bosh (head) qismida aynan chop etish uchun maxsus CSS kiritilishi kerak:

```
<style type="text/css" media="print">
  @page {
    size: auto; /* sahifa o'lchami (A4 varaqqa) moslanadi */
    margin: 0; /* (URL chiqish uchun) hoshiya qoldirmaydi */
  }
</style>
```

▫ **requestIdleCallback(*funksiya*[, *omillar*])** – bu sinov tariqasida kiritilgan texnologiya bo'lib, uning maqsadi ko'rsatilgan funksiyalarni navbat bilan ishga tushishini ta'minlash. Ko'rsatilishi shart bo'lmagan «omillar»da hozircha faqat timeout berilishi mumkin. Ushbu millisoniya tugaguncha tegishli funksiyani chaqirishga vaqt yetmagan bo'lsa ham keyingi oraliqda bari-bir ishga tushiraveradi. Bu tushunarsiz algoritm ehtimol kelajakda takomillashtirilar. Hozircha uning imkoniyatlarini bemalol «setTimeout» yordamida amalga oshirish mumkin.

▫ **cancelIdleCallback(*navbatRaqami*)** – berilgan navbat raqamini hosil qilgan «requestIdleCallback» faoliyatini bekor qilish.

▫ **scroll(*X*, *Y*)** – sahifaning ko'rinadigan qismi chap yuqori nuqtasini berilgan koordinataga surish.

▪ **scrollBy(*X*, *Y*)** – sahifaning ko'rinadigan qismini unga mos berilgan nuqtalar birligida surish.

▪ **scrollTo(*omillar*)** – ko'rsatilgan joygacha sahifaning ko'rinadigan qismini surish, bunda «omillar» obyekti sifatida quyidagicha indeksli qiymatlar berilishi mumkin:

◊ **top** – tik yo'nalishda, **y** – koordinatasi;

◊ **left** – yotiq yo'nalishda, **x** – koordinatasi;

◊ **behavior** – qanday surilishni aniqlovchi satr. Odatda u «**auto**» qiymat qabul qiladi. Agar surilish ohista bo'lishi kerak bo'lsa «**smooth**», darhol o'tish lozim bo'lsa «**instant**» qiymati beriladi.

Omillar sifatida ham eniga, ham bo'yiga qiymat berishga to'g'ri kelsa, ortiqcha yozuvdan qochish maqsadida «**scrollTo(*X*, *Y*)**» sifatida qo'llash mumkin. Bunda «**scroll(*X*, *Y*)**»dan farqi bo'lmaydi. Misol:

```
scrollTo( { top: 300, behavior: 'smooth' } );
scrollTo( 200, 2000 );
```

5.1.3. Hodisalar

Birinchi kitobimizda HTML elementlariga hodisalar qo'shib JavaScript funksiyalariga bog'lash mumkinligi bayon etilgan. Unda sanab o'tilgan hodisalarni (veb)

sahifa uchun ham biriktirish, ya'ni ularga JavaScript funksiya bog'lash mumkin. Ayni bir hodisa uchun bir nechta funksiya yozilgan bo'lsa, ularning oxirgisi inobatga olinadi.

Sahifaga oid hodisalarning ishga tushish shartlari:

▪ **onafterprint** – chop etish oynasi yopilgandan keyin («print()» ga qarang) ishlaydi;

▪ **onbeforeprint** – chop etish oynasi chiqishdan aval ishlaydi;

▪ **onunload** – tark etilgan paytda: sahifa yopilganda, qayta yuklashda, boshqa ishoratga o'tishda ishlaydi. Bu payt xabarlarni (alert) ko'rsatish befoyda, sababi brauzer o'zining ogohlantiruvchi tanlov oynasini chiqaradi.

▪ **onbeforeunload** – tark etish oldidan;

▪ **onblur** – nofaol holatga o'tganda;

▪ **onchange** – darchalarda o'zgarish yuz berganda;

▪ **onclick** – sichqoncha chertilganda (bosilganda);

▪ **ondblclick** – sichqoncha juft (2 marta) chertilgan paytda;

▪ **onclose** – oyna (sahifa) yopilganda;

▪ **oncontextmenu** – sichqoncha o'ng tugmasi chertilib, ro'yxat oynasi (menyu) ochilganda;

▪ **onfocus** – faollashganda ishga tushadi;

- **ondeviceorientation** — ekran holati: tik yoki yotiqligi o'zgargan holda ishlaydi;

- **onerror** — xatolik yuz berganda ishlaydi; bu uslub sahifadagi barcha xatoliklarni ham tutib olavermaydi. Ularning turlari, sodir bo'ladigan joy va vaziyatlariga qarab o'ziga xos yo'l tutishga to'g'ri keladi.

- **onhashchange** — ishoratdagi panjara (#) belgisidan keyingi yozuv o'zgarganda ishlaydi. bunda sahifa qayta yuklanmaydi, birinchi kitobda bayon etilganidek, faqat «langar tashlangan joyga boradi».

- **onmessage** — boshqa sahifaga ma'lumot yuborilganda ishlaydi [7.2.];

- **onoffline** — server bilan aloqa uzilganda;

- **ononline** — server bilan aloqa tiklanganda;

- **onpagehide** va **onpageshow** — sahifa yopilganda emas, mavjudligi saqlangan holda shunchaki yashirilganda va qayta ko'rsatilganda ishlashi kerak, ammo hali bu hodisalar amaliyotga tatbiq etilmagan.

- **onpopstate** — brauzerdagi oldingi yoki keyingi sahifaga o'tishni ta'minlovchi tugma bosilganda ishlashi kerak, umid qilamizki, yaqin vaqtlarda brauzerlar uni qo'llashni boshlaydi;

- **onstorage** — «localStorage» [5.1.1.] qiymatlari o'zgartirilsa ishlashi kerak, ammo ko'p brauzerlar buni tatbiq etishni lozim hisoblashmayapti.

- **onresize** — sahifa (brauzer) o'lchami o'zgarganda;

Ushbu hodisalarga quyidagi misolda ko'rsatilganidek qiymat sifatida yangi funksiya yozib ko'rsatilishi yoki mavjud qiymatning nomi berilishi mumkin:

```
window.onresize = function(event) {
  console.log('O'lcham ' + window.innerWidth
    + 'x' + window.innerHeight + ' ga o'zgardi');
};
```

Buni hodisani «tutib qolish» deb ham yuritiladi. Ya'ni hamma hodisalar doim eshitib (tekshirilib) turiladi, qaysi biriga funksiya biriktirilgan bo'lsa, uni bajaradi. Shu nuqtayi nazardan, muqobil ravishda, hodisaga «eshituvchi» qo'shishni ta'minlovchi uslubdan foydalanish mumkin. Unda fursatni anglatuvchi «**on**» old qo'shimcha ishlatilmaydi. Sintaksisi:

```
window.addEventListener('hodisa', funksiya, omillar);
```

Quyidagi ikkita misoldagi hodisani «ushlab qolish» teng kuchli muqobillar hisoblanadi:

```
window.onoffline = (event) => {
  console.log("Server bilan aloqa uzildi!");
};
window.addEventListener('online', function(event) {
  console.log("Server bilan aloqa tiklandi.");
});
```

Hodisalar bilan ishlashni mukammal tasavvur qilish uchun, konsolda ushbu ikki misolni yozib, kompyuterni tarmoqdan uzib-ulab sinab ko'rish mumkin.

Oddiy qiymat berishdan «addEventListener»ning farqi shundaki, unda bitta hodisaga bir qancha funksiya biriktirilishi mumkin. Barchasi tartib bilan ketma-ket bajariladi. Katta dasturiy ta'minot yaratilayotganda bitta kodni bir necha marta ishga tushirishga to'g'ri kelsa, avval ayni hodisa qo'shilmaganligini doim tekshirish ortiqcha kod yozishni taqozo etadi. Bunday vaziyatlarning oldini olish maqsadida, takroriy biriktirishga yo'l qo'ymaslik uchun, «omillar» sifatida «`{once: true}`» obyektini kiritish kerak.

Ushbu mavzuda faqat window obyekti uchun qo'llaniladigan asosiy hodisalar bilan tanishdik. Ularning aksariyatini boshqa HTML elementlari uchun ham ishlatish mumkin. Yuqoridagi misolda keltirilgan «event» obyekti bilan batafsil tanishish uchun birinchi kitobda bayon etilganlarni inobatga olgan holda ichma-ich joylashgan HTML elementlarining har biriga bir xil hodisa biriktiramiz:

```
  <script language='javascript'>
    function alertTag(elm) {
      alert(elm.tagName);
    }
  </script>
<!-- … -->
  <aside onclick="alertTag(this)">ASIDE
    <div onclick="alertTag(this)">DIV
      <p onclick="alertTag(this)">P</p>
    </div>
  </aside>
<!-- … -->
```


Ushbu misolda div elementiga chertilganda (sichqoncha bosilganda), ekranga «DIV» xabari chiqqandan so'ng, ortidan «ASIDE» ham chiqadi. «P» elementiga chertilsa; «P», «DIV» va «ASIDE»lar ketma-ket chiqadi. Sababi, ichki obyektda yuz beradigan har bir hodisa, tashqisida ham sodir bo'ladi. Bu jarayonni «qalqish» («**bubbling**») deb yuritiladi. Qalqish yo'nalishi eng ichkisidan tashqi elementlar tomon tartiblangan. Elementning tartib mavqeini o'zgartirishda «addEventListener» yordamida hodisa biriktirilayotganda «omillar» sifatida «`false`» yoki «`{passive: true}`» obyektini berish lozim.

Hodisalar o'ziga xosligi bo'yicha, qachon, qayerda sodir bo'lganligiga qarab quyidagi turlarga bo'linadi:

	Fayllarga oid
`cached`	yuklangan va xotirada saqlangan
`error`	yuklash imkoni bo'lmadi
`abort`	yuklash jarayoni to'xtatildi
`load`	yuklash (muvaffaqiyatli) yakunlandi
`beforeunload`	yuklash amalga oshishidan avval
`unload`	yuklanmoqda
	Tarmoqqa oid
`online`	Brauzer tarmoqqa (serverga) ulandi
`offline`	Brauzer tarmoqdan (serverdan) uzildi
	Faollik
`focus`	Element faollashdi
`blur`	Element nofaol bo'ldi
	CSS xususiyatlari bo'yicha
`animationstart`	Harakatlanish boshlandi
`animationend`	Harakatlanish yakunlandi
`animationiteration`	Harakatlanish takrorlanmoqda

`transitionstart`	CSS xususiyatiga o'tish boshlandi
`transitioncancel`	... o'tish bekor qilindi
`transitionend`	... o'tish yakunlandi
`transitionrun`	... o'tishdan (oldingi kutishdan) avval

Soket[2] bo'yicha

`open`	WebSocket bog'landi
`message`	Ma'lumot olindi
`error`	Aloqa uzildi (*ma'lumot to'la olinmadi*)
`close`	Aloqa yo'lagi berkitildi

FORM (HTML elementi) **bo'yicha**

`reset`	Formadagi qiymatlar bekor qilindi
`submit`	Qiymatlar (serverga) yuborildi

Chop etish

`beforeprint`	Chop etish oynasi ochilishidan avval
`afterprint`	Chop etish oynasi yopilganda

Ovoz bilan matn kiritish qurilmasiga oid

`compositionstart`	Matn yig'ilyapti
`compositionupdate`	Matnga harflar qo'shilyapti
`compositionend`	Matn tayyor bo'ldi yoki bekor qilindi

Tarkibini ko'rsatish holatining o'zgarishi bo'yicha

`fullscreenchange`	Element yoki sahifa to'la ekranda namoyishga o'tsa yoki o'z holiga qaytsa
`fullscreenerror`	To'liq ekran egallashiga cheklov bo'lsa
`resize`	Element o'lchami o'zgarsa
`scroll`	Surgich bilan tarkibi siljitilsa

---

[2] **Soket** — ma'noviy jihatdan **lahim** (bir joydan boshqa aniq yerga eltish uchun yashirin yer osti yo'li) so'ziga to'g'ri keladi. Server bilan mijoz orasidagi bog'lanishni alohida yo'lakda (port orqali) ta'minlash uchun qo'llaniladi.

Qisqa xotira bo'yicha

`cut`	qisqa xotiraga ko'chirib (qirqib) olindi
`copy`	qisqa xotiraga nusxasi olindi
`paste`	qisqa xotiradan nusxasi qo'yildi

Tugmadon bo'yicha

`keydown`	Ixtiyoriy tugma bosildi
`keypress`	Funksional (`Shift`, `Fn`, `CapsLock`, ...) hisoblanmagan tugma bosib turilganda
`keyup`	Tugma qo'yib yuborilganda

Sichqonchaga oid

`auxclick`	Asosiy bo'lmagan tugmasi chertilganda
`click`	Asosiy tugmasi chertilganda
`contextmenu`	Sichqonchaning o'ng (menyu chiqaruvchi) tugmasi chertilganda
`dblclick`	Asosiy tugma ikki marta chertilganda
`mousedown`	Asosiy tugma bosib turilganda
`mouseenter`	Ko'rsatgich (HTML) obyektga kirganda
`mouseleave`	Ko'rsatkich obyektdan chiqqanda
`mousemove`	Ko'rsatgich obyekt ustida yurayotganda
`mouseover`	Ko'rsatkich (HTML) obyekt yoki uning ichki elementlari ustida yurayotganda
`mouseout`	Ko'rsatkich obyekt yoki uning ichki elementlarining biridan chiqqanda
`mouseup`	Asosiy tugma qo'yib yuborilganda
`pointerlockchange`	Ko'rsatkich qulflansa yoki bo'shatilsa
`pointerlockerror`	Ko'rsatkichni qulflash imkoni bo'lmasa
`select`	Matn belgilanganda (juft chertilsa)
`wheel`	Sichqoncha aylantirgichi surilganda

Sudrash holati bo'yicha

`drag`	Belgilangan matn yoki element sudralayotganda (har 3500 microsoniyada)
`dragend`	Sudrash nihoyasida (qo'yib yuborilsa)

`dragenter`	Sudraluvchi «manzilga kirganda»
`dragstart`	Sudrash boshlanganda
`dragleave`	Sudraluvchi «manzil»dan chiqqanda
`dragover`	Sudraluvchi «manzil» ustida
`drop`	Sudraluvchi «manzil»ga tashlanganda
	Media bo'yicha
`audioprocess`	ScriptProcessorNode – Kiruvchi qisqa xotira faoliyat yuritishga tayyor bo'ldi
`canplay`	Ijro uchun yetarlicha qism yuklanganda
`canplaythrough`	Qisqa xotiraga boshqa yuklanmasdan oxirigacha ijroni ta'minlay olsa
`complete`	**`OfflineAudioContext`** o'chirilganda
`durationchange`	**`duration`** xususiyati qiymati o'zgarsa
`emptied`	Fayllar ro'yxati bo'shaganda
`ended`	Ijro (ro'yxat) yakuniga yetganda
`loadeddata`	Ilk kadr yuklandi
`loadedmetadata`	Tavsif fayllar yuklanib bo'ldi
`pause`	Ijro to'xtatib turildi
`play`	Ijro boshlandi (davom ettirildi)
`playing`	To'xtatildi yoki faylning navbatdagi qismi yuklanib ijroga tayyor bo'ldi
`ratechange`	Ijro tezligi o'zgardi
`seeked`	Izlash (o'tkazilganda) nihoyasiga yetdi
`seeking`	O'tkazilganda izlash boshlandi
`stalled`	Ijrochi media ma'lumotlarini olayotganda to'satdan faylni topa olmasa
`suspend`	Media yuklanishi to'xtatildi
`timeupdate`	Media joriy vaqti o'zgardi
`volumechange`	Ovoz balandligi o'zgartirildi
`waiting`	O'z vaqtiga yetarlicha ma'lumot yuklana olmayotgani uchun to'xtab qoldi

Ushbu ro'yxatda keltirilganlardan tashqari yana bir qancha ommalashmagan, ayrim HTML elementlarining

o'ziga xos hodisalari ham mavjud. Hodisani aniqlovchi obyekt qaysi turga mansubligiga qarab o'ziga xos uslub va xususiyatlar qabul qiladi. Quyida ularning eng muhimlarining ro'yxati keltiriladi (o'zgartirish mumkin bo'lganlari kvadrat bilan, faqat o'qish uchun mo'ljallanganlari aylana belgisi bilan berilgan):

- **`bubbles`** — hodisa «qalqish» orqali paydo bo'lganmi deb tekshiruvchi mantiqiy qiymat.
- **`cancelBubble`** — qalqishning navbatdagi elementga ta'sirini bekor qiladi, mantiqiy qiymat.
- **`cancelable`** — bekor qilish mumkinligini ko'rsatuvchi mantiqiy qiymat.
- **`composed`** — qalqish jarayoni sodir bo'lishiga imkoniyat mavjudligini aniqlaydi, mantiqiy qiymat.
- **`composedPath()`** — joriy HTML elementdan boshlab uning barcha tashqi elementlarining JavaScript obyektlari sifatida yagona jadvalga ketma-ket yig'adi.
- **`currentTarget`** — hodisa ayni paytda sodir bo'layotgan element obyekti. Jarayon tugagandan so'ng null qaytaradi.
- **`target`** — qalqish jarayoni qaysi elementdan boshlanganligini aniqlaydi yoki joriy elementni qaytaradi.
- **`timeStamp`** — hodisa yaratilgan lahzadan u sodir bo'lgan paytgacha o'tgan millisoniya, haqiqiy son.
- **`type`** — hodisa nomi, satr.
- **`isTrusted`** — hodisa foydalanuvchi tomonidan sodir etilganligini aniqlovchi mantiqiy qiymat. Masalan,

brauzerda ishorat kiritish orqali sahifa ochilganda – «rost», dasturiy kod bilan «open»ni qo'llab paydo etilsa, «yolg'on» qiymat qaytaradi.

Quyida faqat sichqoncha harakatiga oid asosiy xususiyatlarni keltiramiz:

- **altKey, ctrlKey, shiftKey** — chertilganda mos funksional tugma bosib turilganligini aniqlovchi mantiqiy qiymat.
- **button** — aynan qaysi tugma bosilganini anglatuvchi 0 dan 4 gacha oraliqdagi butun son qaytaradi.
  - **0** – asosiy (chap) tugma
  - **1** – qo'shimcha (surgich) tugmasi
  - **2** – ikkinchi (o'ng) tugma
  - **3** – avvalgi sahifaga o'tkazuvchi tugma
  - **4** – keyingi sahifaga o'tkazuvchi tugma
- **screenX, screenY** – kompyuter ekraniga nisbatan koordinatalari.
- **clientX, clientY** – brauzer tanasiga nisbatan koordinatalari.
- **movementX, movementY** – oxirgi «mousemove» hodisaga nisbatan siljishlar.
- **offsetX, offsetY** – joriy HTML element chegarasiga nisbatan koordinatalar.
- **pageX, pageY** – sahifa tarkibiga ko'ra, surishlar hisobga olingan holdagi koordinatalar.

Ko'rsatkich harakatiga mos HTML elementlar yaratish uchun ushbu xususiyatlar keng qo'llaniladi. Satr sichqoncha va «Shift» tugmasini bosish yordamida belgilangan bo'lsa, ko'rsatgich joylashgan koordinatasini chiqaruvchi misol keltiramiz:

```
elm.addEventListener('click', function(hodisa) {
  if (hodisa.shiftKey) {
    alert(`${hodisa.screenX}x${hodisa.screenY}`);
  }
});
```

HTML elementning CSSni o'zgartirib, ko'rsatilgan koordinataga kerakli vaziyatda akslantirish mumkin. Kelgusi mavzularimizda ham hodisalardan foydalanishga doir o'z o'rnida misollar bilan batafsil bayon etib boramiz.

5.1.4. «location» obyekti

Veb-sahifaning ishoratini va unga bogliq omillarni «**location**» obyekti ifodalaydi. U aslida «window»ning tarkibiy obyekti bo'lsa ham, mantiqiy jihatdan qulay bo'lishi uchun «document.location» tarzda ham murojaat etish mumkin. Quyida «location» obyektining o'ziga xos xususiyatlarini bayon etamiz:

- **location.href** – veb-sahifa ishoratini qaytaradi. Agar unga qiymat berilsa, brauzer ko'rsatilgan ishoratga o'tishga urinadi. Qiymat sifatida to'liq URL yoki joriy

saytga nisbatan yo'l berilishi mumkin. Qiymat panjara (#) belgisidan boshlansa, sahifa qayta yuklanmaydi, berilgan langar mavjud bo'lsa, unga borishni ta'minlaydi (birinchi kitobga qarang – *J.A.*):

```
location.href = 'https://yurt.uz/';
```

▪ **location.protocol** — veb-sahifa qaydnomasini ikki nuqta bilan birga qaytaradi. Unga qiymat berilganda brauzer sahifani ko'rsatilgan qaydnoma bo'yicha qayta yuklashga urinadi. Misol:

```
location.protocol = 'https:';
```

▪ **location.host** — server nomi (domen yoki IP) bilan birga port raqamini qaytaradi. Qiymat berish orqali ko'rsatilgan port bo'yicha qayta yuklash mumkin:

```
location.host = 'yurt.uz:2083';
```

▪ **location.hostname** — server nomini qaytaradi yoki o'zgartiradi:

```
location.hostname                    "yurt.uz"
```

▪ **location.port** — port raqamini qaytaradi yoki o'zgartirib sahifani qayta yuklaydi:

```
location.port = 2083;
```

▪ **location.pathname** — ishoratdagi yo'lni qaytaradi yoki qayta aniqlaydi:

```
location.pathname = '/hikoya/5/';
```

▪ **location.search** — ishoratdagi so'roqdan keyingi qismni qaytaradi yoki o'zgartiradi:

```
location.search                "?a=3"
```


▪ **location.hash** — «langar»ni (panjara belgisidan keyingi nom. Birinchi kitobga qarang – *J.A.*) aniqlaydi:

```
location.hash = '#hikmat';
```

▪ **location.origin** — ishoratning qaydnomasi, sayt nomi va portini jamlangan holda aniqlaydi:

```
location.origin                "https://yurt.uz:2083"
```

Birinchi kitobda URL haqida batafsil yoritilgan. Uning qismlarni ajratib olish va qayta aniqlash uchun location obyekti xizmat qiladi. Agar maxsus kalit bilan murojaat qilinadigan ishorat bo'lsa, unda mos ravishda «`location.username`» va «`location.password`»lardan ham foydalanish mumkin, ammo bu tavsiya etilmaydi.

5.1.5. «history» obyekti

Brauzerning ayni sahifasi qaysi ishoratlarni qanday usulda namoyon etganligining ro'yxatini (tarixini) aniqlovchi «**history**» obyekti «orqaga» va «oldinga» tugmalari bosilganda bajariladigan vazifalarni amalga oshirish hamda joriy ishoratni o'zgartirish imkoniyatini yaratadi. Buning uchun quyidagi uslublaridan foydalaniladi:

▪ **history.back()** — avvalgi sahifaga (mavjud bo'lsa) qaytish (brauzerning «orqaga» tugmasini bosish bilan bir xil);

▫ **history.forward()** — oldingi sahifaga qaytilgan bo'lsa, keyingi sahifaga o'tishni ta'minlaydi (brauzerdagi «oldinga» tugmasini bosilgani singari);

▫ **history.go(*n*)** — yuklanishlar ketma-ketligidagi (tarixidagi) o'rni bo'yicha joriy sahifaga nisbatan nechta avvalgi yoki keyingi holatga o'tishini ta'minlaydi. Agar manfiy son berilsa – orqaga, musbat son berilsa – oldinga nisbatan hisoblaydi. Ko'pgina ommaviy brauzerlarda tarixni ko'rish va o'tish uchun mos milli tugmalar (oldinga yoki orqaga) bir necha soniya bosib turilishi kerak. Shunda paydo bo'ladigan ro'yxatdan keraklisi tanlanadi. Quyidagi misolda keltirilgan ikki xil uslub bir xil vazifani bajaradi:

```
history.go(-1);          history.back();
```

Brauzer sahifasining o'tishlar tarixida necha ishorat mavjudligini bilish uchun quyidagi xususiyatdan foydalanish mumkin:

```
history.length
```

Umuman, «history» bo'yicha oldinga yoki orqaga o'tilganda avvaldan namoyon bo'lgan sahifa yana boshqatdan qayta yuklanadi. Oldingi mavzuda tanishilgani kabi «location» orqali ishorat berilganda, unda ko'rsatilgan manzilga o'tishga urinadi. Lekin sahifani to'la yuklash ortiqcha vaqt va vositalarni sarflashga majbur etishi inobatga olinsa, bunday texnologiya saramasiz hisoblanishi ma'lum. Veb-sahifaga murojaat bo'lganda kerakli ma'lumotlar bilan bir marta yuklanishi va faqat talab etilgandagina ayrim qo'shimcha qiymatlar kiritilishi maqsadga muvofiq hisoblanadi. Buning uchun serverga ma'lum qiymatlarni yuborib, mos natija qaytganda uni kerakli joyda qo'llash texnologiyasi ishlatiladi va bunda yuklanish alohida oqimda sodir bo'lgani uchun sahifani to'la qayta namoyon etish talab etilmaydi. Ushbu texnologiyani **AJAX** (Asynchronous JavaScript and XML) deb nomlanadi. Sababi, oldin ma'lumot almashish XML shaklida amalga oshirilgan, hozirda JSON [2.6.10.] keng ommalashgan.

AJAX yordamida ma'lumot yuklanib namoyon etilar ekan, u brauzer tarixida («history») o'z izini qoldirmaydi. Lekin amaliyotda joriy holatni ishorat orqali ifodalashga ehtiyoj yuqoriligi ayon bo'ldi. Natijada HTML5 ga quyidagi qo'shimcha uslublar ham kiritildi:

▫ **history.pushState(*obyekt*, *sarlavha*, *ishorat*)** — berilgan ishoratni brauzer namoyon etadi, lekin uni yuklashga urinmaydi. Shunchaki manzil satri o'zgaradi, xolos. Hozirda brauzerlar sarlavhani e'tiborga olmayapti, lekin ehtimol, keyinchalik qo'llanilishi mumkin. Veb-sahifa nomlanilishini o'zgartirishda «document.title»-dan foydalansa bo'ladi. Ishorat joriy saytga nisbatan olinadi. Agar domen nomi, porti yoki qaydnomasi boshqacha berilsa, ya'ni boshqa server xizmatiga yo'naltirish qo'llanilsa (unda CORS yoqilmagan bo'lsa [5.3.2.]), xatolik yuz beradi. Bunday cheklov xavfsizlik nuqtayi nazaridan bitta saytning ma'lumotlaridan boshqa saytda to'g'ridan-to'g'ri foydalanishning oldini olish uchun kiritilgan. Ishorat qanday qiymatlar asosida o'zgartiril-

ganini saqlab borish uchun, ularni «obyekt»da jamlab ko'rsatish mumkin. Uni brauzer alohida fayl sifatida saqlaydi. Shuning uchun qayta yuklashdan keyin ham ma'lumotni tiklab olish imkoni mavjud. Ammo umumiy hajmi 640 kilobaytdan oshmasligi talab etiladi. Misol:

```
history.pushState({b:"hikmat", r:14}, "", "/razm");
http://yurt.uz/razm
```

- **history.replaceState(*obyekt*, *sarlavha*, *ishorat*)** — xuddi «pushState» singari ishlaydi, faqat tarixiga keyingi sahifasi sifatida qo'shmaydi, joriy o'rindagisining qiymatlarini almashtiradi.

- **history.state** — joriy sahifadagi «pushState» orqali berilgan obyektni qaytaradi.

```
history.pushState({b:"sher", r:19},"","/aytolmadim");
history.pushState({b:"hikmat", r:11}, "", "/iltijo");
history.state                {b: "hikmat", r: 11}
history.back()
history.state                {b: "sher", r: 19}
```

Brauzerning «orqaga» va «oldinga» o'tkazuvchi tugmalari bosilganda «onpopstate» hodisasi bajariladi. Unga ixtiyoriy funksiya biriktirib qo'yish mumkin. Shunda foydalanuvchi brauzerdagi tugmani bosib yoki mos ravishda muqobil vazifani bajarib, «tarix» («history») bo'yicha sahifalaganda, biriktirilgan funksiyadagi kodlar bajariladi. Misol:


```
window.onpopstate = function(hodisa) {
   document.title = hodisa.state.b;
};
```

5.1.6. «navigator» obyekti

Foydalanuvchi va u ishlatayotgan brauzer haqidagi asosiy ma'lumotlarni «**navigator**» obyekti o'zida mujassamlashtiradi. U ba'zi saytlarga o'zi qayddan o'tish hamda ayrim standart vazifalarni o'z-o'zidan bajarish imkoniyatini yaratish uchun foydalaniladi.

Ushbu obyekt faqat o'qish uchun mo'ljallangan quyidagi xususiyatlarga ega:

- **navigator.appCodeName** — brauzerning ichki kod nomini qaytaradi. Bu brauzer aslida qaysi kod manbasidan olinganini aniqlash uchun xizmat qiladi. Masalan, «Chrome» uchun kutilmaganda quyidagi natija chiqarishi mumkin:

```
navigator.appCodeName    "Mozilla"
```

- **navigator.appName** — brauzerning rasmiy nomini qaytaradi. (Hozircha ushbu qaytariladigan nomlarning to'g'riligiga kafolat yo'q, ya'ni brauzerlar boshqasining kodidan foydalangan bo'lsa, uning nomi qolib ketgan bo'lishi mumkin. Ushbu xususiyatlar ommalashmagani bois ularni aniq o'z nomi bilan ifodalashga e'tibor qaratishmagan bo'lsa kerak (Kelgusida bu tuzatiladi, deb umid qilamiz — *J.A.*):

```
Navigator.appName        "Netscape"
```

o **navigator.appVersion** — brauzer, uning talqini va ishlatilayotgan operatsion tizim haqida to'liq ma'lumot qaytaradi.

```
navigator.appVersion
"5.0 (Macintosh; Intel Mac OS X 10_14_5)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/75.0.3770.100 Safari/537.36"
```

o **navigator.connection** — tarmoqqa ulanganligi haqidagi ma'lumotlarni jamlagan obyekt qaytaradi.

```
navigator.connection {effectiveType:"4g", rtt:100, …}
```

o **navigator.cookieEnabled** — brauzerga ma'lumot saqlovchi fayl – kuki («cookie» [5.2.4.]) bilan ishlashga ruxsat berilganligini aniqlovchi mantiqiy tur qaytaradi.

```
navigator.cookieEnabled          true
```

o **navigator.doNotTrack** — brauzer (yoki qurilma) foydalanuvchining faoliyatini kuzatishga ruxsati borligini ko'rsatadi. Agar ushbu xususiyat «yes» qiymat qaytarsa, kuzatishga taqiq qo'yilganligini bildiradi:

```
navigator.doNotTrack             null
```

o **navigator.geolocation** — brauzerga ruxsat berilganda uning turgan joyi koordinatasini aniqlovchi obyektdan foydalanishga imkon yaratadi.

o **navigator.onLine** — brauzer tarmoqdaligini aniqlovchi mantiqiy qiymat qaytaradi:

```
navigator.onLine         true
```

o **navigator.language** — brauzer, odatda, qaysi tilni qo'llayotganligini ko'rsatadi:


```
navigator.language  "ru-RU"
```

o **navigator.languages** — foydalanuvchi ishlatadigan tillarni faol qo'llanilish tartibida joylashgan jadval sifatida qaytaradi:

```
navigator.languages ["ru-RU","ru","en-US","en","uz"]
```

o **navigator.mimeTypes** — brauzer qo'llaydigan tavsif turlarni (birinchi kitobga qarang — *J.A.*) majmua sifatida qaytaradi. Maxsus qo'shimcha dasturiy ta'minotlar o'rnatish orqali ushbu ro'yxatni kengaytirish mumkin. Ammo o'rnatishdan avval tekshirib olish maqsadga muvofiqdir.

o **navigator.platform** — brauzerning qanday platformada (texnik va operatsion tizimlar majmuasida) ishlayotganini qaytaradi.

```
navigator.platform  "MacIntel"
```

o **navigator.plugins** — brauzerga namoyon etish qobilyatini oshirish uchun qanday qo'shimcha dasturiy ta'minotlar (plaginlar) o'rnatilganligini majmua sifatida qaytaradi:

```
navigator.plugins {0:Plugin,PDFView:Plugin,length:1}
```

o **navigator.productSub** — brauzer qachon «yig'ilgan»ligini ko'rsatadi. Yil, oy va kun tartibidagi sonlar ketma-ketligini barcha brauzerlar ham har gal o'zgartirib bormasligi ushbu xususiyat to'g'ri ma'lumot qaytarishiga shubha uyg'otadi. Hozircha bunday qiymatlar haqqoniyligini tekshirish masalasi kun tartibiga chiqmagan. Masalan, «navigator.product» ham hamma

brauzerda doim bir xil "Gecko" satrini qaytaradi. Lekin kelajakda ushbu qiymatlar o'z vazifasida to'g'ri qo'llanilishini ta'minlashga erishilishiga umid qilamiz:

```
navigator.productSub    "20030107"
```

- **navigator.userAgent** — brauzer haqidagi umumiy ma'lumotni xuddi «navigator.appCodeName» va «navigator.appVersion» birikmasi singari qaytaradi.
- **navigator.serviceWorker** — qayd etish, o'chirish, qayta yuklash va aloqani tekshirishga imkon beruvchi obyekt qaytaradi.
- **navigator.vendor** — brauzerni taqdim etuvchi tashkilot nomini qaytaradi. («vendorSub» xususiyati taqdim etilayotgan talqin raqamini ko'rsatishi kerak, lekin hozircha to'g'ri ishlashiga kafolat yo'q.)

```
navigator.vendor        "Google Inc."
```

Brauzer tafsiflarini ko'rsatuvchi «navigator» obyekti quyidagi uslublarga ham ega:

- **navigator.getBattery()** — qurilmaning quvvatlagichi haqida ma'lumot qaytaradi.
- **navigator.getUserMedia(***ruxsat, rozi, qarshi***)** — ushbu uslub mikrofon va kameradan foydalanishga «ruxsat» so'raydi. Bunga foydalanuvchi «rozi» bo'lganligini yoki «qarshi»lik bildirganligini anlatuvchi tugmalardan birini bosganda, mos o'rindagi funksiyalar bajariladi. Unga «ruxsat» sifatida esa ma'lum shartlar bilan aniqlangan maxsus obyekt kiritiladi. Misol:


```
navigator.getUserMedia(
  {
    audio: true,
    video: { width: 1280, height: 720 }
  },
  function(oqim) {
    var video = document.querySelector('video');
    video.srcObject = oqim;
    video.onloadedmetadata = function(e) {
      video.play();
    };
  },
  function(err) {
    let m = "Foydalanishga ruxsat etilmagan!";
    console.log(m + err.name);
  }
);
```

- **navigator.registerProtocolHandler(***qaydnoma, ishorat, sarlavha***)** — «history.pushState» uchun beriladigan sayt va portdan tashqari qaydnoma ham qat'iy bir xil bo'lishi talabi mavjudligi bilan avvalgi mavzuda tanishdik. Ammo amaliyotda sahifadagi ba'zi ma'lumotlarni qayta ishlashda ayni serverdagi boshqa qaydnoma bilan ishlayotgan xizmat turlariga ham murojaat qilishga to'g'ri keladi. Bunday istisnoni o'rnatish uchun ushbu uslubdan foydalaniladi. Bunda «qaydnoma» mavjudlaridan farq qilishi kerak, odatda, oldiga «web+» qo'shish tavsiya etiladi. Beriladigan «ishorat»da esa albatta «%s» qatnashishi shart. Uning o'rniga joriy sahifa manzili qo'shib yuboriladi, «sarlavha» sifatida ixtiyoriy satr yozish mumkin:

```
navigator.registerProtocolHandler(
  "web+ftp",
  "http://yurt.uz/?a=%s",
  "Fayllar yuklash"
);
```

▪ **navigator.vibrate(*millisoniya*)** — qurilma qo'llasa, ko'rsatilgan millisoniya titrash sodir bo'ladi, aks holda bu buyruq shunchaki «false» qiymat qaytaradi. Ayrim kompyuterlarda «true» qiymat qaytsa-da, titrash sezilmasligi mumkin. Bir necha marta turli vaqtlar bo'yicha titrash paydo qilish lozim bo'lsa, qiymatlar jadval sifatida beriladi:

```
navigator.vibrate([200, 100, 30, 300]);     true
```

▪ **navigator.javaEnabled()** — brauzer «Java» dasturini qo'llash-qo'llamasligini aniqlab, mantiqiy tur qaytaradi.

5.1.7. «performance» obyekti

Brauzerning ishlash saramadorligini amallarni bajarish vaqtiga nisbatan «**performance**» obyekti aniqlaydi. U quyidagi uslub va xususiyatlarga ega:

▪ **performance.setResourceTimingBufferSize(*son*)** — brauzer vositalarining ishlashini o'zaro solishtirib turishga xizmat qiluvchi ichki xotira hajmining eng katta qiymatini o'rnatadi. Unga 150 dan kichik son qo'ymaslik tavsiya etiladi.

▪ **performance.onresourcetimingbufferfull** — yuqorida ta'riflangan ichki xotiracha to'lib qolganda bajariluvchi funksiyani aniqlaydi. Misol:

```
performance.onresourcetimingbufferfull = function() {
  console.log("Solishtirish uchun ichki xotira to'ldi!");
  performance.setResourceTimingBufferSize(200);
};
```

○ **performance.navigation** — kerakli manzildan ma'lumotlar olish uchun nechta qayta yuklanish (sakrash) amalga oshirilib namoyon bo'lishini ko'rsatadi.

○ **performance.timing** — brauzerning barcha vositalari o'z vazifasini bajargan vaqtlarini yagona obyekt sifatida namoyon etadi. Misol:

```
performance.timing      {
                          connectEnd: 1562585851909
                          connectStart: 1562585851909
                          domComplete: 1562585852806
                          domContentLoadedEventEnd: 0
                          domContentLoadedEventStart: 0
                          domInteractive: 1562585852557
                          domLoading: 1562585852327
                          domainLookupEnd: 1562585851909
                          domainLookupStart: 1562585851909
                          fetchStart: 1562585851909
                          loadEventEnd: 1562585852807
                          loadEventStart: 1562585852806
                          navigationStart: 1562585851906
                          redirectEnd: 0
                          redirectStart: 0
                          requestStart: 1562585851912
                          responseEnd: 1562585852357
```

```
  responseStart: 1562585852307
  secureConnectionStart: 0
  unloadEventEnd: 1562585852317
  unloadEventStart: 1562585852317
}
p = performance.timing; // ulanish vaqti
new Date(p.connectEnd); Mon Jul 08 2019 16:37:31 GMT+0500
p.responseEnd -         // yuklanish vaqti, millisoniyada
p.responseStart         50
```

▪ **performance.now()** — sahifa yuklangandan ayni paytgacha o'tgan millisoniya. Brauzer veb-sahifani namoyon etish uchun alohida «window» obyekti yaratadi. O'sha zahoti «performance» hisoblashni boshlaydi.

▪ **performance.mark(*nishon*)** — «nishon» qo'yadi. Ba'zida kodning qaysidir joyi qachon bajarilganligini bilishga to'g'ri kelishi mumkin. Unday vaziyatlarda «performance.now()» talabga to'liq javob berolmasligi tabiiy. Ushbu uslub kodning kerakli qismida istalgan nomdagi «nishon» qo'yib, keyin u bo'yicha vaqtni aniqlash imkonini yaratadi.

▪ **performance.getEntriesByName(*nishon*)** — berilgan nomdagi «nishon»larni anglatuvchi obyektlarni jadval sifatida qaytaradi. Misol:

```
performance.mark("yurt");
// ...sahifa yuklangandan keyin 11 soniya o'tib ishlagan
var p = performance.getEntriesByName("yurt");
p[0].startTime                  11000.72000000027
```

▪ **performance.clearMeasures()** — barcha («measure» turidagi) oraliq nishonlarni o'chiradi.

▪ **performance.clearMarks([*nishon*])** — nomi kiritilgan «nishon»larni o'chiradi, agar bunday nomlar kiritilmasa, mavjud barcha nishonlarni o'chiradi.

▪ **performance.measure(*nom*, *nishon1*, *nishon2*)** — berilgan nom bo'yicha «nishon1»dan «nishon2»gacha o'tgan muddatni aniqlaydi. Bu juda ham kerakli vosita hisoblanib, dastur kodining ayni bir qismidan boshqa belgilangan qismi bajarilguncha qancha vaqt o'tganini bilishga imkon yaratadi. Misol:

```
performance.mark("yurt");
// ... sahifa yuklangandan keyin 14 soniya o'tib ishlagan
performance.mark("ilm");
// ... «yurt» va «ilm» nishonlari 3 soniya tafovut bilan ishlagan
performance.measure('uz', "yurt", "ilm");
var p = performance.getEntriesByName("uz");
p[0].startTime          14001.72000000027
p[0].duration           3008.069999999134
```

▪ **performance.getEntriesByType(*nishon*)** — ko'rsatilgan nomdagi «nishon»larni turi bo'yicha («mark» yoki «measure» qiymatlaridan birini) oladi.

▪ **performance.getEntries()** — barcha nishonlarni jadval sifatida qaytaradi. Unda nafaqat dasturlovchi tomonidan yaratilgan «mark» yoki «measure»lar, balki brauzerning o'zi hosil qilgan «resourceTiming» turidagi nishonlar ham mavjudki, ular sahifadagi har bir yuklangan fayllarni aniqlaydi va o'z nomida mos manzilni ko'rsatadi. Ushbu imkoniyat bilan qaysi fayl qancha muddatda yuklanganini aniqlash mumkin:

```
var e = performance.getEntries();
e.length              33
e[11].startTime       266.52500000000146
e[11].name            "http://yurt.uz/img/tik.gif"
```

5.2. «document» obyekti

JavaScriptning eng muhim asoslaridan yana biri «**document**» obyekti hisoblanadi. Aynan u HTML elementlari ustida dasturiy amallarni bajarishga imkon yaratadi. Aslida «window» obyektining bir xususiyati sifatida ifodalansa ham u juda keng qamrovli imkoniyatlarga ega. Sahifa HTML yoki XML tilida yozilganligiga qarab uning ifodalanishi ham o'ziga xos bo'ladi. Biz faqat HTML nuqtayi nazardan o'rganamiz.

Ushbu «document»ning asosiy vazifasi veb-sahifadagi HTML elementlarini JavaScript dasturlash tili o'zgaruvchisiga mukammal obyekt sifatiga yuklab berishni va shu o'zgaruvchining qiymatlari bo'yicha mos elementni (veb) sahifada o'zgartirilishini ta'minlashdan iborat. Aslida HTML (xHTML, XML, SVG) hujjatning joylashtiruv tilida teglar asosida yozilgan elementlarini dasturlash tili tushunadigan (daraxt shaklida) tarkiblangan mukammal obyektga o'zlashtirish **DOM** (Document Object Model) deb yuritiladi. Uning asosiy obyekti «**Element**» sifatida aniqlanadi.

HTML tilida yozilgan tegning ko'rsatilgan «id» alomati bo'yicha JavaScript o'zgaruvchisiga yuklab olish uchun quyidagi uslubdan foydalaniladi:

```
var element = document.getElementById('tagID');
```

5.2.1. «Element» obyekti

HTMLda yoziladigan «teg»larning JavaScript tilida obyekt sifatidagi muqobili hisoblangan «**Element**»ning quyidagi xususiyatlari mavjud:

- **attributes** — elementning mavjud alomatlari bo'yicha **majmua** hosil qiladi. Majmua – shunday obyektki, uni tashkil qiluvchi ichidagi obyektlarning har biri o'z nomiga, ya'ni takrorlanmas satr qabul qiluvchi «name» xususiyatiga ega bo'lib, shu nomdagi indekslanishidan tashqari yana bittadan nusxasi xuddi jadvaldagidek noldan boshlanuvchi tartib raqamiga ham ega bo'ladi. Agar betakror nomlash imkoni bo'lmasa, faqat tartib raqami bo'yicha joylashtiriladi va berilgan o'rindagi obyektni qaytaruvchi «**item**(*tartibRaqam*)» uslubiga ega bo'ladi. Majmuaning «length» xususiyati aslida nechta obyektdan tashkil topganini ko'rsatib turadi. Uning ichki elementlariga tartib raqami yoki nomi bo'yicha murojaat qilish mumkin. Ushbu holda ular mos alomatning o'girilgan obyekti hisoblanadi. Misol:

```
<div id="asos" class="maydon">
var div = document.getElementById('asos');
// {0: id, 1: class, length: 2, class: class, id: id}
```

```
div.attributes[0].name              // "id"
div.attributes['class'].value       // "maydon"
```

- **classList** — elementning «class» alomatida keltirilgan sinflar ro'yxatini mujassamlashtiruvchi jadval qaytaradi. Misol:

```
<div id="asos" class="c1 c2">
div.classList
// [0: "c1", 1: "c2", length: 2, value: "c1 c2"]
div.classList[1]        // "c2"
```

- **childElementCount** — ichki elementlari soni.
- **children** — ichki HTML elementlari majmuasi.
- **className** — «class» alomatida beriladigan satr.
- **clientHeight** — ichki balandligini piksel hisobida aniqlaydi. Bunda (odatda, CSSda beriladigan qiymat) «padding» inobatga olinadi, «margin» esa yo'q.
- **clientWidth** — ichki kengligi («clientHeight» singari aniqlanadi).
- **firstElementChild** — birinchi ichki elementini «Element» obyekti sifatida aniqlaydi. Mavjud bo'lmasa, null qabul qiladi.
- **lastElementChild** — oxirgi ichki elementi.
- **innerHTML** — element asosidagi yozuvini HTML shaklida qaytaradi.
- **innerText** — element va uning ichki teglari asosidagi ko'rinadigan matnlarni qaytaradi.
- **previousElementSibling** — o'zidan oldingi qo'shni elementni (mavjud bo'lsa) qaytaradi.

- **id** — «id» alomati qiymati, satr.
- **nextElementSibling** — o'zidan keyingi qo'shni elementni (mavjud bo'lsa) qaytaradi.
- **outerHTML** — elementning o'zini va uning ichki tashkil etuvchi teglarining HTML sifatida namoyon etilishi, satr. Qiymat olinayotganda u o'zini yaratgan tegni ham to'laligicha qaytaradi, qiymat berilayotganda esa element asosini emas, o'zini o'zgartiradi.
- **outerText** — element va uning ichki tashkil etuvchi teglarining asosida brauzerda ko'rinishi uchun yozilgan matnlarning birlashmasi, satr.
- **scrollHeight** — elementning ichki tashkil etuvchilarining hajmi hisobga olingan holdagi balandligi, son. Agar CSSda «overflow: hidden» berilmagan bo'lsa, surgich bilan umumiy uzunligini ko'rsatadi.
- **scrollWidth** — umumiy kengligi («scrollHeight» singari aniqlanadi), son.
- **tagName** — element HTML tegining nomi, satr.
- **style** — elementning amaliyotda eng ko'p qo'llaniladigan xususiyati, bu CSSda beriladigan shamoyillarni o'zlashtirish va o'zgartirishdir. Bu mukammal obyekt bo'lib, CSS alomatlarning barchasini o'zida jamlaydi. Faqat CSS dasturlash tili bo'lmagani uchun unda nomlanishlarda chiziqcha ishlatilishi mumkin. JavaScriptning ushbu obyektida esa nomlanishlarda o'sha chiziqchalar tushirib qoldiriladi va undan keyingi ilk harflar katta harf sifatida yoziladi. Demak, «style» obyektining barcha qabul qilishi mumkin bo'lgan ichki indeksla-

rining nomlanishlarini birinchi kitobdan bilib olish mumkin. Unga beriladigan o'lchov qiymatlari esa son bilan emas, satr sifatida to'liq ko'rsatilishi shart. Misol:

```
var div = document.getElementById('maydon');
div.style.marginTop = '100px';
```

Element obyektining o'ziga xos asosiy uslublari:

- **addEventListener**('*hodisa*', *funksiya*, *omillar*) — har bir HTML element uchun unga o'ziga xos hodisa biriktirish mumkin [5.1.3.]. Quyida faqat bitta telefon raqami yozish uchun mo'ljallangan «input» elementiga ortiqcha belgilar yozilishiga yo'l qo'ymaydigan misolni keltiramiz. Unda barcha belgilar raqamlar hamda chiziqcha va bo'sh joylardan iborat bo'lishi shart, faqatgina birinchi belgi sifatida «+» yoki raqam yozish mumkin:

```
<input id="phone" name="phone">
var inp = document.getElementById("phone");
inp.addEventListener('keypress', function(hodisa) {
  var r = true;
  if (this.value === '')  // birinchi belgi
    r = '+0123456789'.indexOf(hodisa.key) >= 0;
  else r = '0123456789 -'.indexOf(hodisa.key) >= 0;
  if (!r) hodisa.preventDefault(); // belgi yozilmaydi
  return r;
});
```

Agar funksiya «yolg'on» qiymat qaytarsa, «input» darchasi faol bo'lganda bosilgan tugma belgisi (harf) paydo bo'lmaydi. Lekin bu mukammal yechim emas, agar darchaga qiymat tugma bosish yordamida emas,

balki qisqa xotiradan nusxasini qo'yish orqali kiritilsa, funksiya bajarilmaydi. Xotiradan matn qo'yish vaqtidagi emas, qo'yilgandan keyingi darcha qiymatini olish uchun jarayon 3-4 millisoniya vaqt sarflashini inobatga olib, funksiya bajarilishini ortga suramiz. Darchani esa yangi lokal o'zgaruvchiga yuklab olamiz:

```
inp.addEventListener('paste', function() {
  var inVal = this;    // input darchasi o'zlashtirildi
  setTimeout(function() {
    var i, r = '', v = inVal.value.trimLeft();
    if (v[0] == '+') r = '+'; else v = '+' + v;
// ilk belgi «+» bo'lsa, oladi, qolganlarini bir xil tekshiradi.
    for (i = 1; i < v.length; i++)
      if ('0123456789 -'.indexOf(v[i]) >= 0) r += v[i];
    inVal.value = r;
  }, 5);
}, false);
```

- **removeEventListener**(*hodisa*, *funksiya*[, *omil*]) — elementga o'rnatilgan tutib qoluvchi «hodisa»dan ko'rsatilgan «funksiya»ni olib tashlaydi.

- **closest**(*saralov*) — berilgan «saralov»ni qanoatlantiruvchi eng yaqin tashqi elementni (topa olmasa, «null»ni) qaytaradi. **Saralov** – CSS nomlanish shartlari asosida (*birinchi kitobga qarang – J.A.*) berilgan satr. JavaScriptda uni «**DOMString**» deb yuritiladi va unga «body», «.class», «#id», «div#id» singari qiymat beriladi. Agar noto'g'ri andoza berilsa, xatolik yuz beradi.

- **getAttributeNames**() — barcha alomatlarni jadval sifatida qaytaradi.

▪ **getAttribute**(*alomat*) — berilgan «alomat» qiymatini (yo'q bo'lsa, «null»ni) qaytaradi. Yuqoridagi misol (davomi) bo'yicha:

```
inp.getAttribute('id')                    phone
```

▪ **getAttributeNode**(*alomat*) — berilgan «alomat» qiymatini majmua sifatida qaytaradi. Misol:

```
var m = inp.getAttributeNode('id')        {…}
m.name                                    id
m.value                                   phone
```

▪ **getBoundingClientRect**() — element o'lchamlari va sahifada joylashish koordinatalarining jamlanmasini obyekt sifatida qaytaradi. Bu «**DOMRec**» deb nomlanib, HTML elementni chegaralovchi to'g'ri to'rtburchakni aniqlovchi «x», «y», «width», «height», «left», «right», «top», «bottom» singari ma'no jihatdan tushunarli indekslarga ega obyekt.

▪ **getClientRects**() — veb-sahifada namoyon etilgan obyektning har bir ichki elementi uchun **DOMRec** obyekti hosil qilib, jamlab, jadval sifatida qaytaradi. Dastavval Microsoft buni matnning har bir satri uchun qo'llagan. Endilikda uni jadvalning (<table>) har bir katagi, yoki ro'yxat(<ul>)ning har bir qatori singari murakkab HTML obyektlariga tatbiq etish mumkin.

▪ **getElementsByClassName**(*nomlar*) — sinfi bo'yicha berilgan «nomlar»ga taalluqli elementlarni jadval sifatida qaytaradi. CSSda «class» alomatiga bir nechta nomni bo'sh joy bilan ajratib kiritish mumkin bo'lgani

kabi, bu yerda «nomlar» ham xuddi shunday tarzda beriladi.

▪ **getElementsByTagName**(*nom*) — berilgan «nom»-dagi teglarni akslantirgan barcha elementlarni yagona jadval sifatida qaytaradi.

▪ **hasAttribute**(*nom*) — berilgan «nom»dagi alomat tegda mavjudligini tekshiradi, mantiqiy qiymat.

▪ **insertAdjacentHTML**(*o'rin*, *html*) — ushbu uslub ko'rsatilgan «o'rin»ga «html» matnni akslantirib, mos element sifatida qo'shadi. Bu uslub «innerHTML»ga nisbatan ancha tez ishlaydi. Joriy elementga nisbatan aniqlanuvchi «o'rin» quyidagi qiymatlardan biriga teng:

- **'beforebegin'** — elementdan avval;
- **'afterbegin'** — ichida, boshiga (birinchi);
- **'beforeend'** — ichida, oxiriga;
- **'afterend'** — elementdan keyin;

```
inp.insertAdjacentHTML('afterend',
  '<input type="text" name="ismi" id="ismi" />');
```

▪ **matches**(*saralov*) — mantiqiy tur qaytarib, berilgan «saralov» (DOMString) bo'yicha element mos kelishini tekshiradi. Misol:

```
<input id="phone" name="phone">
var p = document.getElementById("phone");  {…}
p.matches('input')                         true
p.matches('div')                           false
```

▪ **querySelector**(*saralov*) — berilgan saralov bo'yicha mos kelgan birinchi elementni qaytaradi.

▪ **querySelectorAll**(*saralov*) — berilgan saralov bo'yicha mos keluvchi barcha elementlarni jadval sifatida qaytaradi.

▪ **remove**() — elementni o'chiradi. Aslida esa qo'shnilari bilan hosil qilgan tarkiblangan «daraxt»dan aynan ushbu element hosil qilgan «shox»ni olib tashlaydi. Ta'kidlab o'tish lozimki, element butunlay yo'q bo'lmaydi, obyekt sifatida qoladi. Butkul o'chirish uchun unga yo'naltiruvchi barcha ko'rsatkichlarni bekor qilish kerak. Tajribadan ayonki, dasturlashda ko'pgina kutilmagan xatoliklar, aynan shu jihatni nazarga olmaslikdan kelib chiqadi.

▪ **removeChild**(*ichki element*) — berilgan «ichki element»ni («**child**» deb yuritiladi) olib tashlaydi.

```
a = document.getElementById('mainArticle');
b = a.querySelector('div#main');
c = a.querySelector('div.article');
b.remove();
a.removeChild(c);
```

▪ **appendChild**(*ichki element*) — elementning ichida «ichki element» hosil qiladi [5.2.3.].

▪ **requestFullscreen**() — elementni to'la ekranda namoyon bo'lishini ta'minlaydi. Agar muvaffaqiyatli bajarilsa (ya'ni brauzer qo'llasa va cheklovlar bo'lmasa), «document.fullscreenElement» xususiyati aynan shu elementga ko'rsatkich yaratadi. To'la ekran vaziyatidan «document.exitFullscreen()» uslubi yordamida oddiy holatga qaytishi mumkin. Misol:


```
function TolaEkranYokiOddiyOlcham(elem) {
  if (!document.fullscreenElement) {
    elem.requestFullscreen().catch(
      err => {
        alert('To'la ekranda o'tishda xatolik yuz berdi:'
          + ` ${err.message} (${err.name})`);
      }
    );
  } else { document.exitFullscreen(); }
}
```

▪ **requestPointerLock**() — element uchun sichqoncha ko'rsatkichini «qulflab» qo'yadi. Ya'ni aynan u uchun faoliyatini cheklaydi. Ushbu maxsus imkoniyat hozircha hamma brauzerlarda qo'llanmaydi. Umid qilamiz, yaqin kelajakda bu noqulaylik bartaraf etiladi.

▪ **scrollIntoView**(*omillar*) — berilgan «omillar» obyekti bo'yicha sahifani element joyiga mos ravishda suradi. Bunda omillar mantiqiy tur yoki quyidagi indekslarga ega obyekt sifatida berilishi mumkin:

◊ **behavior** — surish qay tarzda amalga oshirilishini aniqlaydi. U «**auto**», «**instant**» (ilkis) yoki «**smooth**» (ohista) qiymatlaridan birini qabul qiladi.

◊ **block** — tik yo'nalishda tekislash. U elementga nisbatan «**start**» (tepa), «**center**» (o'rta), «**end**» (past) yoki «**nearest**» (eng yaqin) qismi ko'rinadigan holatgacha sahifani suradi;

◊ **inline** — yotiq yo'nalishda tekislash, qiymatlari «block»niki singari;

Agar «omillar» sifatida hech narsa berilmasa yoki «true» kiritilsa, `{block: "start", inline: "nearest"}` obyekti sifatida qabul qilinadi. «Yolg'on» qiymat esa ushbu `{ block: "end", inline: "nearest" }` obyektga teng kuchli. Misol:

```
elm.scrollIntoView({behavior:"smooth", block:"end"});
```

- **setAttribute**(*alomat*, *qiymat*) — elementning berilgan «alomat»iga kiritilgan «qiymat»ni yuklaydi.

- **setAttributeNode**(*majmua*) — berilgan «majmua» bo'yicha alomatni omillari bilan qo'shadi yoki mavjud bo'lsa, qiymatini o'zgartiradi [5.2.3.].

- **removeAttribute**(*alomat*) — elementning ko'rsatilgan «alomat»ini o'chiradi. Misol:

```
<input type="checkbox" name="male">
chk = document.querySelector('input[type=checkbox]');
chk.setAttribute("checked", "checked"); // tanlandi
chk.removeAttribute("checked"); // tanlov o'chirildi
```

- **setPointerCapture**(*hodisa.pointerId*) — sichqoncha ko'rsatgichining hodisasini elementga bog'laydi. Bu bilan hodisaning o'zgarish qiymatlarini siqchoncha elementni tark etganda ham olib turishi ta'minlanadi.

- **releasePointerCapture**(*hodisa.pointerId*) — element bilan u bog'langan «**id**»li hodisa uziladi.

5.2.2. «document»ning o'ziga xos xususiyatlari

JavaScriptda «document» obyekti veb-sahifani to'liq idrok etish uchun mo'ljallangan. Shuning uchun ham uning keng qamrovli imkoniyatlari mavjud. Xususan, «Element» obyektining [5.2.1.] ham aksariyat xususiyat va uslublarini o'zida mujassamlashtiradi. Ulardan tashqari yana o'ziga xos quyidagi xususiyatlarga ega:

- **document.characterSet** — sahifa kodlamasi.

- **document.compatMode** — sahifaning namoyon etish usuli. Dastavval teglar maxsus brauzerlarning o'ziga xos shartlariga moslab yozilgan. Umumiy standart ishlab chiqilgandan keyin, avvalgi kodlar risoladagidek ishlamagan. Hozirda bunday muammo deyarli bartaraf etilgan, ammo eski kodlardan butkul voz kechib bo'lmagani uchun, gohida farqlashga to'g'ri keladi. Odatdagi usulda namoyon etilgan bo'lsa – «CSS1Compat», maxsus brauzer standartlari asosida aniqlangan bo'lsa – «BackCompat» kabi satr qaytaradi.

- **document.contentType** — sahifaning tavsif-turi. (MIME type. *Birinchi kitobga qarang – J.A.*)

- **document.doctype** — veb-sahifaning ilk tegida aniqlangan kelishuvlarning jamlanmasini obyekt sifatida qaytaradi. HTMLning avvalgi talqinlarida «systemId» va «publicId» indekslari orqali mos alomatlari ham qaytariladi, HTML5da `<!DOCTYPE html>` singari ko'rsatilgan bo'lsa, faqat nomini qaytaradi:

```
document.doctype.name                    "html"
```

- **document.documentElement** — ilk ichki elementni qaytaradi. HTML sahifa uchun u **html** tegi bo'yicha aniqlangan element hisoblanadi.
- **document.documentURI** — sahifa ishorati.
- **document.inputEncoding** — sahifa kodlamasi.
- **document.activeElement** — ayni paytda faol bo'lgan HTML element.
- **document.anchors** — sahifadagi barcha «langar»-larni (*birinchi kitobda batafsil tushuntirilgan — J.A.*) majmua tarzda qaytaradi.
- **document.alinkColor** — sahifa tanasida faol ishorat uchun belgilangan rangni aniqlaydi.
- **document.bgColor** — sahifaning orqatasvir rangi;
- **document.fgColor** — oldtasvir yoki matn rangi;
- **document.linkColor** — ishoratlar rangi;
- **document.vlinkColor** — tashrif buyurilgan (bosib ochilgan) ishoratlar rangini aniqlaydi.
- **document.body** — tana (body) elementini mujassam obyekt sifatida qaytaradi.
- **document.defaultView** — sahifaning window [5.1.] obyektiga murojaatni ta'minlaydi.
- **document.designMode** — sahifaning tahrir qilish imkoniyatini aniqlaydi. Odatiy qiymati: «Off». Agar «On» qiymat berilsa, ayrim xususiyatlar orqali sahifani o'zgartirish imkoni ortadi.
- **document.dir** — sahifa namoyon bo'lish yo'nalishini aniqlaydi. Qiymati: «rtl» (o'ngdan chapga) va «ltr» (chapdan o'ngga)

- **document.domain** — sayt (domen) nomini qaytaradi.
- **document.embeds** — sahifadagi <embed> elementlarni majmua sifatida qaytaradi.
- **document.forms** — sahifadagi formalarni majmua sifatida qaytaradi.
- **document.head** — sahifaning <head> elementini qaytaradi.
- **document.images** — sahifadagi <img> tegi orqali aniqlangan rasmlarni majmua sifatida qaytaradi.
- **document.lastModified** — sahifaga oxirgi murojaat bo'lgan vaqt. Bu so'nggi biror o'zgartirish kiritilgan vaqt emas, odatda u joriy vaqtni qaytaradi.
- **document.links** — sahifadagi barcha ishoratlarni majmua sifatida qaytaradi.
- **document.plugins** — sahifaga bog'langan qo'shimcha plaginlar ro'yxatini majmua sifatida qaytaradi.
- **document.readyState** — sahifaning yuklanish holatini ko'rsatadi. U quyidagi qiymatlardan birini qabul qilishi mumkin:
  - ◊ **loading** — yuklanish jarayonida;
  - ◊ **interactive** — sahifaning o'zi yuklanib bo'ldi, unda DOM [5.2.] obyektlari bilan ishlash mumkin, bunda «DOMContentLoaded» hodisasi chaqiriladi;
  - ◊ **complete** — sahifa va unga biriktirilgan boshqa fayllar va rasmlar ham yuklanib bo'lindi, bunda «load» hodisasi ishga tushadi.

○ **document.referrer** — joriy sahifa qaysi ishorat orqali ochilganini ko'rsatadi. Brauzerda ilk ochilgan sahifa uchun bu qiymat bo'sh satr bo'ladi. Biror ishorat orqali boshqa sahifaga o'tilganda, unda ushbu qiymat avvalgi sahifa manzilini qaytaradi.

○ **document.scripts** — sahifadagi <script> teglarini majmua sifatida qaytaradi. JavaScriptda yozilgan matnlarni «`document.scripts[0].text`» singari kod yozib olish mumkin.

▪ **document.title** — sahifa sarlavhasini aniqlaydi.

○ **document.URL** — sahifa ishoratini qaytaradi.

5.2.3. «document»ning o'ziga xos uslublari

JavaScriptda eng asosiy amallar «document» obyekti orqali amalga oshiriladi. Quyida ushbu obyektning o'ziga xos muhim uslublari misollar yordamida bayon etiladi.

▪ **adoptNode(*element*)** — boshqa sahifa elementini o'zlashtiradi. Ya'ni boshqa sahifada mavjud elementlarning nusxasini yaratib, joriy sahifaga qo'shish imkoniyatiga ega. Misol tariqasida `<iframe>`ga yuklangan rasmlarni o'zlashtirishni keltiramiz, «document» sifatida freymga murojaat qilish uchun «contentDocument» xususiyatidan foydalaniladi:

```
var f = document.querySelector('iframe'),
  rasm = f.contentDocument.querySelectorAll('img');
```



```
var maydon = document.getElementById('images');
rasm.forEach(function(elm) {
  maydon.appendChild(document.adoptNode(elm));
});
```

▪ **createAttribute(*nom*)** — berilgan «nom»da alomat hosil qiladi, keyin uni elementga biriktirish mumkin [5.2.1.]. Misol:

```
var elm = document.getElementById("elmInput");
var alm = document.createAttribute("maxLength");
alm.value = 12;
elm.setAttributeNode(alm);
```

▪ **createComment(*izoh*)** — «izoh» (`<!-- «izoh» -->` tarzda) hosil qiladi. Misol:

```
var elm = document.createComment('JavaScript izohi');
document.body.appendChild(elm);
```

▪ **createElement(*teg nomi*)** — berilgan «teg nomi»ga mos element hosil qiladi. Ya'ni, ko'rsatilgan nomdagi teg asosida yangi HTML obyekti yaratadi. Uni qaysi element ichiga joylash lozim bo'lsa, «`appendChild`» yordamida unga qo'shish mumkin.

▪ **createDocumentFragment()** — teg nomi avvaldan aniq bo'lmagan element hosil qiladi. U joriy sahifaga bog'liq bo'lmagan bir qancha elementlardan iborat tuzilma yaratishda keng qo'llaniladi. Uning ichida keraklicha element hosil qilib, keyin umumlashmadan yagona obyekt sifatida foydalanish mumkin. Misol:

```
var elm = document.getElementById('sel1'),
  f = document.createDocumentFragment();
```

```
var i, sel = document.createElement('select');
for (i = 0; i < elm.options.length; i++)
   f.appendChild(elm.options[i])
sel.appendChild(f);
   // yangi tayinlanuvchi darcha hosil bo'ldi.
```

▪ **createNodeIterator(*asos*, *ko'rinish*, *saralash*)** — sanoqli tugunlar sifatida element yoki uning qismini ifodalash. **Tugun** — bu (**node**) HTML elementini yoki uning qismini yoxud alomatini JavaScript obyekti ko'rinishida namoyon bo'lishidir. Sanoqli tugun bo'lsa, uning ichidagilariga «nextNode()» uslubi bilan murojaat qilish mumkin. Bunda:

◊ **asos** — qaysi HTML element asosida sanoqli tugun hosil qilinishini ko'rsatadi;

◊ **ko'rinish** — tugun sifatida nimani ko'rsatish lozimligini aniqlaydi; nomlari bo'yicha ma'nosi tushunarli bo'lgan quyidagi qiymatlardan birini qabul qiladi:

NodeFilter.SHOW_ALL	-1
NodeFilter.SHOW_ATTRIBUTE	2
NodeFilter.SHOW_CDATA_SECTION	8
NodeFilter.SHOW_COMMENT	128
NodeFilter.SHOW_DOCUMENT	256
NodeFilter.SHOW_DOCUMENT_TYPE	512
NodeFilter.SHOW_DOCUMENT_FRAGMENT	1024
NodeFilter.SHOW_ELEMENT	1
NodeFilter.SHOW_PROCESSING_INSTRUCTION	64
NodeFilter.SHOW_TEXT	4

◊ **saralash** — topilgan tugunni saralanadigan ro'yxatga qo'shish-qo'shmaslikni aniqlaydi; u quyidagi qiymatlardan birini qabul qiladi:

```
NodeFilter.FILTER_ACCEPT
NodeFilter.FILTER_SKIP
NodeFilter.FILTER_REJECT
```

Sahifadan barcha ishoratlarni olib sanoqli tugun hosil qiluvchi holatga misol keltiramiz:

```
var tugun = document.createNodeIterator(
  document.body,              // sahifa tanasi
  NodeFilter.SHOW_ELEMENT,    // HTML elementlarni olish
  function(node) {
    return node.nodeName.toLowerCase() === 'a'
      ? NodeFilter.FILTER_ACCEPT
      : NodeFilter.FILTER_REJECT;
  }
);
var tugunlar = [], tuguncha;
while (tuguncha = tugun.nextNode()) {
  tugunlar.push(tuguncha);
}   // tugunlarga ishoratlar elementlari yig'ildi.
```

▪ **createTreeWalker(*asos*, *ko'rinish*, *saralash*)** — bu uslub «createNodeIterator»ning ayni muqobili bo'lib, faqat tugunlarni daraxt ko'rinishidagi bitta shoxdan, ya'ni faqat qo'shni teglar orasidan izlaydi. Yuqoridagi misolda sahifadagi barcha ishoratlarni olish imkoni bo'lgan bo'lsa, quyidagi yo'l bilan faqat <body>ning asosida joylashgan, boshqa teglar ichiga kirmagan larni olish mumkin, xolos:

```
var tugun = document.createTreeWalker(
  document.body,
  NodeFilter.SHOW_ELEMENT,
  { acceptNode: function(node) {
    return node.nodeName.toLowerCase() === 'div'
      ? NodeFilter.FILTER_ACCEPT
      : NodeFilter.FILTER_REJECT;
  } }
);
var m = [];
while (tugun.nextNode())
  m.push(tugun.currentNode);
```

▪ **createRange()** — oraliq yaratadi. Bir qancha turli elementlarni yagona qolipga solish uchun o'ziga xos jihatlarga ega JavaScript obyekti — «**oraliq**» yaratib, undan foydalansa bo'ladi. Uning uchun avval quyidagi uslublar yordamida chegaralari aniqlab olinishi kerak:

```
var oraliq = document.createRange();
oraliq.setStart(tugun1, boshQiymat);
oraliq.setEnd(tugun2, oxirQiymat);
```

▪ **createTextNode(*matn*)** — berilgan «matn»dan iborat element hosil qiladi. Berilgan matnni ixtiyoriy teg asosining so'nggiga «appendChild» uslubi yordamida qo'shish (ulash) mumkin.

▪ **elementFromPoint(*x*, *y*)** — berilgan koordinatada joylashgan elementni qaytaradi. (Koordinatani sichqoncha hodisasidan olish mumkin.) Misol:

```
var elem = document.elementFromPoint(200, 440);
elem.style.color = 'red';
```


▪ **exitPointerLock()** — «requestPointerLock» bilan sichqoncha ko'rsatkichini yashirish bekor qilinadi. (Bular hozircha tajriba sifatida kiritilgan, qo'llash tavsiya etilmaydi.)

▪ **importNode(*tugun*, *to'liq*)** — berilgan «tugun» sifatidagi elementni o'zlashtiradi. Agar «to'liq» mantiqiy qiymati «rost» bo'lsa, barcha ichki elementlari ham birgalikda olinadi, aks holda faqat o'zigagina qaraladi. Misol:

```
var bori = document.getElementById("maydon");
var yasa = document.importNode(bori, true);
document.body.appendChild(yasa);
```

▪ **getSelection()** — joriy faol elementni maxsus indeksli obyekt sifatida qaytaradi.

▪ **hasFocus()** — sahifa ayni paytda faolligini mantiqiy tur sifatida qaytaradi. Foydalanuvchi bu uslubni faqat joriy sahifada istifoda qilyaptimi, yoki boshqa sahifa yoxud dasturga o'tishi amalga oshirilganmi, ana shuni aniqlash uchun juda muhim vositadir.

▪ **open()** — sahifani JavaScript yordamida teglar yozish uchun go'yoki yangi varaq sifatida namoyon qiladi. Bunda barcha mavjud yozuvlar tozalanib, yangi oqim ochiladi.

▪ **write(*html*)** — yozilgan HTML teglarni sahifada ifodalaydi. Birinchi marta «write» ishlatilganda, «open» vazifasi ham bajariladi. Alohida qator sifatida yozish uchun «**writeln**» ko'rinishida ham yozish mumkin, unda yozuv so'nggida bo'luvchi belgi ham qo'shiladi.

- **close()** — oqimni yopadi. Undan keyin yozilgan «write» xuddi ilk ishlatilganday vazifani bajaradi:

```
document.open();
document.write("<h1>Mukammal dasturlash</h1>");
document.write("<h1>Ikkinchi kitob</h1>");
document.close();
```

- **queryCommandEnabled(*buyruq*)** — berilgan «buyruq»ni brauzer qo'llash-qo'llamasligini aniqlovchi mantiqiy tur qaytaradi.

- **queryCommandSupported(*buyruq*)** — ushbu uslub ham xuddi «queryCommandEnabled» singari vazifani bajaradi. Farqi shundaki, buyruqni brauzerda qo'llash imkoniyati (masalan, «designMode» yordamida) cheklangan bo'lishi mumkin. Bunday hollarda u «yolg'on» qiymat qaytaradi.

- **execCommand(*buyruq*, *ko'rsat*, *omillar*)** — berilgan «buyruq»ni kiritilgan «omillar» asosida ishga tushiradi, «ko'rsat» mantiqiy turi yordamida maxsus interfeys chiqishi belgilanadi. Misol:

```
if ( document.queryCommandEnabled("popupNode") )
  document.execCommand("popupNode", false, null);
```

- **registerElement(*yangicha-nom*, *omillar*)** — HTMLda mavjud bo'lmagan yangicha nomdagi tegni o'z omillari bilan hosil qilish uchun qo'llaniladi. E'tiborga olish lozimki, nomda albatta chiziqcha qatnashishi shart. (2019-yil mart oyidan ushbu uslub M73 kelishuviga binoan eskirgan deb topildi, qo'llash tavsiya etilmaydi — *J.A.*) Misol:


```
var newTeg = document.registerElement("yangi-teg");
document.body.appendChild(newTeg);
```

5.2.4. document.cookie

Veb-sahifa — serverga murojaat qilinganda u taqdim etgan fayllar asosida namoyon etiladigan interfeys ekanligini yaxshi bilamiz. Aksariyat hollarda server bilan muloqot tarzda axborot almashishga to'g'ri keladi. Buning uchun maxsus usulda unga ma'lumot jo'natiladi. Server har gal qabul qilish jarayonida kim tomonidan yuborilayotganini so'raydi. Bunday talab brauzerga har bir foydalanuvchi uchun alohida ma'lumot saqlash zaruriyatini yuklaydi. Ushbu vazifani «cookie» (kuki – bu so'z **pishiriq** degan ma'noni anglatadi) bajaradi. U aslida hajmi uncha katta bo'lmagan matnli fayl bo'lib, o'ziga xos qolipga ega:

```
nom=nomi; expires=vaqt; path=yo'l; domain=sayt; secure
```

Bunda o'zgaruvchining «nomi» berilishidan tashqari, u qachongi «vaqt»gacha saqlanishi lozimligi, ishoratdagi «yo'l» qanday bo'lganda ishlashi va qaysi «sayt» uchun bajarilishi lozimligi ko'rsatilishi mumkin. Agar yo'l va sayt ko'rsatilmasa yoki bo'sh bo'lsa, «cookie»ni o'rnatgan sahifadagina ishlashini anglatadi. Odatda, brauzerlar lokal faylni ochganda, u uchun kuki yaratmaydi. Xavfsizlik talabi bo'yicha SSL qo'llanadigan hollardagina (https qaydnomasi bo'yicha) kuki yuborilishi

shart bo'lsa, «secure» so'zi qo'shiladi. Kukida saqlangan qiymat ko'rsatilgan vaqtgacha brauzerning maxsus o'z ichki fayllaridan birida saqlanadi. Vaqt ko'rsatilmagan bo'lsa, brauzer yopilishi bilan o'zgaruvchi o'chib ketadi.

Misol sifatida ikkita o'zgaruvchi saqlaymiz:

```
document.cookie = "ismi="
  + escape("Ali Jon")
  + "; expires=20/02/2080 00:00:00";
document.cookie = "yosh=28; expires="
  + new Date(2080, 1, 20).toGMTString();
```

```
ismi=Ali%20Jon;
ismi=Ali%20Jon;
yosh=28;
```

Kuki bu oddiy o'zgaruvchi emas. Unga bir necha qiymat berilganda, barchasini bitta satrga biriktirib boraveradi. Ma'lum qiymatni olib tashlashga to'g'ri kelsa, uning yashash vaqti sifatida ayni hozirni yoki o'tib ketgan muddatni berish kifoya:

```
document.cookie
```

```
ismi=Ali%20Jon;
yosh=28;
```

```
document.cookie = "yosh="
  + qiymat + "; expires="
  + new Date().toGMTString();
```

```
ismi=Ali%20Jon;
```

Bunday o'chirishda qanday qiymat berishning ahamiyati yo'q. Umri tugagan o'zgaruvchi chiqarib tashlanadi. Qiymat berilganda, avval o'sha nomdagi o'zgaruvchi mavjud bo'lsa, qo'shimcha yozilmaydi, yangilanadi. Ushbu imkoniyatdan qiymatni o'zgartirishda foydalanish mumkin. Kukidan foydalanar ekanmiz, kod yozishda qulaylik yaratish uchun u bilan ishlashni soddalashtiradigan zarur funksiyalar yaratib olishimiz kerak:


```
function saqlash(nomi, qiymat, soat) {
  var d = new Date();
  d.setTime(d.getTime() + soat*60*60*1000);
  var muddat = "expires=" + d.toUTCString();
  document.cookie = nomi + "=" + qiymat + ";"
    + muddat + ";path=/";
}
saqlash('kitob',  2,   72);
saqlash('sahifa', 235, 72);
function olish(nomi) {
  var n = nomi + "=";
  var k = decodeURIComponent(document.cookie);
  var q, i, j = k.split(';');
  for (i = 0; i < j.length; i++) {
    c = j[i];
    while (c.charAt(0) == ' ') c = c.substring(1);
    if (c.indexOf(n) == 0)
      return c.substring(n.length, c.length);
  }
  return undefined;
}
olish('kitob')
olish('sahifa')
```

```
"2"
"235"
```

Kitoblardan birini tanlab o'qish uchun veb-sahifa yaratadigan bo'lsak, har bir foydalanuvchiga oxirgi o'qiyotgan joyini saqlab borish juda qulaylik yaratadi. Bexosdan brauzer qayta yuklansa, yopib yuborilganda yoki server bilan aloqa uzilganda, joriy o'rindan qayta ko'rish mumkin bo'lishi kerak. (Hozirda to'g'ridan-to'g'ri kino ko'rishga mo'ljallangan ayrim saytlar ushbu usuldan foydalanishmoqda.) Buning uchun kitob tanlangan-

da va varaqlanganda, ularning maxsus qiymatlari kukida saqlanadi. Agar surgich ishlatilsa, uning o'rnini ham inobatga olsa bo'ladi. Buning uchun maxsus funksiya yaratamiz:

```
function ochish (kitob, sahifa) {
  if (kitob && sahifa) {
    saqlash('kitob',  kitob,  7 * 24);
    saqlash('sahifa', sahifa, 168);
    saqlash('surish', window.pageYOffset, 168);
  } else {
    var k = olish('kitob');
    var b = olish('sahifa');
    var s = olish('surish');
    location.href = '#' + k + '-' + b;
    scrollTo({top: s, behavior: 'smooth' });
    return [k, b];
  }
}
```

Ushbu «ochish» nomli funksiyaga kitob va sahifasining tartib raqami berilsa, ularni hamda surgich holatini kukida bir hafta saqlaydi. Omillar berilmaganda ishoratda panjara belgisidan keyin mos sonlarni chiziqcha bilan ajratib qaytaradi hamda surgich holatini ohista oxirgi vaziyatga o'tkazadi.

Odatda, ro'yxatdan o'tishni talab qiluvchi ko'pgina saytlar kukida maxsus nom va kalit so'zlarni saqlaydi. Bu xavfsizlik nuqtayi nazaridan tavsiya etilmaydi, ayniqsa, shaxsiy ma'lumotlar yoki pul o'tkazmalari bilan ishlaydigan dasturiy ta'minotlarda. Kelgusida SSL bilan ishlamaydigan saytlar uchun kukidan foydalanishni cheklash rejalashtirilmoqda. Shuning uchun faqat unga

bog'lanib qoladigan kodlar yozish tavsiya etilmaydi, o'rniga «localStorage» [5.1.1.] qo'llash maslahat beriladi. Kukida vaqtinchalik foydalaniladigan yoki yashirin maqsadlarda ishlatish mumkin bo'lmagan ma'lumotlarni saqlash maqsadga muvofiq hisoblanadi.

5.3. Darchalar qiymatlari

Birinchi kitobda web-orazni (foydalanuvchi interfeysini) jozibador qilishni ko'rsatishga oid bilimlar bilan batafsil tanishildi. Keyingi muhim qadam kiritilgan ma'lumotlarni kutilgandagidek saqlash va qayta ishlashni amalga oshirish hisoblanadi. Buning uchun JavaScriptning imkoniyatlaridan foydalanish taqozo etiladi.

5.3.1. Qiymatni olish va o'zgartirish

Oddiy HTML forma yaratib, undagi darchalarning qiymatlarini ko'rsatilgan manzilga yuborish imkoniyati mavjudligi bilan tanishgan edik. Shu o'rinda bunga yana bir misol keltirib o'tamiz:

```
<form id="xodim" action="/yubor/" method="GET">
  <label for="ismi">Ismi:</label>
  <input type="text" name="ismi" required>
  <br>
  <label for="yosh">Yoshi:</label>
```

```
    <input type="number" name="yosh" min="9" max="60">
    <br>
    <label>Jinsi:</label>
    <label>
      <input type="radio" name="jins" value="1"> Erkak
    </label>
    <label>
      <input type="radio" name="jins" value="2"> Ayol
    </label>
    <br>
    <label for="malumoti"> Ma'lumoti </label>
    <select name="malumoti">
      <option value="1">O'rta</option>
      <option value="2">Oliy</option>
      <option value="3">Ega emas</option>
    </select>
    <br>
    <label>
      <input type="checkbox" name="azo"> A'zoligi
    </label>
    <input type="reset"  value="Tozalash">
    <input type="submit" value="Yuborish">
</form>
```

Albatta, HTMLdagi forma darchalarining o'ziga yarasha kamchiliklari mavjud. Masalan, sahifa yuklanishida barcha darchalar bo'sh bo'lib tursa ham `<select>` tayinlanuvchi darchasining birinchi elementi tanlangan holda namoyon bo'ladi. Boshlang'ich qiymatni ko'rsatib turish har doim ham maqbul yechim hisoblanmaydi, dastavval bo'sh holatda turishi uchun birinchi elementi sifatida «`<option></option>`» singari yozish mumkin, ammo tanlash vaqtidagi ro'yxatda u

ham ko'rinib qoladi. Ko'rinmasligini esa CSS yordamida ta'minlash mumkin:

```
<option style="display: none;"> </option>
```

Odatda, bunday ro'yxatlar dasturiy yo'l bilan takrorlash operatorlari qo'llangan holda yaratiladi. Unda yuqoridagidek element qo'shilishi doim ham qulay hisoblanavermaydi. Shuning uchun tayinlanuvchi ro'yxat namoyon bo'lgandan keyin unga JavaScript yordamida mavjud bo'lmagan qiymatni berish orqali, boshlang'ich holda bo'sh holatda ko'rinishini ta'minlash mumkin:

```
document.forms[0].elements[4].value = -1;
```

Ushbu kodda sahifadagi birinchi formaning beshinchi elementi `<select>` ekanligi inobatga olingan. Yuqorida HTML elementlarining takrorlanmas «id» alomati bilan o'zlashtirish batafsil bayon etilgan [5.2.]. Lekin o'ta zaruriyat bo'lmaganda «id» qo'llamaslik tafsiya etiladi. Elementlarga imkoni boricha dinamik-universal usullarda murojaat qilish kerak. Ko'p vaziyatlarda bitta tuzilgan interfeys kodidan foydalanib boshqasini yaratishga to'g'ri keladi va tabiiyki, nusxa ko'chiriladi. Kod juda katta bo'lgan hollarda barcha «id»larni o'zgartirib chiqish va ularga betakror nom qo'yish mushkul vazifaga aylanib qoladi. Kiritilayotgan «id» butun veb-sahifa uchun yagona bo'lishi qat'iy talab etiladi, elementlarning «name» alomatlariga beriladigan qiymatlar esa bitta formada takrorlanmasligi kifoya.

Yuqoridagi HTML formada ism kiritiladigan darchaga qiymat kiritilishi majburiyligi hamda yosh uchun

chegara 9 dan 60 gachaligi ko'rsatilgan. Brauzer undagi «submit» tugmasi bosilganda, yuborishdan oldin ushbu shartlarni bajarilganligini tekshiradi. Qaysi darchada qanday kamchilik topsa, unga mos maxsus ogohlantiruvchi xabarcha chiqaradi. Bunday tekshiruvni dasturlash tilida «**tasdiqlash**» deb yuritiladi.

Ammo o'ziga xos qandaydir qo'shimcha tekshiruv kiritish lozim bo'lsa, maxsus JavaScript funksiya yozishga to'g'ri keladi. Buning uchun formadagi barcha darchalarning yagona majmua sifatida o'zlashtiradigan «`document.forms[0].elements`»dan foydalanib, quyidagi misolni keltiramiz:

```
window.onsubmit = function() {
  let i, q = -1, e = document.forms[0].elements;
  for (i = 0; i < e.length; i++)
    if (e[0].value.search(/[\d]/g) >= 0) q = i;
  if (q >= 0) {
    e[q].focus();
    alert('Ism darchasida raqam qatnashmasligi shart!');
    return false;
  }
};
```

5.3.2. AJAX orqali ma'lumot almashish

Avvalgi mavzuda server bilan ma'lumot almashishni AJAX orqali amalga oshirish naqadar muhimligi ta'kidlab o'tildi. Endi ushbu jarayon amaliyotda qanday bajarilishi bilan aniq misollar yordamida tanishib o'tamiz.

Avvalo, dasturlashda har bir narsani, nafaqat o'zgaruvchi hamda o'zgarmaslarni, balki funksiya va hatto fayl-jildlarni ham nomlashga alohida e'tibor qaratish zarur. Bu tizimni avtomatlashtirishga juda katta yordam beradi.

Server bilan oldi-berdi qiladigan mukammal JavaScript funksiya tuzamiz. U murojaat boshlanganda foydalanuvchiga ogohlantiruvchi (kutishni iltimos qiluvchi) harakatlanish chiqishini, bosilgan tugma yoki obyekt o'z holatini o'zgartirishi (aytaylik, javob kelguncha nofaol bo'lishi) ta'minlanishini, qandaydir xatolik yuz berganda uning qaysi fayldagi nechanchi qatordaligini hamda imkon qadar aniq sababini ko'rsatuvchi betakror vositalarga ega bo'lishi lozim.

Foydalanuvchi serverga murojaat bo'lishini anglatuvchi biror harakat amalga oshirganda, odatda, biror JavaScript funksiyani bajartiradi, ya'ni ishga tushiradi. Shunday funksiyalar ikki qismdan tashkil topishi lozim. Birinchidan, harakatga mos ravishda interfeysdan hisob-kitoblar va o'zgarishlarni amalga oshirish hamda kerakli ma'lumotlarni serverga yuborish. Ikkinchi bosqichda esa serverdan kelgan javob asosida belgilangan ishlarni bajarish. Serverdan javob qancha vaqtda va qanday ko'rinishda kelishi aniq emas. Kutilmagan xatoliklar yuz berishi ham mumkin. Shuning uchun server bilan aloqa qiladigan har bir funksiya ichidagi ikkita bosqichning o'zaro bog'liqligini ta'minlovchi alohida maxsus funksiya bo'lishi shart. Biz aynan shunday funksiyaga misol tuzamiz va uni shartli ravishda «`aloqaAJAX`» deb nomlaymiz. Uni ishlatuvchi, yuqorida

ta'kidlanganiday, ikki bosqichdan iborat funksiyalarni «chaqiruvchi» deb yuritamiz. Demak, barcha chaqiruvchi funksiyalar server bilan oldi-berdini ta'minlashi uchun yagona «aloqaAJAX» funksiyaga murojaat qiladi.

Bunday funksiya yaratishdan avval asosiy vazifani bajaruvchi hamda «va'da» [3.3.3.] qaytaruvchi JavaScriptga yangi kiritilgan «**fetch**» maxsus funksiyasini (oldin «XMLHttpRequest» qo'llanilardi) batafsil bayon etib o'tish lozim :

fetch(*ishorat*[,*omillar*]).then(*funksiya*).catch(*funksiya*)

Bunda «then» va «catch» va'daning keyingi bosqichlari [3.3.3.] hisoblanadi va ular qatnashishi shart emas. Undagi «funksiya»lar mos ravishda muvaffaqiyatli amalga oshganda yoki xatolik sodir bo'lganda bajariladi. «Ishorat» oddiy veb-manzilni aniqlaydi. «Omillar» sifatida quyidagicha indeksli qiymatlarga ega yagona obyekt ko'rsatilishi mumkin:

- **method** — serverga murojaat qilish uslubi: POST, GET, CONNECT, DELETE, HEAD, OPTIONS, PATCH, PUT, TRACE kabi qiymatlardan birini qabul qiladi.
- **headers** — bosh qismida brauzerning tavsif ma'lumotlari bilan birga yuboriladigan qiymatlar.
- **body** — so'rovning tana qismi, unda serverga yuboriladigan ma'lumotlar jamlanadi. E'tibor qilish lozimki, agar uslub HEAD yoki GET bo'lsa, tana qismi mavjud bo'lmaydi.
- **mode** — usul «cors», «no-cors» yoki «same-origin» qiymatlaridan birini qabul qiladi.

- **credentials** — serverga kukini qanday yuborishni aniqlaydi, «omit», «same-origin» yoki «include» qiymatlaridan birini qabul qiladi.
- **cache** — so'rovni saqlab turish yo'rig'i. Serverga ayni bir xil so'rov bilan qayta-qayta samarasiz murojaat qilaverishning oldini oladi, «default», «only-if-cached», «no-store», «reload», «no-cache» yoki «force-cache» singari qiymatlardan birini qabul qiladi.
- **redirect** — boshqa ishoratga sakrashni nazorat qiladi. Unga odatiy qiymati «follow» berilsa, sakrashlar hech qanday to'siqsiz amalga oshiriladi. Agar «error» berilgan bo'lsa, boshqa ishoratga yo'naltirilgan holda xatolik yuz beradi va jarayon to'xtaydi. Sakrashlarni o'ziga xos ravishda boshqarish lozim bo'lsa, «manual» qiymat berish lozim.
- **referrer** — ko'rsatilgan ishoratga sakrashni ta'minlaydi. Unga aniq ishorat yoki «no-referrer» qiymatini berish mumkin. U «redirect» bilan mutanosib ravishda ishlaydi. Odatiy qiymati «client» sanaladi.
- **referrerPolicy** — qanday turdagi so'rovlar yo'naltirilishi lozimligini aniqlaydi. U ushbu «unsafe-url», «no-referrer-when-downgrade», «no-referrer», «origin», «strict-origin-when-cross-origin» hamda «strict-origin», «same-origin» va «origin-when-cross-origin» qiymatlardan birini qabul qiladi.
- **integrity** — yuborilayotgan so'rov serverga to'liq yetib borganini tekshirish uchun qo'llaniladigan maxsus o'girilgan satr. Ma'lumki, ma'lumotlar tarmoqda

paketlar tartibida uzatiladi va qaydnomalar qanchalik ishonchli bo'lmasin, orada qandaydir paketlar yo'qolib qolishi mumkin. Serverda to'liq yig'ilgandan keyin so'rovni maxsus kodlash tizimi yordamida o'giriladi. Masalan, «md5» doim 32 ta belgidan iborat satr qaytaradi va u bitta paketning ichiga sig'adi. Serverga kelgan so'rovni o'girish yordamida hosil bo'lgan 32 ta belgili satrni JavaScript yuborgan so'rov bilan taqqoslanadi. Farq bo'lsa, shunga munosib javob qaytarishni ta'minlash mumkin.

- **keepalive** — serverga murojaatlar juda ko'p bo'lganda joriy so'rovni navbatga qo'yishni va yo'qolib qolmasligini ta'minlash uchun «rost» qiymat beriladi.
- **signal** — so'rovni bekor qilish vaziyatida qanday amallar bajarilishini aniqlash uchun qo'llaniladi.

Ushbu elementlarning barchasini qo'llash shartmas, ularning har biri o'z odatiy qiymatiga ega va brauzer tomonidan boshqariladi. Faqat kerakli elementlarga qiymat beriladi va zarurlarining qiymati o'zgartiriladi.

Ishorat sifatida qaydnomasi, domeni va porti ayni saytniki bilan bir xil bo'lgan manzil ko'rsatilishi talab etiladi, aks holda server kutilgan javob qaytarmasligi mumkin. Odatda, ko'pgina serverlarda boshqa manzildan kelgan so'rovlarni rad etuvchi dasturiy ta'minot mavjud. Uni «**CORS**» (Cross-Origin Resource Sharing — manbalararo vositalar ulashish) deb yuritiladi. Albatta, talabga qarab, tizim ma'muri serverni boshqa manzillardan ham ma'lumot qabul qilishga mo'ljallab sozlashi mumkin. Faqat qaydnoma bir xil bo'lishi shart.

Masalan, «https» bilan ishlaydigan sayt «http» qaydnomali manzilga murojaat qilmasligi kerak, chunki ularning serverga bog'lanish texnologiyasi bir-biridan butunlay farq qiladi.

Misol tariqasida maxsus «id» alomatiga ega bo'lgan HTML formada «submit» tugmasi xodimlarga tegishli barcha kiritilgan va tanlangan ma'lumotlarni serverga saqlash uchun yuboradigan «xodimSaqla» funksiyasini ishga tushiradigan bo'lsin. Agar saqlash muvaffaqiyatli amalga oshsa, forma yashirinishi va tegishli xabar chiqishi lozim. Xatolik yuz bersa ham o'ziga xos tarzda ogohlantirishi kerak. HTML kodi deyarli oldingi mavzudagi [5.3.1.] singari bo'lgani uchun faqat JavaScript kodining o'zini keltiramiz. Chaqiruvchi funksiya nomini serverda murojaat qilinayotganda qo'llaniladigan manzil «yo'li» (path) sifatida ham ishlatuvchi va aynan o'sha funksiya qaytgan javobni yuboruvchi «aloqaAJAX» kodi:

```
function aloqaAJAX(data) {
  var btnClk, method = 'post', dataType = 'json';
  // Yuboriluvchi qiymatlar obyekti berilmagan bo'lsa, bo'sh
  if (!data) data = {};
  // Bosilgan tugmani nofaollashtirib turish
  if (data.btn) {
    if (typeof data.btn == "string")
      btnClk = document.getElementById(data.btn);
    else btnClk = data.btn;
    btnClk.setAttribute("disabled", true);
    delete(data.btn);
  }
  // Tavsif-ma'lumot:
  let mime = "application/x-www-form-urlencoded;charset=UTF-8";
```

```
// Yuboriluvchi uslubni aniqlash, berilmagan bo'lsa, post:
if (data.method) {
  method = data.method;
  delete(data.method);
}
// Qaytuvchi ma'lumot turi, berilmagan bo'lsa: JSON
if (data.dataType) {
  dataType = data.dataType;
  delete(data.dataType);
}
// Chaqiruvchi funksiya nomini olyapti
var functionName = '',
  fn = arguments.callee.caller;
if (fn)
  fn = /\W*function\s+([\w\$]+)\(/.exec(fn);
if (fn) functionName = fn[1].toString();
// Kuting ma'nosidagi harakatlanish chiqarish mumkin

fetch(location.origin + "/" + functionName, {
  method: method,
  headers: { 'Content-type': mime },
  body: new URLSearchParams(data)
}).then(
  function (natija) {
    // Bosilgan tugmani qayta faollashtirish
    btnClk.removeAttribute("disabled");
    // Kutish harakatlanuvchisi (bo'lsa) bekor qilinadi
    let r;
    console.log('Natija qaytdi:', natija);
    if (natija.status !== 200) {
      alert('HTTP status: ' + natija.status);
      return false;
    }
```



```
    switch (dataType) {
      case 'json' :
        r = natija.json();
      break;
      case 'blob' :
        r = natija.blob();
      break;
      case 'form' :
        r = natija.formData();
      break;
      case 'array':
        r = natija.arrayBuffer();
      break;
      default: r = natija.text();
    }
    return r;
  }
).then(
  function (result) {
    let q = {"qaytdi": true, data: result };
    if (result && functionName)
    window[functionName](q);
  }
).catch(
  function (error) {
    console.log('Xatolik yuz berdi', error);
  }
);
}
```

Birinchi «then» qaytargan «r»ni ikkinchisi «result» sifatida qabul qiladi va ustida amallar bajaradi. Xatolik yuz bersa, «catch»ga bu haqida obyekt yuboradi.

Muvaffaqiyatli amalga oshirilganda chaqiruvchi funksiyaning o'ziga qayta murojaat qiladi va unga elementida «"qaytdi": true» mavjud maxsus obyekt yuboradi. Buni inobatga olgan chaqiruvchi funksiya quyidagi ko'rinishda ikki qismdan iborat tarzda yozilishi kerak:

```
function xodimSaqla(data) {
  var f = document.getElementById("xodim");
  if (data && data.qaytdi) {
    // Serverdan javob qaytdi
    f.innerHTML = "<p>Ma'lumotlar saqlandi.</p>";
  } else {
    // Serverga so'rov yuborish jarayoni
    var i, yubor = {};
    for (i = 0; i < f.length; i++) {
      yubor[f[i].name] = f[i].value;
      if (f[i].type == 'submit')
        yubor["btn"] = f[i];
    }
    aloqaAJAX(yubor);
  }
}
```


6-BO'LIM

SHAKLLARNI HARAKATLANTIRISH

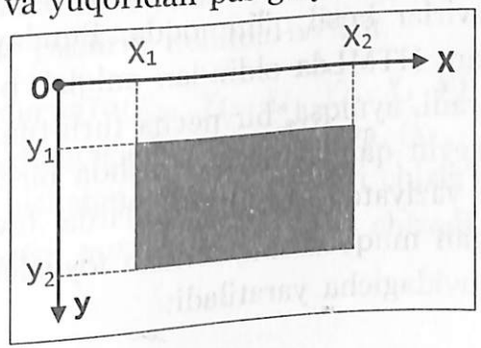
6.1. «canvas» obyekti

Birinchi kitobda bayon etilgani kabi HTMLning <canvas> tegi xuddi <img> singari tasvirni akslantirishga qaratilgan. Faqat <canvas> ichida JavaScript orqali turli shakllar chizish imkoniyati mavjud. Teg sintaksisini eslatib o'tish maqsadga muvofiq:

```
<canvas id="nomi" width="300" height="300"></canvas>
```

Albatta, «nomi» veb-sahifa uchun takrorlanmas identifikator bo'lishi shart. Eni va bo'yi piksel hisobida aniqlanib, ko'rsatilmasa, odatda 300x150 o'lchamda bo'ladi. Nomi orqali <canvas> obyektini JavaScript o'zgaruvchisiga o'zlashtirib olinadi va unga maxsus uslublar yordamida turli shakllar chiziladi.

U to'g'ri to'rtburchak sifatida aniqlanadi va undagi har bir nuqta Dekart koordinata tizimi bo'yicha chapdan o'ngga va yuqoridan pastga yo'nalishida aniqlanadi:



JavaScriptda canvas obyektida to‘g‘ri to‘rtburchak chizish uchun maxsus uslublar mavjud:

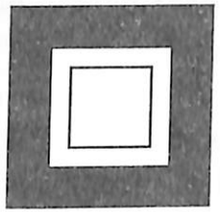
- **fillRect(*X₁, Y₁, eni, bo‘yi*)** — to‘g‘ri to‘rtburchak shaklidagi maydon hosil qiladi.

- **strokeRect(*X₁, Y₁, eni, bo‘yi*)** — to‘g‘ri to‘rtburchak shaklidagi chegara (perimetr) chizadi.

- **clearRect(*X₁, Y₁, eni, bo‘yi*)** — to‘g‘ri to‘rtburchak shaklidagi maydonni tozalaydi, ya’ni u joyni o‘chirib, shaffof rang beradi.

Shakllar chizishdan avval canvas maydonini ikki o‘lchamli shaklda o‘zlashtirib olamiz va uchta ichma-ich kvadrat chizamiz:

```
const canvas = document.getElementById('canvas');
if (canvas && canvas.getContext) {
   var ch = canvas.getContext('2d');
   ch.fillRect(20, 20, 100, 100);
   ch.clearRect(40, 40, 60, 60);
   ch.strokeRect(50, 50, 40, 40);
}
```



HTMLda aniqlangan shakllar chizishga mo‘ljallangan maydonni «ch» o‘zgaruvchisiga yuklab olindi va unda bilvosita tasvirlar hosil qilinmoqda. Bunday maydonni har doim ham HTMLda oldindan aniqlab borish noqulaylik tug‘diradi, ayniqsa, bir necha turli-tuman shakllar hosil qilib keyin qandaydir ko‘rinishda birlashtirish lozim bo‘lgan vaziyatda. Bunday hollarda JavaScript o‘zi hosil qiladigan muqobil maydondan foydalanish tavsiya etiladi. U quyidagicha yaratiladi:


```
var chiz = new Path2D();
```

Ushbu usulda yoki <canvas> tegini o‘zlashtirish orqali hosil qilingan rasm chizish imkoniyatini yaratuvchi obyektni kelgusida «tasvir maydoni» deb yuritamiz.

JavaScriptda canvas maydoniga tasvir chizish SVG-dagiga o‘xshab ketadi, ammo bunda yoy, ellips va to‘g‘ri to‘rtburchakdan tashqari tayyor shakllar mavjud emas. Boshqa tasvirlarni go‘yoki qalamda qog‘ozga chizilgan kabi, «path» imkoniyatlaridan foydalanib, yopiq chiziq tasvirlanadi va lozim bo‘lsa, berilgan rangga bo‘yaladi. Buning uchun quyidagi uslublar qo‘llaniladi:

- **beginPath()** — yangi yopiqchiziq boshlanganini anglatadi. U yopiqchiziqlarni bir-biridan farqlash uchun qo‘llaniladi, har biri uchun alohida amallar bajarish mumkin, masalan, turlicha rang berish.

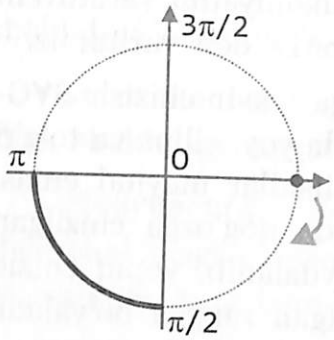
- **moveTo(*X₁, Y₁*)** — «qalam» uchini berilgan nuqtaga keltirish. Shu joydan boshlab chizishni anglatadi.

- **lineTo(*X₂, Y₂*)** — oxirgi nuqtadan berilgan nuqtagacha to‘g‘ri chiziq chizadi.

- **stroke()** — «qalam»ni chizishdan to‘xtatib, hosil bo‘lgan shaklni ko‘rsatadi. Ushbu buyruq berilmagunca chiziqlar yashirin holatda bo‘ladi.

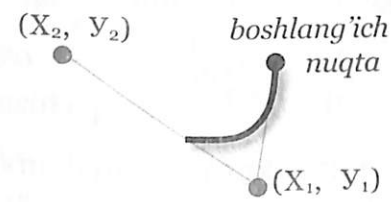
- **bezierCurveTo(*X₁, Y₁, X₂, Y₂, X, Y*)** — berilgan (X, Y) koordinatagacha (X_1, Y_1) va (X_2, Y_2) nazorat nuqtalariga nisbatan «**Beziyer**» egri chizig‘ini (*birinchi kitobda batafsil yoritilgan – J.A.*) chizadi.

▪ **arc(*X*, *Y*, *radius*, *boshNuqta*, *so'ngNuqta*[, *aks*])** — yoy chizadi. Yoy markazi Dekart koordinatalar tizimi markazida joylashgan deb qaralganda, X o'qida yotgan nuqtadan boshlab, soat millari yo'nalishida chiziladi. Yoy ushbu aylanada yotadi, (X, Y) nuqta aylananing markazi va undan keyingi qiymat radiusini aniqlaydi. Keyingi ikki qiymat radian hisobida boshlang'ich va yakuniy chegaralarini bildiradi. Agar soat miliga qarshi yo'nalishda aniqlanishi kerak bo'lsa, «aks»ga «rost» qiymat beriladi:




```
chiz.beginPath();
chiz.arc(50, 50, 30, Math.PI / 2, Math.PI);
chiz.stroke();
```

▪ **arcTo(*X₁*, *Y₁*, *X₂*, *Y₂*, *radius*)** — berilgan radius bo'yicha (X_1, Y_1) va (X_2, Y_2) nazorat nuqtalari hosil qilgan kesmagacha «Beziyer» egri chizig'ini hosil qiladi. Ushbu yoysifat chiziq turli shakllarni bir-biriga ulashda foydalaniladi.



▪ **ellipse(*X*, *Y*, *radiusX*, *radiusY*, *burchak*, *boshNuqta*, *so'ngNuqta*[, *aks*])** — ellips asosida yoy chizadi. Xuddi «arc» singari aniqlanadi. Faqat ellips qovunsifat ko'rinishda bo'lgani uchun unda ikkita radius mavjud hamda uni istalgan radian «burchak»ka og'dirish mumkin.

▪ **rect(*X₁*, *Y₁*, *X₂*, *Y₂*)** — shu ikki nuqtada uchlari qarama-qarshi yotuvchi to'g'ri to'rtburchak chizadi.

▪ **closePath()** — oxirgi nuqtada «beginPath»dan so'ng o'rnatilgan boshlang'ich nuqtagacha kesma chizadi. Ya'ni shakl umumiy yopiq chiziq hosil qilmagan bo'lsa, uni «yopadi».

▪ **createLinearGradient(*X₁*, *Y₁*, *X₂*, *Y₂*)** — chiziqli tuslanuvchi rang yo'nalishini (X_1, Y_1) dan (X_2, Y_2) nuqtaga tomon o'rnatadi.

▪ **addColorStop(*qism*, *rang*)** — berilgan «rang» yuqorida keltirilgan «createLinearGradient» orqali aniqlangan kesmaning qayerida o'rnatilishini belgilaydi. Bunda «qism» sifatida noldan birgacha haqiqiy son beriladi va kesmadagi mos o'rnini anglatadi.

▪ **fillStyle** — ushbu xususiyatga shaklni bo'yash rangi beriladi. Oddiy rangdan tashqari, yuqoridagi usulda aniqlangan tuslanuvchi rang yoki shunday qiymatlarga ega JavaScript o'zgaruvchisini o'zlashtirsa bo'ladi:

```
let g = chiz.createLinearGradient(0, 0, 200, 0);
g.addColorStop(0,   'gray');
g.addColorStop(0.7, 'white');
g.addColorStop(1,   'black');
chiz.fillStyle = g;
chiz.fillRect(5, 10, 250, 250);
```

▪ **fill()** — yopiq chiziqni belgilangan tusda bo'yaydi. Agar «fillStyle» aniqlanmagan bo'lsa, qora rang bilan to'ldiriladi.

▪ **clip([*maydon*[, *uslub*]])** — Path2D yordamida aniqlangan «maydon»ning chizilgan qismini kesib ajratib oladi, keyingi chizmalar faqat o'sha maydoncha ichida ko'rinadigan bo'ladi. «Uslub» sifatida chetki hadlar bo'yicha to'la qamrab olingan maydonni olish uchun «**nonzero**» yoki yopiq chiziqda hadlar bir-biri bilan kesishganda ustma-ust yotgan qismni olib tashlab, faqat bitta qatlamli shaklni ajratib olish uchun «**evenodd**» qiymati beriladi. Quyidagi misolda doirani ajratib olish ko'rsatilgan:

```
chiz.beginPath();
chiz.arc(150, 150, 100, 0, Math.PI * 2);
chiz.clip(); // maydonning doira qismi olindi
chiz.fillStyle = 'gray'; // tag qatlam rangi
chiz.fillRect(0, 0, 300, 300); // to'q qismi
chiz.fillStyle = 'lightgray'; // ustki rang
chiz.fillRect(0, 0, 200, 200); // och qismi
```

▪ **isPointInPath(X, Y[, *uslub*])** — berilgan (x, y) koordinatali nuqta chizilgan shakl ichida yotishini tekshiradi, mantiqiy qiymat qaytaradi.

▪ **isPointInStroke(X, Y)** — xuddi «isPointInPath» singari ishlaydi, faqat bunda «uslub» ko'rsatilmaydi va shaklning ichki qismidagi ochiq joyda yotgan nuqta ko'rsatilsa ham «yolg'on» qiymat qaytaradi. Masalan, «arc» bilan aniqlangan shakl uchun «isPointInPath» doirada, «isPointInStroke» esa aylanada yotadigan nuqtalar uchun «rost» qiymat qaytaradi. Ikkisini aniq farqlash uchun «**c**» o'zgaruvchi bilan aniqlangan maydonda sichqoncha ko'rsatkichi aylanaga kirganda sariq, doira

qismiga o'tganda esa qizil tusga kiruvchi kulrang hoshiyali yumaloq shaklni tasvirlovchi misol keltiramiz:

```
c.arc(150, 150, 100, 0, 2 * Math.PI);
c.lineWidth = 50;
c.strokeStyle = 'gray';
c.fill();
c.stroke();
canvas.addEventListener('mousemove',
  function(e) { // sichqoncha harakatini o'qish funksiyasi
    if (c.isPointInStroke(e.clientX, e.clientY)) {
      c.strokeStyle = 'yellow';
    } else
      if (c.isPointInPath(e.clientX, e.clientY)) {
        c.strokeStyle = 'red';
      } else c.strokeStyle = 'gray';
    c.clearRect(0, 0, canvas.width, canvas.height);
    c.fill(); // maydonni to'la tozalab, qaytadan chizadi
    c.stroke();
  }
);
```

▪ **rotate(*burchak*)** — maydonni soat millari yo'nalishida radian sifatida berilgan «burchak»ka og'diradi. 360°li burchakka o'girish uchun **π/180** ga ko'paytirish kifoya. Misol:

```
chiz.rotate(20 * Math.PI / 180);
chiz.fillRect(150, 50, 50, 10);
```

▪ **scale(*n*, *k*)** — masshtab: X o'qi bo'yicha **n**, Y o'qi bo'yicha **k** marta shaklni kattalashtiradi. Kichiklashtirish lozim bo'lsa, birdan kichik son berish kifoya.

▪ **translate(*X*, *Y*)** — mos ravishda X va Y o'qi bo'yicha shaklni berilgan birlikka suradi.

▪ **transform(*n11*, *n12*, *n21*, *n22*, *n31*, *n32*)** — uch o'lchamli matritsa hosil qilib (unda $n_{13}=0, n_{23}=0, n_{33}=1$), har bir koordinatani u orqali akslantiradi (ko'paytirib hisoblaydi). Beriladigan oltita sonning har biri alohida ahamiyat kasb etadi: n_{11} — yotiq masshtab, n_{12} — tik og'ish, n_{21} — yotiq og'ish, n_{22} — tik masshtab, n_{31} — yotiq surilish, n_{32} — tik surilish. Misol:

```
chiz.transform(1, 0.1, 0.2, 1, 0, 0);
chiz.fillRect(0, 0, 100, 20);
```

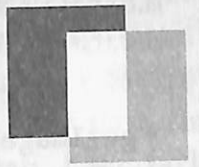
▪ **setTransform(*n11*, *n12*, *n21*, *n22*, *n31*, *n32*)** — avval o'rnatilgan o'giruvchi matritsani bekor qilib, berilgan qiymatlar bo'yicha qaytadan aniqlaydi.

▪ **globalAlpha** — shaffoflik darajasi, misol:

```
chiz.globalAlpha = 0.5;
chiz.fillStyle = 'gray';
chiz.fillRect(10, 10, 100, 100);
chiz.fillStyle = 'lightgray';
chiz.fillRect(60, 30, 100, 100);
```

▪ **globalCompositeOperation** — turli rangdagi qatlamlar ustma-ust tushgan joylarni qanday tusda suvashtirib ko'rsatishni aniqlaydi. Aniqlangan turdagi quyidagi qiymatlardan birini qabul qiladi: «darken», «source-over», «source-in», «source-out», «soft-light», «source-atop», «screen», «overlay», «destination-over», «destination-in», «color-burn», «xor», «color-dodge», «destination-out», «destination-atop», «color», «hue», «lighter», «lighten», «multiply», «copy», «hard-light», «difference», «exclusion», «saturation», «luminosity».


```
chiz.globalCompositeOperation = 'xor';
chiz.fillStyle = 'gray';
chiz.fillRect(10, 10, 100, 100);
chiz.fillStyle = 'lightgray';
chiz.fillRect(60, 30, 100, 100);
```



▪ **drawImage(*rasm*, *X*, *Y*, [*eni*, *bo'yi*, *X2*, *Y2*, *eni2*, *bo'yi2*])** — mavjud <img> obyektidagi rasmni o'zlashtiradi. Bunda rasmning bo'lagi, X va Y koordinatalar hamda «eni» va «bo'yi» bo'yicha, to'g'ri to'rtburchak sifatida ajratib olinadi. Agar omillar beshtadan ko'p berilsa; avval <img> obyekti, so'ngra <canvas> maydoni o'lchamlari nazarda tutiladi.

▪ **createImageData(*eni*, *bo'yi*)** — to'g'ri to'rtburchak sifatida maydonning har bir nuqtasidagi rangni aniqlash uchun maxsus jadval yaratadi. Har bir nuqta rangi uchun to'rtta son saqlaydi va ular mos ravishda qizil, yashil, ko'k va shaffoflikni anglatadi. Demak, jadvalning umumiy uzunligi **«eni»*«bo'yi»*4** ta bo'ladi. Omillardagi «eni» va «bo'yi»ni aniqlovchi sonlar o'rniga avvaldan aniqlangan «imageData»ni yagona qiymat sifatida berish ham mumkin. Bunda mavjud jadvaldan nusxa ko'chirilmaydi. Faqat uning o'lchami o'zlashtiriladi, xolos. Ushbu uslub maydonning har bir nuqtasiga istalgan rangni berish imkoniyatini yaratadi:

```
const m = chiz.createImageData(100, 100);
for (i = 0; i < m.data.length; i += 4) {
  p = (i / 4000) * 25;
  m.data[i + 0] = p;      // qizil qiymati
  m.data[i + 1] = p;      // yashil qiymati
```

```
    m.data[i+2] = p;      // ko'k qiymati
    m.data[i+3] = 255;    // shaffofligi
};
```

- **getImageData(*X*, *Y*, *eni*, *bo'yi*)** — maydonning ko'rsatilgan omillar bo'yicha to'g'ri to'rtburchak qismini o'zlashtirib oladi.

- **putImageData(*jadval*, *X*, *Y*)** — «getImageData» yoki «createImageData» yordamida hosil qilingan «jadval»ni ko'rsatilgan koordinataga chiqaradi. Bu uslublar mavjud shakllarni yuklab olib, faqat ayrim nuqtalarining rangini o'zgartirib, boshqa joyga chiqarish uchun juda qulay vosita hisoblanadi.

- **drawFocusIfNeeded(*element*)** — <canvas> ichidagi faol elementning chegarasini ko'rinadigan qiladi. Umuman olganda, <canvas> tegi ichida boshqa teglar yozish tavsiya etilmaydi. Lekin qat'iy taqiqlanmaganligi bois qo'llash mumkin. Aslida ichki elementlar ko'rinmaydi. Shunday bo'lsa-da, faollashganini ajratib ko'rsatish uchun ushbu uslubdan foydalanish mumkin:

```
<canvas id="canvas">
  <input id="button" value="click">
</canvas>
const canvas = document.getElementById("canvas");
const chiz   = canvas.getContext('2d');
const button = document.getElementById("button");
button.focus();
chiz.beginPath();
chiz.rect(10, 10, 120, 28);
chiz.drawFocusIfNeeded(button);
```


Ushbu bayon etilgan imkoniyatlardan foydalanib, brauzerda <canvas> obyekti ichida ixtiyoriy tasvirlar chizish va ularni sichqoncha harakati hamda vaqt o'tishiga bog'lab harakatlantirish mumkin. Agar dasturlash bilan birga ozgina musavvirlik mahorati ham bo'lsa, qiziqarli o'yinlar yaratishga ham hamma vositalar yetarli. Muhimi, bunday usulda yasalgan obyektlarni akslantirishda kompyuterlar zo'riqmasdan ishlaydi (kam xotira va quvvat sarf etadi).

6.2. Suriluvchan oynalar

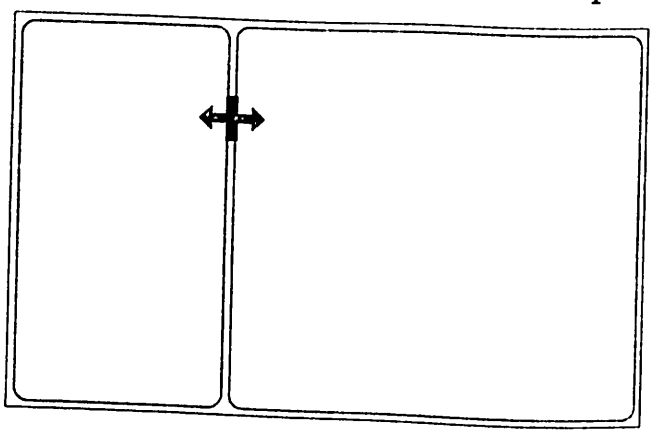
Mukammal dasturlashga qo'yiladigan talablardan asosiysi — bu moslashuvchanlik. Ya'ni ayni bir kod bir necha talablar bo'yicha ham umumiy vazifasini bajara olishidadir. Veb-sahifa yaratishda ham umumiy qolipdan foydalangan holda turli-tuman jozibadorlik xususiyatlari bilan namoyon etish keng qo'llaniladi. Buning uchun nafaqat rang va husnixatlarni o'zgartirish, balki shakllarni ham moslashtirishga to'g'ri keladi.

Shunday talablarga javob beradigan kichik bir misol keltiramiz. Buning uchun amaliyotda keng tatbiq etiladigan, katta ehtiyoj bilan qo'llaniladigan oynalarni surish mashqini ko'rib o'tamiz. Aytaylik, sahifa quyidagi rasmda tasvirlangandek, ikkita bosh oynadan iborat bo'lib, chap tomonida yordamchi ma'lumotlar va o'ng qismida asosiy tarkibi namoyon etiladigan bo'lsin. Chap qismda akslanuvchilarning hajmlari o'zgarsa, o'lchami

ham unga moslashishi hamda surgich paydo bo'lmasligi talab etilgan bo'lsa, albatta, o'ng taraf o'lchami ham o'z-o'zidan o'zgaradi.

Foydalanuvchi uchun oynalar o'lchamini o'zi istaganday sozlash darkor bo'lsa, CSSning imkoniyatlari bunga ojizlik qiladi va uni JavaScript yordamida amalga oshirishni taqozo etadi.

Foydalanuvchi oynalar chegarasini surish uchun, avvalo, sichqoncha ko'rsatkichi u yerga borganda, ko'rinishini o'zgartirishi kerak. Buning uchun ikkala oyna orasida tik chiziq shaklidagi yana bir HTML obyekt yasab, uning uchun ko'rsatkich qiymatini «colresize» beriladi. (Quyidagi rasmdagi shaklni hosil qilamiz.)



Bo'yi oynalar bilan deyarli teng va ensiz bo'ladigan ushbu obyektni ixtiyoriy blokli teg yordamida yaratish mumkin. Lekin, quyida keltiriladigan misolimizda, avvalo o'rganish maqsadi muhim bo'lganligi bois, ma'no jihatdan yotiq chiziq yasaydigan <hr> tegini CSS imkoniyatlari yordamida tik shaklda ifodalaymiz va uni JavaScript kodi orqali hosil qilamiz.

Maydonlarning hajmini o'zgartiruvchi, aniqrog'i, ular orasidagi chegarani suruvchi JavaScript funksiyasini «maydonlar» deb nomlaymiz. Unga HTML teglar identifikatorlarini beramiz. Qolgan amallarni o'zi bajaradigan mukammal funksiya yozib, bu funksiyani «**md.js**» nomli alohida faylda saqlaymiz.

Birinchi kitobda o'rganganlarimiz asosida quyidagicha HTML va CSS kodlaridan iborat fayl yaratamiz:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Mukammal dasturlash. 1-mashq.</title>
    <script src="md.js"></script>
    <style>
      html, body { margin: 4px 2px; }
      #chapQism, #oongQism {
        border: 1px solid lightgray;
        height: calc(100% - 8px);
        position: absolute;
        border-radius: 7px;
        top: 2px;
      }
    </style>
  </head>
  <body onload="maydonlar('chapQism', 'oongQism');">
    <div id="chapQism"></div>
    <div id="oongQism"></div>
  </body>
<html>
```

HTML kodda faqatgina ikkita yaratilgan. Ularning nomlari sahifa yuklanib bo'lgandan keyin chaqiriluvchi JavaScript funksiyasiga berilgan.

CSS yordamida ikkala ning umumiy shamoyili aniqlangan. Har bir nuqta kattaligiga e'tibor qaratish lozim, aks holda andoza buzilib ketishi mumkin. Shuning uchun <body>ning hoshiya kattaligi aniq belgilab olindi. Sababi, u turli brauzerlarda turlicha bo'lishi mumkin. Hoshiyani inobatga olgan holda larning balandliklari va tepadan qancha masofadaligi aniq ko'rsatilgan. Muhimi, «position: absolute;» ko'rsatishni unutmaslik kerak, u oynalarni brauzerning tana qismida biz ko'rsatgan koordinatalarda joylashishini ta'minlaydi.

Alohida (md.js) faylda yozilgan JavaScript kodida tik chiziq — <hr> ustida sichqoncha bosilgandan keyin har uchala obyekt yuzasida harakatlanishini qo'yib yuborilmagungacha kuzatib boraveradi. Uning qiymatini «bos» o'zgaruvchisida saqlab boriladi. Bosilgan holda harakatlanayotgan ko'rsatgichning «X» koordinatasiga mos ravishda tik chiziqning o'rni, chap va o'ng oynalarning o'lchami o'zgaradi:

```
function maydonlar(div1, div2) {
  var chp = document.getElementById(div1), // chap oyna
  ong = document.getElementById(div2),     // o'ng oyna
  tik = document.createElement("hr"),      // tik chiziq
  eni = 400,     // chap oynaning eni
  bos = false;  // sichqoncha bosib turilganligi
  chp.style.width = eni + 'px';
  ong.style.left  = (10 + eni) + 'px';
```



```
  ong.style.width = 'calc(100% - ' + (16 + eni) + 'px)';
  tik.style.width = '4px';  // tik chiziq qalinligi
  tik.style.height = 'calc(100% - 20px)';
  tik.style.position = 'absolute';
  tik.style.left = (6 + eni) + 'px';
  tik.style.cursor = 'col-resize'; // sichqoncha ko'rsatkichi
  tik.style.border = '1px transparent'; // yashirin chiziq
  chp.parentNode.insertBefore(tik,ong); // chiziq yasash
  tik.onmousedown = function() {
    bos = true; // bosib turilganda ko'rsatkich o'zgaradi
    chp.style.cursor = 'col-resize';
    ong.style.cursor = 'col-resize';
  };
  ol = function() {
    bos = false; // qo'yib yuborilganda o'z asliga qaytadi
    chp.style.cursor = 'default';
    ong.style.cursor = 'default';
  };
  tik.onmouseup = ol; // surilayotganda ko'rsatkich o'ng yoki...
  chp.onmouseup = ol; // ... chap oynaga o'tib ketishi mumkin...
  ong.onmouseup = ol; // ... shuning uchun har 3 obyektga ...
  sur = function() { // ... shu funksiya biriktiriladi.
    if (bos) {
      eni = event.x - 10;
      chp.style.width = eni + 'px';
      ong.style.left  = (10 + eni) + 'px';
      ong.style.width = 'calc(100% - ' + (16 + eni) + 'px)';
      tik.style.left  = (6 + eni) + 'px';
    }
  };
  tik.onmousemove = sur; // surish funksiyasi ham har ...
  chp.onmousemove = sur; // ... 3 obyekt uchun ishlaydi, ...
  ong.onmousemove = sur; // ... shunda pirpirash bo'lmaydi.
}
```

6.3. Elementlarni harakatlantirish

JavaScript, eng avvalo, brauzerda namoyon bo'luvchi obyektlarni foydalanuvchi tushunishi va boshqarishi uchun sodda interfeys shaklida namoyon etish vazifasini bajaradi. Har qanday murakkab dasturiy ta'minot ham bir qancha sodda dastur bo'laklarining umumlashmasi hisoblanadi. Shunday ekan, JavaScript yordamida murakkab-mukammal dasturiy ta'minotni vujudga keltirish uchun, avvalo, unga asos bo'luvchi sodda kichik vazifalarni bajaradigan qismlarini yaratish lozim.

Ushbu kitobda mukammal dasturiy ta'minot yaratish uchun JavaScriptning zarur bo'ladigan deyarli barcha imkoniyatlari yoritib o'tildi. Endi ularni qo'llagan holda kichik bir mashq bajaramiz. Unda ixtiyoriy teg nomi asosida HTML obyekt yaratish, uning joylashgan o'rnini sichqoncha harakatiga mos ravishda surish va rangini o'zgartirish ko'rsatib o'tiladi. Bu mashqda shu uchta vazifa qanday bajarilishi ko'rsatiladi va faqat o'rgatish nuqtayi nazaridan bayon etiladi. O'quvchiga ularni takomillashtirgan holda o'zi mustaqil ravishda yangi amaliy mashqlar bajarish tasiya etiladi.

HTML fayl yaratib, ichiga asosiy teglar bilan birgalikda kerakli elementlarni hosil qiluvchi kodlar ham yozamiz:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Mukammal dasturlash. 2-mashq.</title>
```



```
    <script src="md.js"></script>
    <style>
      .bosildi {
        cursor: grabbing !important;
        box-shadow: 2px 2px 3px gray;
      }
    </style>
  </head>
  <body>
    <input type="number" min="2" max="9" value="3">
    ta
    <input type="text" value="button">
    <input type="button" value="yarat"
        onclick="yarat(this);">
    <input type="color">
  </body>
</html>
```

Birinchi darchada miqdor kiritiladi, ikkinchisida qanday nomdagi teg yaratilishi ko'rsatiladi. Uchinchi tugma «yarat» bosilganda kerakli elementlarni tegishli hodisalari bilan paydo qiluvchi funksiya — «**yarat**» chaqiriladi (ishga tushiriladi). Unga omil sifatida o'sha tugmaning o'zi (**this**) berilyapti. Sababi, uning yordamida JavaScript joriy elementdan tashqari yonidagilarni ham topib olishi qulay bo'ladi. Endi yuqoridagi HTML faylda ko'rsatilgan (ulangan) «md.js» fayli tarkibini batafsil izohlari bilan birga keltiramiz:

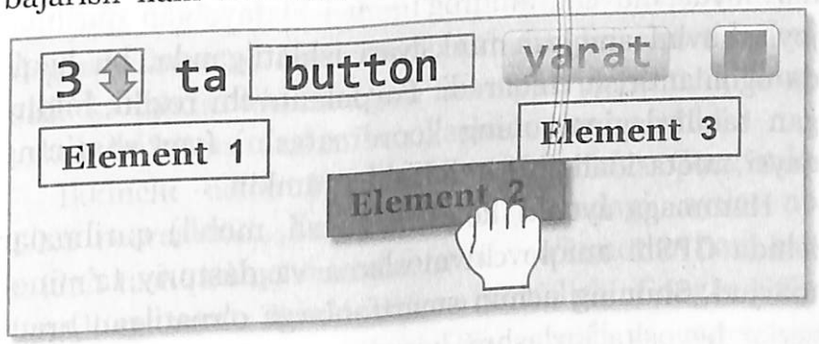
```
function yarat(btn) { // funksiya yaratilmoqda
  var elm = btn.previousElementSibling, // oldindagisi
    // oldindagi element qiymati «teg»ga o'zlashtiriladi,
    // koordinata va surishning ilk qiymatlari aniqlanadi
```

```
  teg = elm.value, i, sur = dx = dy = 0,
  son = elm.previousElementSibling.value;
  // «son»ga ikkita oldindagi element qiymati yuklandi
for (i = 0; i < son; i++) { // «son»ta takrorlash
  elm = document.createElement(teg);
  // «teg» nomli element yaratildi
  elm.style.position = 'absolute';
  elm.style.left = (i * 100) + 'px';
  // bir-birining ustiga tushmasligi uchun chapdan surildi
  elm.style.border = '1px solid gray'; // chegarasi
  elm.style.cursor = 'grab'; // sichqoncha ko'rsatgichi
  elm.innerText = 'Element ' + (i+1); // ichki matn
  document.body.append(elm); // element yasaldi
  elm.onmousedown = function(e) { // sichqoncha bosildi
    this.classList.add("bosildi"); // CSS qo'shildi
    sur = true;    // sichqoncha bosilganini eslab qolish
    dx = e.offsetX; // elementning ayni qaysi joyi ...
    dy = e.offsetY; // ... (nuqtasi) bosilganini bilish
  };
  elm.onmouseup = function(e) { // qo'yib yuborilganda
    this.classList.remove("bosildi"); // CSS o'chirish
    sur = false; // sichqoncha tugmasi qo'yib yuborildi
  };
  elm.onmouseout = function(e) { // chiqib ketganda
    this.classList.remove("bosildi");
    sur = false; // ko'rsatgich element ustida emas
  };
  elm.onmousemove = function(e) {
    if (sur) { // bosib turilgan bo'lsa, element suriladi
      // sichqoncha koordinatasidan «joy» farqi ayiriladi
      this.style.left = e.x - dx;
      this.style.top  = e.y - dy;
    }
  };
```



```
  elm.ondblclick = function(e) { // juft chertilganda
    let c; // rang darchasi qiymatini elementga berish
    c = document.querySelector('input[type="color"]');
    this.style.backgroundColor = c.value;
  };
 } // funksiya nihoyasida tugmani nofaollashtiradi
 btn.setAttribute("disabled", true);
}
```

Quyidagi rasmda ushbu kodlar bajarilgandan keyin brauzerda hosil bo'ladigan obyektlar tasvirlangan. Unda elementlar sonini HTMLda ko'rsatilgandek 2 tadan 9 tagacha o'zgartirish mumkin, «yarat» yozuvli tugma bosilgandan keyin u nofaol ko'rinishga kelgan. Ikkinchi elementning sichqoncha yordamida surilayotgani akslantirilgan, unga avval ikki marta bosib, oxirgi darchada tanlangan rang berilgan. Ushbu mashqni «button» o'rniga boshqa ixtiyoriy blokli teg nomiga almashtirib bajarish ham mumkin.



7-BO‘LIM

AMALIY MASHQLAR

7.1. Joylashuvni aniqlash

Kundalik hayotimizda juda ko‘p bora joy hamda yo‘nalishlarni topib beruvchi xaritaga asoslangan dasturiy ta’minotlardan foydalanganmiz. Hozirgi zamonaviy qurilmalar yordamida Yer sharining aynan qaysi nuqtasida joylashganimizni aniqlash muammo bo‘lmay qolgan. Zamonaviy brauzerlar ham bunday imkoniyatga ega. Ammo odamning ayni paytda qayerdaligi shaxsiy ma’lumot hisoblanib, uni kuzatib borish ko‘pgina qonunlar bilan cheklangan. Shuni inobatga olgan holda, brauzerdan foydalanuvchi, aniqrog‘i – u ishlatayotgan qurilma, joylashuvini aniqlash funksiyasi ishlatilganda, bu haqida ogohlantirish chiqaradi. Foydalanuvchi rozilik bildirgan taqdirdagina, uning koordinatasini (yer sharining qaysi nuqtasidaligini) aniqlashi mumkin.

Hammaga ayonki, ko‘chma (uyali, mobil) qurilmalar ichida GPSni aniqlovchi moslama va dasturiy ta’minot mavjud. Shuning uchun smartfonlarga o‘rnatilgan brauzerlar bevosita joylashuv koordinatasini olish imkoniyatiga ega. Ammo ko‘chmas kompyuterlarga o‘rnatilgan brauzer qay tariqa koordinatani olishi qiziqarli masala. Axir ularda GPS uzatgich mavjud emas. Javob esa oddiy — aniqlamaydi.

Chrome brauzerining Google imkoniyatlari yordamida oddiy kompyuter uchun koordinatani qanday aniqlash usulini bayon etamiz. Qolgan qurilmalardagi dasturiy ta’minotlar ham undan foydalanib yoki shunday qonuniyatlar bilan ishlovchi uslublarni qo‘llab joylanuvni aniqlaydi.

Birinchi uslub WiFi orqali. Android tizimi o‘rnatilgan qurilma biror «WiFi»ga ulansa va ayni paytda GPS koordinatasini aniqlash imkoni mavjud bo‘lsa, bu haqidagi ma’lumotlarni o‘zida saqlash bilan birga Google serveriga ham yuboradi. Shuning uchun ham «WiFi» yoniq paytda smartfonlar joylashuvni tez aniqlaydi. Shunday qilib, Google deyarli barcha internetga ulangan «WiFi»larning takrorlanmas raqami bilan u joylashgan manzilini va hattoki unga qaysi qurilmalardan ulanganini o‘z bazasida saqlaydi. Keyin ma’lum «WiFi»-ga ulangan boshqa qurilmalar (noutbuk, oddiy kompyuter yoki boshqalar) uchun ham aynan o‘sha manzilda degan axborotni beraveradi. (Ammo bu shaxsiy daxlsizlikka tahdid hisoblangani uchun ko‘pgina tanqidlarga sabab bo‘ldi va undan voz kechish tavsiya etildi.)

Ikkinchi uslub IP manzil yordamida. Googlening DNS serverlarida har bir IP bilan undagi saytlar nomidan tashqari qo‘shimcha tavsif ma’lumotlar ham saqlanadi masalan: koordinata va yo‘nalish. Google o‘ziga kelayotgan so‘rovning qaysi IPlar orqali va qanchadan vaqt sarflab kelayotganiga nisbatan taxminiy koordinatani hisoblab o‘z serverlariga yuboradi.

Koordinatani bilish uchun albatta internet bo‘lishi shart, aniqroq ko‘rsatilishi uchun esa brauzerlarning

oxirgi talqinlaridan foydalanish tavsiya etiladi. Joylashuvni https://www.google.com/maps/ manziliga kirib xaritadan o'ng taraf pastroqdagi ⊙ tasvirli tugmani bosib, aniqlashga ruxsat berish tasdiqlangandan so'ng, ko'k doira shaklda ko'rish mumkin. O'zida GPS moslama bo'lmagan qurilmalarda ko'rsatilgan joylashuvning to'g'riligiga ishonib bo'lmaydi. Google kompaniyasida dunyoning yetuk olimlari faoliyat olib boradi, koordinatani aniqlash uchun yuqoridagi ikki uslubdan boshqa ko'pgina usullari ham qo'llaniladi, lekin aniq joyni ko'rsatish uchun JavaScriptning o'z ichki imkoniyatlari mavjud emas. Brauzer esa unga imkon qadar yaqinroq koordinatani berishga harakat qiladi.

Quyida bitta tugmani bosganda Google xaritasi yordamida joylashuvni ko'rsatadigan veb-sahifaga misol keltirilgan:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title> Mukammal dasturlash. 3-mashq. </title>
    <script>
      function qayerdaman() {
        if (!navigator.geolocation) {
          alert("Aniqlashning imkoni mavjud emas.");
          return false;
        }
        function topdi(joy) {
          location.href = "https://google.com/"
             + "maps/?q=" + joy.coords.latitude
             + "," + joy.coords.longitude;
        }
```



```
        function xato() {
          alert("Joylashuvni aniqlab bo'lmadi.");
        }
        navigator.geolocation
          .getCurrentPosition(topdi, xato);
      }
    </script>
  </head>
  <body>
    <input type="button" value="Joylashuvni ko'rsat"
      onclick="qayerdaman();">
  </body>
</html>
```

7.2. Sahifalararo ma'lumot almashish

Odatda, veb-sahifa serverga ma'lumot yuboradi va undan qaytgan javobni olib, belgilangan tartibda veb-orazda namoyon etadi. Ushbu kitob faqat JavaScript haqida bo'lganligi bois, unda server tarafdagi dasturlash bayon etilmadi. E'tiborlisi, bitta veb-sahifadan ikkinchisiga ma'lumot uzatish imkoniyati JavaScriptning o'zida ham mavjud.

Faqat ikkala sahifa orasida bog'lanish bo'lishi kerak. Avvaldan ochiq bo'lgan istalgan sahifalar orasida bog'lanish yaratib bo'lmaydi. Misol:

```
<!DOCTYPE html>
<html>
  <head>
```

```
    <meta charset="utf-8">
    <title> Mukammal dasturlash. 4.1-mashq. </title>
    <script>
      var sahifa;
      function och() {
        sahifa = window.open('qabul.html', 'noma');
      }
      function yubor() {
        let m = document.getElementById('matn').value;
        sahifa.postMessage(m, '*');
        // «*» o'rniga qabul qiluvchi faylni ko'rsatsa bo'ladi
      }
    </script>
  </head>
  <body>
    <textarea id="matn">Yuboriluvchi matn</textarea>
    <input type="button" value="  " onclick="och()">
    <input type="button" value="  " onclick="yubor()">
  </body>
</html>
```

Ikkita mustaqil sahifa orasida aloqani ta'minlash uchun asosiy sahifada «open» funksiyasi yordamida boshqasi ochiladi va uni o'zlashtirgan o'zgaruvchi bog'lanishni ta'minlab beradi. Yuqorida serverga ma'lumot yuborish uchun qaydnoma, port va sayt nomi bir xil bo'lishi lozimligi bir necha bor ta'kidlangan edi [5.1.5.] [5.3.2.]. Ammo ushbu «postMessage» orqali bunday to'siqni aylanib o'tish mumkin. Chunki «open» funksiyasida ixtiyoriy manzil ko'rsatsa bo'ladi.

Quyida qabul qiluvchi sahifada qanday kod yozilishi ko'rsatilgan:


```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Mukammal dasturlash. 4.2-mashq.</title>
    <script>
      window.addEventListener("message", keldi);

      function keldi(hodisa) {
        var m = document.getElementsByTagName('h1')[0];
        m.innerHTML = hodisa.data;
      }
    </script>
  </head>
  <body>  <h1>Salom!</h1>  </body>
</html>
```

Yuqoridagi fayl brauzer yordamida ochiladi. Undagi birinchi ( ) tugma bosilganda ikkinchi fayl akslantiriladi va «Salom!» yozuvi paydo bo'ladi. Birinchi sahifadagi darchaga ixtiyoriy matn yozib, « » nishonchali tugma bosilsa, ushbu matn ikkinchi sahifada paydo bo'ladi. Chunki unda «message» hodisasi qo'shilgan va kelgan ma'lumotni ilk <h1> tegiga chiqaruvchi «**keldi**» funksiyasi biriktirilgan.

7.3. Ijrochilar bilan ishlash

Ijrochi — (player) bu mavjud fayl asosida lavha namoyish qiluvchi hamda tovush chiqarishni ta'min-

lovchi, masalan, videolarni ijro etadigan yoki ovozlarni eshittiradigan vosita hisoblanadi.

Ijroni ta'minlash uchun zarur ma'lumotlar birinchi kitobning beshinchi bo'limida berilgan bo'lib, unda `<audio>` yoki `<video>` elementlaridan qanday yo'sinda foydalanish lozimligi va ularning o'ziga xos alomatlari batafsil yoritilgan. Endi shu o'rinda ijroni boshqarish uchun lozim bo'ladigan uslub va xususiyatlarini keltirib o'tamiz:

- **`buffered`** — ijrochi xos xotiraga faylning qancha bo'lagini yuklab olganini ko'rsatadi. Ushbu maxsus obyekt quyidagi elementlarga ega:
  - ◊ **length** — bo'laklar soni
  - ◊ **start(n)** — **n**-bo'lak boshlanishi (soniyada)
  - ◊ **end(n)** — **n**-bo'lak nihoyasi (soniyada)
- **`controls`** — boshqaruv tugmalarining ko'rinish-ko'rinmasligini aniqlaydi. Mantiqiy turdagi qiymat berilsa, unga mos ravishda tugmalar paydo bo'ladi yoki g'oyib bo'ladi. Qiymat berilmaganda joriy holati bo'yicha mantiqiy tur qaytaradi.
- **`currentSrc`** — ijro etiladigan fayl ishorati.
- **`currentTime`** — ijro etilayotgan joriy vaqtini soniyada qaytaradi yoki o'rnatadi (controls singari).
- **`defaultMuted`** — yaratilgan paytda ovoz o'chirilgan holda bo'lganmi yoki yo'qligini aniqlaydi.
- **`duration`** — ijro umumiy necha soniyadan iboratligi haqida ma'lumot qaytaradi.

- **`defaultPlaybackRate`** — qanday tezlikda ijro etilishini aniqlaydi. –1; –0.5; 0.5; 1; 2 qiymatlaridan birini qaytaradi. Manfiy son orqa tarafga (teskari yo'nalishda) ijro etilayotganini bildiradi.
- **`ended`** — ijro nihoyasiga yetgan yoki yetmaganligini bildiruvchi mantiqiy tur qaytaradi.
- **`loop`** — ijro yakuniga yetgach yana boshidan boshlash shartini o'rnatadi yoki qaytaradi (controls singari).
- **`muted`** — ovoz o'chirilganligini aniqlaydi (o'rnatadi yoki qaytaradi).
- **`networkState`** — tarmoq holatini aniqlaydi. Quyidagi qiymatlardan birini qaytaradi:
  - **0** — fayl yuklanmagan;
  - **1** — faylning kerakli qismi yuklanib bo'lgan, ayni paytda tarmoqdan foydalanilmayapti;
  - **2** — fayl yuklanmoqda;
  - **3** — fayl topilmadi;
- **`paused`** — ijro to'xtatib turilganligi haqida mantiqiy tur qaytaradi.
- **`playbackRate`** — ijro etish tezligini qaytaradi yoki o'rnatadi. Qiymati turli brauzerlarda turlicha oraliqda bo'lishi mumkin.
- **`played`** — ijro etib bo'lingan bo'laklarni aniqlaydi (buffered singari).
- **`readyState`** — ijroning ayni holatini aniqlaydi. Quyidagi qiymatlardan birini qaytaradi:

0 — noma'lum holatda, tayyorligi aniq emas;

1 — tavsif xabarlari o'qildi;

2 — joriy kadr (millisoniya) yuklandi, lekin keyingisi uchun ma'lumot yetarli emas;

3 — joriy va keyingi kadr yuklandi va tayyor;

4 — ijro uchun ma'lumot yetarli;

- **seekable** — o'tish uchun vaqtincha yuklangan bo'laklarni o'zida mujassamlashtirgan obyekt.

- **src** — yangi media fayl ishoratini o'rnatadi yoki eskisini qaytaradi.

- **textTracks** — <track> tegi yordamida aniqlangan matnlarni obyekt sifatida qaytaradi (Birinchi kitobning 5.4-mavzusiga qarang — *J.A.*).

- **volume** — ovoz balandligini o'rnatadi yoki qaytaradi. Qiymati noldan birgacha bo'lgan haqiqiy son.

- **loop** — ijro yakuniga yetgach yana boshidan boshlash shartini o'rnatadi yoki qaytaradi.

Ijro etish vaqt va turli jarayonlar bilan bog'liq. Foydalanuvchi ijro tarzini o'zgartirish, masalan, to'xtatish va davom ettirish singari ko'pgina amallar bajarishni istashi tabiiy. Buning uchun yuqorida bayon etilgan xususiyatlardan tashqari «hodisalar» mavzusida [5.1.3.] bayon etilgan vositalar ham keng qo'llaniladi.

Hodisalarni qo'llashda ikki xil usuldan foydalanish mumkin: «addEventListener» uslubi yordamida yoki nomlanishiga «on» old qo'shimchasini qo'shib, xususiyat sifatida funksiyani o'zlashtirish yo'li orqali (Har ikkisi ham quyidagi misolda keltirilgan).

Tovushli faylni ijro etish uchun faqat muhim vositalarni qo'llagan holda muxtasar misol keltiriladi. Unda ba'zi muhim jihatlarga e'tibor qaratish lozim (Lavha ham kerakli o'zgartirishlar bilan shu asnoda qo'llanilishi mumkin).

Sahifa ilk yuklanganda tovush fayliga hali murojaat bo'lmaganligi sababli, uning hatto mavjudligi tekshirilmaydi. Shuning uchun JavaScript yordamida, avvalo, faylning ko'rsatilgan manzilda borligini va uning ijroga yaroqliligini aniqlash lozim bo'ladi.

Ikkinchi navbatda, bosiladigan tugma ijro holatiga mosligi ta'minlanishi lozim. Hali fayl yuklanib tekshiruvdan o'tmagan holati (), yuklanayotgan vaziyati (), ijroga tayyorligi (), to'xtatib turilganligi (), ijro davom etayotganligi () hamda yaroqsizligi () bir-biridan yaqqol ajragan holda ayni bitta tugma nishonchalari sifatida ko'rsatiladi.

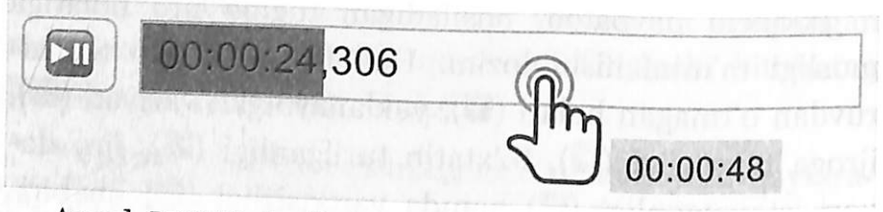
Tugma yonidagi (div) maydonchada ayni paytda qaysi joyi ijro etilayotganini ko'rsatish bilan kerakli o'ringa o'tkazish imkoniyati ham yaratiladi.

Yana bir muhim jihat, sichqoncha bir marta bosilganda boshqa amal, ikki marta bosilganda boshqa amal bajarilishi uchun uddaburonlik bilan yo'l tutishga to'g'ri keladi. Aslida juft bosilganda u ikki marta alohida bosilgandek hisoblanadi. Faqat oraliq vaqt o'ta qisqa bo'lsa, juft bosilgan deb inobatga olish lozim. Tugma ustida sichqoncha juft bosilsa, butkul to'xtatish

vazifasini bajarishi uchun ijrochining vaziyatini o'zgartirish bilan birga, bir zum «yakun» (⏹) nishonchasini ko'rsatib, so'ng boshidan boshlashni bildiruvchiga (▶) almashtiramiz.

Albatta, ijrochi ayni paytdagi joyni soniyalarda qaytaradi. Lekin foydalanuvchiga tushunarli bo'lishi uchun uni vaqt sifatida ko'rsatish kerak. Buni amalga oshiradigan qo'shimcha funksiya yozamiz. U kerak bo'lsa, millisoniyani ham qaytaradi.

Yuqoridagi talablar bo'yicha hosil qilingan shamoilda 24-soniyada to'xtatib qo'yilgan tovush ijrochisini 48-soniyaga o'tkazish lahzasi quyidagicha namoyon bo'ladi:



Avval HTML kodini keltiramiz, u tugma va maydon uchun CSSni ham o'z ichiga oladi:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title> Mukammal dasturlash. 5-mashq. </title>
    <style type="text/css">
      #tugma { // maxsus nishonchali tugma uchun
        width: 32px;
        float: left;
        margin-right: 6px;
      }
```



```
      #maydon {  // ijro foizini aniqlovchi to'rtburchak
        width: 1px;  // dastlabki foiz sifatida
        height: 17px;
        float: left;
        background: gray;
        font-size: 12px;
        line-height: 18px;// ichki yozuv balandligi
      }
      #maydon:before {  // maydon chegarasi
        content: '';
        position: absolute;
        border: 1px solid lightgray;
        height: 16px;
        width: 200px;  // maydon eni o'lchami
        cursor: pointer;
      }
    </style>
    <script src="audio.js"></script>
  </head>
  <body>
    <div style="width: 240px;">
      <button id="tugma">▶</button> // ilk holati
      <div id="maydon"></div>
    </div>
    <audio id="ijrochi">
      <source src='https://yurt.uz/vatan.mp3' />
    </audio>
  </body>
</html>
```

Ushbu HTML sahifada biriktirilgan «audio.js» nomli JavaScript faylning tarkibi quyidagicha:

```
var ijroEt = {    // oddiy obyekti sifatida yaratildi
  div: 'maydon', // HTMLdagi «maydon» (div)ning idsi
  eni: 200,      // «maydon»ning eni (ijro foizi uchun)
  yuklash: function() {
    ijroEt.tugma.innerHTML = '🔄';
    ijroEt.holati = setInterval(function() {
      if (ijroEt.ijrochi.readyState != 4) {
        if (ijroEt.ijrochi.readyState == 0) {
          ijroEt.tugma.innerHTML = '🔄';
          alert('Fayl yuklanishda xatolik!');
          clearInterval(ijroEt.holati);
          return false;
        }
      } else {     // fayl ijro uchun tayyor
        ijroEt.tugma.innerHTML = '▶';
        clearInterval(ijroEt.holati);
      }
    },
    40);
  },
  vaqti: function(timeInSeconds, millisoniya) {
    let nolla = function(num, size) {
      return ('000' + num).slice(size * -1);
    }, // oldidan nollar bilan to'ldiruvchi funksiya
      time    = parseFloat(timeInSeconds).toFixed(3),
      hours   = Math.floor(time / 60 / 60),
      minutes = Math.floor(time / 60) % 60,
      seconds = Math.floor(time - minutes * 60),
      mSeconds = time.slice(-3);
    time  = nolla(hours, 2) + ':' + nolla(minutes, 2);
    time += ':' + nolla(seconds, 2);
    if (millisoniya) time += ',' + nolla(mSeconds, 3);
    return time; //soniyani vaqt ko'rinishida qaytaradi
  },
```



```
  foizi: function() {
    ijroEt.holati = setInterval(function() {
      let m = document.getElementById(ijroEt.div),
      hozir = ijroEt.ijrochi.currentTime,
      hamma = ijroEt.ijrochi.duration,
      f = hozir / hamma * ijroEt.eni;
      if (hozir >= hamma) { //yakunlangan bo'lsa
        f = ijroEt.eni + 1;
        ijroEt.tugma.innerHTML = '▶';
      }
      m.style.width = parseInt(f) + 'px';
      if (!ijroEt.ijrochi.paused) // vaqtni ko'rsatadi
        m.innerHTML = ijroEt.vaqti(hozir, true);
    }, 500); // har yarim soniyada foizi yangilanib turadi
  },

  bosdi: function(ijrochi, maydon) { //bitta bosganda
    if (ijroEt.ijrochi.paused) { //ijro etmoqda
      ijroEt.ijrochi.play();
      ijroEt.tugma.innerHTML = '⏸';
      ijroEt.foizi();
    } else {  //ijro to'xtatib turildi
      ijroEt.ijrochi.pause();
      ijroEt.tugma.innerHTML = '⏯';
      clearInterval(ijroEt.holati);
    }
  }
};

window.onload = function() {  //sahifa yuklanganda
  ijroEt.ijrochi = document.getElementById("ijrochi");
  ijroEt.tugma = document.getElementById("tugma");
  ijroEt.yuklash(); // sahifa yuklanishi bilan tekshiradi
```

```
    ijroEt.tugma.addEventListener("click", function() {
    // aynan 1 marta bosgan bo'lsa, 0.3 soniyadan so'ng ishlaydi
      clearTimeout(ijroEt.bos1);
      ijroEt.bos1 = setTimeout(ijroEt.bosdi, 300);
    });

ijroEt.tugma.addEventListener("dblclick",function(){
    clearTimeout(ijroEt.bos1);
// juft chertilsa 0.7 soniya ⏹ ko'rinib, so'ng ⏺ ga o'zgaradi
    ijroEt.tugma.innerHTML = '⏹';
    setTimeout("ijroEt.tugma.innerHTML = '⏺'", 700);
    ijroEt.ijrochi.pause();
    ijroEt.ijrochi.currentTime = 0;
  });
  ijroEt.ijrochi.onended = function() {
    let m = document.getElementById(ijroEt.div);
    m.innerHTML = ' Ijro yakunlandi.';
  };
  ijroEt.maydon = document.getElementById(ijroEt.div);
  ijroEt.maydon.addEventListener("click",function(e){
    let p = e.x - this.offsetLeft;
    if (p < 0) p = 0;
    if (p > ijroEt.eni) p = ijroEt.eni;
    p = ijroEt.ijrochi.duration * p / ijroEt.eni;
    // bosilgan joyga mutanosib ravishda ijroni o'tkazish
    ijroEt.ijrochi.currentTime = p;
  });
  // sichqoncha ko'rsatkichi yoniga mos vaqtni ko'rsatish
  ijroEt.maydon.onmouseover = function(e) {
    let p = e.x - this.offsetLeft,
      v = ijroEt.ijrochi.duration * p / ijroEt.eni;
    this.setAttribute("title", ijroEt.vaqti(v));
  };
};
```


7.4. Matnni o'qitish

Hozirda barcha ommabop brauzerlarda matnni o'qib berish imkoniyati mavjud. Afsuski, hozircha o'zbek tilini o'qitish imkoniyati kiritilmagan. Lekin jahon tillaridan bizga ko'proq tanish bo'lgan rus va ingliz tillarida yozilgan matnlarni o'qitish vositalaridan bemalol foydalanishimiz mumkin.

Amaliy mashq yordamida o'qitish qanday amalga oshirilishi bilan tanishib boramiz. Avvalo, ijrochi faoliyatini boshqaruvchi tugma hosil qilamiz. Unda ijroni boshlash, to'xtatib turish va yakunlashni bildiruvchi belgilar mos ravishda almashib turadi. Keyin o'qitish uchun tizimda mavjud ovozlarni tanlash imkoniyatini beruvchi element yasaymiz. So'ngra o'qiladigan matn kiritish uchun darcha qo'yamiz. Ushbu imkoniyatlarni quyidagi HTML kod orqali interfeysini yaratamiz:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Mukammal dasturlash. 6-mashq.</title>
    <script src="md.js"></script>
  </head>
  <body>
    <input  id="ijro" type="button">
    <select id="ovozlar" style="display: none;">
      <!-- Ro'yxat yuklanguncha ko'rinmay turadi -->
    </select>
    <br>
```

```
    <textarea>
      Это не обычная книга, это бестселлер.
    </textarea>
  </body>
<html>
```

Shuni inobatga olish kerakki, «speechSynthesis» sahifa yuklangandan keyin biroz o'tib ovozlar ro'yxatida kerakli qiymatlarni beradi. Shuning uchun ilk «ijro» tugmasini bosganimizdan so'ng ro'yxat mavjud bo'lmasa, hosil qilish funksiyasi alohida yozilgan. Quyida «md.js» faylidagi JavaScript kod qanday bayon etilganligi kerakli izohlar bilan keltiriladi:

```
class Mutolaa {
  constructor(tugmaID, matnID, tanlaID) {
    this.tugma  = document.getElementById(tugmaID);
    this.matni  = document.getElementById(matnID);
    this.tanla  = document.getElementById(tanlaID);
    this.bosdi0 = '▶';
    this.bosdi1 = '⏸';
    this.bosdi2 = '⏹';
    this.tovush = window.speechSynthesis;
    this.tillar = [];
    this.tugma.value = this.bosdi0;
    let f = "Mutolaa.tinglash()";
    this.tugma.setAttribute('onclick', f);
    self = this;
  }
  // HTMLdagi tayinlov darchasida mavjud ovozlardan ...
  // ... faqat rus tiliga oidlarini (tizimda odatiy bo'lgan ...
  // ... ovoz tanlangan holda) yuklab beruvchi funksiya:
 ovozTanlash() {
    var n, r, sel = self.tanla,
      ovoz = window.speechSynthesis.getVoices();
```



```
    self.tillar = ovoz;
    if (sel.style.display == 'none') {
      sel.style.display = 'inline';
      for (n = 0; n < ovoz.length; n++)
      if (ovoz[n].lang.indexOf('ru') >= 0) {
        var l = document.createElement('option');
        l.textContent = ovoz[n].name +
          ' (' + ovoz[n].lang + ')';
        if (ovoz[n].default) {
          l.setAttribute('selected', 'selected');
          r = ovoz[n].name;
        }
        l.setAttribute('data-lang', ovoz[n].lang);
        l.setAttribute('data-name', ovoz[n].name);
        sel.appendChild(l);
      }
    }
    return r;
  }
  static tinglash() {
    var i, o, t, matn, text, sel = self.tanla;
    if (sel && sel.selectedOptions[0]) {
      sel = sel.selectedOptions[0];
      sel = sel.getAttribute('data-name');
    } else sel = self.ovozTanlash();
    if (self.tugma.value == self.bosdi0) {          //▶
      self.tovush.cancel();
      text = self.matni.value;
      matn = new SpeechSynthesisUtterance(text);
      matn.onend = function() {
        // O'qishdan to'xtaganda tugma ▶ ga o'zgaradi
        self.tugma.value = self.bosdi0;
      };
      // Tanlangan ovozda ijroni ta'minlash
      for (i = 0; i < self.tillar.length ; i++)
```

```
        if (self.tillar[i].name === sel)
          matn.voice = self.tillar[i];
      self.tovush.speak(matn);
      self.tugma.value = self.bosdi1;
    } else if (self.tugma.value == self.bosdi1) { //
      self.tovush.pause(matn);
      self.tugma.value = self.bosdi2;
    } else {                                        //
      self.tovush.resume(matn);
      self.tugma.value = self.bosdi1;
    }
  }
}
 //Sahifa yuklanishi bilan «Mutolaa» sinfi ishga tushadi
document.addEventListener('DOMContentLoaded',
  function() { new Mutolaa("ijro", "matn", "ovozlar") }
);
 //Sahifa yopilganda yoki qayta yuklanganda ...
 //... o'qish davom etavermasligi uchun uni bekor qiladi.
window.onbeforeunload = function (e) {
  var s = window.speechSynthesis;
  if (s.paused || s.speaking) s.cancel();
};
```

Brauzerda quyidagicha interfeys namoyon bo'ladi:

Milena (ru-RU)

Это не обычная книга, это бестселлер.

7.5. Qat'iy tartibdan foydalanish

Taraqqiyot o'z-o'zidan ba'zi mavjud unsurlar iste'-moldan chiqishiga yoki o'zgartirilishiga sabab bo'ladi. Internet ibtidosida brauzer faoliyatini takomillashtirish uchun muhim vosita sifatida yaratilgan, umumjahon tarmog'ining taraqqiyotida asosiy sababchilardan biri bo'lgan, to hozirgi kungacha ommaviy qo'llanilib, mu-kammalashtirilib kelinayotgan ilk dasturlash tili – Java-Scriptning asoslari ham bundan mustasno emas.

7.5.1. Qat'iy tartib qo'llanilishi

Qat'iy tartib «ECMAScript5» talqinidan boshlab joriy etilgan. Takomillashtirish mobaynida ba'zi funk-siyalar o'rniga muqobili qo'llanish tavsiya etilsa yoki unga murojaat shakli qisman o'zgartirilgan bo'lsa, av-val tuzilgan dasturlar o'z vazifasini bajarishdan to'xtab qolmasligi uchun, eski kodlarni ham yangi talqinda qo'llanilaverilishi ta'kidlab o'tilgan edi. Ammo yangi texnologiya bejiz kiritilmaydi, kelgusida faqat undagi qonuniyatlardan foydalanish maqsadga muvofiq hisob-lanadi. Har bir kodni yangicha qonuniyatlarga mos kelishini tekshirib borish uchun maydonning boshida «`"use strict";`» deb yozish kerak. Biriktiriladigan JavaScript kodli fayl to'liq yangi qonuniyatlar asosida yozilgani aniq bo'lsa, faylning boshida yoziladi. Faqat-gina bitta funksiyaning o'zi yangicha tartibda yozilishi lozim bo'lsa, uning ilk satriga ushbu «`"use strict";`»

jumlasi qo'shiladi. Bu qat'iy tartib yoqilganligini anglatadi va unda eskicha uslubda ishlab turgan ayrim kodlarda xatoliklar ko'rsatilishi mumkin. Masalan: kodlama to'g'ri qo'llanilmaganda, xavfsiz hisoblanmagan (global obyektga murojaat vaqtida) amal bajarilganda yoki buzg'unchi (bemisl cheksiz takrorlayveradigan) funksiyaga murojaatlar bo'lganda.

Eski yozilgan kodlardan odatdagidek foydalanib, yangi yozilayotgan fayllarda qat'iy tartib o'rnatilishi tavsiya etiladi. Zero, mavjud qonuniyatlarga aniq rioya etish — dasturiy mahsulotni takomillashtirishda yuzaga kelishi mumkin bo'lgan muammolarning oldini oladi. Boshqa dasturlash tillarini o'zlashtirishga ehtiyoj bo'lsa, unga qiyoslab o'rganishni soddalashtiradi, chunki JavaScriptdan boshqalari bu qadar «kechirimli» emas.

Qat'iy tartibning odatdagidan farqini yaqqol ko'rish uchun, misol tariqasida aynan bir xil yozilgan ikki funksiya quyida keltiriladi:

```
function odatiy(omil, omil) {
  gm = 9.8;
  return gm * omil;  // ikkinchi omil inobatga olinadi
}

function qattiy (omil, omil) { // yozilishdagi xato
  "use strict";
  mg = 9.8;                    // bajarishdagi xato
  return mg * omil;
}
```

Ikkinchi funksiya e'lon qilingan zahotiyoq ikkita bir xil nomli omil berish mumkin emasligi haqida ogohlantiruvchi xatolik sodir bo'ladi. Bu esa «yozilishdagi»

(sintaksis) xato hisoblanadi. Ikkinchi o'zgaruvchi nomini almashtirish orqali ushbu kamchilik bartaraf etilganda funksiya muvaffaqiyatli yaratiladi, lekin ishga tushirilgan paytda «bajarishdagi» xato sodir bo'ladi. Sababi, qiymat berishdan avval o'zgaruvchi e'lon qilinmagan.

Yuqoridagi qat'iy tartibli funksiya to'g'ri bo'lishi uchun quyidagi ikki o'zgartirishni kiritish lozim:

```
function qattiy(omil, omil2) { Omillar nomi har xil
  "use strict";                O'zgaruvchi e'lon qilindi
  let mg = 9.8;
  return mg * omil;
}                              29.400000000000002
odatiy(2, 3)                   19.6
qattiy(2, 3)
```

7.5.2. Qat'iy tartib talablari

Qat'iy tartib qo'llash tavsiya etilar ekan, uning odatiy tartibdan farqlarini aniq sanab ko'rsatib o'tish maqsadga muvofiqdir. Quyida e'tibor qilish lozim tafovutlar bayon etiladi:

- **O'zgaruvchini e'lon qilish shart**. Har bir o'zgaruvchini, shu jumladan obyektni, qo'llashdan avval «var», «let» yoki «const» yordamida e'lon qilish talab etiladi. JavaScript aslida o'zgaruvchiga qiymat berib, uni qo'llab, so'ngra keyingi qatorlarda e'lon qilishga ham imkon beradi. Bunday qo'llanishni «yuksalish» (**hoisting**) deb yuritiladi. Qat'iy tartibda bunga yo'l qo'yilmaydi.

▪ **O‘zgaruvchini butkul o‘chirish taqiqlanadi.** O‘zgaruvchining o‘zini, xususan, obyekt va funksiyani nomi orqali butunlay yo‘qotish mumkin emas. Obyektning ichki elementlari uchun, agar ular maxsus xususiyat sifatida aniqlanmagan bo‘lsa, bu usulni qo‘llashga ruxsat etiladi:

```
"use strict";
var j = [1, 12, 88];
delete j;                 Xato!
delete j[1];              ruxsat etiladi
console.log(j);           [1, empty, 88]
delete j.length;          Xato!
```

▪ **Funksiya omillarining nomlari takrorlanmasligi kerak.** Bir xil nomda bir necha o‘zgaruvchi berish mantiqsizlik sanaladi. Yuqorida misol keltirilgan.

▪ **Sakkizlik sanoq tizimi bevosita qo‘llanmaydi.** Sakkizlik sanoq tizimida yozilgan sondan yoki maxsus belgidan to‘g‘ridan-to‘g‘ri foydalanish taqiqlanadi:

```
x = "\072";                :
y = 011;                   9
```

▪ **Obyektning faqat o‘qishga mo‘ljallangan xususiyati o‘zgartirilmaydi.** Obyektning elementiga yozish imkoniyati cheklangan bo‘lsa yoki «get» funksiya sifatida aniqlanganda, unga qiymat berilganda «xato» qaytaradi. Odatiy holatda qiymatni o‘zlashtirmaydi va xatolik yuz berdi deb kod bajarishdan to‘xtamaydi:


```
var o = { a: 88 };
Object.defineProperty(o, "a", {writable: false});
o.a = 1110;                                          xato
var f = { get x() { return 10; } };
f.x = 20;                                            xato
```

▪ **Maxsus nomlanishlarni o‘zgaruvchi sifatida ishlatib bo‘lmaydi.** Hozircha bu «let», «arguments», «eval», «implements», «interface», «static», «package», «public», «private», «protected», «yield», «NaN» kabi keng qo‘llaniluvchi ommabop so‘zlar uchungina tatbiq qilingan. Kelajakda dasturlovchi bexosdan tizimda mavjud nomlanishlarning o‘ziga xos vazifasini o‘zgartirib yubormasligi uchun boshqa maxsus kalit so‘zlar ham qo‘llanilishiga umid qilamiz. Misol:

```
var eval = 'Nimadir';                                xato
var arguments = 'qiymat';                            xato
var parseFloat = 99;                                 99
var parseInt = 10.9;                                 10.9
```

▪ **Biriktirilgan holda «with»dan foydalanish taqiqlanadi.** Ichki xossa va tamoyillarini qo‘llashda tashqi obyektning nomini qayta-qayta yozavermaslik uchun «with»dan foydalanish imkoni cheklangan:

```
with (Math) { x = cos(3.1415); y = sin(1); };        xato
```

▪ **Funksiyaning ichki qismida «this» qo'llanilsa, «undefined» qaytaradi.** Odatiy holda, aniq obyekt berilmaganda, funksiya ichidagi «this» kalit so'zi global «window»ga ko'rsatgich hisoblanadi. Misol:

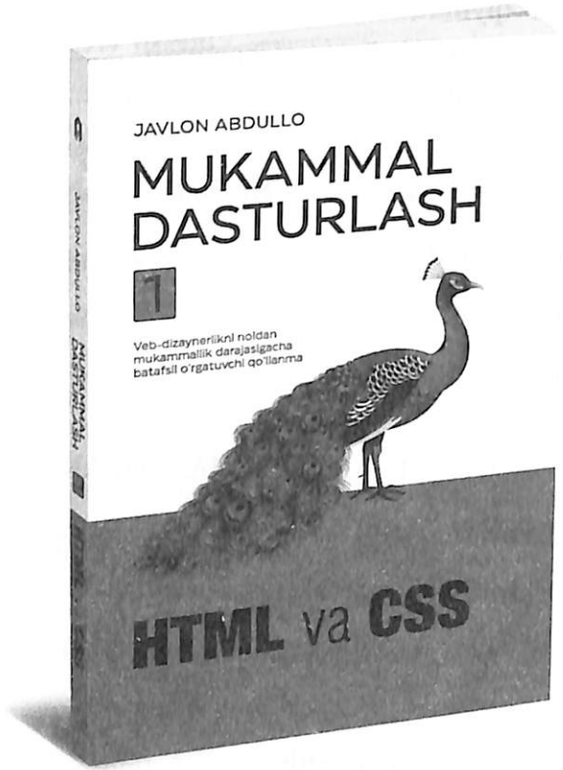
```
"use strict";
function f() { console.log(this); }      undefined
f();
```


Maxsus zaxiralangan kalit so'zlar

Foydalanilgan adabiyotlar

1. «**Mukammal dasturlash. 1-kitob: HTML va CSS**», Javlon Abdullo. «*Akademanshr*» nashriyoti, 2021.
2. «**Axborot-kommunikatsiya texnologiyalari izohli lug'ati**» Amirov D.M. Vb 2-nashr. *Toshkent,* 2010.
3. «**Learn JavaScript and Ajax with W3Schools**», Hege Refsnes. Vb, «*Wiley*», 2010.
4. «**Professional JavaScript for Web Developers**», Nicholas Zakas. «Wrox», 2005.
5. «**Creating Applications with Mozilla**», David Boswell. Vb, «*O'Reilly Media*», 2002.
6. «**You Don't Know JS: Scope & Closures**», Kyle Simpson. «*O'Reilly Media*», 2014.
7. «**Eloquent JavaScript, A Modern Introduction to Programming**», Marijn Haverbeke. 3-nashr, «*No Starch Press*», 2018.
8. «**Speaking JavaScript: An In-Depth Guide for Programmers**», Axel Rauschmayer. «*O'Reilly Media*», 2014.
9. «**JavaScript and JQuery: Interactive Front-End Web Development**», Jon Duckett. «*Wiley*», 2014.
10. «**The Principles of Object-Oriented JavaScript**», Nicholas Zakas. «*No Starch Press*», 2014.
11. «**JavaScript for impatient programmers**», Axel Rauschmayer. Fran Caye, 2019.
12. «**Exploring ES6: Upgrade to the Next Version of JavaScript**», Axel Rauschmayer. «*Leanpub*», 2015.
13. «**Programming JavaScript Applications: Robust Web Architecture with Node, HTML5 & Modern JS Libraries**», Eric Elliott. «*O'Reilly Media*», 2014.
14. «**Pro JavaScript Techniques**», John Resig v. b. 2-nashr, «*APress*», 2016.
15. «**JavaScript для FrontEnd-разработчиков. Написание. Тестирование. Развертывание**», Кириченко А.В., «*Наука и техника*», 2020.

MUNDARIJA



Ushbu kitob
«Mukammal dasturlash. 1-kitob: HTML va CSS»ning
mantiqiy davomi hisoblanadi

Ilmiy-ommabop nashr

Javlon ABDULLO

MUKAMMAL DASTURLASH

2-kitob

JavaScript

Muharrir: Abdulla SHAROPOV
Badiiy muharrir: Bahriddin BOZOROV
Texnik muharrir: Dilshod NAZAROV
Sahifalovchi: Inomjon O'SAROV
Musahhih: Mahfuza IMOMOVA

Nashriyot litsenziyasi: AI №134, 27.04.2009
Terishga berildi: 08.10.2020-y.
Bosishga ruxsat etildi: 09.02.2022-y.
Ofset qog'ozi. Qog'oz bichimi: 60x84 $^{1}/_{16}$.
Cambria garniturasi. Ofset bosma.
Hisob-nashriyot t.: 15,1. Shartli b.t.: 17,6.
Adadi: 3000 nusxa.
Buyurtma № 6

«Akademnashr» nashriyotida tayyorlandi va chop etildi.
100156, Toshkent shahri Chilonzor tumani 20^{A}-mavze 42-uy.

Tel.: (+99871) 216-87-81
e-mail: info@akademnashr.uz
web: www.akademnashr.uz

ISBN 978-9943-6501-6-9