Javascriptlə proqramlaşdırma dünyasına giriş

Şükür Hüseynov

2021

Müəllif: Şükür Hüseynov

Email: Shukur.huseynov.1996@mail.ru

Youtube: “Şükür Hüseynovla Proqramlaşdırma”

Dizayn: Vasif Səfərov

“Javascriptlə proqramlaşdırma dünyasına giriş” kitabı müəllifin üçüncü proqramlaşdırma kitabının ikinci nəşridir. Daha əvvəlki C/C++ və Php kitabları internet üzərində pulsuz şəkildə yayımlanmışdır. Youtube üzərində “Şükür Hüseynovla Proqramlaşdırma” kanalını ziyarət edərək həmin kitabların yükləmə linkini əldə edə bilərsiniz. Bu kitabların yazılma məqsədlərindən biri də Azərbaycan dilindəki resurs azlığı problemini aradan qaldırmaqdır.

Mündəricat

Müəllif haqqında

Şükür Hüseynov, 1996-cı ildə Bakı şəhərində dünyaya gəlmişdir. 2014-cü ildə, Bakı şəhərində yerləşən 55 saylı tam orta məktəbi bitirmişdir və həmin il Azərbaycan Dövlət Neft və Sənaye Universitetinin Elektronika, Telekommunikasiya və Radiotexnika mühəndisliyi ixtisasına qəbul olmuşdur. 2018-ci ildə universiteti başa vurmuşdur. Müəllif, 14-15 yaşlarından etibarən proqramlaşdırma ilə məşğuldur. İndiyə qədər Veb və stolüstü proqramlaşdırma, həvəskar səviyyədə isə mobil, oyun və qurğu proqramlaşdırması ilə məşğul olmuşdur. Daha çox stolüstü, oyun və sistem proqramlaşdırmasına maraq duymaqdadır.

Müəllif, Proqramlaşdırmadan əlavə Psixologiya və Poeziya ilə də məşğul olmaqdadır. Bu kitabdan əlavə elektron variantda yayımlanan 2 ədəd proqramlaşdırma kitabının (C/C++ və Php) və bir ədəd şeir kitabının müəllifidir.

Əlavə olaraq qeyd edək ki, “Şükür Hüseynovla Proqramlaşdırma” adlı Youtube kanalında proqramlaşdırmaya aid videolar yayımlanmaqdadır. Həmçinin, kanalı ziyarət edərək daha əvvəl yayımlanmış C/C++ və Php

kitablarının yükləmə linklərini əldə edə bilərsiniz.

Kitab haqqında

Kitabın yazılma səbəblərindən ən birincisi, Azərbaycan dilində proqramlaşdırma və texnologiya cəhətdən resurs azlığının olmasıdır. Son illərdə Azərbaycan dilində resursların müəyyən qədər artırılması bizi sevindirməkdədir. Ancaq, hələ də bu sahədə, eyni zamanda bir çox sahədə Azərbaycan dilində resurs çatışmamazlığı mövcuddur. Biz də Azərbaycan dilində müxtəlif kitablar yazaraq və videolar çəkərək öz dilimizdə olan resurs sayını artırmağa çalışmaqdayıq.

Kitab proqramlaşdırmaya tək texniki sahə olaraq yox, eyni zamanda, fəlsəfi və emosional olaraq da yanaşmaqdadır. Kitabdakı mövzular, daha əvvəl Html və Css dilləri istisna olmaqla heç bir proqramlaşdırma dili ilə məşğul olmayan şəxslərin rahatlıqla öyrənəcəyi tərzdə yazılmışdır.

Mövzuları öyrənərkən çətinlik çəkdiyiniz mövzular olarsa, mövzunu yenidən oxumağa çalışın. Eyni zamanda, kodları tam mənimsəyənə kimi kitaba baxaraq, sonra kitaba baxmadan yazmağa çalışın. Bu kitabdan başa düşə bilmədiyiniz mövzunu başqa yerlərdən oxuyun, videolara baxın, mütləq öyrənəcəksiniz. Sadəcə proqramçı olmağa **qəti qərar verin və bu**

qərara bütün ruhunuzu, ürəyinizi qoyun. Bu halda siz bir müddət öyrəndikdən və layihələr hazırladıqdan sonra güclü proqramçı ola biləcəksiniz. Sadəcə şübhə etməyin, inamlı olun, öyrənəcəyinizə tam inanın, mütləq öyrənəcəksiniz.

Proqramlaşdırmaya giriş

Əziz oxucu, 20-ci əsrin möcüzəsi dedikdə ağlıma gələn ilk şey proqramlaşdırmadır. Bəs niyə proqramlaşdırma? Təsəvvür edin, bir vaxt hesablama məqsədilə yaradılmış bir qurğu vasitəsilə bugün Gta 5, Point blank kimi oyunlar hazırlanır. Bir vaxt sadəcə hesablama məqsədilə yaradılmış qurğu vasitəsilə bu gün üz tanıma sistemləri hazırlanır, insanlar üzünə görə kompüter vasitəsilə tanınır, robotlar səsləri tərcümə edir, robotlar insanlara ehtiyac duymadan yeni şeylər hazırlaya bilirlər. Zavodlarda proseslər avtomatlaşdırılır, cibimizə yerləşdirə biləcəyimiz balaca bir qurğu vasitəsilə dünyanın başqa bir yeri ilə əlaqə saxlaya bilirik, oyunlar oynayırıq, işlərimizi görürük, bir çox əməliyyatlar aparırıq. Bunları hər birini hazırlamaq üçün qurğu daxilində proqramlaşdırmadan istifadə olunur. İndi siz mənə deyin, saya-saya bitirə bilməyəcəyim bu qədər mükəmməl şey qarşısında mən proqramlaşdırmanı möcüzə adlandırmaya bilərəmmi?

Proqramlaşdırma eyni zamanda fantaziyadır. Bir çox mövzu var və siz öz fantaziyanıza uyğun olaraq o mövzularla bir çox şey hazırlaya bilərsiniz. Məsələn, oyunlar

fantaziya deyilmi? Məsələn, adicə kiçik hesablamalar və qrafikalarla bir çox oyunlar hazırlana bilir. Siz adicə 5 ədəd ulduz simvolunu yan-yana qoyub (*****) heç bir qrafika olmadan bunu ilan təsəvvür edib onunla hərəkətli oyun hazırlaya bilərsiniz. Hər şey sizin fantaziyanıza qalmışdır. Bir neçə şəkli götürüb öz fantaziyanıza uyğun olaraq, biraz da məntiqinizi işə salaraq onlarla maraqlı oyunlar, layihələr hazırlamağınız mümkündür. Artıq hər şey sizin fantaziyanıza, məntiqinizə və qərarlılığınıza qalmışdır.

İlk vaxtlar proqramlaşdırma həddindən artıq çətin idi. İnsana yaxın ilk proqramlaşdırma dili Assembler dili idi. Assembler dilində insanın işlətdiyi sözlərdən də istifadə olunsa da Assemblerdə nəyisə proqramlaşdırmaq çox çətin bir şeydir. Assembler kodları bəzən qorxuducu ola bilir. Bugün virusların, driverlərin yazılmasında, sistem proqramlaşdırmasında Assemblerdən istifadə olunmaqdadır. Assembler hər nə qədər çətin dil olsa da, təmin etdiyi sürət, kiçik yaddaş və qurğularla əlaqəsi onu möhtəşəm bir dil etməkdədir. Hətta Nasanın 1969-cu ildə Kosmosa göndərdiyi Apollo 11-in kodları Assemblerdə yazılmışdır və bu kodlar

internetə də yerləşdirilmişdir. Aşağıdakı kiçik Assembler kodu insanı qorxutmağa kifayət edir.

```
          global    _start

          section   .text
_start:   mov       rax, 1                  ; system call for write
          mov       rdi, 1                  ; file handle 1 is stdout
          mov       rsi, message            ; address of string to output
          mov       rdx, 13                 ; number of bytes
          syscall                           ; invoke operating system to do the write
          mov       rax, 60                 ; system call for exit
          xor       rdi, rdi                ; exit code 0
          syscall                           ; invoke operating system to exit

          section   .data
message:  db        "Hello, World", 10      ; note the newline at the end
```

Gördüyümüz kimi, kodlar qorxuducudur. Ancaq, qətiyyən narahat olmayın, çünki indiki proqramlaşdırma alətləri Assemblerdən qat-qat asandır. Zaman keçdikcə C, C++, Java, C# kimi dillər yaranmışdır və onlarla proqram hazırlamaq daha da asan hala gəlmişdir.

Proqramlaşdırma dilində kodun yazılıb maşın dilinə dönüşdürülməsi prosesi kompilyasiya adlanmaqdadır. Proqram yazılıb kompilyasiya olunduqda sonra komputerdə işlədilə bilir. Həmin proqram ona uyğun olan başqa komputerlərə də daşınıb işlədilə bilir. Ancaq, bunun üçün proqram komputerə quraşdırılmalıdır.

Sonradan veb brauzerlər ortaya çıxmışdır. Deməli, bu veb brauzerlər olduqdan sonra, artıq, proqramın komputerə quraşdırılmasına ehtiyac duyulmur. Yazılmış proqram vahid bir serverdə saxlanılır və bu brauzer o serverin ünvanına görə o serverə qoşulur. Server isə brauzerə proqramın məlumatlarını göndərir. Burada proqram dedikdə, veb səhifələri nəzərdə tuturuq. Veb səhifələr kodlaşdırılaraq serverə yerləşdirilir. Brauzer oraya qoşulduqda serverdən məlumatları götürür və bizə göstərir. Serverə yerləşdirilmiş veb səhifənin görünüş hissəsi Html, Css və Javascript dilləri ilə hazırlanır. Bu dillərlə işləmək nisbətən daha rahatdır. Html, Css və Javascript kodları brauzer tərəfindən serverdən götürülür və brauzer daxilində görünüşə çevrilərək bizə göstərilir. Görünüş hissəsindən əlavə saytın proqramlaş-dırma hissəsi də olur. Bu hissə Php, Perl, Asp və başqa dillərlə yazılır. Bu hissə isə serverdə işlənilir və brauzerə yalnız işlənmiş kodlardan alınmış nəticələr göndərilir. Sonda brauzer yalnız Html, Css və Javascript kodlarını götürür və onları görünüşə çevirərək bizə göstərir. Biz brauzer daxilində həmin kodlara da baxa bilirik. Ancaq, dediyimiz kimi, Php, Perl kimi dillər serverdə işləndiyinə görə, brauzer onların

kodlarını görə bilməz, yalnız onlardan alınmış nəticəni Html daxilində görə bilər.

Javascriptə giriş

Yuxarıda da dediyimiz kimi, brauzerə serverdən Html, Css və Javascript kodları gəlməkdədir. Brauzer isə həmin kodları görünüşə çevirərək bizə göstərməkdədir. Html və Css ilə yalnız görünüşün hazırlanmasına baxmayaraq, Javascriptlə məntiqi əməliyyatlar da yerinə yetirilə bilir. Bu səbəbdən də Javascript vasitəsilə animasiyalar, oyunlar da hazırlamaq mümkün olur. Javascript bugün çox güclü bir texnologiyadır və bir çox əlavə kitabxanaları yaradılmışdır. Yaradılmış köməkçi texnologiyalar vasitəsilə Javascript kodlarını yazmaq daha asan şəklə gəlmişdir. Hətta bugün yaradılmış müxtəlif Javascript texnologiyaları vasitəsilə mobil tətbiqlər də hazırlamaq mümkündür.

Javascripti öyrənmək üçün ilk olaraq az da olsa Html və Css biliklərinə ehtiyac duyulur. Əgər, biraz da olsa Html və Css biliklәriniz varsa, o zaman Javascripti öyrənməyə başlaya bilərsiniz. Ancaq yaxşı olar ki, html və css

dillərini ən azı orta səviyyədə öyrənib daha sonra Javascriptə keçəsiniz.

Dediyimiz kimi, Javascript kodları Html və Css kodları ilə birgə brauzer tərəfindən işləndiyinə görə Javascripti öyrənmək üçün əlavə bir proqram quraşdırmağa ehtiyac yoxdur. Ehtiyacımız olan tək şey Chrome, Opera, Internet Explorer, Safari kimi bir brauzer və kodları yazmaq üçün Notepad, Notepad++ və ya başqa bir redaktordur. Javascript kodlarını sadəcə olaraq Html faylının içində Html kodlarının arasına yaza bilərik. Aşağıdakı Html və arasındakı Javascript kodunu yazaq və "index.html" olaraq yadda saxlayaq, sonra isə brauzerlə açaq.

```
<html>

<head>

<title>Javascript</title>

</head>

<body>

<script>

alert("Hello Javascript!");
```

</script>

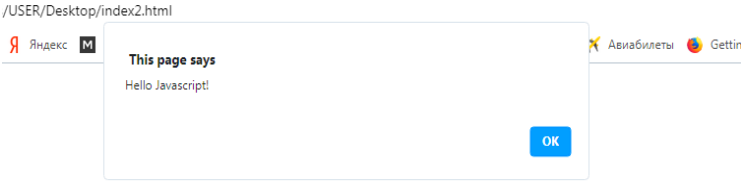
</body>

</html>

Nəticədə, səhifəni açarkən içərisində "Hello Javascript!" yazılan bir bildiriş qutusu qarşımıza çıxacaq.

Javascript kodları Html daxilində sadəcə **<script>** və **</script>** etiketləri arasında yazılmaqdadır. Bu etiketlər arasında yazılan kodlar artıq Javascript kodları sayılır. Yuxarıdakı kodda bu etiketlər arasında sadəcə **alert("Hello Javascript!");** yazmışıq. Bu da yazdığımız ilk Javascript kodu oldu. Deməli, Javascript daxilində bir çox funksiyalar mövcuddur ki, bunlardan biri də **alert** funksiyasıdır. Bu funksiyanın məqsədir hər-hansı bir məlumatı bildiriş qutusu daxilində istifadəçiyə göstərməkdədir. Funksiyanın adı yazıldıqdan sonra mötərizə açılır. Mötərizədən sonra Dırnaq işarələri arasında xəbərdarlıq qutusunda görünməsini istədiyimiz yazını yazırıq. Sonra isə açdığımız dırnaq işarəsini və mötərizəni bağlayırıq. Sətrin sonunda isə nöqtəli vergül qoyuruq. Sətrin sonunda nöqtəli vergül qoymaq bir çox proqramlaşdırma dilinə xasdır.

Beləliklə ilk javascript kodumuzu yazmış oluruq. Nəticədə aşağıdakı pəncərəni görəcəyik.



Javascriptlə yazıların göstərilməsi

Proqramlaşdırmada adicə yazıları da istifadəçiyə göstərmək üçün müxtəlif funksiyalardan istifadə olunur. Javascriptdə də bunun üçün müxtəlif funksiyalar mövcuddur. Bunlardan biri yuxarıda göstərilmiş **alert** funksiyasıdır. Bu funksiya istifadəçiyə məlumatı xəbərdarlıq şəklində göstərməkdədir.

<html>

<head>

<title>Javascript</title>

</head>

<body>

<script>

alert("Hello Javascript!");

</script>

</body>

</html>

Gördüyümüz kimi, **alert** sözünü yazırıq və mötərizə açırıq. Ondan sonra dırnaq işarəsi

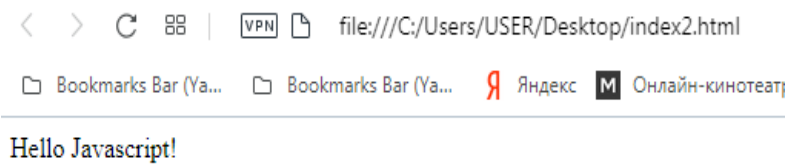
qoyub İstədiyimiz yazını yazırıq. Sonda isə açılmış dırnaq və mötərizəni bağlayırıq və sətirin sonunda nöqtəli vergül qoyuruq. İlk olaraq açdığımız dırnaq işarəsi və mötərizəni sonda bağlamağımız mütləqdir. Nəticədə aşağıdakı pəncərəni görürük.

Yazıları, ifadələri istifadəçiyə göstərmək üçün işlədə biləcəyimiz funksiyalardan biri də **document.write** funksiyasıdır. Burada **document** obyektdir, **write** isə onu içindəki funksiyadır. Obyekt anlayışına gələcək mövzularda daha ətraflı baxacağıq. Funksiyanın istifadəsi **alert** funksiyasına oxşardır, sadəcə, burada **alert** funksiyasından fərqli olaraq yazı sadəcə brauzer üzərində yazılır, bildiriş şəklində gəlmir. Aşağıdakı koda baxaq.

```
<script>

document.write("Hello Javascript!");

</script>
```

Yuxarıdakı Html kodları eyni olduğu üçün sadəcə Javascript hissəsini yazırıq. Siz yuxarıdakı kodlarda olduğu kimi bu kodu da Html kodunun arasında yaza bilərsiniz.

Gördüyümüz kimi burada, **document.write** yazırıq, sonra isə mötərizə və dırnaq işarəsi açırıq. Sonra isə istədiyimiz yazını, ifadəni yazıb açdığımız dırnaq işarəsi və mötərizəni bağlayırıq. Sətrin sonunda isə nöqtəli vergül qoyuruq. Kodu yadda saxlayıb işlətdikdə, brauzer üzərində **"Hello Javascript!"** ifadəsinin yazıldığını görürük.



İstifadə etdiyimiz **document.write** funksiyasının daxilində html kodlarını yaza və onları da çap edə bilərik. Aşağıdakı nümunəyə baxaq.

```
<script>

document.write("<h1>Hello Javascript!
</h1>");

</script>
```

Nəticədə “Hello Javascript!” yazısı böyük ölçüdə, yəni başlıq şəklində çıxacaq.

Funksiyalardan bir neçə dəfə ardıcıl şəkildə istifadə etməyimiz də mümkündür. Aşağıdakı nümunəyə baxaq.

```
<script>

alert("Welcome!");

document.write("Hello Javascript!");

document.write("<br>");

document.write("<marquee>Action</marquee>");

</script>
```

Gördüyümüz kimi, funksiyalardan istədiyimiz sayda istifadə edə bilirik.

Ədədlər və hesablamalar

Javascriptdə və eyni zamanda bir çox proqramlaşdırma dilində yazılar və ədədlər fərqləndirilməkdədir. Məsələn, **document.write** funksiyasında yazıları dırnaq işarəsi arasında yazırdıq. Ancaq, ədədləri yazarkən dırnaq işarəsi qoymağa ehtiyac olmur. Aşağıdakı koda baxaq.

```
<script>

document.write(5);

</script>
```

Gördüyümüz kimi, burada funksiya daxilində ədəd yazdığımıza görə dırnaq işarəsini qoymağa ehtiyac duymuruq. Ancaq, ədəd yerinə yazı yazsa idik, o zaman, dırnaq işarəsindən istifadə etməliydik.

```
<script>

document.write("Hello Javascript!");

</script>
```

Ədədləri və yazıları fərqli olaraq görməyimiz mütləqdir. Çünki, ədədlər üzərində hesablama əməliyyatları da apara bilərik. Burada ədədi

ədəd tipi, yazını isə yazı tipi olaraq qəbul edək. Növbəti mövzularda bu anlayışları daha ətraflı izah edəcəyik.

Ən sadə hesablama əməliyyatları toplama, çıxma, vurma və bölmə əməliyyatlarıdır ki, javascript daxilində də bunları asanlıqla yerinə yetirə bilərik. Aşağıdakı koda baxaq:

```
<script>

document.write(5+6);

</script>
```

Kodu yadda saxlayıb brauzerdə nəticəyə baxdıqda 11 yazıldığını görəcəyik. Gördüyümüz kimi, sadəcə “+” simvolundan istifadə edərək toplama əməliyyatını yerinə yetiririk. Toplama əməliyyatından əlavə çıxma, vurma və bölmə əməliyyatlarını yerinə yetirmək üçün müvafiq olara “-”, “*” və “/” simvollarından istifadə edə bilərik. Aşağıdakı nümunəyə baxaq.

```
<script>

document.write(6+3);

document.write("<br>");

document.write(6-3);
```

**document.write("
");**

document.write(6*3);

**document.write("
");**

document.write(6/3);

**document.write("
");**

</script>

Kodu yadda saxlayıb nəticəyə baxdıqda əməliy-yatların yerinə yetirildiyini və nəticələrin alt-alta çap olunduğunu görəcəyik.

Apardığımız hesablamalar nisbətən daha mürəkkəb də ola bilər. Aşağıdakı nümunə ilə fikrimizi daha da dəqiqləşdirək.

<script>

document.write(6+3*4);

</script>

Nəticədə “6+3*4” ifadəsi hesablanaraq çap olunur. Diqqət edək ki, cavab 18 alınır. Çünki, adi riyaziyyatda olduğu kimi burada da ilk öncə vurma və bölmə, sonra isə toplama və çıxma əməliyyatları yerinə yetirilir. Yəni ilk olaraq 3*4 hesablanır, sonra isə nəticənin, yəni 12-nin

üzərinə 6 gəlinir və 18 cavabı alınır. Əgər ilk öncə toplama hissəsinin yerinə yetirilməyini istəyiriksə, o zaman, toplama hissəsini mötərizələr daxilində yaza bilərik. Aşağıdakı nümunəyə baxaq.

<script>

document.write((6+3)*4);

</script>

Burada cavab 36 alınır. Çünkl, toplama əməliyyatı mötərizə daxilində yazıldığı üçün ilk olaraq toplama əməliyyatı aparılır, sonra isə alınmış ədəd 4-ə vurulur. Aparacağımız hesabalam daha da mürəkkəb şəkildə ola bilər. Aşağıdakı nümunəyə baxaq.

<script>

document.write((6+3)*4+(7+2)/3);

</script>

Burada ilk öncə 6+3 hesablanır və 9 alınır. Həmin 9 rəqəmi 4-ə vurulur və 36 alınır. Sonra isə 36-nın üzərinə ikinci hissənin nəticəsi gəlinir. İkinci hissədə 7 və 2 toplanır və 9 alınır. Sonra isə alınmış 9 rəqəmi 3-ə bölünür və ikinci hissənin nəticəsi olaraq 3 alınır. Son olaraq ilk

hissədən alınmış 36 və 3 toplanaraq 39 nəticəsi alınır.

Dəyişən anlayışı

Kainat yarandığı gündən özündə saysız-hesabsız dəyişənləri saxlamaqdadır. Bu dəyişənlər müxtəlif yerlərdə müxtəlif şəkildə olaraq fərqli şəraitlər yaratmaqdadırlar. Məsələn, yerdəki temperatur hal hazırda fərqli yerlərdə fərqli dəyərdədir. Ortalama 20 dərəcə qəbul edək. Yerdəki temperatur özü bir dəyişəndir. Bəs nəyə görə biz onu dəyişən adlandırırq? Çünki o daima dəyişir. Məsələn, otaqda əvvəlcə 20 dərəcə olur, sonra biraz keçir artır və 22 olur, sonra isə azalaraq 19 olur. Gündüz temperatur daha yüksək olur, gecə isə daha az olur. Gördüyümüz kimi, temperatur daima dəyişir. Bu səbəbdən də onu dəyişən olaraq qəbul edə bilirik. Yerdə temperatur elə şəkildədir ki, bu temperaturda insanların yaşaması mümkün olur. Əgər temperatur dəyişənimizin qiyməti 20 yox, 200 olsa idi, onda bizim Yer üzərində yaşamağımız mümkün olmazdı. Gördüyümüz kimi, temperatur dəyişənimiz xüsusi bir aralıqda dəyişərək Yerdə həyat şəraitinin yaranmasına kömək edir.

Məsələn, bizim yaşımız da bir dəyişəndir. Bu dəyişənin qiyməti əvvəlcə 0 olur, sonra 1 il keçir və 1 olur, sonra 1 il də keçir və 2 olur, yəni

bizim 2 yaşımız olur və hər il bu dəyişənin qiyməti 1 vahid artır. Yaşımız daima dəyişdiyinə görə onu da dəyişən adlandıra bilirik.

Dəyişənləri orta məktəb riyaziyyatından da xatırlaya bilərik. Xatırlaya biləcəyimiz üzərə bu dəyişənlər adətən x, y, z olaraq adlandırılırdı. Bunlar sadəcə adlar idi və onlar müxtəlif qiymətlər alırdı. Dəyişən anlayışından istifadə edərək bir çox riyazi problem də öz həllini tapırdı.

Hər yerdə olduğu kimi proqramlaşdırmada da dəyişən anlayışına ehtiyac duyulur. Məsələn, Gta oyununu xatırlayaq. Oyunçunun ilk olaraq 100 canı olurdu. Oyunçunun canı bir dəyişən idi və 100 qiymətini alırdı. Sonra zaman keçdikcə, oyunçumuz xəsarət aldıqda onun canı 70-ə düşürdü. Yəni, dəyişənimizin qiyməti azalaraq 70 olurdu. Sonra yenidən xəsarət aldıqda 50-ə düşürdü. Öldükdə isə 0-a düşürdü. Gördüyümüz kimi, əvvəlcə oyunçunun can dəyişəninin qiyməti 100 olurdu, sonra öldükdə isə 0 olurdu. Bu dəyişənin qiyməti 0 olduqda isə oyun yenidən başlayırdı.

Gördüyümüz kimi, oyunçunun can dəyişəni 0 və 100 aralığında dəyişirdi, dəyişənin

dəyəri 0 olduqda isə bu müəyyən olunaraq oyun yenidən başladılırdı.

Oyunçunun pulu da bir dəyişən sayılır. Məsələn, əvvəlcə 0 dolları olur, sonra işləyir və 100 dolları olur. Sonra yenə işləyir və fərz edək ki, 500 dolları olur. Sonra onu xərcləyir və 300 dolları olur. Gördüyümüz kimi, pul dəyişəni də daima dəyişir və müxtəlif qiymətlər alır.

Gördüyümüz bir çox şeyin dəyişən olduğunu görə bilərik. Məsələn, oyundakı silahların güllə sayı, toplanan qızıl sayı və s. bunların hamısı dəyişən sayılmaqdadır. Bunlarsız bir oyunu təsəvvür etmək çox çətindir. Beləliklə dəyişənin nə qədər əhəmiyyətli olduğunu görə bilirik.

Biz yuxarıdakı nümunədə dəyişənləri oyun üzərində göstərdik, ancaq, demək olar ki, proqramlaşdırmanın hər sahəsində dəyişənlərdən istifadə olunmaqdadır. Elə Javascript daxilində də bu dəyişənlərdən istifadə olunur. Əlbəttə ki, proqramlaşdırma dili xüsusi bir sintaksisə tabe olmağımızı tələb etdiyi üçün, dəyişəni də elan edərkən xüsusi qaydada elan etməliyik. Javascript dilində dəyişənləri elan etmək üçün **var** açar sözünə ehtiyac duyuruq. Bu söz **variable** sözünün qısaldılmasıdır ki, elə

bu sözün mənası da dəyişən deməkdir. Dəyişənin əsas olaraq bir adı və dəyəri mövcud olur. Dəyişənimiz adını özümüz təyin edə bilərik. Aşağıdakı nümunəyə baxaq.

<script>

var x;

</script>

Beləliklə **x** adlı dəyişəni elan etmiş oluruq. Gördüyümüz kimi, dəyişəninn yaradılması kifayət qədər sadədir. Digər funksiyalarda olduğu kimi, dəyişəni elan etdikdən sonra da sətrin sonunda nöqtəli vergül qoymağımız mütləqdir. Kodu yadda saxlayıb brauzerdə səhifəni açdıqda heç nə baş vermədiyini görəcəyik. Çünki, biz sadəcə dəyişəni elan etmişik, o dəyişənlə heç bir əməliyyat aparmamışıq. Aşağıdakı qaydada həmin dəyişənə qiymət verək.

<script>

var x;

x=5;

</script>

Gördüyümüz kimi, sadəcə **x=5** yazaraq həmin dəyişənə **5** qiymətini vermiş oluruq. İndi isə həmin dəyişəni çap edək.

```
<script>

var x;

x=5;

document.write(x);

</script>
```

Kodu yazıb yadda saxlayaq. Brauzerdə nəticəyə baxdıqda 5 rəqəminin çap olunduğunu görəcəyik. Gördüyümüz kimi, brauzerdə çap etmək funksiyası olan **document.write** funksiyasında sadəcə dəyişənin adını yazırıq və o çap olunmuş. Qeyd edək ki, dəyişənlərin adlarını dırnaq arasında **yazmırıq.** Aşağıdakı nümunəyə də baxaq.

```
<script>

var temperatur;

temperatur=20;

document.write("Otagin temperaturu ");

document.write(temperatur);
```

document.write(" derecedir.");

</script>

Nəticəyə baxdıqda “Otagin temperaturu 20 derecedir.” Yazısının çap olunduğunu görəcəyik. Gördüyümüz kimi, burada dəyişənimizin adı “temperatur” olaraq təyin olunmuşdur. Dəyişənimizə istədiyimiz adı verə bilərik, hətta öz adımızı da. Sadəcə dəyişəni adını təyin edərkən bəzi şeylərə diqqət etməliyik. Məsələn, dəyişən adında rəqəm ola bilər, ancaq, rəqəmlə başlaya bilməz. Dəyişən adı **“x1”** ola bilər, ancaq, **“1x”** ola bilməz. Dəyişən adında yalnız və yalnız hərflərdən, rəqəmlərdən və “_” simvolundan istifadə edə bilərik.

Eyni sətirdə bir neçə dəyişəni də elan etməyimiz mümkündür, Bunun üçün sadəcə onları vergüllə ayırmalıyıq. Aşağıdakı nümunəyə baxaq.

<script>

var x, y;

x=5;

y=10;

document.write(x);

**document.write("
");**

document.write(y);

</script>

Nəticədə **x** və **y** dəyişənlərinin qiymətlərinin alt-alta çap olunduğunu görəcəyik.

Hesablama əməliyyatlarını dəyişənlər üzərində də aparmağımız mümkündür. Aşağıdakı nümunəyə baxaq.

<script>

var x, y;

x=5;

y=10;

document.write(x+y);

</script>

Nəticədə x və y dəyişənlərinin qiymətləri toplanaraq çap olunacaq. Dəyişənlərin qiymətlərinin cəmini üçüncü bir dəyişəndə də yadda saxlaya bilərik.

<script>

var x,y,z;

x=5;

y=10;

z=x+y;

document.write(z);

</script>

Burada z dəyişəninin qiyməti x və y dəyişənlərinin qiymətlərinin cəminə bərabər olur. Sonra isə z dəyişəninin qiyməti **document.write** funksiyası vasitəsilə çap olunur.

Dəyişənlərə qiymət olaraq ədədlərdən əlavə yazıları da verə bilərik. Aşağıdakı nümunəyə baxaq.

<script>

var x;

x="Deyisen";

document.write(x);

</script>

Gördüyümüz kimi, burada dəyişənə **"Deyisen"** mənimsətmiş oluruq. Dəyişənə yazıları

mənimsədərkən onları dırnaq işarəsi arasında yazmağımız mütləqdir.

Dəyişənin qiyməti yazı olsa da, biz onlar üzərində toplama əməliyyatı apara bilərik. Bu zaman sadəcə yazılar yan-yana yazılacaq. Aşağıdakı nümunəyə baxaq.

<script>

var x,y;

x="Birinci Deyisen ";

y="Ikinci Deyisen";

document.write(x+y);

</script>

Nəticədə dəyişənlərin qiymətləri yan-yana yazılmış olacaq. Toplama yerinə çıxma, vurma və ya bölmə yazsaq xəta ilə qarşılaşacağıq.

Dəyişənləri öyrəndikdən sonra maraqlı bir funksiya olan **prompt** funksiyasına baxa bilərik. Funksiyanın istifadəsi həm sadədir, həm də maraqlı bir funksiyadır. Aşağıdakı nümunəyə baxaq.

<script>

var ad;

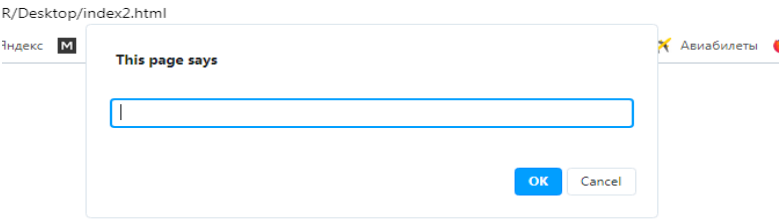
ad=prompt();

document.write("Xos geldiniz ");

document.write(ad);

</script>

Kodu yadda saxlayıb brauzerdə səhifəni işlətdikdə aşağıdakı kimi bir görüntü çıxacaq.



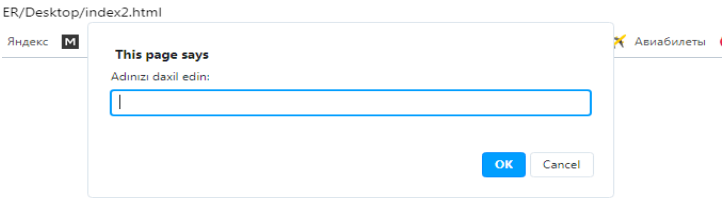
Deməli bu funksiya alert funksiyasına çox oxşar bir funksiyadır. Hər iki funksiyada da bir xəbərdarlıq qutusu çıxır. Ancaq, alert funksiyası sadəcə istifadəçiyə xəbərdarlıq qutusu daxilində məlumat göstərmək üçün istifadə olunur. İstifadə etdiyimiz **prompt** funksiyasında isə, xəbərdarlıq qutusu daxilində bir sahə yaranır və istifadəçi o sahəyə istədiyi yazını daxil edə bilir. İstifadəçi yazını daxil edib "Ok" düyməsinə basdıqdan sonra **ad=prompt();** yazdığımız

üçün daxil edilən yazı **ad** dəyişəninə ötürülmüş olur. Bundan sonra isə biz ad dəyişənini çap edirik.

İstifadə etdiyimiz **prompt** funksiyasının daxilinə yazı da yazmağımız mümkündür. Bu zaman daxil etdiyimiz yazı istifadəçiyə təqdim olunmuş xananın yuxarısında görünəcək. Aşağıdakı nümunəyə baxaq.

```
<script>

var ad;

ad=prompt("Adınızı daxil edin:");

document.write("Xos geldiniz ");

document.write(ad);

</script>
```

Gördüyümüz kimi, **prompt** funksiyasının daxilində dırnaq işarələri arasında bir ifadə yazırıq. Nəticə aşağıdakı kimi olur.



Gördüyümüz kimi, həmin yazı boş xananın yuxarısında yazılmış olur.

Boş xanada istifadəçidən ədəd daxil etməsini tələb edib həmin ədəd üzərində hesablama əməliyyatı da apara bilərik. Məsələn, elə edək ki, biz açılan pəncərədə bir ədəd daxil edək, ekranda isə həmin ədədin kvadratı çap olunsun. Bunun üçün sadəcə ədədi götürüb özünə vurmağımız kifayətdir. Aşağıdakı nümunəyə baxaq.

```
<script>

var x;

x=prompt("Kvadratini tapmaq istediyiniz ededi daxil edin:");

document.write(x*x);

</script>
```

Məsələn, 5-in kvadratını tapmaq üçün 5-i 5-ə vurmalıyıq, yəni özünə. Burada da həmin şeyi edirik, x dəyişəninə ötürülmüş ədədin kvadratını tapmaq üçün x-i özünə vururuq. Nəticədə x-in kvadratını çap etmiş oluruq. Əgər təqdim olunmuş mətn sahəsinə ədəd yox, yazı yazılarsa, o zaman xəta çıxacaqdır.

Hesablama funksiyaları

Bundan əvvəlki mövzularda toplama, çıxma vurma və bölmə əməliyyatlara baxmışdıq. Bu mövzuda isə ədədin kök altısını, faktorialını, bucağın sinus və ya kosinusu tapmaq kimi daha mürəkkəb əməliyyatlara baxacağıq. Bu əməliyyatları aparmaq üçün Javascript dilində hazır funksiyalar mövcuddur. Bu funksiyalar **Math** obyekti altında yerləşməkdədir. Bu funksiyalardan biri **sqrt** funksiyasıdır. Bu funksiya ədədin kök altısını tapmaq üçün istifadə olunur. Aşağıdakı nümunəyə baxaq.

```
<script>

var x;

x=Math.sqrt(9);

document.write(x);

</script>
```

Nəticədə 9-un kök altısı tapılaraq çap edilir. Burada **Math** sözünün ilk hərfinin böyüklə yazıldığını unutmayaq. Digər riyazi funksiyalara da nəzər yetirək. Qeyd edək ki, bu funksiyaları əzbərləməyə ehtiyac yoxdur. Sadəcə, ehtiyac duyulduqda funksiyanın istifadə qaydasına

baxıb istifadə edə bilərsiniz. Sadəcə bilək ki, Javascriptdə bu tip funksiyalar da mövcuddur.

Math.abs funksiyası. Bu funksiya ədədin modulunu tapmaq üçün istifadə olunur. Aşağıdakı nümunəyə baxaq.

```
<script>

var x;

x=Math.abs(-5);

document.write(x);

</script>
```

Nəticədə 5 çap olunmuş olur.

Math.acos funksiyası. Ədədin arkkosinusunu tapmaq üçün istifadə olunur. Dəyər radianla verilir. Nümunə:

```
<script>

var x;

x=Math.acos(0.8);

document.write(x);

</script>
```

Math.asin funksiyası. Ədədin arksinusunu tapmaq üçün istifadə olunur. Dəyər radianla verilir. Nümunə:

<script>

var x;

x=Math.asin(0.8);

document.write(x);

</script>

Math.atan funksiyası. Ədədin arktangesini tapmaq üçün istifadə olunur. Dəyər radianla verilir. Nümunə:

<script>

var x;

x=Math.atan(5);

document.write(x);

</script>

Math.ceil funksiyası. Ədədi ən yaxın yuxarı tam ədədə yuvarlaqlaşdırmaq üçün istifadə olunur. Nümunə:

<script>

var x;

x=Math.ceil(4.3);

document.write(x);

</script>

Nəticədə 4.3 ədədinə ən yaxın yuxarı tam ədəd 5 olduğu üçün, 5 çap olunmuş olur.

Math.cos funksiyası. Ədədin kosinusunu tapmaq üçün istifadə olunur. Dəyər radianla verilir. Nümunə:

<script>

var x;

x=Math.cos(2);

document.write(x);

</script>

Math.exp funksiyası. E^x qiymətini tapmaq üçün istifadə olunur. Nümunə:

<script>

var x;

x=Math.exp(2);

document.write(x);

</script>

E ədədinin təxmini qiyməti 2.71-ə bərabərdir. Ancaq, daha dəqiq qiymətləri daha uzun yazılmaqdadır.

Math.floor funksiyası. Ədədi ən yaxın aşağı tam ədədə yuvarlaqlaşdırmaq üçün istifadə olunur. Nümunə:

<script>

var x;

x=Math.floor(2.6);

document.write(x);

</script>

Burada 2.6 ədədinə ən yaxın aşağı tam ədəd 2 olduğu üçün, nəticədə 2 çap olunur.

Math.log funksiyası. Ədədin natural loqarifmasını tapmaq üçün istifadə olunur. Nümunə:

<script>

var x;

x=Math.log(2.718281828459045);

document.write(x);

</script>

Math.max funksiyası. Daxil edilmiş ədədlərdən ən böyüyünü tapmaq üçün istifadə olunur. Bir neçə parametr ala bilər. Nümunə:

<script>

var x;

x=Math.max(5,4,7,8);

document.write(x);

</script>

Nəticədə 8 çap olunur. Çünki, daxil edilən ədədlərdən ən böyüyü 8-dir.

Math.min funksiyası. Daxil edilmiş ədədlərdən ən kiçiyini tapmaq üçün istifadə olunur. Bir neçə parametr ala bilər. Nümunə:

<script>

var x;

x=Math.min(5,4,7,8);

document.write(x);

</script>

Nəticədə 4 çap olunur. Çünki, daxil edilən ədədlərdən ən kiçiyi 4-dür.

Math.pow funksiyası. Bir ədədin üstünü tapmaq üçün istifadə olunur. Məsələn, 2^3 ifadəsininin qiymətini bu funksiya ilə tapa bilərik. Aşağıdakı nümunəyə baxaq.

<script>

var x;

x=Math.pow(2,3);

document.write(x);

</script>

Nəticədə ifadə hesablanır və 8 çap olunur.

Math.round funksiyası. Ədədi ən yaxın tam ədədə yuvarlaqlaşdırmaq üçün istifadə olunur.

<script>

var x;

x=Math.round(4.6);

document.write(x);

</script>

Burada 4.6 ədədi 5-ə yaxın olduğu üçün nəticədə 5 alınır və 5 çap olunur. Aşağıdakı nümunəyə də baxaq:

<script>

var x;

x=Math.round(4.4);

document.write(x);

</script>

Burada isə 4.4 ədədi 4-ə yaxın olduğu üçün nəticədə 4 alınır. Bu funksiya **ceil** və **floor** funksiyalarına oxşardır. Ancaq, **ceil** funksiyası ancaq yuxarıya doğru yuvarlaqlaşdırırdı. Məsələn, **ceil** funksiyasında həm 4.4 olduqda, həm 4.6 olduq da 5-ə yuvarlaqlaşdırırdı. Baxmayaraq ki, 4.4 ədədi 4-ə daha yaxındır. Digər **floor** funksiyasında isə həmin ədədlər 4-ə yuvarlaqlaşdırırdı. Burda da baxmayaraq ki, 4.6 ədədi 5-ə daha yaxındır. Ancaq **round** funksiyasında ədəd hansı tam ədədə yaxındırsa ona da yuvarlaqlaşdırır. Ona görə də, parametr 4.4 olduqda 4 alınır, 4.6 olduqda isə 5 alınır.

Çünki, 4.4 ədədi 4-ə daha yaxındır, 4.6 ədədi isə 5-ə daha yaxındır.

Math.sin funksiyası. Ədədin sinusunu tapmaq üçün istifadə olunur. Dəyər radianla verilir. Nümunə:

<script>

var x;

x=Math.sin(0.3);

document.write(x);

</script>

Math.tan funksiyası. Ədədin tangensini tapmaq üçün istifadə olunur. Dəyər radianla verilir. Nümunə:

<script>

var x;

x=Math.tan(3);

document.write(x);

</script>

Nəticədə ədədin tangensi tapılmış olur.

Yuxarıda bir sıra əsas riyazi funksiyaları qeyd etmiş olduq. **Math** obyektinin daxilində riyazi funksiyalardan əlavə riyazi sabitlər də mövcuddur. Məsələn, Math. PI sabiti pi ədədinin dəyərini qaytarmaqdadır. Aşağıdakı nümunəyə baxaq.

```
<script>

var x;

x=Math.PI;

document.write(x);

</script>
```

Nəticədə pi ədədinin dəyəri çap olunmuş olur. Əlavə olaraq bir sıra sabitlərin də adını qeyd edək.

Math.E sabiti. Eyler ədədinin qiymətini göstərməkdədir. Təxminən 2.71-ə bərabərdir.

Math.SQRT2 sabiti. 2 ədədinin kök altısını göstərməkdədir.

Math.SQRT1_2 sabiti**.** 0.5 ədədinin kök altısını göstərməkdədir.

Math.LN2 sabiti. 2-nin natural loqarifmasını göstərməkdədir.

Math.LN10 sabiti. 10-un natural loqarifmasını göstərməkdədir.

Math.LOG2E sabiti. E ədədinin 2 əsaslı loqarifmasını göstərir.

Math.LOG10E sabiti. E ədədinin 10 əsaslı loqarifmasını göstərir.

Təsadüfi seçim funksiyası

Javascript dilində və bir çox proqramlaşdırma dillərində təsadüfi seçim funksiyası mövcuddur. Bu funksiya vasitəsilə təsadüfi ədədlər seçmək mümkündür. Məsələn, Snake oyununu xatırlayaq. Oyunda snake qarşısındakı yeməyi yedikdən sonra yemək başqa bir yerdə yaranırdı. Həmin yeməyin yarandığı yer isə təsadüfi olaraq seçilməlidir. Çünki, əvvəlcədən təyin olunsa, hər dəfə yemək eyni yerdə olacaq və bu da oyunu maraqsızlaşdıracaq. Buna görə də, təsadüfi seçim funksiyası çox vacibdir. Məsələn, lotereya oyunu da hazırlaya bilərik. Bu oyunda da, ədədləri seçmək üçün təsadüfi seçim funksiyasından istifadə etməliyik. Təsadüfi seçimlər oyunlarda və bir çox yerdə istifadə olunmaqdadır və vacib funksiyalardan biridir.

Javascript daxilində təsadüfi ədəd seçmək üçün **Math.random** funksiyasından istifadə olunur. Funksiya heç bir parametr almır. Aşağıdakı nümunəyə baxaq.

```
<script>

var x;

x=Math.random();
```

document.write(x);

</script>

Brauzerdə səhifəni işlətdikdə 0 və 1 arasında təsadüfi bir ədəd çap olunduğunu görəcəyik. Brauzerdəki səhifəni hər dəfə yenilədikdə hər dəfə fərqli-fərqli ədədlər çıxacaq. Yuxarıdakı kodda ədəd 0 və 1 arasında uzun bir ədəd olur. Biz onu müəyyən aralıqdakı tam bir ədədə çevirə bilərik. Fərz edək ki, aşağıdakı ədəd çap olundu:

0.8819406369037288

Biz bu ədədi 100-ə vursaq **88,19406369037288** olmuş olacaq. Elə bu ədədi də aşağıya doğru yuvarlaqlaşdırsaq 88 alacağıq. Nəticədə, hər dəfə 0 ilə 99 arasında tam ədəd almış olacağıq. Aşağıdakı nümunəyə baxaq.

<script>

var x;

x=Math.random()*100;

document.write(Math.floor(x));

</script>

İlk olaraq təsadüfi ədədi tapırıq və 100-ə vuruq. Sonra isə ədədi çap etdikdə **Math.floor** funksiyası ilə onu yuvarlaqlaşdırırıq və tam bir qiymət alırıq. Əgər x-in üzərinə 1 gəlsək, o zaman, ədəd 1 və 100 arasında olacaq. 100-ün yerinə 1000 yazsaydıq, ədəd 0 və 999 arasında olacaqdı.

Təsadüfi olaraq seçmək istədiyimiz ədəd xüsusi bir aralıqda da ola bilər. Məsələn, istəyə bilərik ki, seçəcəyimiz ədəd 5 və 20 arasında olsun. Bu zaman necə edək? Ədədi 20-ə vursaq 0 və 20 arasında bir ədəd alacağıq. İndi biz elə etməliyik ki, bu 0 və 20 arasında ədəd yalnız 5 və 20 arasında olsun. Bu zaman, ədədi 20-ə yox, 15-ə vura bilərik. Ədədi 15-ə vurduqda, ədəd 0 və 15 arasında olacaq. Onun da üzərinə 5 gəldikdə, artıq ədədimiz 5 və 20 arasında olmuş olacaq. Məsələn, 4 tapılsa 9 olacaq. 14 tapılsa 19 olacaq. Aşağıdakı koda baxaq.

```
<script>

var x;

x=Math.floor(Math.random()*15+5);

document.write(x);

</script>
```

Bu zaman ədəd 15-ə vurulur. 15-ə vurulduğu üçün ədəd 0 və 14 aralığında olur. Onun üzərinə 5 gəldikdə isə ədəd artıq 5 və 20 arasında olmuş olur. Burada diqqət edək ki, təsadüfi ədədi vurduğumuz 15 ədədi, aralığında təsadüfi ədəd tapmaq istədiyimiz iki ədədin fərqinə bərabərdir, yəni 20-15 ifadəsinə bərabərdir. Təsadüfi ədədi 15-ə vurduqdan sonra ifadənin üzərinə gəldiyimiz 5 isə iki ədəddən ən kiçiyidir. O zaman belə bir nəticəyə gələ bilərik. Təsadüfi ədədi vuracağımız ədəd iki ədədin fərqinə, üzərinə gələcəyimiz ədəd isə onlardan ən kiçiyinə bərabərdir. O zaman kodumuzu aşağıdakı şəkildə yaza bilərik:

```
<script>

var x,max,min;

min=5;

max=19;

x=Math.floor(Math.random()*(max-
min+1)+min);

document.write(x);

</script>
```

Beləliklə, sadəcə **max** və **min** dəyişənlərinin qiymətlərini dəyişməklə istədiyimiz aralıqda təsadüfi ədəd seçə bilərik. Burada **max** dəyişəni yuxarı qiymət, **min** dəyişəni isə aşağı qiymətdir. Burada **max-min+1** yazaraq fərqin üzərinə 1 gəlməsəydik, o zaman, 5 və 19 arasında qiymətlər tapılacaqdı. Çünki 15-ə vurduqda 0 və 14 arası qiymətlər tapılırdı.

Şərt operatoru

Demək olar ki, həyatımızın hər bir nöqtəsində şərtlərdən istifadə etməkdəyik. Məsələn, yolu keçərkən biz beynimizdə şərt qoyuruq, əgər işıq qırmızıdırsa dayan, yaşıldırsa keç. Beləliklə biz qoyduğumuz şərtlə əməliyyatlar yerinə yetiririk. Qoyduğumuz şərt yaşıl işığın yanmasıdır. Əgər yaşıl işıq yanarsa, bu halda şərt ödənilmiş olur, o zaman biz də yolu keçirik. Əgər qoyduğumuz şərt ödənirsə biz əməliyyatları yerinə yetiririk, yəni, yaşıl yanarsa yolu keçirik.

Məsələn, oyunlarda şərt qoyulur, əgər canımız 0 olarsa, o zaman, xarakterimiz ölür, oyun yenidən başlayır. Burada şərt canımızın 0 olmağıdır. Əgər bu şərt ödənərsə xarakterimiz ölür və oyun yenidən başlayır.

Proqramlaşdırmada da bu şərtlərdən geniş istifadə olunmaqdadır. Çünki, şərtlər çox vacibdir, bir çox yerdə istifadə olunmaqdadır. Javascript dilində şərt yaratmaq üçün **if** operatorundan istifadə olunur. Aşağıdakı nümunəyə baxaraq fikrimizi daha da aydınlaşdıraq.

<script>

var x;

x=prompt("5+3=?");

if(x==8) document.write("Cavab duzdur.");

</script>

Bu nümunədə **prompt** funksiyası vasitəsilə **"5+3=?"** sualını soruşuruq. Əgər istifadəçi 8 daxil edərsə deməli cavabı doğrudur. Ancaq, 8 daxil etməzsə deməli yalnış cavab verib. Bu nöqtədə artıq bizim şərt operatoruna ehtiyacımız var. Yuxarıdakı nümunədə **if(x==8)** yazaraq şərt operatorundan istifadə etmiş oluruq. Şərt operatorunun istifadə qaydası bu şəkildədir. Sadəcə **if** sözü yazılır, sonra mötərizə açılır və mötərizə daxilində şərt yazılır. Şərtimiz x dəyişəninin, yəni daxil olunan ədədin 8-ə bərabər olub-olmamasıdır. Bunun üçün də, biz **x==8** yazırıq. Bərabərliyi yoxlayarkən iki ədəd **=** işarəsi yan-yana yazılır. Sonra isə mötərizəni bağlayırıq. Bundan sonra isə əgər şərt ödənərsə yerinə yetirəcəyimiz əməliyyatı yazırıq. Əgər şərt ödənərsə, yerinə yetirəcəyimiz əməliyyat “Cavab doğrudur.” Sözünü çap etməkdir. Nəticədə, əgər istifadəçi 8 daxil edərsə, o zaman, cavabın doğru olduğu ona bildirilir.

Şərt operatoruna aid başqa bir nümunə də göstərək. İstifadəçidən bir şifrə soruşaq, əgər şifrəni bilərsə ona “Xoş gəldiniz” deyək. Şifrəmiz kiçik hərflərlə “javascript12” olsun. Aşağıdakı koda baxaq.

```
<script>

var x;

x=prompt("Sifreni daxil edin:");

if(x=="javascript12") document.write("Xos geldiniz.");

</script>
```

Burada istifadəçi bir şifrə daxil edir və bu şifrə x dəyişəninə ötürülür. Aşağıda isə bu şifrənin **"javascript12"** olub-olmadığını yoxlayırıq. Digər yerlərdə olduğu kimi burada da yazılar dırnaq işarəsi içində yazılaraq yoxlanılır. Əgər daxil olunmuş şifrə “javascript12” olarsa istifadəçiyə “xoş gəldiniz” deyirik.

Şərt daxilində bərabərlikdən əlavə ədədlərin böyüklük və kiçikliyini də yoxlamaq mümkündür. Bunun üçün sadəcə “>” və “<” işarələrindən istifadə olunur. Təsəvvür edək ki, bir film göstəririk. Ancaq, filmə baxmaq üçün yaşınız 7-dən çox olmalıdır. Bunun üçün

istifadəçidən yaşının neçə olduğunu soruşmalıyıq. Əgər yaşı 7-dən böyük olarsa, ona filmi göstəririk. Aşağıdakı nümunəyə baxaq:

<script>

var yas;

yas=prompt("Yasiniz:");

if(yas>7) document.write("Filmi izleye bilersiniz.");

</script>

Burada istifadəçidən yaşını soruşuruq və “yas>7” ifadəsi ilə yaşın 7-dən böyük olub-olmadığını yoxlayırıq. Əgər yaş 7-dən böyük olarsa, şərt ödənir və istifadəçiyə filmi izləyə biləcəyi deyilir. Əgər yaş 7-dən kiçik olarsa heç bir şey çap olunmur.

Burada bir şeyə diqqət edək ki, yaş 7 daxil olunarsa, şərt ödənməmiş olur, çünki bizim şərtimiz yaşın 7-dən böyük olmasıdır. Ancaq, biz yaşı 7-dən böyük və həmçinin 7-ə bərabər olanlara da icazə vermək istəyiriksə, o zaman, **böyükdür** ifadəsindən yox, **böyük bərabərdir** ifadəsindən istifadə etməliyik. Yəni, **“yas>7”** yerinə **“yas>=7”** yazmalıyıq. Bu zaman yaşı 7 olanlara da icazə vermiş olacağıq.

<script>

var yas;

yas=prompt("Yasiniz:");

if(yas>=7) document.write("Filmi izleye bilersiniz.");

</script>

İndi isə fərz edək ki, səhifəmiz yalnız uşaqlar üçündür və səhifəyə yalnız 13 yaşından kiçik uşaqlar daxil ola bilər. Bunun üçün istifadəçidən yaşını götürməli və 13-dən kiçik olub-olmadığını yoxlamalıyıq. Aşağıdakı koda baxaq:

<script>

var yas;

yas=prompt("Yasiniz:");

if(yas<13) document.write("Otaga daxil ola bilersiniz.");

</script>

Gördüyümüz kimi, burada istifadəçidən yaşını götürürük və “**<**” işarəsi ilə yaşın 13-dən kiçik olub-olmadığını yoxlayırıq. Əgər istifadəçinin

yaşı 13-dən kiçikdirsə, o zaman, otağa daxil ola biləcəyini deyirik.

Burada diqqət edək ki, otağa yalnız yaşı 13-dən kiçik olanlar daxil ola bilər. Əgər 13 daxil etsək, daxil ola bilməyəcəyik. Əgər 13 daxil etdikdə də, otağa daxil olmasını istəyiriksə, o zaman, **kiçikdir** ifadəsindən yox, **kiçik bərabərdir** ifadəsindən istifadə etməliyik. Yəni, **yas<13** yerinə **yas<=13** yazmalıyıq. Aşağıdakı nümunəyə baxaq.

```
<script>

var yas;

yas=prompt("Yasiniz:");

if(yas<=13) document.write("Otaga daxil
ola bilersiniz.");

</script>
```

Beləliklə, yaşı 13 olanlar da otağa daxil ola biləcəklər.

İndi isə fərz edək ki, bir otağımız var və bu otağa bütün yaşda olanlar girə bilərlər, ancaq, yalnızca yaşı 20 olanlar girə bilməzlər. Bu zaman, biz istifadəçidən yaşını soruşmalıyıq və onun yaşının 20-dən fərqli olub-olmadığını

yoxlamalıyıq. Şərt daxilində fərqliliyi yoxlamaq üçün **“!=”** ifadəsindən istifadə olunur. Aşağıdakı nümunəyə baxaq:

```
<script>

var yas;

yas=prompt("Yasiniz:");

if(yas!=20) document.write("Otaga daxil
ola bilersiniz.");

</script>
```

Beləliklə, yalnızca yaş 20-dən fərqli olduqda istifadəçiyə otağa daxil ola biləcəyini deyirik. Əgər yaş 20 olarsa, heç bir şey çap olunmur.

Kassir müştərimizə kömək

Kassir müştərimiz tez-tez hesablamalar aparır və bu hesablamalarını asanlaşdıracaq bir səhifə yaratmağımızı xahiş edir. Biz də bunun üçün iki dəfə prompt funksiyasından istifadə edib 2 qiymət götürə və onları toplaya bilərik. Kodumuza baxaq:

```
<script>

var x,y;

x=prompt("Birinci məhsulun pulu");

y=prompt("İkinci məhsulun pulu");

alert(x*1+y*1);

</script>
```

Beləliklə dostumuzun problemini həll etmiş oluruq. Burada dəyişənləri toplayarkən niyə 1-ə vurduğumuzu gələcək mövzularda izah edəcəyik.

Tərcüməçi müştərimizə kömək

Tərcüməçi müştərimiz bəzən sözlərin tərcüməsini unudur. Buna görə, bizdən xahiş edir ki, onun üçün sözləri yazıb tərcüməsini tapa biləcəyi bir səhifə hazırlayaq. Biz də prompt funksiyası və if operatorundan istifadə edərək bunu edə biləcəyimizi görürük. Sadəcə prompt vasitəsilə sözü götürüb onu if operatoru ilə yoxlayıb tərcüməsini çap edə bilərik. Kodumuza baxaq:

```
<script>

var soz;

soz=prompt("Sozu daxil edin:");

if(soz=="apple"){

alert("Alma");

}

if(soz=="orange"){

alert("Portagal");

}

if(soz=="smart"){
```

alert("Ağıllı");

}

</script>

Beləliklə dostumuz üçün 3 sözün tərcüməsini tapa biləcəyi bir səhifə yaratmış oluruq. Bu sözlərin sayını istədiyimiz qədər artıra bilərik.

Riyaziyyatçı müştərimizə kömək

Bu dəfə riyaziyyatçı müştərimiz bizdən hesablamalarını aparmaq üçün bir proqram istəməkdədir. Tələbi isə bu şəkildədir ki, ilk olaraq iki mətn sahəsi olsun. Birinci sahəyə istədiyi ədədi daxil etsin. İkinci sahə isə əməliyyatın nömrəsi olsun. İkinci sahəyə 1 daxil etdikdə İlk ədədin kök altısı, 2 daxil etdikdə kvadratı, 3 daxil etdikdə isə ədədin natural loqarifması tapılsın. Kodumuza baxaq:

```
<script>

var x,y;

x=prompt("Ededi daxil edin");

y=prompt("Emeliyyati daxil edin. 1- kok alti, 2 kvadrat, 3 natural loqarifma");

if(y==1){

alert(Math.sqrt(x));

}

if(y==2){

alert(Math.pow(x,2));

}

if(y==3){
```

alert(Math.log(x));

}

</script>

Burada x dəyişəni hesablama aparılacaq əməliyyatı, y dəyişəni isə əməliyyatın növünü bildirir. Əgər, y dəyişəni 1 olarsa x-in kökü tapılır. Əgər y dəyişəni 2 olarsa x-in kvadratı tapılır. Sonda isə əgər y dəyişəni 3 olarsa x-in natural loqarifması tapılır.

Lotereyaçı müştərimizə kömək

Lotereyaçı müştərimiz bizdən təsadüfi ədədlər yerləşdirilmiş lotereya cədvəli yaratmağımızı xahiş edir. Biz sadəcə öyrəndiyimiz təsadüfi seçim funksiyasından istifadə edərək bu cədvəli yarada bilərik. Kodumuza baxaq:

```
<script>

var a,b,c,d,e,f;

a=Math.floor(Math.random()*100);

b=Math.floor(Math.random()*100);

c=Math.floor(Math.random()*100);

d=Math.floor(Math.random()*100);

e=Math.floor(Math.random()*100);

f=Math.floor(Math.random()*100);

document.write("<table border=1
width=500>");

document.write("<tr>");
```

```
document.write("<td>" + a + "</td>" +
"<td>" + b + "</td>" + "<td>" + c +
"</td>");

document.write("</tr>");

document.write("<tr>");

document.write("<td>" + d + "</td>" +
"<td>" + e + "</td>" + "<td>" + f +
"</td>");

document.write("</tr>");

document.write("</table>");

</script>
```

Burada 6 ədəd dəyişən elan edirik və hər dəyişənə 1 və 100 arası təsadüfi ədəd köçürürük. Beləliklə lotereya biletini hazırlamış oluruq. Səhifə hər dəfə yeniləndikdə fərqli biletlər görünəcək.

Şərtlər ikinci hissə

Şərtin ödəndiyi hallardan əlavə şərtin ödənmədiyi hallar da mövcuddur. Yuxarıda yazdığımız kodlardan birinə bir daha baxaq.

```
<script>

var x;

x=prompt("5+3=?");

if(x==8) document.write("Cavab duzdur.");

</script>
```

Deməli burada x dəyişəninin qiyməti 8 olarsa cavabın düz olduğunu yazırıq. Ancaq, x dəyişəni 8 olmazsa heç bir əməliyyat yerinə yetirmirik. Bəs biz x dəyişəni 8 olmazsa cavabın səhv olduğunu demək istəyiriksə, o zaman nə etməliyik? Artıq burada **if** operatorunun yanında işlənən **else** hissəciyindən istifadə etməliyik. Bu hissəcik şərtin ödənmədiyi halda əməliyyatları yerinə yetirmək üçün nəzərdə tutulub. Aşağıdakı nümunəyə baxaq.

```
<script>

var x;

x=prompt("5+3=?");
```

if(x==8) document.write("Cavab duzdur.");

else document.write("Cavab sehvdir.");

</script>

Gördüyümüz kimi, **if** yazdığımız sətirdən sonra **else** yazırıq və onun qarşısında da yerinə yetirəcəyimiz əməliyyatı yazırıq ki, bu əməliyyatda da istifadəçiyə cavabının səhv olduğunu deyirik. Beləliklə, daxil edilmiş cavab 8 olarsa, o zaman, cavabın düz olduğunu deyirik. Əks halda isə, yəni şərt ödənməyibsə istifadəçiyə cavabının səhv olduğunu deyirik.

Yazacağımız bir neçə nümunə ilə bu hissəni daha yaxşı anlayacağımızı düşünürəm. Məsələn, istifadəçidən şifrəni daxil etməsini tələb edək. Əgər şifrə “javascript” sözü olarsa, o zaman, doğru daxil etdiyini deyək. Ancaq, “javascript” sözünü daxil etməzsə, yəni, şərt ödənməzsə, o zaman, yalnış daxil etdiyini deyək. Kodu aşağıdakı şəkildə yazaq.

<script>

var x;

x=prompt("Parol:");

if(x=="javascript") document.write("Xos geldiniz.");

else document.write("Parol sehvdir.");

</script>

Beləliklə, daxil edilen parol “javascript” oalrsa, istifadəçiyə xoş gəldiyini deyirik. Əks halda isə, parolun səhv olduğunu ona bildiririk.

Bu hissəyə aid son bir nümunə də yazaq. Uşaqlar üçün bir oyun otağımız olsun və otağa yalnız 12 yaş və 12 yaşdan kiçik olanlar daxil ola bilsin. Dİgər şəxslər isə daxil ola bilməsin. Aşağıdakı koda baxaq.

<script>

var x;

x=prompt("Yasinizi daxil edin");

if(x<=12) document.write("Oyun otagina xos geldiniz :)");

else document.write("Otaga yalniz 12 yasli ve yasi 12-den asagi olan usaqlar daxil ola biler.");

</script>

Beləliklə, otağa daxil olmaq istəyən şəxsin yaşı 12-dən böyük olduqda, otağa daxil ola bilməyəcəyini deyirik.

Şərt operatorlarında öyrənməli olduğumuz vacib bir hissə də **else if** hissəsidir. Bu hissə şərtdə müxtəlif halları yoxlamaq üçün nəzərdə tutulmuşdur. Fərz edək ki, bir oyun otağımız var. Bu oyun otağına yalnız yaşı 12-dən aşağı olan uşaqlar daxil ola bilər, yaşı 12-dən böyük olanlar isə daxil ola bilməz. Ancaq, yaşı 12 olanlar yalnız xüsusi icazə ilə daxil ola bilər. Bu zaman bizim üç halımız var. Birinci hal yaşın 12-dən kiçik olmasıdır. Bu halda şəxsə otağa daxil olmağa icazə verəcəyik. İkinci hal isə yaşının 12-ə bərabər olmasıdır. Bu halda isə şəxsə xüsusi icazənin lazım olduğunu bildirəcəyik. Digər hallarda isə, yəni, şəxsin yaşı 12-dən böyük olduqda otağa daxil ola bilməyəciyini deyəcəyik. Burada iki şərt yoxladığımız üçün **else if** hissəsinə mütləq ehtiyacımız var. Aşağıdakı nümunəyə baxaq.

```
<script>

var x;

x=prompt("Yasinizi daxil edin");
```

if(x<12) document.write("Oyun otagina xos geldiniz :)");

else if(x==12) document.write("Daxil olmaq ucun size xususi icaze lazimdir.");

else document.write("Otaga yalniz 12 yasli ve yasi 12-den asagi olan usaqlar daxil ola biler.");

</script>

Burada artıq yaşın 12-dən kiçik olduğu halda başqa əməliyyat, 12-ə bərabər olduğu halda başqa əməliyyat, 12-dən böyük olduğu halda isə başqa əməliyyatlar yerinə yetirmiş oluruq.

Burada əvvəlcə birinci şərtə baxılır, yəni yaşın 12-dən kiçik olmasına. Əgər şərt ödənərsə, əməliyyat yerinə yetirilir və bundan sonra **else** və **else if** hissəsinə baxılmır. Ancaq, birinci şərt ödənməzsə yaşın 12-ə bərabər olması yoxlanılır, yəni **else if** hissəsinə baxılır. Burada şərt ödənərsə əməliyyatlar yerinə yetirilir və bundan sonrakı **else** hissəsinə baxılmır. Ancaq, ikinci şərt də ödənməsə, artıq **else** hissəsindəki əməliyyatlar yerinə yetirilir. Çünki yuxarıdakı şərtlərin heç bir ödənmirdi.

Başqa bir nümunə də yazaraq mövzunu daha yaxşı mənimsəməyə çalışaq. Bir pəncərədə istifadəçidən otaqdakı temperaturun neçə dərəcə olduğunu soruşaq. Əgər, temperatur 10 dərəcə olarsa, o zaman, havanın soyuq olduğunu deyək. Əgər temperatur 20 dərəcə olarsa havanın isti olduğunu deyək. Əgər bu şərtlərin heç biri ödənməzsə, o zaman, proqnoz verə bilmədiyimizi bildirək. İndi isə kodumuzu yazaq.

```
<script>

var x;

t=prompt("Temperaturu daxil edin:");

if(t==10) document.write("Hava
soyuqdur.");

else if(t==20) document.write("Hava
istidir.");

else document.write("Proqnoz verile
bilmedi.");

</script>
```

Beləliklə **else if** hissəsinə aid daha bir nümunə yazmış olduq.

Şərt daxilindəki əməliyyatları fiqurlu mötərizə daxilində də yazmağımız mümkündür. Aşağıdakı nümunəyə baxaq.

```
<script>

var x;

t=prompt("Temperaturu daxil edin:");

if(t==10){

document.write("Hava soyuqdur.");

}

else if(t==20){

document.write("Hava istidir.");

}

else{

document.write("Proqnoz verile bilmedi.");

}

</script>
```

Gördüyümüz kimi, şərti yazdıqdan sonra fiqurlu mötərizə açırıq, əməliyyatı yerinə yetiririk, sonda isə fiqurlu mötərizəni bağlayırıq. Fiqurlu

mötərizəni bağladıqdan sonra **else if** hissəsinə keçirik və bu hissədə də eyni şəkildə yazırıq. Sonda **else** hissəsində isə **else** sözündən sonra birbaşa fiqurlu mötərizə açırıq və əməliyyat-lardan sonra bağlayırıq. Qeyd edək ki, şərt ödənərsə yerinə yetiriləcək əməliyyatların sayı 1-dən çoxdursa, həmin əməliyyatları fiqurlu mötərizə daxilində yazmağımız mütləqdir. Aşağıdakı nümunəyə baxaq.

```
<script>

var x;

t=prompt("Temperaturu daxil edin:");

if(t==10){

document.write("Hava soyuqdur.");

document.write("<br>Cunki temperatur
10 derecedir.");

}

</script>
```

Gördüyümüz kimi, burada şərt ödənərsə iki fərqli yazı yazılacaq. Əgər şərt ödənməzsə, heç bir yazı yazılmayacaq, çünki fiqurlu mötərizələr arasındakı əməliyyatlar şərtin ödəndiyi hala

aiddir. Yəni, hər iki əməliyyat şərt ödənərsə baş verəcək.

Funksiyalar

Biz əvvəlki mövzularda bir sıra funksiyalardan istifadə etmişdik. Məsələn, **Math.sqrt** funksiyasını xatırlayaq. Bu funksiyaya bir parametr verirdik və onun kök altısını tapırdı.

```
<script>

var x=Math.sqrt(9);

document.write(x);

</script>
```

Javascriptdəki bu tip mövcud funksiyalardan əlavə biz özümüz də yeni funksiyalar yarada bilərik. Məsələn, elə funksiya yarada bilərik ki, sadəcə bir ədədi, yazını çap etsin, elə bir yeni funksiya yarada bilərik ki, funksiya daxil etdiyimiz parametrin kubunu tapıb 4-ə bölsün. Yəni, istədiyimiz şəkildə yeni funksiyalar yarada bilərik.

Qeyd edək ki, funksiyalar parametrli və parametrsiz ola bilər. Biz çətinlik yaratmamaq üçün ilk öncə sadəcə parametrsiz funksiyalara baxacağıq. İlk funksiyamızı aşağıdakı kimi yaradaq:

<script>

function funksiya(){

}

</script>

Beləliklə funksiyamızı yaratmış oluruq. Funksiyanı yaratmaq üçün **function** sözündən istifadə edirik. İlk olaraq **function** yazırıq və məsafə qoyuruq. Bundan sonra isə funksiyanın adını yazırıq. Burada funksiyamızın adını elə **funksiya** olaraq qeyd etmişik. Funksiyaya istədiyimiz adı qoya bilərik. Sadəcə, adda yalnız hərflərdən, rəqəmlərdən və «_» simvolundan istifadə edə bilərik. Eyni zamanda, funksiyanın adı rəqəmlə başlaya bilməz. Funksiyanın adını yazdıqdan sonra onun yanında bir ədəd mötərizə açırıq və bağlayırıq. Bu mötərizəni niyə yazdığımızı gələcək mövzularda izah edəcəyik. Bundan sonra isə fiqurlu mötərizələri açırıq və bağlayırıq. Funksiyamızın əməliyyatları bu fiqurlu mötərizələrin arasında olacaqdır. Yuxarıdakı kodu yadda saxlayıb işlətdikdə səhifədə heçnə baş vermədiyini görəcəyik. Yazdığımız kodda heç bir səhv yoxdur, sadəcə, funksiyanı elan etsək də funksiyanı işlətməmişik. Funksiyanı işlətmək üçün

funksiyanı çağırmalıyıq. Burada funksiyanı çağırsaq belə heçnə baş verməyəcək çünki funksiyanın daxilində heç bir əməliyyat yazmamışıq. Gəlin funksiyamızın daxilində sadəcə bir əməliyyat yazaq və funksiyanı çağıraraq onu işə salaq. Aşağıdakı koda baxaq:

```
<script>

function funksiya(){

alert("Ilk funksiyamizi ise saldiq.");

}

document.write("Asagida funksiyamizi cagiririq. <br>");

funksiya();

</script>
```

Gördüyümüz kimi, burada funksiyamızın daxilində, fiqurlu mötərizələrin arasında bir əməliyyat yazdıq ki, bu əməliyyatımız da sadəcə bir xəbərdarlıq mesajı idi. Aşağıda isə **funksiya();** yazaraq funksiyamızı çağırdıq və işə saldıq. Gördüyümüz kimi, funksiyanı çağırmaq və onu işə salmaq çox sadədir. Beləliklə, burada ilk funksiyamızı yaratdıq və onu çağıraraq işə saldıq.

Burada haqlı olaraq bir sual verə bilərsiniz.

Funksiyadan istifadə etmədən də biz onsuzda ekrana xəbərdarlıq mesajı çıxara bilirik. Bəs nəyə görə burada funksiyadan istifadə etdik?

Bu sualı verməyiniz tamamilə təbiidir. Bu suala cavab olaraq deyək ki, proqramlaşdırmada funksiyalar işimizi çox rahatlaşdırmaqdadır. Eyni əməliyyatları bir funksiya daxilində yazaraq yüzlərlə yerdə işlədə bilərik, bununla da daha az kod yazaraq daha çox iş görə bilirik. Ümumilikdə deyək ki, funksiyalar proqramlaşdırmadakı işimizi çox rahatlaşdırmaqdadır. Funksiyakardan istifadə etdikcə bunu daha yaxşı anlayacağınıza əminəm.

Kənardan fayl çağırma

Yazacağımız Javascript kodları çox mürəkkəb olduqda həmin kodların bir hissəsini başqa faylda yaza və sonra da həmin faylı əsas səhifəmizə əlavə edə bilərik. Aşağıdakı kodu yeni bir boş fayla yazaq:

```
function cap(){

alert("OK");

}
```

Kodu yazdıqdan sonra 1.js adı ilə yadda saxlayaq. Burada diqqət edək ki, fayl .js sonluğu ilə bitdiyi üçün, yəni javascript faylı olduğu üçün kodları <script> və </script>etiketləri arasında yazmırıq.

Növbəti olaraq aşağıdakı kodu başqa bir faylda yazaq:

```
<script src="1.js"></script>

<script>

cap();

</script>
```

Bu faylı isə 1.html olaraq yadda saxlayaq. Yadda saxladığımız faylların hər ikisinin eyni qovluqda olduğuna diqqət edək, məsələn, hər ikisinin masaüstündə. Yadda saxladığımız 1.html faylını brauzerdə açdıqda 1.js səhifəsində yaratdığımız cap() funksiyasının işə düşdüyünü görəcəyik. Funksiya ayrı bir faylda yazılsa da biz həmin faylı səhifəyə əlavə etdiyimiz üçün həmin fayldakı kodları da istifadə edə bilirik. Həmin faylı html səhifəmizə çağırmaq üçün aşağıdakı koddan istifadə etdik:

<script src="1.js"></script>

Burada src atributuna həmin faylın adını yazaraq onu html səhifəmizə çağırmış olduq. Beləliklə oradakı kodları html səhifəmizdə işlədə bilərik.

Əvvəldə də dediyimiz kimi, kodlar mürəkkəb olduqda, layihəmizin kodlarını bir yox, bir neçə faylda yazmağa ehtiyac duya bilirik. Buna görə də bu üsuldan istifadə etməyimiz lazım gəlir.

Proqramlaşdırmada hadisə

C/C++, Pascal kimi dilləri öyrənərkən başlanğıcda əməliyyatlar yalnız klaviatura vasitəsilə həyata keçirilir. Məsələn, 1 düyməsinə basanda məlumat çıxır, 2 düyməsinə basanda proqramdan çıxır və s. Ancaq, sonradan C++ Builder, Qt kimi mühitlər vasitəsilə görünüşü daha yaxşı olan proqramlar hazırlanır. Həmin proqramlarda «event» yəni «hadisə» anlayışından istifadə olunur və hadisələr siyahı şəklində təqdim olunur. Məsələn, düyməyə basılması hadisə hesab olunur, düyməyə iki dəfə ardıcıl basılması hadisə hesab olunur, klaviaturanın hər-hansı bir düyməsinə basılması, mausun hərəkət etdirilməsi və s. hadisə sayılır. Bu hadisələrin əsasında müəyyən əməliyyatlar yerinə yetirilir. Məsələn, hər-hansı bir düyməyə basdıqda qeydiyyat forması açılır. Qeydiyyat formasının açılması düyməyə basılma hadisəsinin əsasında baş verir. Bir çox əməliyyatlar da hadisələr əsasında baş verir.

Eyni hal Javascript üçün də keçməkdədir. Bizim indiyə qədər yazdığımız bütün əməliyyatlar səhifə açılarkən yerinə yetirilirdi. Ancaq, saytlarda əməliyyatlar çox hallarda hər-hansı hadisələr əsasında baş verir. Məsələn,

«Qeydiyyatdan keç» düyməsinə basıldıqdan sonra qeydiyyat əməliyyatı aparılır. Bu o deməkdir ki, biz qeydiyyatdan keçmək üçün lazım olan əməliyyatları «Qeydiyyatdan keç» düyməsinin klik olunma hadisəsi əsasında yerinə yetirməliyik.

Oyunlarda əsas xarakter klaviaturadakı bir düyməyə basdıqdan sonra hərəkət edir. O zaman biz hərəkət üçün lazım olan əməliyyatları klaviaturanın düyməsinə klik olunma hadisəsinin əsasında yazmalıyıq.

Digər mühitlərdə olduğu kimi Javascriptdə də vəziyyət eynidir. Bir çox əməliyyatlar hər-hansı bir hadisənin baş verməsi əsasında aparılır.

Javascriptdə hadisə

Yuxarıda da dediyimiz kimi, səhifə daxilində bir çox əməliyyat hadisələr əsasında baş verir. Məsələn, biz istəyirik ki, düyməyə basıldıqda ekrana xəbərdarlıq qutusu çıxsın. Bunun üçün Javascriptdə biz **onclick** adlanan hadisədən istifadə etməliyik. Hadisə zamanı işə düşəcək kodları **<script>** və **</script>** etiketləri arasında yazmaya da bilərik. Bu kodları sadəcə html elementinin içərisində yaza bilərik. Aşağıdakı nümunəyə baxaq:

```
<button onclick="alert('Ok');">Daxil ol</button>
```

Heç bir əlavə kod yazmadan Html səhifəmizdə sadəcə olaraq bu kodu yazaq və yadda saxlayaq. Gördüyümüz kimi, burada javascript kodunu yazmaq üçün əlavə **<script>** etiketinə ehtiyac duymuruq, sadəcə element daxilində **onclick=** yazırıq və dırnaq arasında javascript kodlarını yazırıq. Bu o deməkdir ki, bu javascript kodları sadəcə düyməyə klik olunduqda işə düşəcək.

Burada **onclick** hadisəsi daxilində bir neçə əməliyyat da yaza bilərik. Aşağıdakı nümunəyə baxaq:

**<button onclick="document.write('Yazi yazilir
'); alert('Ok');">Daxil ol</button>**

Burada isə düyməyə klik olunduqda həm səhifədə yazı yazılacaq, həm də, xəbərdarlıq qutusu görünəcək. Ancaq, burada bir məsələyə mütləq toxunmalıyıq. Gördüyümüz kimi, **onclick** hadisəsi daxilində bir neçə əməliyyat yazdıqda yazdığımız kod qəlizləşir. Elə ola bilər ki, bu hadisənin daxilində 100 sətir kod yazmağa ehtiyac duyaq. Bəs bu zaman nə edəcəyik? Bu zaman edəcəyimiz şey, hadisə üçün yazılacaq kodları element daxilinə yazmaq yerinə bir funksiyaya yazmaq, element daxilində isə sadəcə həmin funksiyanı çağırmaq olacaq. Aşağıdakı nümunəyə baxaq:

<button onclick="funksiya();">Daxil ol</button>

<script>

function funksiya(){

**document.write("Ok
");**

alert("Ok");

document.write("Ok2");

}

</script>

Gördüyümüz kimi burada bir funksiya elan edirik və düyməyə klik olunan zaman baş verəcək əməliyyatları həmin funksiyanın daxilinə yazırıq. Düymənin **onclick** hadisəsində isə həmin funksiyanı çağırırıq. Beləliklə işimiz sadələşir. Artıq düyməmizdə heç bir dəyişiklik etmədən funksiyamızın daxilində istədiyimiz qədər kod yazıb işlədə bilərik. Beləliklə, funksiya yaratmağın vacibliyini bu mövzuda daha yaxşı anlamış oluruq.

Onclick funksiyasında bənzəyən **ondblclick** funksiyasına da baxaq. Bu funksiya düyməyə və ya bir elementə iki dəfə ardıcıl basıldıqda müəyyən əməliyyatları yerinə yetirmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

<button ondblclick="funksiya();">Daxil ol</button>

<script>

function funksiya(){

alert("Duymeye ardicil iki defe basildi.");

}

</script>

Burada düyməyə iki dəfə ardıcıl tez-tez klik etdikdə, xəbərdarlıq qutusunun çıxdığını görəcəyik.

Yuxarıda baxdığımız 2 hadisədən əlavə javascript daxilində bir çox hadisələr mövcuddur. Ancaq, hələki bu iki hadisə ilə kifayətlənək və bir sıra mövzuları da öyrənək. Gələcək mövzularımızda isə həmin hadisələrə ayrı-ayrılıqda baxacağıq.

Lotereyaçı müştərimizin çətinliyi

Lotereyaçı müştərimiz daha əvvəl yaratdığımız biletdən razıdır. Ancaq, istəyir ki, bir düymə olsun və həmin düyməyə klik olunduqda bilet yaransın. Bunun üçün sadəcə bir düymə yaradıb əvvəl etdiyimiz əməliyyatı düyməynin onclick hadisəsinə aid funksiyaya yazmağımız kifayətdir.

```
<button onclick="bilet();">Bilet Yarat</button>

<script>

function bilet(){

var a,b,c,d,e,f;

a=Math.floor(Math.random()*100);

b=Math.floor(Math.random()*100);

c=Math.floor(Math.random()*100);

d=Math.floor(Math.random()*100);

e=Math.floor(Math.random()*100);

f=Math.floor(Math.random()*100);
```

```
document.write("<table border=1
width=500>");

document.write("<tr>");

document.write("<td>" + a + "</td>" +
"<td>" + b + "</td>" + "<td>" + c +
"</td>");

document.write("</tr>");

document.write("<tr>");

document.write("<td>" + d + "</td>" +
"<td>" + e + "</td>" + "<td>" + f +
"</td>");

document.write("</tr>");

document.write("</table>");

}

</script>
```

Riyaziyyatçı müştərimiz çətinliyi

Riyaziyyatçı müştərimiz daha əvvəl onun üçün hazırladığımız proqramda çətinlik çəkir və onun üçün daha rahat bir interfeys yaratmağımızı xahiş edir. İstəyir ki, Hər əməliyyata aid bir düymə olsun və o, həmin düyməyə klik etdikdə mətn sahəsi çıxsın. Mətn sahəsinə yazacağı ədəd üzərində isə hesablama aparılsın. Biz də bu istəyi koda çeviririk.

```
<button onclick="kok_alti();">Kok alti</button>

<button onclick="kvadrat();">Kvadrat</button>

<button onclick="natural();">Natural loqarifma</button>

<script>

function kok_alti(){

var x=prompt("Ededi daxil edin:");

alert(Math.sqrt(x));

}

function kvadrat(){
```

```
var x=prompt("Ededi daxil edin:");
alert(Math.pow(x,2));
}
function natural(){
var x=prompt("Ededi daxil edin:");
alert(Math.log(x));
}
</script>
```

Beləliklə üç düymə yaradırıq və hər düymənin klik hadisəsinə bir funksiya ötürürük. Beləliklə, düymələrə basıldıqda qutu ekrana çıxır və ora hesablama aparmaq istədiyimiz ədədi yazırıq. Sonda isə uyğun əməliyyat yerinə yetirilir.

Tərcüməçi müştərimizin çətinliyi

Tərcüməçi müştərimiz də hər dəfə yeni söz yazarkən səhifəni yeniləmək istəmir və ona görə də, bizdən xahiş edirik bir düymə yaradaq və həmin düyməyə klik olunduqda tərcümə üçün pəncərə açılsın. Bunun üçün sadəcə tərcümə əməliyyatını bir düymənin onclick hadisəsinə ötürülən funksiyanın içərisinə yazmağımız kifayətdir.

```
<button onclick="tercume();">Tərcümə et</button>

<script>

function tercume(){

var soz;

soz=prompt("Sozu daxil edin:");

if(soz=="apple"){

alert("Alma");

}

if(soz=="orange"){

alert("Portagal");
```

```
}
if(soz=="smart"){
alert("Ağıllı");
}
}
</script>
```

Beləliklə tərcüməçi dostumuz artıq ikinci dəfə söz tərcümə edərkən səhifəni yeniləməyə ehtiyac duymayacaq.

Document obyektinə giriş

Javascriptin olmazsa olmazla-rından biri də **document** obyektidir. Javascriptdə element-lərlə işləmək, onlarda dəyişiklik aparmaq, yeni elementlər yaratmaq, elementləri silmək və bir sıra əməliyyatlar aparmaq üçün **document** obyekti mövcuddur. Bu obyekt altında bir çox funksiya və xüsusiyyətlər mövcuddur. Aşağıdakı nümunəyə baxaq:

```
<div id="yazi"></div>

<script>

document.getElementById("yazi").innerHTML="Ikinci yazi";

</script>
```

Burada artıq document obyektindən istifadə etmiş oluruq. Burada **getElementById()** funksiya, **innerHTML** isə xüsusiyyətdir. Deməli burada, **getElementById** funksiyası vasitəsilə «id» dəyəri «yazi» olan elementi seçmiş oluruq. Növbəti olaraq seçilmiş elementin **innerHTML** xüsusiyyətini bir yazıya bərabərləşdiririk. Beləliklə, elementin içərisinə həmin yazı yazılmış olur.

Burada method və xüsusiyyətlərin yazılışına diqqət etməyimiz mütləqdir. Məsələn, **getElementById** funksiyasının adını yazarkən, ilk söz istisna olmaqla digər sözlərin baş hərflərini böyüklə yazmalıyıq. Digər **innerHTML** xüsusiyyətində isə, gördüyümüz kimi «HTML» sözü böyüklə yazılmalıdır.

Elementin daxilindəki yazını bir dəyişənə ötürüb onu ekrana çıxarmağımız da mümkündür. Aşağıdakı nümunəyə baxaq:

```
<div id="yazi">This is Javascript!</div>

<script>

var
x=document.getElementById("yazi").inner
HTML;

alert(x);

</script>
```

Nəticədə element daxilindəki yazını xəbərdarlıq qutusu ilə ekrana çıxarmış oluruq.

Bu əməliyyatları hadisələrin daxilin- də yazmağımız mümkündür. Aşağı-dakı nümunəyə baxaq:

** This is Css! **

<button type="button"
onclick="funksiya()"
id="duyme">Change</button>

<script>

function funksiya(){

document.getElementById("yazi").innerHT
ML="This is Javascript!";

}

</script>

Beləliklə, düyməyə basdıqda div elementinin daxilindəki yazı dəyişmiş olacaq.

Kassir müştərimizin interfeysi

Kassir müştərimiz bu dəfə hesablanacaq ədədlərin input elementinə yazılmasını istəyir. Biz də bunun üçün iki ədəd input elementi yaradıb onların içərisindəki yazıları dəyişənlərə köçürəcəyik. Ondan sonra onları toplayıb ekrana çıxaracağıq.

```
<input type="text" id="eded1">

<input type="text" id="eded2">

<button
onclick="hesabla();">Hesabla</button>

<script>

function hesabla(){

var x,y;

x=document.getElementById("eded1").val
ue*1;

y=document.getElementById("eded2").val
ue*1;

alert(x+y);

}
```

</script>

Beləliklə input elementlərindən ədədlər götürülərək dəyişənlərə köçürülür. Sonda isə onlar toplanaraq ekrana çıxarılır. Ədədlərin götürülərkən niyə 1-ə vurulduğunu gələcək mövzularda izah edəcəyik.

Riyaziyyatçı müştərimizin interfeysi

Riyaziyyatçı müştərimiz bir ədəd mətn sahəsi və üç ədəd düymə yaratmağımızı istəyir. Birinci düyməyə basıldıqda kök altı, ikinci düyməyə basıldıqda kvadrat, üçüncü düyməyə basıldıqda natural loqarifmanın hesablanmasını istəyir. Biz burada sadəcə 3 ədəd funksiya yaradıb hər funksiyanı ona uyğun düymənin onclick hadisəsində çağıracağıq. Kod baxaq:

```
<input type="text" id="eded">

<button onclick="kok_alti();">Kok alti</button>

<button onclick="kvadrat();">Kvadrat</button>

<button onclick="natural();">Natural loqarifma</button>

<script>

function kok_alti(){

var x=document.getElementById("eded").value;

alert(Math.sqrt(x));
```

```
}

function kvadrat(){

var
x=document.getElementById("eded").valu
e;

alert(Math.pow(x,2));

}

function natural(){

var
x=document.getElementById("eded").valu
e;

alert(Math.log(x,2));

}

</script>
```

Beləliklə hər düyməyə basıldıqda ona uyğun əməliyyat yerinə yetirilir.

Lotereyaçı müştərimizin interfeysi

Lotereyaçı müştərimiz bu dəfə istəyir ki, lotereya biletinin yuxarısında bir düymə olsun və o düyməyə basdıqda bilet yenilənsin. Bunun üçün sadəcə funksiyanı düymənin onclick hadisəsində çağıracağıq. Funksiya daxilində cədvəl və içərisindəki təsadüfi ədədlər bilet adlı dəyişənə əlavə olunur. «Bilet+=» yazaraq yeni hissənin köhnə hissənin üzərinə gəlməsini təmin edirik. Ən sonda isə bilet yazısını id dəyəri «bilet» olan div elementinin içərisinə köçürürük.

```
<button onclick="bilet();">Bilet Yarat</button>

<div id="bilet"></div>

<script>

function bilet(){

var a,b,c,d,e,f;

var bilet="";

a=Math.floor(Math.random()*100);

b=Math.floor(Math.random()*100);

c=Math.floor(Math.random()*100);
```

```
d=Math.floor(Math.random()*100);

e=Math.floor(Math.random()*100);

f=Math.floor(Math.random()*100);

bilet+="<table border=1 width=500>";

bilet+="<tr>";

bilet+="<td>" + a + "</td>" + "<td>" + b
+ "</td>" + "<td>" + c + "</td>";

bilet+="</tr>";

bilet+="<tr>";

bilet+="<td>" + d + "</td>" + "<td>" + e
+ "</td>" + "<td>" + f + "</td>";

bilet+="</tr>";

bilet+="</table>";

document.getElementById("bilet").innerH
TML=bilet;

}

</script>
```

Tərcüməçi müştərimizin interfeysi

Tərcüməçi olan müştərimiz bu dəfə bizdən daha güclü bir interfeys istəməkdədir. Müştərimiz istəyir ki, iki ədəd yazı sahəsi olsun. Birinci yazı sahəsinə sözü yazdıqda tərcüməsi avtomatik olaraq ikincidə görünsün. Bunun üçün ilk olaraq Mətn sahəsinə, yəni yaradılmış inputa yazılmış yazının götürülmə qaydasına baxaq.

```
<input type="text" id="yazi" class="form-control">

<button onclick="funksiya();">Goster</button>

<script>

function funksiya(){

var x=document.getElementById("yazi").value;

alert(x);

}

</script>
```

Burada düyməyə basdıqda input elementinə yazılmış yazı xəbərdarlıq qutusunda görünəcək.

Funksiya daxlində ilk olaraq id dəyəri «yazi» olan input elementini seçirik. Ondan sonra isə «value» xüsusiyyəti vasitəsilə onun içərisinə yazılmış yazını götürürük və x dəyişəninə köçürürük. İnput elementinin içərisindəki yazı «value» xüsusiyyəti vasitəsilə götürülməkdədir. Beləliklə input elementinə yazılmış yazı x dəyişəninə köçürülmüş olur. Bundan sonra sadəcə x dəyişənini ekrana çıxarmış oluruq.

İndi isə «oninput» hadisəsinə baxaq. Bu hadisə input elementinə yeni bir simvol daxil etdikdə müəyyən əməliyyatları yerinə yetirmək üçün istifadə olunur. Nümunəyə baxaq:

```
<input type="text" id="soz"
oninput="funksiya();">

<script>

function funksiya(){

var
soz=document.getElementById("soz").valu
e;

alert(soz);

}

</script>
```

Beləliklə mətn sahəsinə hər dəfə yeni simvol əlavə olunduqda funksiya çağırılacaq və əməliyyatlar işə düşəcək.

Lazım olan hissələri öyrəndikdən sonra tərcümə sistemimizi hazırlaya bilərik. Bunun üçün biz ikinci bir input elementi də yaradaq və elə edək ki, birinci elementə sözü yazdıqda onun tərcüməsi ikinci elementin içərisində görünsün. Bunun üçün funksiya daxilində birinci elementə yazılmış sözü götürürük və daha əvvəl də işlətdiyimiz üsulla onun tərcüməsini tapırıq. Ondan sonra sözün tərcüməsini ikinci input elementinə köçürürük. Tərcümənin avtomatik olması üçün funksiyanı birinci input elementinin «oninput» hadisəsində çağıracağıq. Beləliklə hər dəfə inputa yeni simvol daxil edildikdə tərcüməsi yoxlanılacaq. Əgər, söz varsa tərcüməsi yazılacaq. İndi isə koda baxaq.

```
<input type="text" id="soz"
oninput="funksiya();">

<input type="text" id="tercumesi">

<script>

function funksiya(){

var
soz=document.getElementById("soz").value;

if(soz=="apple"){
```

```
document.getElementById("tercumesi").value
="alma";

}

if(soz=="orange"){

document.getElementById("tercumesi").value
="portagal";

}

if(soz=="car"){

document.getElementById("tercumesi").value
="masin";

}

}

</script>
```

Beləliklə birinci input elementinə sözü yazdıqda ikinci elementdə avtomatik olaraq tərcüməsini görəcəyik.

Document obyekti ilə css xüsusiyyətləri

Document obyekti vasitəsilə elementin css xüsusiyyətlərinin dəyiş-dirilməsi mümkündür. Məsələn, biz elə edə bilərik ki, düyməyə basdıqda yazının rəngi dəyişilsin. Bu obyekt vasitəsilə css xüsusiyyətlərinin dəyişdirilməsi üçün aşağıdakı sintaksisdən istifadə olunur:

```
document.getElementById("id").style.xususiyyet="yeni deyer";
```

Bu şəkildə elementin istədiyimiz css xüsusiyyətini dəyişmiş oluruq. Aşağıdakı nümunəyə baxaq:

```
<div id="yazi">This is Css!</div>

<button type="button"
onclick="funksiya()"
id="duyme">Change</button>

<script>

function funksiya(){

document.getElementById("yazi").style.color="red";

}
```

</script>

Beləliklə, düyməyə basdıqda yazının rəngi qırmızı olmuş olur. Bu şəkildə elementin istənilən css xüsusiyyətini dəyişə bilərik.

Yazdığımız **style** özü də bir obyektdir və içərisində bir çox xüsusiyyətləri daşımaqdadır. Onlardan bəzilərinə baxaq:

1) **background** xüsusiyyəti — elementinin fonunu dəyişmək üçündür. Aşağıdakı nümunəyə baxaq:

```
<div id="yazi">This is Css!</div>

<button type="button"
onclick="funksiya()"
id="duyme">Change</button>

<script>

function funksiya(){

document.getElementById("yazi").style.background="red";

}

</script>
```

Beləliklə, elementin fonunu qırmızı etmiş oluruq.

2) **backgroundColor —** Yalnızca fon rəngini dəyişmək üçündür. Nümunə:

** This is Css! **

<script>

document.getElementById("yazi").style.backgroundColor="red";

</script>

Beləliklə elemenin fon rəngi dəyişmiş olur.

3) **backgroundImage —** Fon şəklini təyin etmək üçün istifadə olunur.

4) **border** — Çərçivəni təyin etmək üçün istifadə olunur. Nümunə:

** Javascript **

<script>

document.getElementById("yazi").style.border="5px solid black";

</script>

Nəticədə qara rəngdə, 5px ölçüsündə və solid növündə çərçivə yaranmış olur. Çərçivə növləri

css-də mövcuddur. Solid növündən əlavə çərçivənin aşağıdakı növləri də mövcuddur:

dotted, dashed, double, groove, ridge, inset, outset, none, hidden

Sadəcə **solid** sözünün yerinə bu sözlərdən birini yazaraq daha fərqli çərçivələr əldə edə bilərsiniz.

5) **borderColor** — Çərçivənin rəngini təyin etmək üçün istifadə olunur. Nümunə:

```
<div id="yazi" style="border: 5px dotted black;">Javascript</div>

<br>

<button onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

document.getElementById("yazi").style.borderColor="red";

}

</script>
```

Beləliklə düyməyə basdıqda div elementinin çərçivəsinin rəngi qırmızı olmuş olur.

6) **borderBottom** — Elementin aşağı çərçivəsini təyin etmək üçün istifadə olunur. Nümunə:

```
<div id="yazi" style="border: 5px dotted black;">Javascript</div>

<br>

<button onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

document.getElementById("yazi").style.borderBottom="10px solid black";

}

</script>
```

Nəticədə düyməyə basdıqda elementin aşağı çərçivəsi dəyişmiş olur.

7) **borderTop** — Elementin yuxarı çərçivəsini təyin etmək üçün istifadə olunur.

** Javascript **

**
**

<button onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

document.getElementById("yazi").style.borderTop="10px solid black";

}

</script>

Nəticədə düyməyə basdıqda div elementinin yuxarı çərçivəsi dəyişmiş olur.

8) **borderLeft** — Elementin sol çərçivəsini təyin etmək üçün istifadə olunur.

** Javascript **

**
**

<button onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

**document.getElementById("yazi").style.bo
rderLeft="10px solid black";**

}

</script>

Nəticədə düyməyə basdıqda div elementinin sol çərçivəsi dəyişmiş olur.

9) **borderRight** — Elementin sağ çərçivəsini təyin etmək üçün istifadə olunuur.

**<div id="yazi" style="border: 5px dotted
black;">Javascript **

**
**

**<button
onclick="funksiya();">Deyis</button>**

<script>

function funksiya(){

**document.getElementById("yazi").style.bo
rderRight="10px solid black";**

}

</script>

Nəticədə düyməyə basdıqda div elementinin sağ çərçivəsi dəyişmiş olur.

10) **borderLeftColor** — Elementin sol çərçivəsinin rəngini təyin etmək üçün isifadə olunur.

** Javascript **

**
**

<button onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

document.getElementById("yazi").style.borderLeftColor="red";

}

</script>

Nəticədə düyməyə basdıqda elementin sol çərçivəsinin rəngi dəyişmiş olur.

11) **borderRightColor** — Elementin sağ çərçivəsinin rəngini təyin etmək üçün istifadə etmək üçün istifadə olunur. Nümunə:

** Javascript **

**
**

<button onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

document.getElementById("yazi").style.borderRightColor="red";

}

</script>

Nəticədə düyməyə basdıqda elementin sağ çərçivəsinin rəngi dəyişmiş olur.

12) **borderTopColor** — Elementin yuxarı çərçivəsinin rəngini təyin etmək üçün istifadə olunur. Nümunə:

** Javascript **

**
**

<button onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

document.getElementById("yazi").style.borderTopColor="red";

}

</script>

Nəticədə düyməyə basdıqda elementin yuxarı çərçivəsinin rəngi dəyişmiş olur.

13) borderBottomColor — Elementin aşağı çərçivəsnin rəngini təyin etmək üçün istifadə olunur.

** Javascript **

**
**

<button onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

document.getElementById("yazi").style.borderBottomColor="red";

}

</script>

Nəticədə düyməyə basdıqda elementin aşağı çərçivəsinin rəngi dəyişmiş olur.

14) fontFamily — Elementin daxilindəki yazıların fontunu təyin etmək üçün istifadə olunur.
Nümunə:

** Javascript **

**
**

<button onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

document.getElementById("yazi").style.fontFamily="Arial";

}

</script>

Nəticədə düyməyə basdıqda yazının fontu dəyişmiş olur.

15) fontSize — Elementin daxilindəki yazılarını ölçüsünü təyin etmək üçün istifadə olunur. Nümunə:

** Javascript **

**
**

<button onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

document.getElementById("yazi").style.fontSize="20px";

}

</script>

Nəticədə yazının ölçüsü 20px olmuş olur.

16) **left** — Elementin soldan olan məsafəsini təyin etmək üçün istifadə olunur. Nümunə:

** Javascript **

**
**

<button onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

document.getElementById("yazi").style.left="200px";

}

</script>

Nəticədə elementin soldan məsafəsini dəyişmiş oluruq.

17) **lineHeight** — Mətndə sətirlər arası məsafəni təyin etmək üçün istifadə olunur. Nümunə:

** Javascript**

**
**

Ikinci setir

** **

**
**

<button onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

document.getElementById("yazi").style.lineHeight="3";

}

</script>

Nəticədə düyməyə basdıqda sətirlər arası məsafə dəyişmiş olur.

18) **margin** — Elementin kənardan olan məsafəsini müəyyən etmək üçün istifadə olunur. Nümunə:

** Javascript **

**
**

<button onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

document.getElementById("yazi").style.margin = "50px 10px 20px 30px";

}

</script>

Beləliklə düyməyə basdıqda elementin kənardan məsafələri dəyişmiş olacaq. Burada 50px yuxarıdan məsafə, 10px sağdan məsafə, 20px aşağıdan məsafə, 30px isə soldan məsafə olmuş olur. Yalnız 50px yazmış olsaydıq elementin hər tərəfdən məsafəsi 50px olmuş olacaqdı.

19) **marginBottom** — Elementin aşağıdan olan məsafəsini təyin etmək üçün istifadə olunur. Nümunə:

** Javascript **

**
**

<button onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

document.getElementById("yazi").style.marginBottom = "50px";

}

</script>

Nəticədə düyməyə basdıqda elementin aşağıdan məsafəsi 50px olmuş olur.

20) **marginLeft** — Elementin soldan olan məsafəsini təyin etmək üçün istifadə olunur.

** Javascript **

**
**

<button onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

document.getElementById("yazi").style.marginLeft = "50px";

}

</script>

Nəticədə düyməyə basdıqda elementin soldan məsafəsi 50px olmuş olur.

21) **marginRight** — Elementin sağdan olan məsafəsini təyin etmək üçün istifadə olunur.

```
<div id="yazi" style="border: 5px dotted black;">Javascript</div>

<br>

<button onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

document.getElementById("yazi").style.marginRight = "50px";

}

</script>
```

Nəticədə düyməyə basdıqda elementin sağdan məsafəsi 50px olmuş olur.

22) **marginTop** — Elementin yuxarıdan olan məsafəsini təyin etmək üçün istifadə olunur.

** Javascript **

**
**

<button onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

document.getElementById("yazi").style.marginTop = "50px";

}

</script>

Nəticədə düyməyə basdıqda elementin yuxarıdan məsafəsi 50px olmuş olur.

23) **padding, padddingBottom, paddingTop, paddingRight, paddingLeft** — Bu xüsusiyyətlər elementin daxili məzmununun kənardan məsafəsini təyin edir. İstifadə qaydaları margin xüsusiyyətləri kimidir.

24) **maxHeight, maxWidth, minHeight, minWidth** – Bu xüsusiyyətlər uyğun olaraq elementin maksimum uzunluğunu, maksimum

enini, minimum uzunluğunu, minimum enini təyin edir. Yazılış qaydası aşağıdakı kimidir:

document.getElementById("yazi").style.maxHeight = "1px";

25) **display —** Elementin görüntülənmə formasını təyin etmək üçün istifadə olunur Aşağıdakı nümunəyə baxaq:

** Javascript **

**
**

<button onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

document.getElementById("yazi").style.display = "none";

}

</script>

Nəticədə düyməyə basdıqda elementin display xüsusiyyəti dəyişərək none olmuş olur. Buna görə də element görünmür. Bu xüsusiyyət css-

də mövcuddur və block, inline, inline-block və s. qiymətlər də ala bilməkdədir.

26) **cursor** — elementin üzərinə gələrkən mausun formasını dəyişmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

```
<div id="yazi">Javascript</div>

<script>

document.getElementById("yazi").style.cursor = "pointer";

</script>
```

Beləliklə yazının üzərinə gəldikdə əl işarəsi yaranır. Xüsusiyyət bir çox qiymətlər ala bilər ki, buna ətraflı css dərsliklərində baxa bilərsiniz.

27) **opacity** — elementin şəffaflığını təyin edir. Nümunə:

```
<div id="yazi">Javascript</div>

<script>

document.getElementById("yazi").style.opacity = "0.5";

</script>
```

Bu xüsusiyyət 0 və 1 arası qiymətlər almaqdadır.

28) **width** və **height** xüsusyyətləri — Elementin enini və uzunluğunu təyin edir. Qiymət piksellə verilir.

29) **textAlign** — Elementin daxilindəki yerləşməni təyin edir. Qiymət olaraq center, left, right kimi qiymətlər alır.

30) **visibility** — Elementin görünüşünü müəyyən edir. Məsələn, elementin gizlədilməsi üçün «hidden» qiymətini alır.

Yuxarıda bir sıra lazımlı stle obyektinə məxsus xüsusiyyətləri yazdıq və istifadə qaydasını göstərdik. Burada yazmadığımız bir sıra xüsusiyyətlərin də olduğunu qeyd etməliyik. Ümumi siyahya internetdə axtarış edərək və ya aşağıdakı ünvana daxil olaraq baxa bilərsiniz:

www.w3schools.com/jsref/dom_obj_style.asp

Müştərilərimizin rəng seçimləri

Müştərilərimiz müxtəlif zövqlərdə olduqları üçün səhifənin fon rəngini dəyişmək istəyirlər. Biz də color tipli input elementindən istifadə edərək onlara bu imkanı verə bilərik. Koda baxaq:

```
<body id="body">

<input type="color" id="color"
onchange="color();">

<script>

function color(){

var
color=document.getElementById("color").
value;

document.getElementById("body").style.b
ackgroundColor=color;

}

</script>
```

Nəticədə rəng qutusundan rəngi seçib qutudan çıxdıqdan sonra səhifənin fon rənginin dəyişdiyini görəcəksiniz.

İlk olaraq onu qeyd edək ki, burada «onchange» hadisəsindən istifadə etməkdəyik. Bu hadisə element üzərində dəyişiklik olduqda müəyyən əməliyyatların yerinə yetirilməsi üçün istifadə olunur. Yuxarıdakı nümunədə bunu görə bilərsiniz. Rəngdə dəyişiklik edib rəng pəncərəsini bağladıqdan sonra səhifənin fon rənginin dəyişdiyini görəcəksiniz.

Hadisə əsasında çağırılan funksiyanın içərisində ilk olaraq input elementindəki rəng dəyəri color adlı dəyişənə ötürülür. Sonra isə bu dəyişəndəki qiymət səhifənin body etiketinin fon rəngi olaraq təyin olunur. Buradakı «backgroundColor» xüsusiyyəti səhifənin fon rəngini təyin etmək üçün istifadə olunur.

Javascriptdə form elementləri ilə işləmək

Javascriptdə form elementləri ilə işləmək mümkündür. Bunun üçün elə document obyektinin funksiya və xüsusiyyətlərindən istifadə edə bilərik. Form elementləri ilə işləmək üçün document obyektinin xüsusi funksiya və obyektləri mövcuddur. Məsələn, **value** xüsusiyyəti vasitəsilə bir **input** elementinə yazılmış dəyəri əldə edə bilərik. Aşağıdakı nümunəyə baxaq:

```
<input type="text" id="yazi">

<br>

<button
onclick="funksiya();">Ok</button>

<script>

function funksiya(){

var
x=document.getElementById("yazi").value
;

alert(x);

}

</script>
```

Nəticədə input elementinə bir yazı yazıb ok düyməsinə basdıqda yazdığımız yazının xəbərdarlıq qutusunda göründüyünü görəcəyik. Apardığımız əməliyyatlar sadədir. İlk olaraq **value** xüsusiyyətindən istifadə edərək input elementinə yazılmış yazını götürürük və x dəyişəninə ötürürük. Sonra isə alert funksiyası vasitəsilə onu xəbərdarlıq şəklində istifadəyə bildiririk.

Biz burada istifadəçidən şifrə tələb etdiyimizi fərz edək. Şifrəmiz «javascript» olsun. Əgər input elementinə daxil olunan şifrə düzgün olarsa şifrənin doğru olduğunu deyək. Əgər şifrə yalnış olarsa şifrənin yalnış olduğunu istifadəçiyə bildirək.

```
<input type="text" id="yazi">

<br>

<button
onclick="funksiya();">Ok</button>

<script>

function funksiya(){

var
x=document.getElementById("yazi").value
;
```

```
if(x=="javascript") alert("Parol
dogrudur.");

else alert("Parol yalnisdir.");

}

</script>
```

Nəticədə parol «javascript» olarsa istifadəçiyə parolun doğru olduğu deyiləcək. Əks halda isə parolun yalnış olduğu deyiləcək. Burada böyük və kiçik hərflərin fərqləndiyinə diqqət edək.

Əgər xəbərdarlıq qutusundan istifadə etmək istəmiriksə, elə öyrəndiyimiz css xüsusiyyətlərini burada tətbiq edə bilərik. Aşağıdakı nümunəyə baxaq:

```
<input type="text" id="yazi"
style="border: 2px solid black;">

<br>

<button type="button"
onclick="funksiya();">Ok</button>

<div id="netice"></div>

<script>

function funksiya(){
```

```
var
x=document.getElementById("yazi").value
;

if(x=="javascript"){

document.getElementById("yazi").style.bo
rderColor="green";

}

else{

document.getElementById("yazi").style.bo
rderColor="red";

}

}

</script>
```

Burada input elementinə daxil olunmuş yazını götürüb parolun doğru olub-olmadığını yoxlayırıq. Əgər parol doğru olarsa, o zaman, input elementinin çərçivəsi yaşıl rəngdə olmuş olacaq. Parol doğru olmasa, o zaman, input elementinin çərçivəsi qırmızı rəngdə olmuş olacaq.

Həm form elementləri üçün, həm də digər vəziyyətlərdə tez-tez istifadə olunan

xüsusiyyətlərdən biri də **length** xüsusiyyətidir. Bu xüsusiyyət bizə mətnin uzunluğunu tapmağa imkan verir. Aşağıdakı nümunəyə baxaq:

```
<input type="text" id="yazi">

<br>

<button
onclick="funksiya();">Ok</button>

<script>

function funksiya(){

var
value=document.getElementById("yazi").v
alue;

var uzunluq=value.length;

alert(uzunluq);

}

</script>
```

Nəticədə «Ok» düyməsinə basdıqda yazdığımız yazının neçə simvoldan ibarət olduğu bizə bildirilir. Burada ilk **value** xüsusiyyəti vasitəsilə input elementinə yazılmış yazını götürüb «value» dəyişəninə ötürürük. Sonra isə **length**

xüsusiyyəti vasitəsilə **value.length** yazaraq yazının uzunluğunu «uzunluq» adlı dəyişənə ötürürük və onu ekrana çıxarırıq.

Məsələn, istifadəçidən şifrə tələb etdiyimizi düşünək. Bu şifrəmizin uzunluğu isə 6 simvoldan kiçik olmamalıdır. Bu zaman bu xüsusiyyət bizim köməyimizə çatır. Aşağıdakı nümunəyə baxaq:

```
<input type="text" id="yazi">

<br>

<button
onclick="funksiya();">Ok</button>

<script>

function funksiya(){

var
value=document.getElementById("yazi").v
alue;

var uzunluq=value.length;

if(uzunluq>6) alert("Icaze verilir.");

else alert("Parolun uzunlugu 6 simvoldan
boyuk olmalidir.");
```

}

</script>

Burada şərt operatoru ilə istiufadəçinin yazdığı parolun uzunluğunun 6 simvoldan böyük olub-olmadığını yoxlayırıq. Əgər parol 6 simvoldan kiçik olarsa bunu istifadəçiyə bildiririk. Təhlükəsizlik məqsədilə istifadəçinin parolunun daha güclü olması lazımdır. Bu isə görəcəyimiz ən sadə tədbirlərdəndir.

İndi isə form elementlərindən biri olan «select» elementinə baxaq. Burada da sadəcə «value» xüsusiyyəti vasitəsilə seçilmiş dəyəri götürə bilərik. Aşağıdakı nümunəyə baxaq:

<select id="select"
onchange="funksiya();">

<option>Azerbaycan dili</option>

<option>Ingilis dili</option>

<option>Rus dili</option>

</select>

<script>

function funksiya(){

```
var
value=document.getElementById("select")
.value;

alert(value);

}

</script>
```

Burada funksiyanı elementin **onchange** hadisəsində çağırmışıq. Beləliklə, hər dəfə seçim dəyişdirildikdə seçilən dəyər xəbərdarlıq qutusunda bildirilir.

Əvvəl istifadə etdiyimiz **onclick** hadisəsi düyməyə klik olunduqda funksiyanı çağırırdı. İndi işlətdiyimiz **onchange** hadisəsi isə elementdə dəyişiklik olunduqda funksiyanı çağırır. Buna görə də select elementində dəyişiklik olunduqda, yəni, seçim dəyişdirildikdə funksiya çağırılır.

İndi isə checkbox tipli input elementi ilə işləmək qaydasına baxaq. Elementin seçilib-seçilmədiyini bilmək üçün **checked** xüsusiyyətindən istifadə etməliyik. Aşağıdakı nümunəyə baxaq:

```
<input type="checkbox" id="check"><br>
```

```
<button type="button"
onclick="funksiya();">Tesdiqle</button>

<script>

function funksiya(){

var
x=document.getElementById("check").che
cked;

alert(x);

}

</script>
```

Əgər checkbox tipli element seçilibsə, bu zaman x dəyişəninə «true» qiyməti ötürülür. Əks halda isə x dəyişəninə «false» qiyməti ötürülür. Buna görə də, elementi seçib düyməyə klik etsək, o zaman, «true» qiyməti ekrana çıxacaq. Əks halda isə «false» qiyməti çıxacaq.

Elementdən aldığımız qiyməti şərt operatoru ilə də yoxlaya bilərik. Aşağıdakı nümunəyə baxaq:

```
<input type="checkbox"
id="check"><br><button type="button"
onclick="funksiya();">Tesdiqle</button>
```

<script>

function funksiya(){

var x=document.getElementById("check").checked;

if(x==true) alert("Element secilib.");

if(x==false) alert("Element secilmeyib.");

}

</script>

Nəticədə checkbox elementi seçilmişsə x dəyişəninin qiyməti «true» olmuş olur və elementi seçildiyi bildirilir. Əks halda isə x dəyişəninin qiyməti «false» olur və elementin seçilmədiyi bildirillir.

Eyni qayda radio tipli input elementi üçün də keçməkdədir. Radio tipli input elementinin də seçilib-seçilmədiyini yoxlamaq üçün **checked** xüsusiyyətindən istifadə edirik. Aşağıdakı nümunəyə baxaq:

**<input type="radio" id="radio">
**

<button type="button" onclick="funksiya();">Tesdiqle</button>

```
<script>

function funksiya(){

var
x=document.getElementById("radio").chec
ked;

if(x==true) alert("Element secilib.");

if(x==false) alert("Element secilmeyib.");

}

</script>
```

Beləliklə element seçilərsə elementin seçildiyi istifadəçiyə bildiriləcək.

Hadisələr (2-ci hissə)

Bundan öncə sadə olması üçün az sayda Javascript hadisələrinə baxdıq. Daha əvvəl də dediyimiz kimi, Javascript hadisələri səhifədə müəyyən hadisələr baş verdikdə, məsələn, düyməyə basıldıqda, düymənin üzərinə gəlindikdə, klaviaturada hər-hansı bir düyməyə basıldıqda və s. hallarda müəyyən əməliyyatların aparılması üçündür. Javascript hadisələrinə sıra ilə baxaq.

1) onclick və onchange hadisələri – bu hadisələri daha əvvəl izah etdiyimiz üçün burada izah etməyə ehtiyac duymuruq.

2) onmouseover hadisəsi – mausla müəyyən elementin üzərinə gəlindikdə müəyyən hadisələrin baş verməsi üçündür. Aşağıdakı nümunəyə baxaq:

```
<div style="height: 100px; width: 100px; background-color: red;" onmouseover="funksiya();"></div>

<script>

function funksiya(){

alert("Elementin uzerine geldiniz.");
```

}

</script>

Beləliklə qırmızı ilə rənglənmiş div elementinin üzərinə gəldikdə xəbərdarlıq yazısı görünəcək.

3) onmouseout hadisəsi – mausla elementin üzərinə gəldikdə, sonra elementin üzərindən kənara çəkildikdə bu hadisə işə düşməkdədir. Aşağıdakı nümunəyə baxaq:

** **

<script>

function funksiya(){

alert("Elementin uzerinden kenara cixdiniz.");

}

</script>

Beləliklə əvvəlcə elementin üzərinə gəlib sonra elementin üzərindən kənara çıxdıqda xəbərdarlıq yazısı gəldiyini görəcəyik.

4) onkeydown hadisəsi – element üzərində hər-hansı bir düyməyə basdıqda müəyyən əməliy-yatları yerinə yetirmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

```
<body id="body"
onkeydown="funksiya();">

<script>

function funksiya(){

alert("Her hansi bir duymeye basdiniz.");

}

</script>

</body>
```

Beləliklə səhifədə hər-hansı bir düyməyə basdıqda düyməyə basdığımızı göstərən bir xəbərdarlıq qutusu gələcək.

Bu funksiyanı işlədərkən istifadəçinin hansı düyməyə basdığını da öyrənə bilərik. Bunun üçün funksiyaya bir parametr ötürməliyik. Aşağıdakı nümunəyə baxaq:

```
<body id="body"
onkeydown="funksiya(event);">
```

<script>

function funksiya(event){

alert(event.keyCode);

}

</script>

</body>

Burada **event.keyCode** yazaraq düyməyə basıldıqda həmin düymənin kodunu ekrana çıxarırıq. Hər düyməyə basdıqda onun uyğun kodunun ekrana çıxdığını görəcəksiniz. Beləliklə istənilən düymənin kodunu öyrənə bilərik. Həmin kodu öyrəndikdən sonra həmin düymə ilə əməliyyatlar apara bilərik. Məsələn, səhifədə sağa hərəkət düyməsinə (oyunlarda istifadə olunan hərəkət oxları) basdıqda 39, sola hərəkət düyməsinə basdıqda isə 37 yazıldığını görəcəyik. Çünki sağa hərəkət düyməsinin kodu 39, sola hərəkət düyməsinin kodu isə 37-dir. Kodları öyrəndikdən sonra istifadəçi bu düyməyə basarkən onu müəyyən edib əməliyyatlar apara bilərik. Nümunəyə baxaq:

<body id="body"
onkeydown="funksiya(event);">

```
<script>
function funksiya(event){
if(event.keyCode==39){
alert("Saga hereket etdiniz");
}
else if(event.keyCode==37){
alert("Sola hereket etdiniz");
}
}
</script>
</body>
```

Beləliklə istifadəçi sağa hərəkət etdikdə sağa hərəkət etdiyi, sola hərəkət etdikdə isə sola hərəkət etdiyi ona bildiriləcək. Buna aid başqa bir nümunə də yazaq:

```
<body id="body"
onkeydown="funksiya(event);">
<div style="width: 100px;
height:100px;background-color: red;"
id="div"></div>
```

```
<script>

var x=0;

function funksiya(event){

if(event.keyCode==39){

x=x+5;

document.getElementById("div").style.ma
rginLeft=x;

}

else if(event.keyCode==37){

x=x-5;

document.getElementById("div").style.ma
rginLeft=x;

}

}

</script>

</body>
```

Beləliklə sağa hərəkət düyməsini basdıqda element sağa, sola hərəkət düyməsini basdıqda isə sola hərəkət edəcək. Burada ilk olaraq bir x

dəyişəni elan edirik və ona o qiymətini veririk. Bizim x dəyişənimiz elementin soldan olan məsafəsini bildirir. Əgər istifadəçi sağa hərəkət düyməsini basarsa x=x+5 yazaraq onu 5 vahid artırıq və marginLeft xüsusiyyəti ilə elementin soldan məsafəsini x-ə bərabər edirik. Beləliklə soldan məsafə 5 vahid artmış olur. İstifadəçi hər dəfə sağa hərəkət düyməsinə basdıqda x-in qiyməti 5 vahid artır və elementin soldan məsafəsi x-ə bərabərləşdirilir. Beləliklə elementin soldan məsafəsi artır və hərəkət yaranır. Sola hərəkət düyməsinə basdıqda isə bunun əksi baş verir. O halda x-in qiyməti 5 vahid azalır və element sola doğru hərəkət edir.

5) onkeyup hadisəsi – onkeydown hadisəsindən fərqli olaraq düymə sıxıldıqda deyil, düymə sıxılıb buraxıldıqdan sonra funksiyanı çağırır və müxtəlif əməliyyatları işə salır. Aşağıdakı nümunəyə baxaq:

```
<body id="body"
onkeyup="funksiya(event);">

<script>

function funksiya(event){

alert(event.keyCode);
```

}

</script>

</body>

Burada düyməni basılı saxladıqda heç bir şey olmadığını görəcəyik. Ancaq basdığımız düyməni buraxdıqdan sonra funksiyanın çağırıldığını görəcəyik.

6) onfocus hadisəsi – elementə fokuslandıqda müəyyən əməliyyatların baş verməsi üçündür. Aşağıdakı nümunəyə baxaq:

<input type="text" onfocus="funksiya();" id="input">

<script>

function funksiya(){

document.getElementById("input").style.backgroundColor="red";

}

</script>

Beləliklə input elementinə fokuslandıqda, üzərinə basdıqda fon rəngi qırmızı olacaq.

7) onblur hadisəsi - bu hadisəni onfocus hadisəsinin əksi olaraq başa düşə bilərik. İstifadəçi elementə fokuslandıqdan sonra elementin üzərindən kənara çəkilsə, kənar bir yerə klik etsə o zaman müəyyən hadisələr işə düşəcək. Aşağıdakı nümunəyə baxaq:

```
<input type="text" onfocus="funksiya();" onblur="funksiya2();" id="input">

<script>

function funksiya(){

document.getElementById("input").style.backgroundColor="red";

}

function funksiya2(){

document.getElementById("input").style.backgroundColor="white";

}

</script>
```

Əvvəlcə elementə fokuslanaq. Fokuslandıqda fon rənginin qırmızı olduğunu görəcəyik. Sonra isə kənar bir yerə klik edək. O zaman fon rənginin ağ olduğunu görəcəyik.

8) onload hadisəsi – səhifə yükləndikdən sonra bir sıra əməliyyatların aparılması üçündür. Aşağıdakı nümunəyə baxaq:

```
<html>

<body onload="funksiya();">

<script>

function funksiya(){

alert("Ok");

}

</script>

</body>

</html>
```

Bu funksiyanın istifadəsinə bəzi hallarda ehtiyac duyula bilir.

9) onsubmit hadisəsi – form submit olunduqda müəyyən əməliyyatların aparılması üçündür. Aşağıdakı nümunəyə baxaq:

```
<form onsubmit="funksiya();">

<input type="text">
```

<input type="submit">

</form>

<script>

function funksiya(){

alert("Ok");

}

</script>

Nəticədə form submit olunduqda əməliyyatlar baş verəcək.

10) oninput hadisəsi – input elementinə yeni bir şey daxil olunduqda müəyyən əməliyyatların baş verməsi üçündür. Aşağıdakı nümunəyə baxaq:

<input type="text" oninput="funksiya();" id="input">

** **

<script>

function funksiya(){

var x=document.getElementById("input").value;

document.getElementById("div").innerHTML=x;

}

</script>

Beləliklə input elementinə yazılar yazdıqca həmin yazının aşağıda göründüyünü görəcəyik.

Yuxarıda istifadə olunacaq hadisələrdən bir neçənə baxdıq. Lazım olduqda bu hadisələri sadə axtarışla axtarış sistemləri vasitəsilə tapa bilərsiniz. Aşağıdakı ünvanda isə digər hadisələrə də baxa bilərik:

https://www.w3schools.com/jsref/dom_obj_event.asp

Biz adətən hadisələri aşağıdakı şəkildə yazırdıq:

<button onclick="funksiya();">Click</button>

<script>

function funksiya(){

alert("Ok");

}

</script>

Ancaq, hadisələri yazmağın başqa bir yolu da var. Bu yol addEventListener funksiyasından istifadə etməkdir. Aşağıdakı nümunəyə baxaq:

<button id="duyme">Click</button>

<script>

document.getElementById("duyme").addEventListener("click",funksiya);

function funksiya(){

alert("Ok");

}

</script>

Beləliklə elementi seçərək ona click hadisəsini və ona uyğun funksiyanı əlavə etmiş oluruq. Burada funksiyanın birinci parametri hadisənin adı, ikinci parametri isə çağırılacaq funksiyanın adıdır.

Aşağıdakı yazılış da mümkündür:

```
<button id="duyme">Click</button>

<script>

document.getElementById("duyme").addE
ventListener("click", function(){

alert("Ok");

});

</script>
```

Beləliklə funksiyanı ayrıca elan etməyə ehtiyac duymuruq. Bir elementə bir neçə ədəd hadisə əlavə etməyimiz də mümkündür. Aşağıdakı nümunəyə baxaq:

```
<button id="duyme">Click</button>

<script>

document.getElementById("duyme").addE
ventListener("click",funksiya);

document.getElementById("duyme").addE
ventListener("mouseover",funksiya2);

function funksiya(){

alert("Click etdiniz.");

}
```

function funksiya2(){

alert("Uzerine geldiniz.");

}

</script>

Beləliklə düymənin üzərinə gəlindikdə və düyməyə klik olunduqda fərqli funksiyalar işə düşəcəkdir.

Kalkulyator hazırlamaq

Kalkulyator hazırlamaq üçün ilk öncə aşağıdakı kodumuzu yazaq və izah edək.

```
<input type="text" id="eded1">

<input type="text" id="eded2">

<button
onclick="hesabla();">Hesabla</button>

<script>

function hesabla(){

var eded1,eded2,netice;

eded1=document.getElementById("eded1"
).value*1;

eded2=document.getElementById("eded2"
).value*1;

netice=eded1+eded2;

alert(netice);

}

</script>
```

Burada ilk öncə iki ədəd input elementi və bir ədəd düymə yaradırıq. Burada sadəcə düyməyə basdıqda həmin inputlara yazılan ədədlərin cəmini göstərməliyik. Bunun üçün bir funksiya yaradıb həmin funksiyanı düymənin onclick hadisəsinə yazırıq. Funksiyanın daxilində isə 3 ədəd dəyişən yaradırıq. Birinci dəyişənə birinci input elementinin dəyərini, ikinciyə ikinci input elementinin dəyərini ötürürük. Bunun üçün daha əvvəl öyrəndiyimiz document obyektinin **value** xüsusiyyətindən istifadə edirik. Onların dəyərlərini götürdükdən iki dəyişənə ötürdükdən sonra, bu iki dəyişənin cəmini üçüncü dəyişənə yazırıq və üçüncü dəyişəni ekrana çıxarırıq. Beləliklə iki ədədin cəmini tapan sadə hesablayıcımız hazır olmuş olur.

Burada əsas diqqət etməli olduğumuz nöqtə input elementindən daxil edilmiş yazını götürdükdən sonra onu 1-ə vurmağımızdır.

eded1=document.getElementById("eded1").value*1;

Bunun səbəbi inputdan dəyəri götürdükdə götürülən dəyərin yazı tipində olmasıdır. Yazıları isə ədəd kimi toplamaq olmaz. Onları toplamazdan əvvəl ədədə çevirməyimiz mütləqdir. Bunun üçün də sadəcə

götürdüyümüz dəyəri 1-ə vururuq və beləliklə yazı ədədə dönüşmüş olur.

İndi isə 4 əməliyyat üçün (toplama, çıxma, vurma və bölmə) kalkulyator yaradaq. Bunun üçün səhifəyə sadəcə bir ədəd seçim qutusu əlavə edək və istifadəçi hansı əməliyyatı aparacağını oradan seçsin. Aşağıdakı koda baxaq:

```
<input type="text" id="eded1">

<input type="text" id="eded2">

<select id="select">

<option>+</option>

<option>-</option>

<option>*</option>

<option>/</option>

</select>

<button
onclick="hesabla();">Hesabla</button>

<script>

function hesabla(){
```

```
var eded1,eded2,select,netice;

eded1=document.getElementById("eded1"
).value*1;

eded2=document.getElementById("eded2"
).value*1;

select=document.getElementById("select"
).value;

if(select=="+") netice=eded1+eded2;

if(select=="-") netice=eded1-eded2;

if(select=="*") netice=eded1*eded2;

if(select=="/") netice=eded1/eded2;

alert(netice);

}

</script>
```

Burada əlavə olaraq bir seçim qoyuruq və onun dəyərini əlavə bir dəyişənə yazırıq. Əgər seçilən dəyər «+» olarsa, o zaman, «netice» adlı dəyişənə iki ədədin cəmini, «-» olarsa fərqini, «*» olarsa hasilini, «/» olarsa nisbətini ötürürük və sonda onu ekrana çıxarırıq. Beləliklə də 4 əməliyyatlı hesablayıcımız hazır olmuş olur. Bu

hesablayıcını css ilə dizayn edərək öz saytınızda da işlədə bilərsiniz.

Atributlarla işləmək

Elementləri yaradarkən onlara href, src, class, id kimi bir sıra atributlar təyin edirik. Məsələn:

<img src="1.jpg" class="sinif1">

Burada elementə class və src atributlarını vermişik. Javascript vasitəsilə bu atributların dəyərini əldə edə və ya onları dəyişdirə bilərik. Elementin atributunun dəyərini əldə etmək üçün **getAttribute()** funksiya-sından istifadə etməliyik. Aşağıdakı nümunəyə baxaq:

```
<img src="1.jpg" id="sekil">

<br>

<button onclick="funksiya();">Elde et</button>

<script>

function funksiya(){

var x=document.getElementById("sekil").getAttribute("src");

alert(x);
```

}

</script>

Burada ilk öncə elementi seçirik və **getAttribute** funksiyası ilə «src» atributunu götürürük. **getAttribute** funksiyasının daxilində götürəcəyimiz atributun adını yazmalıyıq. Buna görə də yuxarıda funksiyanın içərisində «src» yazmışıq. Bu funksiya vasitəsilə müxtəlif atributların dəyərlərini götürə bilərik.

Atributların dəyərlərini götürməkdən əlavə onların dəyərlərini dəyişdirməyimiz də mümkündür. Məsələn, şəklin src atributunu dəyişərək ekranda görünən şəkli dəyişə bilərik. Bunun üçün **setAttribute** funksiyasından istifadə etməliyik. Aşağıdakı nümunəyə baxaq.

<img src="1.jpg" id="sekil">

**
**

<button onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

document.getElementById("sekil").setAttribute("src","2.jpg");

}

</script>

Burada ilk öncə ekranda «1.jpg» adlı şəkil görünür. Düyməyə basdıqdan sonra isə «img» elementinin «src» atributu dəyişir və «2.jpg» olur. Buna görə də ekranda «2.jpg» adlı şəkil görünür.

Gördüyümüz kimi, **setAttribute** funksiyası iki parametr almaqdadır. İlk parametr dəyişəcəyimiz atributun adıdır. Biz «src» atributuna yeni qiymət verəcəyimiz üçün birinci parametr olaraq «src» yazırıq. İkinci parametr olaraq isə «src» atributunun alacağı yeni qiyməti yazırıq ki, bu qiymət də «2.jpg» qiymətidir. Beləliklə düyməyə basdıqda şəkil dəyişmiş olur.

Elementin class atributunu, yəni sinifini dəyişərək onun dizaynında dəyişiklik edə bilərik. Aşağıdakı nümunəyə baxaq:

<style>

.sinif1{

color: red;

```
}

.sinif2{

color: green;

}

</style>

<a href="#" class="sinif1"
id="link">Burada yazi var</a>

<button
onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

document.getElementById("link").setAttri
bute("class","sinif2");

}

</script>
```

Gördüyümüz kimi burada iki sinif yaratmışıq və linkə birinci sinifi vermişik. Düyməyə basdıqda isə bu sinifi dəyişirik və beləliklə elementin görünüşünü də dəyişmiş oluruq. Burada siniflər yalnızca rəngi təyin edir.

Zaman funksiyaları

Javascript dilində zaman əməliyyatları aparmaq üçün zaman obyekti mövcuddur. Bu obyekt vasitəsilə saatı, günü, ili əldə etmək və bir sıra əməliyyatlar aparmaq mümkündür. Aşağıdakı koda baxaq:

<script>

var zaman=new Date();

document.write(zaman);

</script>

Burada **new Date();** yazaraq zaman obyektini yaratmış oluruq. Növbəti sətirdə isə **document.write(zaman);** yazaraq mövcud tarixi və saatı çap etmiş oluruq. Nəticə aşağıdakı kimi olur:

Mon Apr 13 2020 14:41:07 GMT+0400 (+04)

Sizdə isə nəticə olaraq səhifəni işlətdiyinizi tarixi göstərəcək. Beləliklə mövcud tarixi əldə etmiş oluruq. Zaman obyekti daxilində bir sıra funksiyalar da mövcuddur. Bu funksiyalardan bəzilərinə baxaq:

1) **getFullYear()** funksiyası — Bu funksiya yalnızca hal-hazırkı ili götürmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

```
<script>

var zaman=new Date();

var il=zaman.getFullYear();

document.write("Hal hazirki il: "+il);

</script>
```

2) **getMonth()** funksiyası — Bu funksiya hal-hazırki ayın nömrəsini götürmək üçün istifadə olunur. Aşağıdakı koda baxaq:

```
<script>

var zaman=new Date();

var ay=zaman.getMonth()+1;

document.write("Hal hazirki ay: "+ay);

</script>
```

Nəticədə hal-hazırki ayın nömrəsini götürmüş oluruq. Burada ayın dəyərinin üzərinə 1 gəldiyimizə diqqət edək. Bunun səbəbi, proqramlaşdırmada saymanın 0-dan başlamış

olmasıdır. Yəni, **getMonth()** funksiyası Yanvar ayı üçün bizə 0 dəyərini verəcəkdir. Çünki ayların dəyəri 0-11 aralığında verilir. Bunun üçün biz həmin dəyərin üzərinə 1 gələrək həmin dəyəri 1-12 aralığında etmiş oluruq.

3) **getDate()** funksiyası — Bu funksiyaı hal-hazırki günü götürmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

```
<script>

var zaman=new Date();

var gun=zaman.getDate();

document.write("Hal hazirki gun: "+gun);

</script>
```

Nəticədə hal-hazırki günü əldə etmiş oluruq.

4) **getHours()** funksiyası — Hal-hazırki saatı götürmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

```
<script>

var zaman=new Date();

var saat=zaman.getHours();
```

document.write("Hal hazirki saat: "+saat);

</script>

Beləliklə hal-hazırki saatı çap etmiş oluruq.

5) **getMinutes()** - Bu funksiya hal-hazırki dəqiqəni götürmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

<script>

var zaman=new Date();

var deqiqe=zaman.getMinutes();

document.write("Hal hazirki deqiqe: "+deqiqe);

</script>

Nəticədə hal-hazırki dəqiqə çap olunmuş olur.

6) **getSeconds()** funksiyası — Bu funksiya hal-hazırki saniyə dəyərini götürmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

<script>

var zaman=new Date();

var saniye=zaman.getSeconds();

document.write("Hal hazirki saniye: "+saniye);

</script>

Nəticədə saniyə dəyərini götürmüş oluruq. Səhifəni 1 saniyədən 1 yenilədikdə bu dəyərin dəyişdiyini görəcəyik.

7) **getDay()** funksiyası — Bu funksiya həftənin gününün nömrəsini götürmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

<script>

var zaman=new Date();

var gun=zaman.getDay();

document.write("Hal hazirda: "+gun+"-ci gundur.");

</script>

Nəticədə həftənin gününün nömrəsini əldə etmiş olacağıq. Məsələn, hal-hazırda gün bazar ertəsidirsə, bu 1-ci gün olduğu üçün 1 rəqəminin çap olunduğunu görəcəyik.

8) **getMilliseconds()** funksiyası — Bu funksiya hal-hazırki millisaniyəni götürmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

<script>

var zaman=new Date();

var millisaniye=zaman.getMilliseconds();

document.write("Hal hazirki millisaniye: "+millisaniye);

</script>

Beləliklə saniyədən kiçik olan millisaniyəni əldə etmiş oluruq.

İndi isə bu funksiyalardan bir neçəsini bir yerdə işlədək.

<script>

var zaman=new Date();

document.write(zaman.getHours() + ":" + zaman.getMinutes() + ":" + zaman.getSeconds());

</script>

Beləliklə mövcud saat, dəqiqə və saniyəni səhifədə göstərmiş olacağıq. Səhifəni bir saniyədən bir yenilədikdə zamanın dəyişdiyini görəcəyik. Bu zamanı avtomatik dəyişdirmək qaydasına isə növbəti mövzuda baxacağıq.

Yuxarıda **getDay()** funksiyasına baxdıq. Bu funksiya həftənin gününü rəqəmlə götürmək üçün istifadə olunurdu. İndi isə gəlin bu funksiyadan və şərt operatorundan istifadə edərək həftənin gununu Azərbaycan dilində sözlə yazaq.

```
<script>

var zaman=new Date();

var gun=zaman.getDay();

if(gun==1) document.write("Bazar
ertesi");

else if(gun==2)
document.write("Cersenbe axsami");

else if(gun==3)
document.write("Cersenbe");

else if(gun==4) document.write("Cume
axsami");

else if(gun==5) document.write("Cume");

else if(gun==6) document.write("Senbe");

else if(gun==7) document.write("Bazar");

</script>
```

Beləliklə həftənin gününü Azərbaycan dilində çap etmiş oluruq.

Oxşar nümunəni aylar üçün də yazmağımız mümkündür. Nəticədə ayın nömrəsinə uyğun olaraq ayın adını Azərbaycan dilində səhifədə göstərə bilərik. Aşağıdakı nümunəyə baxaq.

```
<script>

var zaman=new Date();

var ay=zaman.getMonth()+1;

if(ay==1) document.write("Yanvar");

else if(ay==2) document.write("Fevral");

else if(ay==3) document.write("Mart");

else if(ay==4) document.write("Aprel");

else if(ay==5) document.write("May");

else if(ay==6) document.write("Iyun");

else if(ay==7) document.write("Iyul");

else if(ay==8) document.write("Avqust");

else if(ay==9)
document.write("Sentyabr");
```

```
else if(ay==10)
document.write("Oktyabr");

else if(ay==11)
document.write("Noyabr");

else if(ay==12)
document.write("Dekabr");

</script>
```

Beləliklə ayın adı Azərbaycan dilində çap olunmuş olacaq.

Zaman aralığı ilə əməliyyatlar

Javascriptdəki ən maraqlı əməliyyatlardan biri də zaman aralığı ilə əməliyyatlardır. Bunun üçün Javascriptdə xüsusi funksiyalar mövcuddur. Məsələn, bir saniyədən bir səhifədə saatı yazmaq istəyiriksə, o zaman, bu funksiyalardan istifadə etməliyik. Bu funksiyaları hədsiz çox yerdə istifadə edə bilərik. Məsələn, bu funksiyaları oyunlarda, animasiya-larda istifadə edə bilərik.

Öyrənməli olduğumuz əsas funksiya **setInterval()** funksiyasıdır. Bu funksiya əməliyyatı zaman aralığı ilə daima yerinə yetirmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

```
<script>

function zamanlama(){

var zaman=new Date();

document.write(zaman.getSeconds()
+"<br>");

}

setInterval(zamanlama,1000);
```

</script>

İlk olaraq burada bir funksiya yaratmışıq. Funksiyamızın adı zamanlamadır və sadəcə saniyəni səhifədə çap edir. Sonra isə **setInterval(zamanlama,1000);** yazmışıq. Gördüyümüz kimi, funksiya iki parametr almışdır. Birinci parametr yaratdığımız zamanlama funksiyasının adıdır. İkinci parametr isə 1000-dir. Bu o deməkdir ki, 1000 **millisaniyədən** bir, yəni, 1 saniyədən bir zamanlama adlı funksiyamız işə düşəcəkdir. Yəni hər 1 saniyədən bir zamanlama adlı funksiyamız çağırılır və işə düşür. Beləliklə zaman aralığı ilə əməliyyat yaratmış oluruq. Burada hər bir saniyədən bir saniyə alt-alta yazılır. Ancaq, alt-alta olmaması üçün, sadəcə bir sətirdə yenilənməsi üçün, bir ədəd div elementi yaradıb saatı daima onun içərisində yeniləyə bilərik. Aşağıdakı nümunəyə baxaq.

```
<div id="saat"></div>

<script>

function zamanlama(){

var zaman=new Date();
```

```
document.getElementById("saat").innerHT
ML=zaman.getSeconds();

}

setInterval(zamanlama,1000);

</script>
```

Burada artıq hər saniyə zaman yenilənməkdə-dir. Əgər saat, dəqiqə və saniyəni də səhifədə göstərmək istəyiriksə o zaman aşağıdakı kimi yazmalıyıq:

```
<div id="saat"></div>

<script>

function zamanlama(){

var zaman=new Date();

document.getElementById("saat").innerHT
ML=zaman.getHours()
+":"+zaman.getMinutes()
+":"+zaman.getSeconds();

}

setInterval(zamanlama,1000);

</script>
```

İndi isə bu funksiyadan istifadə edərək 10-dan 0-a qədər bir geri sayım yaradaq.

```
<div id="zaman"></div>

<script>

var a=10;

function zamanlama(){

document.getElementById("zaman").innerHTML=a;

a--;

}

setInterval(zamanlama,1000);

</script>
```

Burada ilk olaraq bir a dəyişəni elan edirik və qiymətini 10-a bərabər edirik. Sonra isə funksiya daxilində div elementinə bu dəyişəni yerləşdiririk. Beləliklə div elementində 10 yazılır. Sonra isə **a--** yazaraq a dəyişəninin qiymətini bir vahid azaldırıq. Beləliklə artıq a-nın qiyməti 9 olmuş olur. Buna görə də, funksiya ilk dəfə çağırıldıqda səhifədə 10 yazılır. İkinci dəfə çağırıldıqda isə səhifədə 9 yazılır. Çünki, funksiya hər dəfə çağırıldıqda a dəyişəninin

qiyməti 1 vahid azalır. Beləliklə geri sayım yaranır. Burada ən əsas diqqət edək ki, a dəyişənini funksiyadan əvvəldə elan edirik. Çünki, dəyişəni funksiya içində elan edib qiymət versək, funksiya hər dəfə çağırıldıqda dəyişənin qiyməti yenidən 10 olacaq və geri sayım yaranmayacaq.

Yuxarıdakı kodda diqqət etsək, geri sayım 0-a çatdıqdan sonra yenidən azalaraq -1, -2 və s. olaraq saymağa başlayır. Ancaq, istəyə bilərik ki, sayma 0-a qədər olsun və 0-a çatdıqda vaxt bitsin. Bunun üçün sadəcə funksiya daxilindəki əməliyyatları a dəyişəninin 0-dan böyük olduğu halında yerinə yetirməliyik. Əks halda isə xəbərdarlıq göndərməliyik. Aşağıdakı koda baxaq:

```
<div id="zaman"></div>

<script>

var a=10;

function zamanlama(){

if(a>0){

document.getElementById("zaman").inner
HTML=a;
```

```
a--;

}

else{

document.getElementById("zaman").innerHTML=a;

alert("Vaxt bitdi.");

}

}

setInterval(zamanlama,1000);

</script>
```

Bu kodu yazıb işlətdikdə, 10 saniyədən sonra geri sayımın bitdiyini və xəbərdarlıq edildiyini görəcəyik. Ancaq, burada da bir problem yaranmaqdadır. Buradakı a dəyişənimizin qiyməti 0-a çatsa belə, funksiya yenidən 1 saniyədən bir çağırılmaqdadır. Buna görə də, davamlı olaraq hər 1 saniyədən bir xəbərdarlıq qutusunu görəcəyik. Bu səbəbdən də zamanlamanı dayandırmağımız mütləqdir. Zamanlamanı dayandırmağımız üçün **clearInterval()** funksiyasından istifadə edəcəyik. Aşağıdakı nümunəyə baxaq:

```
<div id="zaman"></div>

<script>

var a=10;

function zamanlama(){

if(a>0){

document.getElementById("zaman").inner
HTML=a;

a--;

}

else{

document.getElementById("zaman").inner
HTML=a;

alert("Vaxt bitdi.");

clearInterval(timer);

}

}

var timer=setInterval(zamanlama,1000);

</script>
```

Burada artıq 10 saniyə bitdikdən sonra **clearInterval()** funksiyası vasitəsilə 1 saniyədən bir çağırmanı ləğv edirik. Burada yazılışda kiçik bir dəyişiklik etmiş olduq. Əsas funksiyanı aşağıdakı şəkildə yazdıq:

var timer=setInterval(zamanlama,1000);

Belə yazdıqda da eyni şeylər baş verir. Sadəcə 10 saniyə bitdikdən sonra **clearInterval(timer);** yazdığımıza görə funksiyanın 1 saniyədən bir çağırılmasını ləğv etmiş oluruq. Burada **timer** dəyişənin adıdır.

Yazdığımız **setInterval()** funksiyası vasitəsilə müxtəlif animasiyalar hazırlamağımız da mümkündür. Aşağıdakı koda baxaq:

<img src="car.jpg" id="car" style="width: 200px;">

<script>

var a=0;

function zamanlama(){

a++;

document.getElementById("car").style.marginLeft=a+"px";

}

var timer=setInterval(zamanlama,10);

</script>

Burada bir maşın şəklini səhifəyə yerləşdirmişik. Hər 10 millisaniyədən bir isə zamanlama funksiyasını işlədirik. Bu funksiya daxilində isə hər dəfə a dəyişəninin qiyməti 1 vahid artır və **style.marginLeft** xüsusiyyəti ilə maşınımızın soldan məsafəsini dəyişirik, yəni a dəyişəninə bərabər edirik. Bu dəyişənimiz isə funksiya hər çağırıldıqda 1 vahid artdığına görə, hər dəfəsində maşınımız sağa doğru getmiş olur. Yuxarıda **a+"px"** yazmağımızın səbəbi isə, ölçü vahidinin piksel olmasıdır.

Öyrənəcəyimiz zaman funksiyalarından biri də setTimeout funksiyasıdır. Bu funksiya, müəyyən bir zaman sonra bir əməliyyatı yalnız bir dəfə yerinə yetirmək üçün istifadə olunur. Məsələn, istəyirik ki, 3 saniyə sonra istifadəçiyə sadəcə bir dəfə bildiriş göndərək. Bunun üçün bu funksiyadan istifadə edə bilərik. Bu funksiyanın istifadə qaydası **setInterval** funksiyasının istifadə qaydasına çox oxşardır. Aşağıdakı nümunəyə baxaq:

<script>

function zamanlama(){

alert("3 saniye kecdi.");

}

setTimeout(zamanlama,3000);

</script>

Nəticədə 3 saniyə sonra sadəcə bir dəfə **zamanlama** funksiyası işə düşür və səhifədə bildiriş görünür.

Animasiyaların hazırlanması

Javascript vasitəsilə 2 və 3 ölçülü animasiyaların hazırlanması mümkündür. Maraqlı animasi-yalar hazırlamaq biraz da fantazi-yamıza bağlı bir şeydir. Məsələn, zaman aralığı ilə elementi soldan sağa hərəkət etdirə bilərik ki, bu da sadə bir animasiya sayılır. Aşağıdakı nümunəyə baxaq:

```
<div style="width: 100px; height: 100px;
background-color: red;"
id="element"></div>

<script>

var x=0;

setInterval(hereket,10);

function hereket(){

x++;

document.getElementById("element").styl
e.marginLeft=x+"px";

}

</script>
```

Nəticədə 10 millisaniyədən bir hərəkət funksiyası çağırılır və element soldan sağa doğru hərəkət etmiş olur. Burada **x+"px";** yazmağımızın səbəbi, ölçünü piksellə verməli olmağımızdır. Buradakı x dəyişəninin qiyməti hər funksiya çağırıldıqda bir vahid artırılır və elementin soldan məsafəsi yeni qiymətə bərabər olur, yəni bir vahid artmış olur. Beləliklə də animasiya yaranmış olur.

Burada elementin soldan nə qədər uzaqlaşdığını bilmək üçün x dəyişənini başqa bir elementin içərisində göstərə bilərik. Aşağıdakı nümunəyə baxaq:

```
<div style="width: 100px; height: 100px;
background-color: red;"
id="element"></div>

<div id="yazi"></div>

<script>

var x=0;

setInterval(hereket,10);

function hereket(){

x++;
```

document.getElementById("element").style.marginLeft=x+"px";

document.getElementById("yazi").innerHTML=x;

}

</script>

Nəticədə ekranda yazılan x-in qiymətinin daima artdığını və bununla bərabər də elementin soldan uzaqlaşdığını görəcəyik. Burada elementin soldan uzaqlaşmasının səbəbi x-in dəyərinin daima artmasıdır ki, bunu da burada açıq şəkildə görə bilərik.

Əgər animasiyanın bir düyməyə klik olunduqdan sonra başlamağını istəyiriksə, bu zaman, yuxarıdakı javascript kodlarını ayrı bir funksiyanın içinə yazmalı və o funksiyanı da bir düymənin «onclick» hadisəsində çağırmalıyıq. Aşağıdakı nümunəyə baxaq:

** **

** **

```
<button
onclick="funksiya();">Saga</button>

<script>

function funksiya(){

var x=0;

setInterval(hereket,10);

function hereket(){

x++;

document.getElementById("element").styl
e.marginLeft=x+"px";

document.getElementById("yazi").innerHT
ML=x;

}

}

</script>
```

Beləliklə düyməyə klik etdikdən sonra animasiya başlamış olur.

Elementlər üçün müxtəlif xüsusiyyətlərdən istifadə edərək çox fərqli animasiyalar yaratmağımız da mümkündür.

Parametrli funksiyalar

Biz bundan əvvəlki mövzularda parametr-siz funksiya yaratmağı və istifadə etməyi öyrənmişdik. Çətinlik və qarışıqlıq yaranmasın deyə, ilk olaraq yalnızca parametrsiz funksiyalar barədə yazmışdıq. Çünki bir çox yerdə parametrsiz funksiyaların istifadəsi vacib idi. Bu mövzumuzda isə parametrli funksiyaları öyrənəcəyik.

İlk öncə parametrli funksiyanın nə olduğunu anlamağa çalışaq. Məsələn, riyaziyyatda adətən parametrli funksiyalardan istifadə olunur. Məsələn:

Y=f(x)=x*x;

Bu funksiya ədədin kvadratını tapmaq üçün olan funksiyadır. Biz burada x-in yerinə müxtəlif qiymətlər verdikdə funksiya daxilində x-i həmin qiymətlə əvəzləyib nəticəni hesablayırıq. Məsələn:

F(3)=3*3=9;

F(5)=5*5=25;

Gördüyümüz kimi, əvvəlcə funksiyanı təyin edirik, bu funksiya x-in kvadratını tapmaq üçün

olan funksiyadır. Sonra isə funksiyaya qiymətlər verib nəticəni hesablayırıq. Burada F(3) yazdıqda sadəcə x olan yerlərə 3 yazılır və hesablama aparılır. Proqramlaşdırmada da tamamilə eyni məntiqdən istifadə olunur. Aşağıdakı nümunəyə baxaq:

<script>

function yaz(x){

return x*x;

}

document.write(yaz(3));

</script>

Burada ilk öncə funksiyamızı yaradırıq. Funksiyamıza sadəcə bir x parametri veririk. Funksiya parametrləri mötərizələrin arasında yazılır. Gördüyümüz kimi, funksiya daxilində «return x*x» yazmışıq. Bu o deməkdir ki, funksiyanın nəticəsi x-in kvadratı olacaq. Yəni, funksiyamız bizə daxil etdiyimiz ədədin kvadratını verəcək. Buradakı «return» sözü isə funksiyanın verəcəyi qiyməti bildirmək üçün yazılır. Növbəti olaraq 3-ün kvadratını tapmaq üçün yaz(3) yazırıq və onu çap edirik. Beləliklə 3-ün kvadratını tapmış oluruq.

Parametrli funksiyanı daha yaxşı anlamaq üçün müxtəlif nümunələr. Məsələn, aşağıdakı funksiya daxil olunan ədədin 2 qatını bizə verəcək:

```
<script>

function yaz(x){

return x*2;

}

document.write(yaz(3));

</script>
```

Nəticədə ədədin 2 qatını verən bir ədəd funksiya yaratmış oluruq.

Funksiyaya istədiyimiz sayda parametr daxil edə bilərik. Ancaq, həmin parametrləri daxil edərkən mötərizə daxilində onları vergüllə ayırmalıyıq. Aşağıdakı nümunəyə baxaq:

```
<script>

function yaz(x,y){

return x+y;

}
```

document.write(yaz(3,4));

</script>

Nəticədə funksiyamız daxil edilmiş parametrlərin cəmini tapmış olur. Burada funksiya üçün iki ədəd parametr var. Bunlardan biri x, digəri isə y parametridir. Gördüyümüz kimi, bu parametrləri vergüllə ayırmışıq. Funksiya daxilində **return x+y;** yazmaqla isə funksiya nəticəsinin x və y dəyişənlərinin cəminə bərabər olduğunu bildirmiş oluruq. Funksiyanı çağırdıqda isə, yaz(3,4) yazmaqla x parametrinə 3, y parametrinə isə 4 qiymətini vermiş oluruq. Çünki funksiyada birinci parametrimiz x, ikinci parametrimiz isə y idi. Buna uyğun olaraq da funksiyanı çağırdıqda birinci parametrə 3, ikinci parametrə isə 4 vermişdik. Beləliklə x dəyişəni 3 qiyməti ilə, y dəyişəni isə 4 qiyməti ilə əvəzlənmiş olur.

Funksiya daxilində paramterlərlə istədiyimiz əməliyyatları apara bilərik. Məsələn, iki ədədi bir-birinə bölmək üçün bir funksiya yazaq:

<script>

function yaz(x,y){

if(y!=0) return x/y;

else return "Emeliyyat mumkun deyil.";

}

document.write(yaz(3,2));

</script>

Gördüyümüz kimi, burada y parametrinin 0-dan fərqli olub-olmadığını yoxlayırıq. Əgər y parametri 0-dan fərqli olarsa, o zaman, funksiyanln nəticəsində x parametrini y paramterinə bölürük. Nəticədə funksiyaya parametr olaraq ötürdüyümüz birinci ədədi ikinci ədədə bölmüş olacağıq. Ancaq, əgər y parametri 0-a bərabər olarsa, funksiya nəticəsi olaraq əməliyyatın mümkün olmadığı yazısı təyin olunacaq. Gördüyümüz kimi, funksiyanın nəticəsi yazı da ola bilər. Sonda funksiyanı çağırarkən ikinci parametri 0-dan fərqli təyin etsək, o zaman, birinci ədəd 2-ciyə bölünəcək və nəticə yazılacaq. Ancaq, ikinci parametr 0 olarsa, o zaman, əməliyyatın mümkün olmadığı yazılacaq.

Zamanla müxtəlif nümunələr yazdıqca funksiyaları daha yaxşı anlaya, funksiyaların əhəmiyyətini daha yaxşı başa düşə bilərsiniz.

Dövr operatorları. While opratoru

Bir otaqda 100 ədəd qutu olduğunu və bu qutulardan neçəsinin boş, neçəsinin dolu olduğunu öyrənmək istədiyimizi fərz edək. Bunun üçün qutuları bir-bir açıb qutuların dolu olub-olmadığına baxmalıyıq. Buna oxşar əməliyyatları proqramlaşdırmada da yerinə yetirərkən eyni üsuldan istifadə etməliyik. Yazacağımız xəyali proqram qutuları bir-bir açıb hansının dolu, hansının boş olmasını yoxlamalıdır. 100 dəfə qutuları yoxlamaq isə bir xeyli vaxtımızı alacaq. Proqramlaşdırmada sırf bu cür ardıcıl əməliyyatları qısa şəkildə yerinə yetirmək üçün, eyni bir əməliyyatı bir neçə dəfə təkrarlamaq üçün bir sıra operatorlar mövcuddur. Bu operatorlardan biri də **while** operatorudur. Bu operator, bir şərt ödəndiyi müddətcə bir əməliyyatı yerinə yetirmək üçündür. Operatorun istifadə qaydası if operatoruna oxşardır. Əvvəlcə operatorun adı yazılır, sonra mötərizə daxilində şərt, sonra isə fiqurlu mötərizələr arasında əməliyyatlar yazılır. Nümunə bir proqramda bu operatordan istifadə edək.

```
<script>

var i=0;
```

```
while(i<5){

document.write("Ok<br>");

}

</script>
```

Kodu işlətdiyimizdə səhifədə sonsuz bir dövrə yaranmış olacaq. Yəni alt-alta yazı yazılmağa davam edəcək və bu heç vaxt bitməyəcək. Buna görə də səhifə daim yüklənə, yüklənməsi heç vaxt bitməyə bilər. Bunun səbəbi a dəyişəninin qiymətinin həmişə 5-dən kiçik olmasıdır. «While» operatorunda ilk olaraq yazdığımız şərt yoxlanılır. Burada yazdığımız şərt a dəyişəninin 5-dən kiçik olmasıdır. Şərt ödəndiyi üçün ilk olaraq əməliyyatlar yerinə yetirilir. Əməliyyatlar bir dəfə yerinə yetirildikdən sonra, operator yenidən əvvələ qayıdaraq şərti bir daha yoxlayır. Şərt doğru olarsa yenidən əməliyyatlar aparılır. Bu proses daimi olaraq davam edir. Yuxarıdakı nümunədə də şərt həmişə ödənəcəyinə görə, əməliyyatlar həmişə yerinə yetirilir. Ancaq, biz hər dövrün sonunda a dəyişəninin qiymətin bir vahid artırsaq, onda, nəticə başqa cür olacaq.

İndi isə, gəlin əməliyyatları sonsuz sayda deyil, 5 dəfə yerinə yetirək. Bunun üçün kodumuzda kiçik bir dəyişiklik edək.

```
<script>

var i=0;

while(i<5){

document.write("Ok<br>");

i++;

}

</script>
```

Burada ilk olaraq i dəyişəninin qiyməti 0-dır. İlk şərtdə dəyişənin qiymətinin 5-dən kiçik olub-olmaması yoxlanılır. Həqiqətən də 0 rəqəmi 5-dən kiçikdir. Şərt ödəndiyinə görə əməliyyatlar yerinə yetirilməyə başlanır. İlk əməliyyatımız sözün çap olunmasıdır. Sonrakı əməliyyatımız isə dəyişənin qiymətinin 1 vahid artmasıdır. İlk dövrün sonunda dəyişənin qiyməti 1 vahid artaraq 1 olur.

Əməliyyatlar bir dəfə yerinə yetirildikdən sonra operator başa qayıdır və şərti yenidən yoxlayır. Bu dəfə i dəyişəninin qiyməti 1-ə

bərabərdir. Həqiqətən də 1 rəqəmi 5-dən kiçik olduğu üçün şərt ödənir və operator əməliyyatları yerinə yetirməyə başlayır, eyni zamanda əməliyyatların sonunda i++ ifadəsi ilə dəyişəninin qiymətini 1 vahid artıraraq 2 edir.

Artıq əməliyyatlar iki dəfə yerinə yetirildi. Növbəti dəfə yenə operator əvvələ qayıdaraq I dəyişənin qiymətinin 5-dən kiçik olub-olmamasını yoxlayır. Dəyişənin hal-hazırki qiyməti 5-dən kiçik olduğu üçün, yəni şərt ödəndiyi üçün, əməliyyatlar yerinə yetirilməyə başlayır və eyni zamanda I dəyişəninin qiyməti 1 vahid artaraq 3 olur.

Üçüncü dəfə əməliyyatlar yerinə yetirildikdən sonra, proqram yenidən əvvələ qayıdır və I dəyişəninin qiymətinin 5-dən kiçik olub-olmamasını yoxlayır. Bu dəfə də I dəyişəninin qiyməti, yəni 3, 5-dən kiçik olduğu üçün şərt ödənir və əməliyyatlar yerinə yetirilməyə başlayır, eyni zamanda I dəyişəninin qiyməti 1 vahid artaraq 4 olur.

Dördüncü dəfə də əməliyyatlar yerinə yetirildikdən sonra, operator yenidən əvvələ qayıdır və I dəyişəninin 5-dən kiçik olub-olmamasını yoxlayır. Bu dəfə də I dəyişəninin qiyməti, yəni 4, 5-dən kiçik olduğu üçün şərt

ödənir və əməliyyatlar yerinə yetirilməyə başlayır, eyni zamanda i++ yazıldığı üçün, I dəyişəninin qiyməti 1 vahid artaraq 5 olur.

Artıq əməliyyatlar 5 dəfə yerinə yetirilib və i dəyişəninin qiyməti 5-ə bərabər olub. 5-ci dəfə əməliyyatlar yerinə yetirildikdən sonra, operator yeindən əvvələ qayıdaraq I dəyişəninin qiymətinin 5-dən kiçik olub olmadığını yoxlayır. Bu dəfə I dəyişəninin qiyməti 5-dir və 5-dən kiçik deyil, 4 olsa idi doğru olacaqdı. Beləliklə şərt ödənmir, şərt ödənmədiyinə görə, görə operator bu dəfə əməliyyatları yerinə yetirmir və operatorun işi sona çatmış olur. Bundan sonra artıq operatordan sonrakı əməliyyatlar yerinə yetirilməyə başlayır.

Bu operatora aid bir neçə nümunə daha yazaraq operatorun işini daha yaxşı anlamağa çalışaq. Gəlin 1-dən 10-a qədər ədədləri alt-alta çap edən bir proqram yazaq.

```
<script>

var i=1;

while(i<=10){

document.write(i+"<br>");

i++;
```

```
}

</script>
```

Dəyişənimizin qiyməti əvvəlcə 1 olaraq təyin olunmuşdur. İlk olaraq operatorda dəyişənin qiymətinin 11-dən kiçik-bərabər olub-olmaması yoxlanılır. Şərt doğru olduğu üçün əməliyyatlar yerinə yetirilməyə başlayır. Əməliyyatlarımız i dəyişənin çap olunmasından və sonda bir vahid artırılmasından ibarətdir. İlk olaraq I dəyişəninin qiyməti 1 çap olunur və bir vahid də artaraq 2 olur. Sonra yenidən operator başa qayıdır və şərt yoxlanılır. Doğru olduğu üçün əməliyyatlar yerinə yetirilir, I-nin qiyməti çap olunur və bir vahid artaraq 3 olur. Bu proses I-nin qiyməti 11 olana qədər davam edir. Dəyişənin qiyməti 11 olduqda artıq şərt ödənmədiyinə görə əməliyyatlar başlamır və operator öz işini bitirmiş olur.

İndi isə, 1-dən 5-ə qədər ədədlərin cəmini tapan bir proqram yazaq.

```
<script>

var i=1, cem=0;

while(i<=5){

cem=cem+i;
```

```
i++;

}

document.write(cem);

</script>
```

Burada hər dövrdə i-nin qiqyməti 1 vahid artmış olur və I-nin yeni qiyməti cem dəyişəninin üzərinə gəlinir. Cem dəyişənin ilk qiyməti 0-dır. İlk dövrün sonunda 1 olur. Növbəti dövrdə 3 olur, növbəti dövrdə 6, növbəti dövrdə 10, sonuncu dövrdə isə 15 olur və operator öz işini bitirmiş olur.

İndi isə, 1-dən 5-ə qədər ədədlərin hasilini tapan bir proqram yazaq.

```
<script>

var i=1, hasil=1;

while(i<=5){

hasil=hasil*i;

i++;

}
```

```
document.write(hasil);

</script>
```

Burada hasil dəyişəninin ilk qiyməti 1 olaraq təyin olunmuşdur. İlk dövrdə hasil dəyişəni 1-ə vurulur, ikinci dövrdə 2-ə vurularaq 2 olur, üçüncü dövrdə 3-ə vurularaq 6 olur, dördüncü dövrdə 4-ə vurularaq 24 olur, beşinci və sonuncu dövrdə isə 5-ə vurularaq 120 olur və operator öz işini tamamlayır.

Burada diqqət etməli olduğumuz əsas məqam, cəmin tapılarkən ilk qiymətin 0, hasilin tapılarkən ilk qiymətin 1 olaraq təyin olunmasıdır. Cəm tapılarkən ilk qiymət 0 olaraq təyin olunur, çünki, növbəti ədəd 0-ın üzərinə gəlinəcək və cəm alınacaq. Hasildə isə 1 olur, çünki növbəti ədəd 1-ə vurulacaq. Əgər hasildə də ilk qiymət 0 olsa, onda növbəti ədəd 0-a vurular və yeni qiymət də 0 olardə. Nəticədə isə dəyişənin son qiyməti 0 olaraq qalardı və istədiyimiz nəticəni ala bilməzdik.

İndi isə, gəlin 1 ilə 100 arasında olan cüt ədədləri çap edək. Əvvəlcə yalnız 1 ədədin cüt olub-olmadığını yoxlamaq üçün lazım olan koda baxaq:

```
<script>
```

```
var i=1;

while(i<=100){

if(i%2==0){

document.write(i+"<br>");

}

i++;

}

</script>
```

Burada əsas if hissəsinə diqqət edək. «i%2» ifadəsi, dəyişənin 2-ə bölünməsindən alınan qalığı göstərir. Məsələn, ədədimiz 51-dirsə, 2-ə bölünməsindən alınan qalıq 1 olur. Çünki, 51 tək ədəddir. Cüt ədədlərdə isə, qalıq 0 olur. Bundan istifadə edərək ədədin cüt yoxsa tək olmasını ayırd edə bilirik. Yəni, əgər ədədin 2-ə bölünməsindən alınan qalıq 0 olarsa ədəd cütdür, 1 olarsa ədəd təkdir. Beləliklə 1-dən 100-ə qədər olan cüt ədədləri çap etmiş oluruq. Şərt hissəsində **if($i%2==0){** yazmaqla, 1-dən 100-ə qədər olan tək ədədləri çap edə bilərik.

Dövr operatorları. For operatoru

Yuxarıdakı nümunələrdə dövr yaratmaq üçün «while» operatorundan istifadə etdik. Ancaq, dövr yaratmaq üçün «while» operatorundan əlavə «for» operatorundan da istifadə olunur. «While» operatorunda dövr sayı əvvəlcədən bilinməsə də dövrü başlatmaq olurdu və dövr sayı sonsuz da ola bilərdi. For operatoru isə əsasən dövr sayı məlum olduqda istifadə olunur. Operatorun istifadə qaydasına aid bir nümunə göstərək:

```
<script>

var i;

for(i=1;i<=5;i++){

document.write(i+"<br>");

}

</script>
```

While operatorunda i dəyişəninin artımını yalnız operatorun əməliyyatların daxilində göstərə bilirdik. Ancaq, for operatorunda, əməliyyatlar başlamazdan öncə i dəyişəninin ilk qiymətini təyin edirik. İlk parametrdə i dəyişəninin ilk qiyməti təyin olunur. İkinci

parametrdə şərt yazılır, şərtimiz dəyişənin 5-dən kiçik-bərabər olmasıdır. Üçüncü parametr isə i-nin artımıdır. Operatorun işləmə qaydası da while operatorunun işləmə qaydasına oxşardır. İlk olaraq I-nin ilk qiyməti təyin olunduqdan sonra i-nin qiymətinin 5-dən kiçik-bərabər olub-olmaması yoxlanılır. Şərt ödənirsə əməliyyatlar başlayır, sonda isə i-nin qiyməti 1 vahid artırılır və yenidən əvvələ qayıdılır. İ-nin qiyməti 1 vahid artdıqdan sonra şərt yenidən yoxlanılır. Şərt ödənərsə proses eyni qaydada yerinə yetirilir. Bu proses şərt ödənməyənə qədər davam edir. Dəyişənin qiyməti 5-ə çatdıqda əməliyyatlar şərt ödəndiyinə görə yerinə yetirilir və dəyişənin qiyməti bir vahid artırılaraq altı olur. Yenidən əvvələ qayıdılır və şərt yoxlanılır. Şərt ödənilmədiyinə görə əməliyyatlar yerinə yetirilmir və operator öz işini bitirir.

While operatoru ilə yazdığımız bir çox nümunəni eynilə for operatoru ilə də yazmağımız mümkündür. Məsələn, 1-dən 100-ə qədər olan cüt ədədlərin tapılması üçün lazım olan proqramı for operatorundan istifadə edərək də yaza bilərik.

```
<script>

var i;
```

```
for(i=1;i<=100;i++){

if(i%2==0){

document.write(i+"<br>");

}

}

</script>
```

Nəticədən 1-dən 100-ə qədər olan cüt ədədlər alt-alta çap olunmuş olacaq. Dövr sayı əvvəlcədən məlum olan dövrlər üçün for operatorunun istifadə olunması daha məqsədəuyğundur.

For operatoru ilə başqa bir nümunə də yazaq. Məsələn, ədədin faktorialını hesablamaq üçün for operatorundan istifadə edə bilərik. Faktorialın nə olduğunu xatırlayaq, faktorial 1-dən təyin etdiyimiz ədədə qədər olan ədədlərin hasilidir. Məsələn, 4 faktorial 4*3*2*1 deməkdir. Deməli bir ədədin faktorialını tapmaq üçün, 1-dən başlayaraq həmin ədədə qədər olan ədədləri bir-birinə vurmalıyıq. Aşağıdakı nümunədə proqramla 5 faktorialı hesablayaq.

```
<script>

var i, faktorial=1;

for(i=1;i<=5;i++){

faktorial=faktorial*i;

}

document.write(faktorial);

</script>
```

Yazdığımız bu kod vasitəsilə 5 faktorialı hesablamış oluruq. İlk olaraq bir faktorial dəyişəni elan edirik və onun qiymətini 1-ə bərabər edirik. Sonra isə for operatoru ilə i dəyişənini 1-dən 5-ə qədər sıralayırıq və faktorial dəyişənini i-nin aldığı hər bir qiymətə vururuq, yəni faktorial dəyişənini 1-dən 5-ə qədər bütün ədədlərə vururuq və nəticədə 5 faktorialı tapmış oluruq.

Dövr operatorları. Do while opratoru

Dövr operatorlarından biri də «do while» operatorudur. Bu operator «while» operatoruna çox oxşardır. Ancaq əsas bir fərqi mövcuddur. «while» operatorunda şərt ödənmədiyi halda əməliyyatlar yerinə yetirilmirdi. «do while» operatorunda isə şərt ödənməsə belə əməliyyatlar ən azı 1 dəfə yerinə yetirilir və operator öz işini bitirir. Bəzi yerlərdə şərt ödənməsə belə əməliyyatların ən azı bir dəfə yerinə yetirilməsinə ehtiyac duyularsa, «do while» operatorundan istifadə oluna bilər. Operatorun istifadəsinə aid kiçik bir nümunə göstərək:

```
<script>

do{

document.write("Ok");

} while(1==2);

</script>
```

Gördüyümüz kimi, ilk olaraq do yazılır, fiqurlu mötərizələr açılır və bu mötərizələrin içində əməliyyatlar yazılır. Əməliyyatlar bitdikdən sonra fiqurlu mötərizə bağlanır və

while(şərt) yazılır. Yuxarıdakı nümunədə şərtimiz 1-in 2-ə bərabər olmasıdır. Təbii ki, 1 və 2 bərabər deyillər, şərt ödənmir. Şərt ödənmədiyinə görə də operator öz işini bitirir. Ancaq yenə də, şərtə baxmazdan öncə operator əməliyyatları bir dəfə yerinə yetirmiş olur. Belə deyə bilərik ki, şərtdən asıllı olmadan operator əməliyyatları bir dəfə yerinə yetirir, əməliyyatları bir dəfə yerinə yetirdikdən sonra şərtə baxır.

Obyektlərin yaradılması

Biz indiyə qədər öyrəndiyimiz mövzularda Javascript daxilində olan hazır obyektlərdən istifadə etmişdik. Məsələn, İçərisində riyazi funksiyalarını daşıyan Math obyektini öyrənmişdik və onun funksiyalarından riyazi hesablamalar üçün istifadə etmişdik. Document obyektinin içərisində olan write funksiyasından yazı çap etmək üçün istifadə etmişdik. Bunlardan əlavə Javascriptdin özündə bir çox obyekt və funksiyalar var ki, onlara irəlidə baxacağıq.

Javascriptin özündə olan hazır obyektlərdən əlavə bizim özümüz də obyekt yarada, onun içərisinə xüsusiyyətlər və funksiyalar yerləşdirə bilərik.

Məsələn, insanı bir obyekt olaraq təsəvvür edə bilərik. İnsanın yaşı, göz rəngi, çəkisi, boyu kimi bir çox xüsusiyyətləri var. Biz insan adlı bir obyekt yaradıb onun içərisinə bu xüsusiyyətləri yerləşdirə bilərik. Obyekt yaratmaq qaydasına baxaq:

```
<script>

var insan={ad:"Shukur"};

document.write(insan.ad);
```

</script>

Dəyişənləri edərkən var açar sözündən istifadə edirdik. Obyektləri yaradarkən də bu qaydada yaradırıq. Dəyişəni elan edib = işarəsi qoyuruq, «{» və «}» işarələri arasında obyektin xüsusiyyət və funksiyalarını yazırıq. Xüsusiyyətin adını dırnaq içində yox, adicə yazırıq. Xüsusiyyətin aldığı qiyməti isə dırnaq içərisində yazırıq. Beləliklə insan adlı obyekt və onun içərisində ad xüsusiyyətini yaratmış oluruq. Obyektin xüsusiyyətinə müraciət edərkən isə sadəcə **insan.ad** yazırıq və müraciət edirik.

Obyektin daxilində birdən çox xüsusiyyətlər və funksiyalar ola bilər. Çox sayda xüsusiyyət və funksiya yaradaırıqsa, o zaman, onları vergüllə ayıraraq elan edə bilərik. Aşağıdakı nümunəyə baxaq:

<script>

var insan={ad:"Shukur",soyad:"Huseynov",yas:23};

**document.write(insan.ad +"
" + insan.soyad);**

</script>

Beləliklə obyekt daxilində bir neçə xüsusiyyət yaratmış oluruq. Yaş ədəd tipində olduğu üçün onu dırnaq arasında yazmağa ehtiyac yoxdur.

İstəyimizə uyğun olaraq obyekt xüsusiyyətlərini ayrı-ayrı sətirlərdə də yaza bilərik. Aşağıdakı nümunəyə baxaq:

<script>

var insan={ad:"Shukur",

soyad:"Huseynov",

yas:24

};

**document.write(insan.ad +"
" + insan.soyad);**

</script>

Gördüyümüz kimi, obyekti bu şəkildə də yazmağımız mümkündür.

Xüsusiyyətlərdən əlavə obyekt daxilində funksiyalar yaratmağımız da mümkündür. Aşağıdakı nümunəyə baxaq:

```
<script>
var insan={ad:"Shukur",
soyad:"Huseynov",
yas:24,
tevellud: function(){
return 2020-this.yas;
}
};
document.write(insan.tevellud());
</script>
```

Burada sadəcə xüsusiyyətlərdən sonra funksiyanın adını yazırıq və : qoyuruq. Sonra isə sadəcə funksiyanı yazırıq.

```
function(){
return 2020-this.yas;
}
```

Funksiyanın içərisində return açar sözü ilə funksiyanın qaytaracağı qiyməti yazırıq. **2020-this.yas** yazaraq təvəllüdü hesablamış oluruq.

2020-dən 24 çıxdıqda 1996 almış oluruq. Burada **this.yas** yazaraq həmin obyektin yaşını götürmüş oluruq. Burada **this** açar sözü ilə obyektin xüsusiyyətlərinə müraciət edə bilirik. Burada tam adı götürmək üçün də kiçik bir funksiya yaza bilərik:

```
<script>

var insan={ad:"Shukur",

soyad:"Huseynov",

yas:24,

tevellud: function(){

return 2020-this.yas;

},

tam: function(){

return this.ad + " " + this.soyad;

}

};

document.write(insan.tam());

</script>
```

Burada tam adlı funksiyamız sadəcə obyektin ad və soyad xüsusiyyətlərini toplayaraq bizə göstərir.

Obyekt daxilində yaratdığımız funksiyalara parametr ötürməyimiz də mümkündür. Aşağıdakı nümunəyə baxaq:

```
<script>

var insan={ad:"Shukur",

soyad:"Huseynov",

yas:24,

deyis: function(x){

this.yas=x;

}

};

insan.deyis(30);

document.write(insan.yas);

</script>
```

Beləliklə insan.deyis(30) yazaraq obyekt daxilindəki yaş xüsusiyyətini dəyişib 30 etmiş oluruq. Bu funksiyamız aşağıdakı şəkildə idi:

deyis: function(x){

this.yas=x;

}

Burada «deyis» funksiyanın adıdır. Funksiyanın içərisində isə this.yas=x obyekt daxilindəki yaş xüsusiyyətini funksiyanın parametrinə bərabərləşdiririk. Ona görə də, **insan.deyis(30);** yazaraq parametri 30 etdikdə obyektdəki yas xüsusiyyətinin qiyməti 30 olmuş olur. Əgər funksiyadan istifadə etməsək yaş sadəcə 24 olaraq qalacaq.

Obyekt daxilində yaratdığımız bir xüsusiyyəti silmək üçün **delete** operatorundan istifadə edə bilərik. Aşağıdakı nümunəyə baxaq:

<script>

var insan={ad:"Shukur",

soyad:"Huseynov",

yas:24

};

delete insan.yas;

document.write(insan.yas);

</script>

Beləliklə **delete insan.yas;** yazaraq insan obyektinin içərisindəki yas xüsusiyyətini silmiş oluruqu. Ona görə də onu çap edərkən **undefined** yazıldığını görəcəyik. Çünki artıq həmin xüsusiyyət silinmişdir.

Obyekti aşağıdakı şəkildə də yarada bilərik:

<script>

function insan(yas,boy,ceki){

this.yas=yas;

this.boy=boy;

this.ceki=ceki;

}

var insan1=new insan(24,180,80);

document.write(insan1.boy);

var insan2=new insan(25,175,75);

document.write(insan2.boy);

</script>

Bu üsulla bir ədəd funksiya yaradıb onunla iki ədəd obyekt yaradırıq. Yaradılacaq obyektlər eyni formadadırsa, yəni, parametrləri eynidirsə bu üsuldan istifadə etmək rahat olacaq. Burada funksiyanın içərisində this.yas=yas yazaraq obyektinin yaşını ötürüləcək yas parametrinə bərabər olacağını bildirirk. Obyekti isə aşağıdakı şəkildə yaradırıq:

var insan1=new insan(24,180,80);

Beləliklə, yaşı 24, boyu 180, çəkini 80 olaraq təyin edirik. Bu xüsusiyyətlərə malik istədiyimiz sayda obyekt yarada bilərik. Burada insan1 və insan2 fərqli obyektlərdir və fərqli xüsusiyyətlər almaqdadır. Bu üsulda yalnız bir obyekt də yarada bilərik.

Prototiplər

Prototiplər Javascript obyektlərinin xüsusiyyətləri bir-birindən miras aldıqları mexanizmdir. Javascript prototip əsaslı bir dildir və bu səbəblə Javascript istifadə edərək bir obyekt yaratdıqda Javascript obyektin içərisinə bir prototip xüsusiyyəti əlavə edir. Prototip xüsusiyyəti təməl olaraq bir obyektdir və bir prototip obyektinə funksiyalar və xüsusiyyətlər əlavə edə bilərik. Bu da digər bütün obyektlərin bu funksiya və xüsusiyyətləri təhvil almasını təmin edir.

Bir obyektin prototip obyekti, funksiyalar və xüsusiyyətləri təhvil aldığı bir prototip obyektinə də sahib ola bilər. Bu ümumilikdə bir prototip zənciri adlanır və fərqli obyektlərin niyə görə istifadə oluna bilən digər obyektlər üzərində təyin edilmiş xüsusiyyətlərə və metodlara sahib olduğunu izah edir.

Gəlin ilk olaraq bir obyekt yaraq.

```
<script>

function telefon(model,ram){

this.model=model;
```

```
this.ram=ram;
}
var t1=new telefon("Xiaomi",4);
document.write(t1.ram);
</script>
```

Bütün Javascript obyektləri, xüsusiyyətləri və funksiyaları bir prototipdən təhvil alır.

-Date obyektindən Date.prototype olaraq miras alınır;

-Array obyektindən Array.prototype olaraq miras alınır;

-Telefon obyektindən Telefon.prototype olaraq miras alınır;

Object.prototype prototip təhvilalma zəncirinin ən üstündədir.

Date obyektləri, Array obyektləri və Telefon obyektləri Object.prototype-dan təhvil alınır.

Bəzən obyektlərə yeni funksiyalar ə ya xüsusiyyətlər əlavə etmək istəyə bilərik. Prototip xüsusiyyəti obyektə yeni xüsusiyyətlər əlavə etməyimizə imkan verir. Koda baxaq:

```
<script>
function telefon(model,ram){
this.model=model;
this.ram=ram;
}
telefon.prototype.qiymet=1000;
var t1=new telefon("Xiaomi",4);
document.write(t1.qiymet);
</script>
```

Prototip xüsusiyyəti obyektlərə yeni funksiyalar əlavə etməyimizə də imkan verir. Nümunə koda baxaq:

```
<script>
function telefon(model,ram){
this.model=model;
this.ram=ram;
```

```
}
telefon.prototype.funksiya=function(){
return this.ram/4;
}
var t1=new telefon("Xiaomi",4);
document.write(t1.funksiya());
</script>
```

Javascriptdə çoxluqlar

Çoxluqları ilk olaraq riyaziyyatdan xatırlaya bilərik. Çoxluqlar içində birdən çox məlumatın saxlanması üçün elan olunan dəyişənlərdir. Məsələn, riyaziyyatda istifadə etdiyimiz A dəyişəninə baxaq:

A={1,2,5,7,4}

Bu çoxluq içərisində 5 ədəd elementi saxlayan çoxluqdur.

Çoxluqları izah etmək üçün ilk olaraq aşağıdakı nümunəyə baxaq:

```
<script>

var a,b,c,d,e;

a=5;

b=4;

c=7;

d=8;

e=2;

</script>
```

Burada 5 ədəd dəyişən yaradıb hər dəyişənə müxtəlif qiymətlər vermişik. Burada dəyişənlərin sayı yüzlərlə də ola bilər. Bu zaman isə ayrı-ayrı dəyişənlərdən istifadə etmək xeyli çətinləşəcək. Bu zaman çoxluqlar bizim köməyimizə çatacaq və işimizi rahatlaşdıracaq. Aşağıdakı nümunəyə baxaq:

<script>

var a=[1,3,5,2,4];

</script>

Gördüyümüz kimi, burada çox qısa şəkildə çoxluq yaradıb onun daxilinə qiymətlər yazırıq. 5 ədəd ayrı dəyişən yazmaq yerinə dəyişənləri bu şəkildə elan etmək işimizi çox-çox asanlaşdırmaqdadır. Yazılış qaydası isə yuxarıda gördüyümüz kimi çox sadədir. Sadəcə dəyişəni elan etdikdən sonra «[« və «]» işarələri arasında çoxluğun qiymətlərini yazırıq və onları vergüllə ayırırıq. Çoxluğun daxilindəki qiymətə aşağıdakı şəkildə müraciət edirik:

<script>

var a=[1,3,5,2,4];

alert(a[0]);

</script>

Beləliklə çoxluğun ilk qiymətini çap etmiş oluruq. Çoxluqdakı bir qiyməti götürmək üçün sadəcə çoxluğun adını və onun yanında da «[» və «]» işarələri arasında elementin sıra nömrəsini yazırıq. Proqramlaşdırmada sayma 0-dan başladığı üçün ilk qiyməti götürmək üçün a[0] yazmışıq. Məsələn, ikinci qiyməti götürmək üçün a[1] yazmalıyıq.

Eyni zamanda qısaca qeyd edək ki, çoxluqlar obyektlərin xüsusi növləridir.

Çoxluğu aşağıdakı şəkildə də elan edib istifadə etməyimizf mümkündür:

<script>

var a=new Array(1,3,5,2,4);

alert(a[0]);

</script>

Burada çoxluğu yaratmaq üçün **new Array** açar sözlərindən istifadə edirik. Daha əvvəl yazdığımız kodla bu kod eyni işi görməkdədir. Ona görə də, birinci yazılış daha qısa olduğu üçün daha sərfəlidir.

Çoxluq elementinin qiymətini sonradan dəyişməyimiz də mümkündür. Aşağıdakı nümunəyə baxaq:

<script>

var a=[1,3,5,2,4];

a[0]=42;

alert(a[0]);

</script>

Burada **a[0]=42;** yazaraq çoxluğun birinci elementinin qiymətini dəyişmiş oluruq.

Çoxluğun elementlərini ayrı-ayrı götürməkdən əlavə çoxluğunu elementlərini ümumi də ekrana verə bilərik. Aşağıdakı nümunəyə baxaq:

<script>

var a=[1,3,5,2,4];

alert(a);

</script>

Beləliklə çoxluğun elementlə vergüllə ayrılmış şəkildə çap olunmuş olur.

Çoxluğun elementləri dəyişənlərdə olduğu kimi tək ədəd tipində yox, digər tiplərdə də ola bilər. Məsələn:

```
<script>

var
olkeler=["Azerbaycan","Turkiye","Rusiya"];

alert(olkeler[0]);

</script>
```

Bəzi hallarda çoxluğun uzunluğu məlum olmur və onun uzunluğunu tapmağa ehtiyac olur. Bu halda çoxluğun uzunluğunu tapmaq üçün **length** xüsusiyyətindən istifadə olunur.

```
<script>

var
olkeler=["Azerbaycan","Turkiye","Rusiya"];

alert(olkeler.length);

</script>
```

Nəticədə 3 rəqəmi çap olunur. Çünki çoxluğun 3 ədəd elementi var.

Bu xüsusiyyətdən və for operatorundan istifadə edərək elementləri sayı nə qədər olursa olsun,

onları tək-tək çap edə və fərqli vəziyyətlərdə istədiyimiz şəkildə istifadə edə bilərik. Aşağıdakı nümunəyə baxaq:

```
<script>

var
olkeler=["Azerbaycan","Turkiye","Rusiya"];

var uzunluq=olkeler.length;

for(i=0;i<uzunluq;i++){

document.write(olkeler[i]+"<br>");

}

</script>
```

Burada çap etmək üçün **olkeler[i]** yazırıq. Beləliklə hər dövrdə I dəyişəni uyğun rəqəmlə əvəzlənir və çoxluğun uyğun qiyməti çap olunmuş olur.

Bəzən də çoxluq elementlərini azdan çoxa və ya əlifba sırası ilə sıralamağa ehtiyac duyulur. Bu zaman çoxluq üçün **sort()** funksiyasından istifadə olunur.

```
<script>

var olkeler=[3,5,1,2,8];
```

olkeler.sort();

alert(olkeler);

</script>

Nəticədə elementlərin yerlərini dəyişdiyini, azdan çoxa sıralandığını görəcəyik. Elementlər ədəd yox yazı olduqda sıralama əlifba sırasına görə aparılır.

Bəzən çoxluğa yeni element əlavə etmək lazım ola bilər. Bunu **push()** funksiyası ilə edə bilərik. Aşağıdakı nümunəyə baxaq:

<script>

var olkeler=["Azerbaycan","Turkiye","Rusiya"];

olkeler.push("Gurcustan");

alert(olkeler);

</script>

Beləliklə çoxluğa yeni element əlavə olunmuş olur.

Eyni zamanda qeyd edək ki, Javascriptdə çoxluq yaradarkən onun element sayını əvvəlcədən

bilməsək belə yarada bilərik. Aşağıdakı nümunəyə baxaq:

```
<script>

var olkeler=[];

olkeler[0]="Azerbaycan";

olkeler[1]="Turkiye";

olkeler[2]="Rusiya";

alert(olkeler);

</script>
```

Burada **olkeler=[];** yazaraq çoxluq elan etdiyimizi bildiririk. İlk olaraq çoxluğun elementi olmur və boş olur. Riyaziyyat dilində desək bir ədəd boş çoxluq yaratmış oluruq :) Sonradan isə çoxluğa istədiyimiz qədər element əlavə edirik.

Çoxluq funksiyaları

Javascript dilində çoxluqlarla işləmək üçün bir çox funksiyalar mövcuddur. Zaman-zaman bu funksiyalara ehtiyac duyula bilir. Aşağıda bəzi çoxluq funksiyalarına baxaq:

1) **toString()** funksiyası — Çoxluq elementlərini yazı tipinə, yəni string tipinə çevirmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

```
<script>

var
olkeler=["Azerbaycan","Turkiye","Rusiya"];

var yazi=olkeler.toString();

alert(yazi);

</script>
```

Beləliklə çoxluq elementləri vergüllə ayrılaraq string tipinə çevrilir və çap olunur.

2) **join()** funksiyası — Bu funksiya da bundan əvvəlki **toString** funksiyasına oxşar olaraq çoxluq elementlərini birləşdirərək yazıya çevirir. Ancaq, toString funksiyası çoxluq elementlərini birləşdirərkən elementlərin arasına vergül qoyurdu. Bu funksiya isə elementlərin arasına

istənilən simvolu qoya bilir. Aşağıdakı nümunəyə baxaq:

```
<script>

var
olkeler=["Azerbaycan","Turkiye","Rusiya"];

var yazi=olkeler.join(" * ");

alert(yazi);

</script>
```

3) **pop()** funksiyası — Bu funksiya çoxluğun son elementini silmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

```
<script>

var
olkeler=["Azerbaycan","Turkiye","Rusiya"];

olkeler.pop();

alert(olkeler);

</script>
```

Beləliklə çoxluğun son elementi silinir və çap olunur.

4) **push()** funksiyası — çoxluğa yeni element əlavə etmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

```
<script>

var
olkeler=["Azerbaycan","Turkiye","Rusiya"];

olkeler.push("Gurcustan");

alert(olkeler);

</script>
```

Beləliklə çoxluğa yeni element əlavə olunur.

5) **shift()** funksiyası — Bu funksiya çoxluğun ilk elementini silmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

```
<script>

var
olkeler=["Azerbaycan","Turkiye","Rusiya"];

olkeler.shift();

alert(olkeler);

</script>
```

Beləliklə çoxluğun ilk elementi silinmiş olur.

6) **unshift()** funksiyası — Bu funksiya çoxluğun əvvəlinə yeni element əlavə etmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

```
<script>

var
olkeler=["Azerbaycan","Turkiye","Rusiya"];

olkeler.unshift("Gurcustan");

alert(olkeler);

</script>
```

Beləliklə çoxluğun əvvəlinə yeni element əlavə olunmuş olur.

7) splice() funksiyası — çoxluğa yeni elementlər əlavə etmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

```
<script>

var
olkeler=["Azerbaycan","Turkiye","Rusiya"];

olkeler.splice(2,0,"Gurcustan","Amerika","
Erebistan");

alert(olkeler);
```

</script>

Bu metoddakı ilk parametr 2 qiymətini alır. Bu o deməkdir ki, yeni elementlər 2-ci elementdən sonra əlavə olunacaq. İkinci parametr isə 0 qiymətini alır. Bu isə o deməkdir ki, 0 ədəd element silinəcək. Növbəti parametrlər isə çoxluğa əlavə olunacaq qiymətlərdir.

Bu funksiya çoxluqdan elementləri silmək üçün də istifadə oluna bilər. Aşağıdakı nümunəyə baxaq:

<script>

var olkeler=["Azerbaycan","Turkiye","Rusiya"];

olkeler.splice(0,2);

alert(olkeler);

</script>

Beləliklə 0-cı elementdən başlayaraq ilk iki element silinmiş olur.

8) **concat()** funksiyası — Bu funksiya bir neçə çoxluğu birləşdirərək yeni bir çoxluq yaratmaq üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

```
<script>

var
olkeler=["Azerbaycan","Turkiye","Rusiya"];

var
olkeler2=["Amerika","Erebistan","Iran"];

var olkeler3=olkeler.concat(olkeler2);

alert(olkeler3);

</script>
```

Beləliklə olkeler və olkeler2 adlı çoxluğun birləşməsindən olkeler3 adlı yeni bir çoxluq yaranır.

Aşağıdakı nümunədə isə 3 ədəd çoxluğun birləşməsindən yeni bir çoxluq alınır.

```
<script>

var
olkeler=["Azerbaycan","Turkiye","Rusiya"];

var
olkeler2=["Amerika","Erebistan","Iran"];

var olkeler3=["Suriya","Efqanistan"];
```

var olkeler4=olkeler.concat(olkeler2,olkeler3);

alert(olkeler4);

</script>

9) **split()** funksiyası — Bu funksiya bir yazının müəyyən bir simvola görə ayırıb yeni bir çoxluq yaratmaq üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

<script>

var yazi="Baki Azerbaycanin paytaxtidir.";

var array=yazi.split(" ");

alert(array);

</script>

Nəticədə yazı boşluq simvoluna görə parçalanır. Beləliklə də yazının sözlərindən ibarət yeni bir çoxluq yaranır.

10) **reverse()** funksiyası — Çoxluqdakı element ardıcıllığını tərsinə çevirmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

<script>

```
var
olkeler=["Azerbaycan","Rusiya","Turkiye"];

olkeler.reverse();

alert(olkeler);

</script>
```

Nəticədə elementlərin yerləşmə ardıcıllığının tərsinə çevrildiyini görə bilərik.

Window obyekti

Javascript daxilindəki bu obyekt brauzerdəki açılmış pəncərə ilə işləmək üçün nəzərdə tutulur. Bu obyekt vasitəsilə mövcud pəncərə ilə işləmək və onun haqqında məlumatları əldə etmək mümkündür. Məsələn, window obyektindəki close() funksiyasını işlədərək açıq olan pəncərəni bağlaya bilərik. Aşağıdakı nümunəyə baxaq:

```
<button onclick="bagla();">Pencereni bagla</button>

<script>

function bagla(){

window.close();

}

</script>
```

Nəticədə düyməyə basdıqda işlədiyimiz pəncərə bağlanmış olacaq. Maraqlıdır, elə deyilmi? :)

Mövcud pəncərəni bağlamaqdan əlavə yeni bir pəncərə də açа bilərik. Bunun üçün open funksiyasından istifadə edə bilərik. Aşağıdakı nümunəyə baxaq:

<button onclick="yeni();">Yeni pencere aç</button>

<script>

function yeni(){

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

}

</script>

Beləliklə Google yeni pəncərədə açılmış olur.

Eyni zamanda həm open, həm də close funksiyalarından istifadə edərək bir pəncərəni əvvəlcə bir düymə ilə açа, başqa düymə ilə də bağlaya bilərik. Aşağıdakı nümunəyə baxaq:

<button onclick="yeni();">Yeni pencere aç</button>

<button onclick="bagla();">Yeni pencereni bagla</button>

<script>

function yeni(){

pencere=window.open("http://google.az");

}

function bagla(){

pencere.close();

}

</script>

Gördüyümüz kimi burada pəncərəni açarkən sadəcə window.open yox, **pencere=window.open** yazırıq. Burada «pencere» adlı dəyişənimiz yeni açılmış pəncərəni bildirmiş olur. Beləliklə sonradan həmin pəncərə üzərində bu dəyişənin köməkliyi ilə əməliyyatlar apara bilirik.

Pəncərəni bağlayarkən isə **pencere.close();** yazırıq və yeni açdığımız pəncərəni bağlamış oluruq.

İçərisi tamamilə boş olan yeni bir pəncərə yaradıb onun içərisini sonradan doldurmağımız da mümkündür. Aşağıdakı nümunəyə baxaq:

<button onclick="yeni();">Yeni pencere aç</button>

<button onclick="bagla();">Yeni pencereni bagla</button>

<script>

```
function yeni(){

pencere=window.open("","myWindow",
"width=200, height=100");

pencere.document.write("Bu yeni
penceremizdir!");

}

function bagla(){

pencere.close();

}

</script>
```

Beləliklə eni 200, uzunluğu 100 dolan yeni pəncərə yaradırıq və onun içərisinə məlumatlar yazırıq.

Yaratdığımız bir pəncərənin ölçüsünü sonradan dəyişməyimiz mümkündür. Bunun üçün resizeTo() funksiyasından istifadə edə bilərik. Aşağıdakı nümunəyə baxaq:

```
<button onclick="yeni();">Yeni pencere
aç</button>

<button onclick="deyis();">Olcusunu
deyis</button>
```

```
<script>

function yeni(){

pencere=window.open("","myWindow",
"width=200, height=100");

pencere.document.write("Bu yeni
penceremizdir!");

}

function deyis(){

pencere.resizeTo(300,300);

pencere.focus();

}

</script>
```

Beləliklə pəncərənin ölçüsü dəyişmiş olur. Növbəti olaraq **pencere.focus();** yazaraq isə yenidən yeni pəncərəyə fokuslanmış oluruq.

Bu obyekt daxilindəki lazımlı funksiyalardan biri də scrollTo funksiyasıdır. Bu funksiya pəncərənin scrolunun mövqeyini dəyişmək üçündür. Aşağıdakı nümunəyə baxaq:

```
<body style="height: 2000px;"
id="body">

<button
onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

window.scrollTo(0,1000);

}

</script>

</body>
```

Nəticədə düyməyə basdıqda scrolun aşağı endiyini görəcəyik. ScrollTo funksiyasındakı birinci parametr horizontal scrolun mövqeyini, ikinci parametr isə vertical scrolun mövqeyini bildirir. Yuxarıdakı nümunədə biz birinci parametri 0 olaraq qeyd etmişik. Bu o domekdir ki, horizontal scrolun qiyməti 0 olaraq qalaccaq, yəni, dəyişməyəcək. Vertical olaraq isə scrolun qiyməti 1000 olacaq, yəni aşağı enəcək.

Bəzi saytlarda scroll aşağı endikcə müxtəlif əməliyyatların baş verdiyini görə bilərik. Məsələn, bəzi xəbər saytlarında scroll aşağı

endikcə daha çox xəbərlərin yükləndiyini görə bilərik. Bu tip əməliyyatlarda bu funksiyadan istifadə edə bilərik.

Səhifədə aşağı və ya sola getdikdən sonra bəzən scrolun pozisyasını öyrənməyə ehtiyac duyulur. Bu zaman pageXOffset və pageYOffset xüsusiyyətlərindən istifadə etmək olar. Aşağıdakı nümunəyə baxaq:

```
<body style="height: 2000px;"
id="body">

<button onclick="funksiya();"
style="position: fixed;">Deyis</button>

<script>

function funksiya(){

alert(window.pageYOffset);

}

</script>

</body>
```

Səhifəni biraz aşağı endirib düyməyə klik etdikdə yuxarıdan nə qədər məsafədə aşağı endiyimizi piksellə ekranda görə bilərik. PageYOffset xüsusiyyəti tam olaraq vertikal

scrolun yuxarıdan nə qədər məsafədə olduğunu göstərir. PageXOffset isə horizontal scrolun soldan nə qədər məsafədə olduğunu göstərəcək. Bu ədədlərin piksellə göstərildiyini unutmayaq.

Ehtiyac duyulan window obyekti xüsusiyyətlərindən biri də innerWidth və innerHeight xüsusiyyətləridir. Birinci xüsusiyyət pəncərənin enini, ikinci xüsusiyyət isə pəncərənin uzunluğunu göstərir. Bu ölçülərə padding məsafəsi daxildir, ancaq, çərçivə daxil deyil. Aşağıdakı nümunəyə baxaq:

```
<script>

document.write(window.innerWidth +
"<br>" + window.innerHeight);

</script>
```

Beləliklə pəncərənin en və uzunluğunu götürmüş oluruq.

innerWidth və innerHeight xüsusiyyətlərindən əlavə outerWidth və outerHeight xüsusiyyətləri də mövcuddur. Bu xüsusiyyətlərlə əldə olunan ölçüyə en və ya uzunluq, padding ölçüsü, çərçivə və istəyə bağlı olaraq margin ölçüsü də daxildir. Aşağıdakı nümunəyə baxaq:

<script>

**document.write(window.innerWidth + "
" + window.innerHeight);**

**document.write("
");**

**document.write(window.outerWidth + "
" + window.outerHeight);**

</script>

Beləliklə ölçüləri əldə edirik və fərqləri də görmüş oluruq.

Səhifə daxilində neçə ədəd iframe elementinin olduğunu öyrənmək üçün bu obyekt daxilində length xüsusiyyəti mövcuddur aşağıdakı nümunəyə baxaq:

<iframe src="https://google.az"></iframe>

<iframe src="http://shukurhuseynov.com"></iframe>

<script>

document.write(window.length);

</script>

Səhifədə 2 ədəd iframe elementi olduğu üçün 2 çap olunur.

Bəzən səhifədə olan bir məlumatı çap etməyə ehtiyac duyula bilir. Bu zaman print() funksiyasından istifadə oluna bilir. Aşağıdakı nümunəyə baxaq:

```
<button onclick="cap();">Cap et</button>

<br>

Cap olunacaq....

<script>

function cap(){

window.print();

}

</script>
```

Beləliklə düyməyə basdıqda çap etmək üçün pəncərə açılacaq.

Bəzən də səhifədəki bütün məlumatların yox, səhifədəki yalnız bir hissənin, məsələn səhifədəki bir cədvəlin çap olunmasına ehtiyac duyulur. Bunu aşağıdakı şəkildə edə bilərik:

```
<button onclick="cap();">Cap
et</button>

<div id="print">

<table border="1">

<tr><td>Html</td><td>Css</td></tr>

<tr><td>Javascript</td><td>Jquery</td>
</tr>

<tr><td>C++</td><td>Php</td></tr>

</table>

</div>

<script>

function cap(){

yeni=window.open("");

content=document.getElementById("print
").innerHTML;

yeni.document.write(content);

yeni.print();

yeni.close();

}
```

</script>

Burada ilk öncə səhifədə bir cədvəl yaradırıq. Cədvəlimiz div elementinin içərisində yerləşməkdədir. İstifadə çap et düyməsinə basdıqda funksiyamız işə düşür. Funksiya daxilində isə yeni bir pəncərə yaradırıq. Sonra cədvəli yerləşdirdiyimiz div elementinin məzmununu content dəyişəninə ötürürük. Sonra isə bu məzmunu yeni yaratdığımız pəncərənin içərisinə yazırıq. Bundan sonra print() funksiyası vasitəsilə onu çap edirik. Sonda isə həmin səhifəni bağlayırıq. Beləliklə yalnız bir cədvəli çap etmiş oluruq.

Səhifə daxilində bir nöqtədən sonrakı hissənin yüklənməsini dayandırmaq istəyiriksə stop() funksiyasından istifadə edə bilərik. Aşağıdakı nümunəyə baxaq:

Bundan sonrasi yuklenmeyecek...

<script>

window.stop();

</script>

<table border="1">

<tr><td>Html</td><td>Css</td></tr>

<tr><td>Javascript</td><td>Jquery</td></tr>

<tr><td>C++</td><td>Php</td></tr>

</table>

Beləliklə İlk yazıdan sonrakı hissələr yüklənmir və səhifədə görünmür.

Javascript vasitəsilə məlumatları kodlaşdırmaq da mümkündür. Məsələn, btoa funksiyası vasitəsilə yazının base64 alqoritmi ilə kodlaşdırılması mümkündür. Aşağıdakı nümunəyə baxaq:

<script>

var kod=btoa("Hello");

document.write(kod);

</script>

Beləliklə yazı base64 alqoritmi ilə kodlaşdırılmış olur.

Əməliyyatın əksi isə atob funksiyası ilə aparılır. Aşağıdakı nümunəyə baxaq:

<script>

var kod=atob("SGVsbG8=");

document.write(kod);

</script>

Nəticədə Hello yazısının çap olunduğunu görəcəyik. Çünki Hello sözünü btoa funksiyası ilə kodlaşdırdıqda **SGVsbG8=** ifadəsi alınmışdı.

Screen obyekti

Javascriptdə istifadəçinin ekranı haqqında məlumatları özündə saxlayan **screen** obyekti mövcuddur. Bu obyekt bir sıra xüsusiyyətlərdən ibarətdir. Həmin xüsusiyyətlərə bir-bir baxaq.

1) width və height xüsusiyyətləri — istifadəçinin ekranının enini və uzunluğunu bizə piksellə verməkdədir. Aşağıdakı nümunəyə baxaq:

```
<script>

document.write("En: " + screen.width);

document.write("<br>");

document.write("Uzunluq: " +
screen.height);

</script>
```

Beləliklə ekranın enini və uzunluğunu çap etmiş oluruq.

2) availWidth və availHeight xüsusiyyətləri — Windows Taskbar istisna olmaqla ekranın enini və uzunluğunu göstərməkdədir.

```
<script>
```

document.write("En: " +
screen.availWidth);

**document.write("
");**

document.write("Uzunluq: " +
screen.availHeight);

</script>

Beləliklə məlumatları çap etmiş oluruq.

3) colorDepth xüsusiyyəti — Göstərilən təsvirlər üçün rəng palitrasının bit dərinliyini verməkdədir. Nümunə:

<script>

document.write(screen.colorDepth);

</script>

4) pixelDepth xüsusiyyəti - Bu xüsusiyyət istifadəçinin ekranının rəng görüntü imkanını verməkdədir. Nümunə:

<script>

document.write(screen.pixelDepth);

</script>

Beləliklə screen obyektinin xüsusiyyətləri ilə tanış olmuş olduq.

Location obyekti

Location obyekti səhifənin cari URL ünvanı haqqında məlumatları özündə saxlamaqdadır. Obyekt daxilində bir sıra funksiyalar və xüsusiyyətlər mövcuddur. Həmin funksiyalara sıra ilə baxaq:

1) assign funkiyası — başqa bir səhifəni yükləmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

```
<button
onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

location.assign("http://google.az");

}

</script>
```

Beləliklə düyməyə basdıqda yeni səhifə yüklənmiş olur.

2) replace funksiyası — bu funksiya hal-hazırda olan səhifəni başqa biri ilə əvəzləmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

<button onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

location.replace("http://google.az");

}

</script>

Nəticədə düyməyə basdıqda yeni səhifə yüklənmiş olur. Bu funksiyanın əsas xüsusiyyəti ondan ibarətdir ki, bu funksiyadan istifadə edib yeni səhifəyə keçdikdən sonra hal-hazırki səhifəyə qayıtmaq üçün «geri qayıt» düyməsi yaranmır. Yəni əvvəlki səhifəyə geri qayıtmaq olmur.

3) reload funksiyası — cari səhifəni yeniləmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

<button onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

location.reload();

}

</script>

Beləliklə düyməyə basdıqda səhifə yenilənmiş olur.

Yuxarıda baxdığımız funksiyalardan əlavə obyekt daxilində bir sıra xüsusiyyətlər də mövcuddur. Bu xüsusiyyətlərə sıra ilə baxaq.

1) **href** xüsusiyyəti — səhifənin url ünvanını bizə göstərməkdədir. Nümunəyə baxaq:

<script>

document.write(location.href);

</script>

Beləliklə səhifənin ünvanı göstərilmiş olur. Xüsusiyyətə fərqli qiymətlər yazaraq başqa səhifələrə keçid etməyimiz də mümkündür. Aşağıdakı nümunəyə baxaq:

<button onclick="funksiya();">Kecid</button>

<script>

function funksiya(){

location.href="https://google.az";

}

</script>

Beləliklə düyməyə basdıqda Google-a keçmiş oluruq.

2) **host** xüsusiyyəti — Səhifənin yerləşdiyi host adını və portu bizə verməkdədir.

<script>

document.write(location.host);

</script>

Beləliklə domen adını görəcəyik. Hər hansı bir server olmadan sadəcə komputerdə saxladığınız html faylda ünvan görünməyə bilər.

3) **hostname** xüsusiyyəti — səhifənin host adını bizə verməkdədir. Aşağıdakı nümunəyə baxaq:

<script>

document.write(location.hostname);

</script>

Bu xüsusiyyət vasitəsilə səhifənin host adını dəyişəməyimiz də mümkündür. Aşağıdakı nümunəyə baxaq:

<script>

location.hostname="www.google.az";

</script>

Beləliklə host adını dəyişirik və başqa bir sayta keçmiş oluruq.

4) origin xüsusiyyəti — protokolu, host adını və portu bizə verməkdədir. Aşağıdakı nümunəyə baxaq:

<script>

document.write(location.origin);

</script>

5) **pathanme** xüsusiyyəti — səhifənin yerləşdiyi fayl yolunu göstərməkdədir. Aşağıdakı nümunəyə baxaq:

<script>

document.write(location.pathname);

</script>

Beləliklə səhifənin yerləşdiyi fayl yolunu bizə göstərəcək.

6) port xüsusiyyəti — server istifadə etdiyi port nömrəsini bizə göstərir. Nümunə:

<script>

document.write(location.port);

</script>

Port nömrəsi defolt olduqda (əgər bu standart portdursa — 80 və ya 443 kimi) bəzi brauzerlər heç bir şey göstərməyə bilər.

7) protocol xüsusiyyəti — protokolu götürmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

<script>

document.write(location.protocol);

</script>

Beləliklə protokolu ekrana yazdırmış oluruq. Bu xüsusiyyətlə protokolu dəyişməyimiz də mümkündür. Məsələn:

<script>

location.protocol="https:";

</script>

Beləliklə protokolu dəyişmiş oluruq. Protokol dəyərlərinə nümunə olaraq **file:, ftp:, http:, https:, mailto:** və s. göstərə bilərik.

8) **search** xüsusiyyəti — Bu xüsusiyyət vasitəsilə URL ünvanın sorğu hissəsini (querystring) əldə edə və dəyişə bilərik. Real proyektlərdə bu xüsusiyyətlərə ehtiyac duyula bilir.

Səhifənin ünvanını host.html?id=1 şəklində edək və aşağıdakı kodu yazaq:

<script>

document.write(location.search);

</script>

Beləliklə həmin hissənin çap olunduğunu görəcəyik.

Bu xüsusiyyət vasitəsilə həmin hissəni dəyişməyimiz də mümkündür. Məsələn:

<button onclick="funksiya();">Deyis</button>

<script>

function funksiya(){

location.search="?id=5";

}

</script>

Beləliklə düyməyə basıldıqda həmin hissə dəyişmiş olur.

9) **hash** xüsusiyyəti — Bu xüsusiyyət səhifənin URL ünvanında # işarəsindən sonra yazılmış hissəni bizə verməkdədir. Nəticəni diyes işarəsi ilə birgə göstərməkdədir. Səhifənin ünvanının sonuna #javascript yazaq və aşağıdakı kodu yazıb yoxlayaq:

<script>

document.write(location.hash);

</script>

Səhifənin ünvanı aşağıdakı şəkildədir:
file:///home/shukur/Desktop/host.html#javascript

Buna görə də ekrana sadəcə #javascript yazılacaq.

Bu xüsusiyyətdən istifadə edərək dəyəri dəyişmək də mümkündür. Dəyəri dəyişərkən diyes işarəsini yazmağa ehtiyac yoxdur. Aşağıdakı nümunəyə baxaq:

<script>

location.hash="php";

document.write(location.hash);

</script>

Beləliklə URL ünvanda son hissəyə #php yazıldığını və onun ekranda çap olunduğunu görə bilərik.

Navigator obyekti

Bu obyekt brauzerə aid məlumatları özündə saxlamaqdadır. Obyektin xüsusiyyətlərinə sıra ilə baxaq.

1) **appCodeName** xüsusiyyəti — bu xüsusiyyət brauzerin kod adını bizə verməkdədir. Nümunə:

```
<script>
document.write(navigator.appCodeName);
</script>
```

Bütün müasir brauzerlər bu funksiya ilə «Mozilla» qiymətini verməkdədir.

2) **appName** xüsusiyyəti — brauzerin adını verməkdədir. Nümunə:

```
<script>
document.write(navigator.appName);
</script>
```

Firefox, Chrome və Safari brauzerləri «Netspace» qiymətini verməkdədir.

3) appVersion xüsusiyyəti — brauzerin versiya məlumatını bizə göstərməkdədir. Nümunə:

<script>

document.write(navigator.appVersion);

</script>

4) cookieEnabled xüsusiyyət — bu xüsusiyyət brauzerdə çərəzlərin aktiv olub-olmamasını göstərən məntiqi qiymət verməkdədir. Nümunə:

<script>

document.write(navigator.cookieEnabled);

</script>

Əgər brauzerdə çərəzlər aktivdirsə true qiyməti göstəriləcək.

5) language xüsusiyyəti — brauzerin dil versiyasını bizə göstərməkdədir. Nümunə:

<script>

document.write(navigator.language);

</script>

6) onLine xüsusiyyəti — bu xüsusiyyət brauzerin online və ya offline rejimdə olduğunu göstərən bir məntiqi dəyər verməkdədir. Nümunə:

<script>

document.write(navigator.onLine);

</script>

Əgər internet varsa true, yoxdursa false qiyməti göstəriləcək.

7) platform xüsusiyyəti — brauzerin hansı platforma üçün kompilyasiya olunduğunu göstərir. Nümunə:

<script>

document.write(navigator.platform);

</script>

8) product xüsusiyyəti — brauzerin mühərrik adını bizə göstərməkdədir. Nümunə:

<script>

document.write(navigator.product);

</script>

9) userAgeng xüsusiyyəti — bu xüsusiyyət brauzer tərəfindən serverə göndərilmiş user-agent başlığının dəyərini verməkdədir. Nümunə:

<script>

document.write(navigator.userAgent);

</script>

Yuxarıdakı xüsusiyyətlərdən əlavə obyekt daxilində javaEnabled() funksiyası da mövcuddur. Bu funksiya brauzer üçün Javanın aktiv olub-olmadığını göstərən məntiqi dəyər göstərməkdədir. Nümunə:

<script>

document.write(navigator.javaEnabled());

</script>

Qeyd edək ki, Java və Javascript arasında sadəcə ad oxşarlığı mövcuddur. Java dili Oracle şirkətinin sahib olduğu ayrıca bir proqramlaşdırma dilidir.

History obyekti

History obyekti istifadəçinin səyahət etdiyi URL ünvanları özündə saxlamaqdadır. Bu obyekt daxilində bir sıra funksiyalar və xüsusiyyətlər mövcuddur. Onlara sıra ilə baxaq.

1) **back()** funksiyası — Bu funksiyası brauzeri tarix siyahısındakı əvvəlki ünvana qayıtmaq üçün istifadə olunur. Bu funksiya brauzerdəki «geri qayıt» düyməsi ilə eyni işi görməkdədir.

Aşağıdakı nümunəyə baxaq:

```
<button
onclick="funksiya();">Back</button>

<script>

function funksiya(){

history.back();

}

</script>
```

Nəticədə düyməyə basdıqda əvvəlki səhifəyə qayıdacaq. Əgər açdığımız səhifə brauzerdəki ilk səhifədirsə, əvvəlki səhifə olmadığı üçün funksiya işləməyəcək. İlk öncə başqa bir sayta daxil olub, sonra ünvana kodu yazdığımız səhifənin ünvanın yazıb daxil olun. Ondan sonra Back düyməsinə basdıqda əvvəlki sayta qayıtdığımızı görəcəksiniz.

2) **forward()** funksiyası — bu funksiya da bundan əvvəlki funksiyanın yerinə yetirdiyi əməliyyatın əksini yerinə yetirməkdədir. Bu funksiya brauzerdəki «İrəli» düyməsi ilə eyni işi görməkdədir. Aşağıdakı koda baxaq:

```
<button
onclick="funksiya();">Forward</button>

<script>

function funksiya(){

history.forward();

}

</script>
```

Nəticədə düyməyə basdıqda növbəti səhifəyə gedəcək. Əgər növbəti səhifə yoxdursa heç bir şey baş verməyəcək. Əgər səhifə ilkdirsə,

növbəti səhifə yoxdursa, o zaman, səhifəni açdıqdan sonra həmin pəncərədə başqa bir sayta daxil olun və sonra geriyə qayıdın. Bu zaman səhifədəki düyməyə basdıq daxil olduğumuz növbəti sayta getdiyini görəcəksiniz.

3) **go()** funksiyası — bu funksiya brauzerin tarix siyahısındakı növbəti və ya əvvəlki səhifələrə getmək üçündür. Bu funksiya əsas fərqi ondadır ki, bu funksiya ilə tək 1 səhifə əvvələ və ya sonraya deyil, bir neçə səhifə əvvələ və ya sonraya da gedə bilərik. Aşağıdakı nümunəyə baxaq:

```
<button
onclick="funksiya();">next</button>

<script>

function funksiya(){

history.go(2);

}

</script>
```

Burada go funksiyasa parametr olaraq 2 yazmışıq. Bu o deməkdir ki, düyməyə basdıqda növbəti ikinci səhifəyə gedəcək. Məsələn, həmin səhifəni açaq. Həmin səhifədə əvvəlcə bir sayta

girək. Sonra oradan da başqa sayta girək. Sonra iki səhifə əvvələ qayıdaq, yəni yaratdığımız səhifəyə. Bu dəfə düyməyə basdıqda növbəti ikinci səhifəyə getdiyimizi görəcəyik. Əgər bir neçə səhifə əvvələ qayıtmaq istəyiriksə, funksiyaya parametr olaraq mənfi qiymətlər verməliyik. Məsələn, **history.go(-2)** yazsaq, o zaman, əvvəlki ikinci səhifəyə qayıtmış olacağıq.

4) **length** xüsusiyyəti — tarix siyahısındakı URL ünvanların sayını bizə göstərir. Nümunə:

<script>

document.write(history.length);

</script>

Səhifəni açdıqda tarixçədəki URL-lərin sayını görəcəksiniz. Açılmış səhifədə başqa bir sayta daxil olub geriyə qayıtsanız, bu sayın artdığını görəcəksiniz. Göstərilən qiymət ən azı 1 olacaqdır. Çünki başqa saytlara daxil olmasaq belə ən azı səhifənin özünü açmış olacağıq.

String funksiyaları və xüsusiyyətləri

Bəzən proqramlaşdırmada yazı tipli dəyişənlərlə işləməyə ehtiyac duyulur. Bu tip dəyişənlər **string** adlanır. Məsələn, bəzən istifadəçinin yazdığı yazının uzunluğunu bilməyə ehtiyac duyulur. Bəzən isə yazının bir hissəsini kəsməyə ehtiyac duyulur. Bu tip hallardan xüsusi funksiyalardan istifadə olunur.

Javascriptdə də yazı tipli dəyişənlərlə işləmək üçün bir sıra funksiyalar mövcuddur. Məsələn, istifadəçi bir yazı daxil edib və bizə o yazının hərf sayı, yəni, uzunluğu lazımdır. Bu zaman bizə **length** xüsusiyyəti lazımdır. Aşağıdakı nümunəyə baxaq.

```
<script>

var ad,uzunluq;

ad=prompt("Adiniz:");

uzunluq=ad.length;

document.write(uzunluq);

</script>
```

Burada **uzunluq=ad.length** yazaraq ad dəyişənindəki yazının uzunluğunu **uzunluq** dəyişəninə ötürürük və sonra da onu çap edirik.

İndi isə fərz edək ki, istifadəçidən istifadəçidən adını yazmağı tələb edirik. Ancaq, istifadəçi adı 3 simvoldan böyük olmalıdır. Bu zaman istifadəçi adının uzunluğunu şərt oepratoru ilə yoxlamalıyıq. Aşağıdakı nümunəyə baxaq:

```
<script>

var ad,uzunluq;

ad=prompt("Adiniz:");

uzunluq=ad.length;

if(uzunluq<=3) document.write("Istifadeci adi 3 simvoldan boyuk olmalidir.");

</script>
```

Burada istifadəçi adının uzunluğu şərt operatoru ilə yoxlanılır və 3-dən kiçik bərabər olduqda, yəni, istifadəçi adının uzunluğu 3-dən kiçik olduqda və ya 3-ə bərabər olduqda ona xəbərdarlıq olunur.

Əlavə olaraq Javascriptdə bir çox **string** funksiyaları mövcuddur. Onlara sıra ilə baxaq

1) **indexOf** funksiyası – bu funksiya axtarılan yazının birincisinin neçənci simvoldan başladığını göstərir. Aşağıdakı nümunəyə baxaq:

<script>

var x="Javascript mohtesem dildir.
Hemcinin html dili de mohtesemdir.";

document.write(x.indexOf("mohtesem"));

</script>

Burada nəticədə 11 yazılır. Çünki ilk **"mohtesem"** sözü 11-ci simvoldan sonra başlayır. Cümlə daxilində iki ədəd eyni sözdən yazılmağına baxmayaraq, bu funksiya yalnız həmin sözlərdən birincisinin neçənci simvoldan sonra olduğunu göstərir. Əgər axtarılan söz cümlədə olmasa -1 qiyməti qaytarılır.

2) **lastIndexOf** funksiyası – bu funksiya axtarılan yazının sonuncusunun neçənci simvoldan başladığını göstərir. Aşağıdakı nümunəyə baxaq:

<script>

**var x="Javascript mohtesem dildir.
Hemcinin html dili de mohtesemdir.";**

**document.write(x.lastIndexOf("mohtesem"
));**

</script>

Burada “mohtesem” sözü iki dəfə işlənib. Bu funksiya isə sözlərdən sonuncusunun neçənci simvoldan sonra başladığını göstərir. Ona görə də 50 çap olunur. Əgər axtarılan söz cümlədə yoxdursa, o zaman, -1 qiyməti qaytarılır.

3) **search()** funksiyası – bu funksiya cümlə daxilində axtarılan sözün birincisinin neçənci simvoldan sonra başladığını tapmaq üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

<script>

**var x="Javascript mohtesem dildir.
Hemcinin html dili de mohtesemdir.";**

document.write(x.search("mohtesem"));

</script>

Nəticədə 11 çap olunur, çünki axtarılan yazının birincisi 11-ci simvoldan sonra başlayır.

Diqqət etsək **search** və **indexOf** funksiyalarının çox oxşar olduğunu görə bilərik. Hər ikisində də bir sözü axtarırıq və onun yerini tapırıq. Ancaq, əslində bu funksiyaların bir sıra fərqləri var ki, bu fərqlərə də gələcək mövzularda baxacağıq.

4) **slice()** funksiyası – Mətnin qeyd olunan hissəsini kəsib götürmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

```
<script>

var x="Html, Css, Javascript";

document.write(x.slice(6,9));

</script>
```

Beləliklə, x dəyişəninə mənimsədilmiş mətnin 6-cı simvolundan başlayaraq 9-cu simvoluna qədər hissə kəsilərək çap edilir. Burada 6-cı simvoldan sonrakı hissə nəzərdə tutulur. Eyni zamanda, x dəyişəninin özündə heç bir dəyişiklik baş vermir.

Funksiya mənfi qiymətli parametrlər alarsa, o zaman, hissə sondan başlayaraq kəsilir. Aşağıdakı nümunəyə baxaq.

```
<script>
```

var x="Html, Css, Javascript";

document.write(x.slice(-15,-12));

</script>

Burada birinci parametr -15 qiymətini alır. Bu o deməkdir ki, kəsməyə cümlənin sonundan başlayaraq 15-ci simvoldan başlanılır. İkinci parametr isə -12 qiymətini alır. Bu da o deməkdir ki, kəsməyə sondan 15-ci simvoldan başlanıb, sondan 12-ci simvolda bitiririk.

Əgər funksiya tək bir parametr alarsa, o zaman, həmin parametrin aldığı qiymətdən başlayaraq cümlənin qalan hissəsi oxunur. Aşağıdakı nümunəyə baxaq:

<script>

var x="Html, Css, Javascript";

document.write(x.slice(5));

</script>

Beləliklə, 5-ci simvoldan sonrakı hissə kəsilib götürülmüş olur.

6) **substring()** funksiyası – Bu funksiya da mətnin müəyyən hissəsini kəsib götürmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

<script>

var x="Html, Css, Javascript";

document.write(x.substring(6,9));

</script>

Beləliklə 6-cı simvoldan 9-cu simvolda qədər olan hissə oxunmuş olur. Burada 6-cı simvol nəzərə alınmır. Bu funksiya **slice()** funksiyasından fərqli olaraq mənfi qiymətlər ala bilməz. Funksiyanın ikinci parametri yazılmaya da bilər. Bu zaman, birinci qiymətdən sonrakı hissələr kəsilmiş olacaq.

7) **substr()** funksiyası – Bu funksiya da mətnin bir hissəsini kəsib götürmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq.

<script>

var x="Html, Css, Javascript";

document.write(x.substr(6,10));

</script>

Burada 6-cı simvoldan başlayaraq növbəti 10 simvol kəsilib götürülmüş olur.

8) **replace()** funksiyası – Bu funksiya mətndəki bir sözü başqa bir sözlə əvəzləmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

```
<script>

var x="Html is very nice.";

var y=x.replace("Html","Javascript");

document.write(y);

</script>
```

Beləliklə, x dəyişənindəki “Html” sözü “Javascript” sözü ilə əvəzlənir və y dəyişəninə mənimsədilir. Burada x dəyişənində isə heç bir dəyişiklik baş vermir. Sözlər əvəzlənərkən böyük və kiçik hərflər fərqləndirilməkdədir.

9) **toUpperCase()** funksiyası – Yazıdakı kiçik hərfləri böyük hərflərə çevirmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

```
<script>

var x="Html is very nice.";

var y=x.toUpperCase();

document.write(y);
```

</script>

Nəticədə yazıdakı kiçik hərflər böyük hərflərə çevrilmiş olur.

10) **toLowerCase()** funksiyası – Yazıdakı böyük hərfləri kiçik hərflərlə əvəzləmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

<script>

var x="HTML is VERY nice.";

var y=x.toLowerCase();

document.write(y);

</script>

Nəticədə cümlədəki böyük hərflər kiçik hərflərlə əvəzlənmiş olur.

11) **concat()** funksiyası – Bir yazının sonuna başqa yazını əlavə etmək üçün istfadə olunur. Aşağıdakı nümunəyə baxaq:

```
<script>

var yazi1="Html";

yazi2=yazi1.concat(", Css");

document.write(yazi2);

</script>
```

Nəticədə “yazi1” dəyişəninin sonuna “, Css” yazısı artırılaraq “yazi2” dəyişəninə mənimsədilmiş olur.

12) **trim()** funksiyası – Yazının sağ və sol tərəfindəki boşluqları silmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

```
<script>

var yazi1="    Html    ";

yazi2=yazi1.trim();

alert(yazi2);

</script>
```

Nəticədə “yazi1” dəyişəninin sağ və sol tərəfindəki boşluqlar silinərək “yazi2” dəyişəninə mənimsədilir.

13) **charAt()** funksiyası – Yazının qeyd etdiyimiz nömrədəki simvolunu götürmək üçün istifadə olunur. Aşağıdakı nümunəyə baxaq:

```
<script>

var yazi1="Html";

yazi2=yazi1.charAt(0);

document.write(yazi2);

</script>
```

Nəticədə “yazi2” dəyişəninə “yazi1" dəyişəninin 0-cı simvolu mənimsədilir. Proqramlaşdırmada saymanın sıfırdan başladığını qeyd edək. Bu səbəbdən birinci simvolu götürmək üçün 0 yazmalıyıq.

14) charCodeAt() funksiyası – Qeyd edilən nömrədəki simvolun UTF-16 kodunu göstərir. Aşağıdakı nümunəyə baxaq.

```
<script>

var yazi1="Html";
```

yazi2=yazi1.charCodeAt(0);

document.write(yazi2);

</script>

Nəticədə 0-cı simvolun UTF-16 kodu çap olunmuş olur.

Cookies

Çərəzlər, yəni cookies istifadəçinin brauzerində məlumatların saxlanılması üçün istifadə olunmaqdadır. Məsələn, Javascriptdə müəyyən dəyişənlər elan edib onlardan istifadə edirik. Ancaq, brauzeri bağlayıb növbəti dəfə açdıqda daha əvvəl səhifədə qeyd etdiyimiz məlumatlar itmiş olur. Məsələn, formdan istifadəçi adını götürüb ekrana yazırıq. Ancaq, brauzerdən çıxıb yenidən girdikdə həmin adın itdiyini görəcəyik. Çərəzlərdən istifadə edərək isə həmin məlumatı uzun müddət yaddaşda saxlaya bilərik.

Qeyd edək ki, çərəzlərdən istifadə etmək üçün komputerinizdə localhost quraşdırmağınız və ya kodları bir hostingdə yazmağınız lazımdır. Komputerinizdə localhost üçün xampp server, wamp server və s. quraşdıra bilərsiniz.

Çərəzlərdən istifadə edərək məlumatı brauzerdə yadda saxlamaq üçün document.cookie xüsusiyyətindən istifadə edə bilərik. Aşağıdakı nümunəyə baxaq:

```
<script>

document.cookie = "name=Shukur;
expires=Thu, 18 Dec 2025 12:00:00 UTC";
```

alert(document.cookie);

</script>

Beləliklə dəyəri “Shukur” olan, adı isə “name” olan bir ədəd cookie yaratmış oluruq. Yazdığımız digər hissə isə cookie dəyərinin silinmə vaxtıdır.

expires=Thu, 18 Dec 2025 12:00:00 UTC

Beləliklə cookie 2025-ci ilə qədər yaddaşda qalacaqdır. Ancaq, qeyd edək ki, istifadəçi öz brauzerindəki cookie dəyərini istədiyi vaxt silə bilər.

İndi isə həmin səhifədəki kodu dəyişək:

<script>

alert(document.cookie);

</script>

Kodu dəyişib yadda saxlayaq. Növbəti olaraq brauzerdən çıxaq və səhifəni yenidən açaq. Bu dəfə daha əvvəl yadda saxladığımız cookie dəyərinin ekrana çıxdığını görəcəyik. Halbuki ikinci dəfə çərəzi yaratmamışdıq. Çərəzlərin əsas məntiqi də elə budur. Bir dəfə çərəzi yaradırıq, artıq o brauzerin yaddaşında qalmış

olur. Növbəti dəfələrdə isə həmin yaddaşda qalmış məlumatı götürüb istifadə edə bilirik.

İndi isə gəlin form elementindən input dəyərini götürək və onu yadda saxlayaq.

```
<input type="text" id="yazi">

<button onclick="yarat();">Cookie yarat</button>

<button onclick="bax();">Melumata bax</button>

<script>

function yarat(){

var yazi=document.getElementById("yazi").value;

document.cookie = "name=" + yazi + "; expires=Thu, 18 Dec 2025 12:00:00 UTC";

alert("Cookie yaradildi.");

}

function bax(){

alert(document.cookie);
```

}

</script>

Beləliklə düyməyə basdıqda inputa daxil etdiyimiz dəyər yadda saxlanılacaq. Məlumata bax düyməsini basdıqda isə həmin cookie dəyəri görünəcək. Bundan sonra brauzeri bağlayaq və yenidən açaq. Bu dəfə forma məlumat daxil etməyək və sadəcə Məlumata bax düyməsinə basaq. Beləliklə, daha əvvəl yadda saxladığımız cookie dəyərini görəcəyik.

Bir neçə ədəd cookie dəyərini yaratmağımız da mümkündür. Nümunəyə baxaq:

<script>

document.cookie = "name=Shukur;expires=Thu, 18 Dec 2025 12:00:00 UTC";

document.cookie = "surname=Huseynov;expires=Thu, 18 Dec 2025 12:00:00 UTC";

alert(document.cookie);

</script>

Cookie dəyərləri çap olunarkən isə bir yerdə çap olunacaq.

name=Shukur; surname=Huseynov

Təkcə adı götürmək üçün isə cookie dəyərini parçalamağımız lazımdır. Bunun üçün cookie dəyərini “;” simvoluna görə bölməli və ilk hissəni götürməliyik. Nümunəyə baxaq:

```
<script>

document.cookie =
"name=Shukur;expires=Thu, 18 Dec 2025
12:00:00 UTC";

document.cookie =
"surname=Huseynov;expires=Thu, 18 Dec
2025 12:00:00 UTC";

var x=document.cookie.split(";")[0];

alert(x);

</script>
```

Beləliklə təkcə “name=Shukur” hissəsini götürmüş oluruq. Əgər sadəcə adı götürmək istəyiriksə, o zaman bu hissəni də “=” simvoluna görə parçalamağımız lazımdır.

```
<script>
```

document.cookie = "name=Shukur;expires=Thu, 18 Dec 2025 12:00:00 UTC";

document.cookie = "surname=Huseynov;expires=Thu, 18 Dec 2025 12:00:00 UTC";

var x=document.cookie.split(";")[0].split("=")[1];

alert(x);

</script>

Beləliklə həmin hissəni də “=” işarəsinə görə parçalayıb ikinci hissəni götürmüş oluruq. Beləliklə təkcə adı alırıq.

Yaratdığımız cookie dəyərini silmək üçün onun bitmə vaxtını keçmiş vaxta yazmağımız kifayətdir. Nümunə:

<script>

document.cookie = "name=;expires=Thu, 01 Jan 1970 00:00:01 GMT;";

alert(document.cookie);

</script>

Beləliklə “name” adlı cookie dəyərini silmiş oluruq.

Çərəzi yaratdığımız qaydada ona yeni dəyər verərək onun dəyərini dəyişə bilərik. Aşağıdakı nümunəyə baxaq:

<script>

document.cookie = "name=yeni_qiymet; expires=Thu, 18 Dec 2025 12:00:00 UTC";

alert(document.cookie);

</script>

Bu şəkildə cookie dəyərini dəyişmiş oluruq.

ECMAscript 6

ECMAScript 6 (ES6) javascript dili üçün yeni nəsil bir standartdır və 2015-ci il iyunda yayımlanmışdır. Məqsədlərindən biri Javascript dilini böyük, qarışıq tətbiqlər yazmaq üçün uyğun hala gətirməkdir. ES6 ilə bərabər bir sıra xüsusiyyətlər gəlmişdir ki, onlara da növbəti mövzularda baxacağıq. ECMAScript haqqında daha ətraflı internet üzərindən araşdıra bilərsiniz.

Let və Const ifadələri

Let və const ifadələri də var ifadəsi ilə eyni olaraq dəyişən yaratmaq üçün istifadə olunur. Sadəcə bu ifadələrin var ifadəsindən biraz fərqi var. İlk olaraq let ifadəsini nəzərdən keçirək. Bu ifadənin yazılışı var ifadəsi ilə eynidir. Nümunəyə baxaq:

```
<script>

let x=5;

alert(x);

</script>
```

Burada sadəcə dəyişən elan olunur. Bu ifadənin var ifadəsindən fərqini göstərmək üçün aşağıdakı koda baxaq:

```
<script>

var x=4;

var x=5;

alert(x);

</script>
```

Bu kod heç bir problemsiz işləyəcək. Ancaq, let açar sözündə bir adda dəyişəni bir dəfə elan edə bilirik. Yəni, let ilə dəyişəni ikinci dəfə elan etməyə çalışsaq xəta alarıq. Koda baxaq:

```
<script>

let x=4;

let x=5;

alert(x);

</script>
```

Bu kod xəta verəcək və işləməyəcək. Çünki let ilə iki dəfə eyni adda dəyişən yaratmağa çalışmışıq. Gördüyümüz kimi let ilə bir adda dəyişəni iki dəfə yarada bilmirik. Ancaq ikinci dəfə yarada bilməsək də, qiymətini sonradan dəyişə bilirik. Nümunəyə baxaq:

```
<script>

let x=4;

x=5;

alert(x);

</script>
```

Burada kod heç bir problemsiz işləyəcək. Beləliklə let və var ifadələrinin fərqini görmüş oluruq.

Həmçinin, let ifadəsi ilə dəyişənləri bloklar arasında da yarada və orada istifadə edə bilərik. Nümunəyə baxaq.

```
<script>

{

let x=5;

document.write(x);

}

</script>
```

Gördüyümüz kimi dəyişəni bloklar arasında yaradırıq və istifadə edirik. Ancaq həmin dəyişənə blokdan kənarda müraciət etmək mümkün olmur. Nümunəyə baxaq:

```
<script>

{

let x=5;

}
```

document.write(x);

</script>

Nəticədə bu kodda xəta alacağıq və bizə x dəyişəninin təyin olunmadığını bildirəcək.

Yuxarıdakılardan əlavə, let ifadəsinin digər fərqləri də mövcuddur. Məsələn, let ilə if operatorunun daxilində dəyişən yaratdıqda kənardan müdaxilə mümkün olmur. Nümunəyə baxaq:

<script>

if(4>3){

let x=5;

}

alert(x);

</script>

Burada x təyin olunmadığı xətası verərək kod işləməyəcək. Ancaq let yerinə var işlətsək kod heç bir problemsiz işləyəcək.

İndi isə dövrlər daxilində baxaq. Aşağıdakı kodu nəzərdən keçirək.

<script>

for(let i=0;i<=10;i++){

}

alert(i);

</script>

Nəticədə kodda xəta alacağıq və bizə i dəyişənin təyin olunmadığını bildirəcək. Ancaq let yerinə var işlətsə idik, o zaman, heç bir problemsiz işləyəcəkdi. Yəni, let ilə yaratdıqda dövrə dəyişəninə kənardan müraciət mümkün olmur.

İndi isə const ifadəsinə baxaq. Bu ifadə sabit dəyərlər üçündür. Yazılış qaydasına baxaq:

<script>

const x=4;

alert(x);

</script>

Ancaq const ifadəsinin əsas fərqi ondadır ki, onunla yaradılmış parametrə yeni dəyər ötürmək olmaz. Nümunəyə baxaq:

```
<script>

const x=4;

x=5;

alert(x);

</script>
```

Bu kodda xəta alacağıq. Çünki parametrə yalnız bir dəfə qiymət verə bilirik. Const ifadəsi let ifadəsi ilə oxşar xüsusiyyətləriu daşımaqdadır. Aralarındakı fərq sadəcə const ilə yaradılmış parametrin dəyərini sonradan dəyişmək olmamasıdır. Eyni zamanda const ilə yaradılmış parametrin dəyəri elan olunarkən verilməlidir. Məsələn, aşağıdakı kodda xəta alacağıq.

```
<script>

const x;

x=5;

document.write(x);
```

</script>

Çünki const ilə parametr elan olunarkən ona dəyər verilməlidir. Doğru kod aşağıdakı şəkildədir:

<script>

const x=5;

document.write(x);

</script>

Bu kod heç bir problemsiz işləyəcəkdir.

Önəmli bir qeydi də diqqətinizə çatdırmaq istəyirəm. Bəzi mənbələrdə const ifadəsi dəyişənlər mövzusunun içərisində getməkdədir. Ancaq const ilə yaradılmış parametrlər dəyişən deyil. Bunun səbəbi dəyişənlərin zamandan və ya nədənsə asılı olaraq dəyişə bilmə xüsusiyyətini daşımasıdır. Buna görə də adı dəyişəndir. Const ilə yaradılmış parametrlər isə sonradan dəyişə bilmir. Buna görə də const barəsində danışarkən dəyişən yox, sadəcə yaradılmış parametr kimi danışdıq.

Öncədən qəbul edilmiş parametrlər

Öncədən qəbul edilmiş parametrlər vasitəsilə (Default parameters) funksiya parametrlərinə öncədən qiymət verə bilirik. Beləliklə funksiya çağırıldıqda funksiyaya parametr ötürülməsə, o zaman, öncədən qəbul edilmiş parametrlər istifadə olunacaq. Nümunəyə baxaq:

```
<script>
function f(x=5){
document.write(x);
}
f();
</script>
```

Burada funksiyanı çağırarkən heç bir parametr ötürmürük. Buna görə də parametr olaraq öncədən qəbul edilmiş 5 qiyməti götürülür və x 5-ə bərabər olur. Aşağıdakı koda da baxaq:

```
<script>
function f(x=5){
document.write(x);
}
f(8);
</script>
```

Burada artıq 5 ləğv olunur və x-ə 8 ötürülmüş olur. Birdən çox parametr üçün də eyni şeyi tətbiq edə bilərik:

```
<script>
function f(x=5,y=7){
document.write(x+y);
}
f();
</script>
```

Beləliklə x=5 və y=7 olaraq qəbul olunacaq.

Bir öncədən qəbul etiş parametr digəri tərəfindən də istifadə oluna bilər Nümunəyə baxaq:

```
<script>
function f(x=5,y=x+3){
document.write(x+y);
}
f();
</script>
```

Beləliklə y-in qiyməti 8 olmuş olur.

Nəticə olaraq deyə bilərik ki, öncədən qəbul edilmiş parametrlər çox əlverişlidir. Əgər, funksiya çağırılan zaman ötürülən parametrlərdən bəziləri olmasa belə, öncədən qəbul edilmiş parametrlər əməliyyatların yerinə yetirilməsini təmin edəcək.

Çoxlu dəyər mənimsətmə

Çoxlu dəyər mənimsətmə (Destruction Assignment) çoxluqdakı elementləri asanlıqla adi dəyişənlərə mənimsətməyə imkan verir. Nümunəyə baxaq:

```
<script>
var ededler=[5,4,3];
var [a,b,c]=ededler;
document.write(a+"<br>"+b+"<br>"+c);
</script>
```

Beləliklə çoxluqdakı ilk qiymət a dəyişəninə, ikinci qiymət b dəyişəninə, üçüncü qiymət isə c dəyişəninə ötürülmüş oldu.

Qısaldılmış funksiyalar

Qısaldılmış funksiyalar (Arrow functions – hərfi tərcümə deyil) funksiyaları daha qısa şəkildə elan etməyə icazə verir. Nümunəyə baxaq:

```
<script>

var f=()=> {

return 5;

}

document.write(f());

</script>
```

Beləliklə daha qısa şəkildə f() funksiyasını yaratmış oluruq. Mötərizələr arasında isə parametrlər yer alacaq. Nümunəyə baxaq:

```
<script>

var f=(x,y)=> {

return x+y;

}
```

document.write(f(5,4));

</script>

Beləliklə funksiyaya parametr ötürmüş oluruq. Bu yazılışn setInterval() funksiyası daxilində də istifadəsinə baxaq:

<script>

**setInterval(f=()=>{document.write("Sala m
");},1000);**

</script>

Beləliklə hər dəfə function sözündən istifadə etməyimizə ehtiyac qalmır. Burada funksiyanın adını yazmasaq da olar. Nümunə:

<script>

setInterval(()=>{document.write("Salam< br>");},1000);

</script>

Beləliklə yazılışı daha da qısalaşdırmış oluruq.

Qarışıq yazılar

Qarışıq yazılar yazarkən, çox dırnaq işarəsi işlətməyə ehtiyac duyarkən bəzən problemlər yarana bilir. Nümunəyə baxaq:

<script>

document.write("Bu "Javascript" dilidir.");

</script>

Burada dırnaqlar bir-birinə qarışır və problem yaranır. Onun üçün də, ` (klaviaturada esc düyməsinin aşağısında yerləşir) simvolundan istifadə edə bilərik. Nümunəyə baxaq:

<script>

document.write(`Bu "Javascript" dilidir.`);

</script>

Burada artıq problem yaşanmamaqdadır.

Eyni zamanda dəyişənləri mətnin içərisində aşağıdakı şəkildə də yaza bilərik:

<script>

var x="Javascript";

document.write(`Bu ${x} dilidir.`);

</script>

Bu şəkildə dəyişəni yazının içərisində daha rahat şəkildə istifadə edə bilirik. Hətta bu üsulla hesablama kimi işləri də apara bilərik. Nümunəyə baxaq:

<script>

var x=5;

var y=6;

document.write(`Hesablamanin neticesi ${x+y} olacaq.`);

</script>

Nəticədə “Hesablamanin neticesi 11 olacaq.” yazılacaq.

Çoxluqlara müraciət

Burada işlədəcəyimiz üsulla (Spread operator) çoxluqlarla daha rahat işləmək mümkündür. Nümunəyə baxaq:

<script>

```
var array=[5,4,3];

document.write(...array);

</script>
```

Gördüyümüz kimi, adın əvvəlinə üç nöqtə qoyaraq çoxluğa müraciət etmiş oluruq. Başqa bir nümunəyə baxaq:

```
<script>

f=(x,y,z)=>{

return x+y+z;

}

var array=[5,4,3];

document.write(f(...array));

</script>
```

Beləliklə çox rahat şəkildə çoxluq elementlərini funksiyaya parametr olaraq ötürmüş oluruq. Nəticədə x=5, y=4, z=3, funksiyanın nəticəsi isə 12 olmuş olur.

Bu üsulla çoxluqları çox rahat şəkildə birləşdirməyimiz də mümkündür. Nümunəyə baxaq:

<script>

var array1=[5,4,3];

var array2=[7,9,10];

var array3=[...array1,...array2];

document.write(...array3);

</script>

Beləliklə iki çoxluğu birləşdirib elementlərini çap etmiş oluruq.

For..of operatoru

Bu operator dövrləri daha rahat şəkildə yazmaq üçün istifadə olunur. Nümunəyə baxaq:

<script>

```
var x="Javascript";
for(i of x){
document.write(i+"<br>");
}
</script>
```

Beləliklə hərflər alt-alta yazılmış olacaq. Başqa bir nümunəyə baxaq:

```
<script>
var x=[5,6,7];
for(i of x){
document.write(i+"<br>");
}
</script>
```

Beləliklə çoxluq elementləri alt-alta sıralanmış olur.

Set obyekti

Bu obyekt rahat şəkildə elementləri saxlamağa imkan verir. Nümunəyə baxaq:

```
<script>

let s=new Set();

s.add("Yeni element");

s.add("Ikinci element");

console.log(s);

</script>
```

Burada add() funksiyası vasitəsilə rahatlıqda obyektə element əlavə edə bilirik. Bir şeyi qeyd edək ki, bu funksiya təkrar elementləri obyektə əlavə etmir. Nümunəyə baxaq:

```
<script>

let s=new Set();

s.add("Yeni element");

s.add("abc");

s.add("abc");

console.log(s);
```

</script>

Nəticədə obyektdə yalnız bir ədəd “abc” dəyəri olduğunu görəcəyik. Çünki bu funksiya təkrar dəyər əlavə etmir.

Bundan əlavə delete() funksiyası da mövcuddur ki, onunla obyektdən elementləri silməyimiz mümkündür. Nümunəyə baxaq:

<script>

let s=new Set();

s.add("Yeni element");

s.add("abc");

console.log(s);

s.delete("abc");

console.log(s);

</script>

Beləliklə, əvvəl olunmuş “abc” dəyərli element sonradan silinmiş olur.

Bunlardan əlavə size parametri də mövcuddur. Bu parametr obyektdə olan element sayını verir. Nümunə:

<script>

let s=new Set();

s.add("Yeni element");

s.add("abc");

console.log(s.size);

</script>

Nəticədə 2 çıxacaq. Bundan əlavə obyektdəki bütün dəyərləri silən clear() funksiyası da mövcuddur. Nümunəyə baxaq:

<script>

let s=new Set();

s.add("Yeni element");

s.add("abc");

s.clear();

console.log(s);

</script>

Beləliklə sonda obyektdən bütün dəyərlərin silinmiş olduğunu görəcəyik. Əlavə olaraq, has() funksiyası da mövcuddur ki, bu funksiya obyekt

içərisində bir dəyərin olub-olmadığını yoxlayır. Nümunəyə baxaq:

```
<script>

let s=new Set();

s.add("Yeni element");

s.add("abc");

console.log(s.has("abc"));

</script>
```

Əgər, “abc” yazısı obyektdə varsa, o zaman, true qiymətini verir. Əgər, olmasa false qiymətini verər. Aşağıdakı nümunəyə də baxaq:

```
<script>

let s=new Set();

s.add("Yeni element");

s.add("abc");

if(s.has("abc")){

alert("Movcuddur.");

}

else{
```

alert("Movcud deyil.");

}

</script>

Burada da element obyektin içərisində mövcud olduğu üçün mövcud olduğu bildiriləcək.

Map obyekti

Bu obyekt set obyektinə oxşar olsa da, bir sıra fərqləri mövcuddur. Məsələn, burada obyektə dəyər əlavə etdikdə açar söz də əlavə olunmalıdır. Nümunəyə baxaq:

```
<script>
let s=new Map();
s.set("adi","deyeri");
s.set("Proqramlasdirma dili","C/C++");
console.log(s);
</script>
```

Burada set() funksiyası ilə obyektə dəyər əlavə olunur. Funksiyadakı birinci parametr dəyərin açar sözü, ikinci parametr isə özüdür. Burada da has() funksiyası mövcuddur və dəyərin olub-olmadığını yoxlayır. Nümunə:

```
<script>
```

```
let s=new Map();
s.set("adi","deyeri");
s.set("Proqramlasdirma dili","C/C++");
if(s.has("Proqramlasdirma dili")){
document.write("Var");
}
</script>
```

Gördüyümüz kimi, burada yoxlama açar sözlə aparılır. Bundan delete() funksiyası burada da silmə əməliyyatını aparmaqdadır. Nümunə:

```
<script>
let s=new Map();
s.set("adi","deyeri");
s.set("Proqramlasdirma dili","C/C++");
s.delete("adi");
console.log(s);
```

</script>

Gördüyümüz kimi, silinmə açar sözə görə aparılmaqdadır.

Bu obyekt daxilindəki əsas funksiyalardan biri də get() funksiyasıdır. Bu funksiyanın içərisinə açar sözün adını yazırıq və bizə onun aldığı dəyəri verir. Nümunə:

```
<script>
let s=new Map();
s.set("adi","deyeri");
s.set("Proqramlasdirma dili","C/C++");
console.log(s.get("Proqramlasdirma dili"));
</script>
```

Beləliklə açar sözü yazıb dəyərini əldə etmiş oluruq.

Bu obyekt daxilində keys() funksiyası mövcuddur ki, bizə açar sözləri verməkdədir. Nümunə:

```
<script>
let s=new Map();
s.set("adi","deyeri");
s.set("Proqramlasdirma dili","C/C++");
console.log(s.keys());
</script>
```

Nəticədə bizə sadəcə açar sözləri verəcək.

Obyekt daxilində values() funksiyası dəyərləri bizə verməkdədir. Nümunə:

```
<script>
let s=new Map();
s.set("adi","deyeri");
s.set("Proqramlasdirma dili","C/C++");
console.log(s.values());
```

</script>

Beləliklə yalnız dəyərlər bizə verilməkdədir.

Obyekt daxilindəki clear() funksiyası mövcud dəyərləri silməkdədir. Nümunə:

<script>

let s=new Map();

s.set("adi","deyeri");

s.set("Proqramlasdirma dili","C/C++");

s.clear();

console.log(s);

</script>

Beləliklə, sonda obyektin boş olduğunu görəcəyik. Çünki dəyərlər silinmişdir.

Obyekt daxilindəki size parametri obyektdəki element sayını bizə verməkdədir. Nümunə:

```
<script>
let s=new Map();
s.set("adi","deyeri");
s.set("Proqramlasdirma dili","C/C++");
console.log(s.size);
</script>
```

Beləliklə obyekt sayını almış oluruq.

Qeyd: Əlavə olaraq WeakMap və WeakSet obyektlərini də araşdırın.

İkilik və səkkizlik ədədlər

ES 6 ilə birlikdə Javascript daxilində ikilik və səkkizlik say sistemindəki ədədləri də yazmağımız mümkündür. İkilik say sistemindəki ədədi yazmaq üçün sadəcə qarşısına **0b** yazmaq kifayətdir. Nümunə:

<script>

document.write(0b101011);

</script>

Nəticədə 43 çap olunacaq. Çünki yazılan ədədin qarşılığı 43-dür.

8-lik sayt sistemində ədədi yazmaq üçün isə sadəcə qarşısında **0o yazmağımız kifayətdir. Nümunə:**

<script>

document.write(0o543);

</script>

Nəticədə ədədin onluq qarşılığı, yəni, 355 çıxacaq.

Bu ədədlərlə şərtlər daxilində də yoxlama apara bilərik. Nümunə:

<script>

if(0b101011==43){

alert("Beraberlik dogrudur.");

}

</script>

Beləliklə, ədədlərin bərabər olduğu bildirilir.

ES6 Tail Calls

Es6 Tail Calls ilə sonsuz, heç vaxt bitməyəcək funksiyalar xəta verməkdədir. Nümunə:

<script>

function a() {

b();

}

function b() {

a();

}

console.log(a());

</script>

Burada daim funksiyalar bir-birini çağıracağı üçün Javascript xəta verəcəkdir.

Obyektyönümlü proqramlaşdırma

Siniflərin tam olaraq nə olduğunu anlamaq üçün bəzi nümunələrə baxmalıyıq. Buradakı sinif anlayışı həyatdakı sinifləndirməyə oxşardır. Məsələn, bütün maşınları eyni sinifə aid edə bilərik. Bütün maşınların müxtəlif olsa da rəngi var, işlətdiyi yanacaq miqdarı var, qapıları var və digər bir çox xüsusiyyəti var.. Və yaxud heyvanları da buna misal gətirə bilərik. Məsələn, bütün quşları eyni sinifə aid edə bilərik. Hamısının iki ayağı var, rəngi var, cinsi var, adı var, uçma funksiyası və bir çox xüsusiyyəti var. Yəni, gördüyümüz kimi həyatda bir çox obyektlər var ki, onlar bir-birinə çox oxşamaqda və eyni sinifə aid edilməkdədir. Bu yanaşmadan proqramlaşdırmada da istifadə olunmaqdadır. Buna obyektyönümlü proqramlaşdırma deyilməkdədir. Çünki həyatda olduğu kimi bir sinif mövcuddur, məsələn quşlar sinifi. Həmin sinifə aid də quşlar mövcuddur. Burada sinif quşlardır, hər quş isə bu sinifə aid bir obyektdir. Bir çox dillərdə sinifin yaradılması üçün class açar sözündən istifadə olunmaqdadır. Ecmascript 6-dan etibarən javascriptdə də sinif yaratmaq üçün class sözündən istifadə olunmaqdadır. Nümunəyə baxaq:

<script>

class qus {

constructor (ayaq,qanad) {

this.ayaq=2;

this.qanad=2;

}

}

</script>

Bu yazılışla quşlara aid bir sinif yaradılmaqdadır. Burada ilk olaraq constructor funksiyası yaradılmaqdadır. Bu funksiya obyekt yaradılarkən oxa məxsus olacaq ilk qiymətləri təyin etməkdədir. Məsələn, burada funksiyanın içərisində ayaq və qanad parametrimiz var. Hər quşun iki ədəd ayağı və iki ədəd qanadı var. Ona görə də onun içərisində this.ayaq=2 və this.qanad=2 yazaraq onların sayını 2 etmiş oluruq. İndi isə bu sinifə aid bir obyekt yaradaq.

<script>

class qus {

```
constructor (ayaq,qanad) {

this.ayaq=2;

this.qanad=2;

}

}

var qartal=new qus();

document.write("Qanad sayi:" +
qartal.qanad);

document.write("<br>Ayaq sayi:" +
qartal.ayaq);

</script>
```

Gördüyümüz kimi, burada sinifə aid qartal adlı bir obyekt yaradılır. Burada qartal.qanad və qartal.ayaq yazaraq qartalın qanad və ayaq sayını göstərmiş oluruq. Bu yazılış aşağıdakı şəkildə də ola bilər.

```
<script>

class qus {
```

```
constructor (ayaq,qanad) {

this.ayaq=ayaq;

this.qanad=qanad;

}

}

var qartal=new qus(2,2);

document.write("Qanad sayi:" +
qartal.qanad);

document.write("<br>Ayaq sayi:" +
qartal.ayaq);

</script>
```

Beləliklə obyektin ayaq və qanad sayı obyekt yaradılarkən bildirilir. Yuxarida this.ayaq=ayaq yazmaqla obyektdəki ayaq sayının funksiyaya ötürülən ayaq parametrinə bərabər olacağını bildiririk. Sinifin digər funksiyalarında da this.ayaq yazaraq ayaq sayını funksiya daxilində işlədə bilərik.

İndi isə başqa bir xüsusiyyət də əlavə edək. Bu xüsusiyyət yaradılacaq quşun uça bilib-bilməyəcəyini müəyyən etsin. Nümunəyə baxaq:

```
<script>
class qus {
constructor (ayaq,qanad) {
this.ayaq=2;
this.qanad=2;
}
ucma(x){
if(x==1) document.write("Uca bilir<br>");
else document.write("Uca bilmir<br>");
}
}
var qartal=new qus();
qartal.ucma(1);
var pinqvin=new qus();
pinqvin.ucma(0);
</script>
```

Burada ikinci bir ucma funksiyası yaradılır və ona parametr ötürülür. Əgər, parametrin

qiyməti 1 olarsa uça bildiyi yazılır, əks halda isə, uça bilmədiyi yazılır. Biz burada qartal və pinqviin obyektləri yaratmışıq. Qartal uça bildiyi üçün onun uçma funksiyasına 1 qiyməti ötürülür və uça bildiyi yazılır. Pinqvin isə uça bilmədiyi üçün onun uçma funksiyasına 0 qiyməti ötürülür və şərt ödənmədiyi üçün uça bilmədiyi yazılır.

İndi isə başqa bir funksiya da yazaq. Xəyal edək ki, qanadın uzunluğunu 5-ə vurduqda quşun sürəti alınır. Dediyim kimi, sadəcə xəyal edirik, realda yəqin ki fərqlidir. Bu zaman sadəcə bir sürət funksiyası yaratmalıyıq. Həmin funksiya verilən parametri 5-ə vurub bizə qaytarmalıdı. Beləliklə quşun sürətini almış olacağıq. Nümunə:

```
<script>

class qus {

constructor (ayaq,qanad) {

this.ayaq=2;

this.qanad=2;

}

ucma(x){
```

```
if(x==1) document.write("Uca bilir<br>");
else document.write("Uca bilmir<br>");
}
suret(qanadin_uzunlugu){
return qanadin_uzunlugu*5;
}
}
var qartal=new qus();
document.write("Qartarlin sureti: " +
qartal.suret(10));
var serce=new qus();
document.write("<br>Sercenin sureti: " +
serce.suret(3));
</script>
```

Burada ilk olaraq sinifə sürət funksiyasını əlavə etdik. Funksiyanın işi sadəcə ötürülən parametri 5-ə vurub bizə qaytarmaqdı. Ondan sonra obyektlər yaradırıq. Həmin obyektlər üçün isə həmin funksiya ilə sürəti hesablayırıq. Məsələn, qartalın qanadının uzunluğu 10 olduğu üçün

bizə 50 qiymətini verir. Sərçədə isə bu qiymət 15 olur. Bundan sonra öz fantaziyanıza uyğun quşa yeni funksiyalar təyin edə bilərsiniz.

İndi isə gəlin bu modeli telefonlara tətbiq edək. Konstruktorda telefonumuzun modelini, yaddaşını, ramını və kamerasını daxil edəcəyik. Nümunəyə baxaq:

```
<script>

class telefon {

constructor (model,ram,yaddas,kamera) {

this.model=model;

this.ram=ram;

this.yaddas=yaddas;

this.kamera=kamera;

}

}

telefon1=new telefon("xiaomi",4,8,12);

alert(telefon1.model);

</script>
```

Beləliklə telefon sinifi yaradırıq və bir ədəd obyekt təyin edirik. Ardıcıl olaraq telefonumuza məxsus parametrləri də daxil edirik. İndi isə yaddaşa bizim təyin edəcəyimiz ölçüdə neçə ədəd video yerləşəcəyini hesablayan bir funksiya əlavə edək. Nümunəyə baxaq:

```
<script>
class telefon {
constructor (model,ram,yaddas,kamera) {
this.model=model;
this.ram=ram;
this.yaddas=yaddas;
this.kamera=kamera;
}
hesabla(say){
return this.yaddas/say;
}
}
telefon1=new telefon("xiaomi",4,8,12);
```

alert(telefon1.hesabla(1));

</script>

Beləliklə yaddaşı 1 olan 8 ədəd video yerləşə biləcəyini hesablamış oluruq.

İndi isə gəlin sinifi Html ilə birlikdə işlədək və Html elementlərinə tətbiq edək. Fərz edək ki, bizim 3 ədəd divimiz var və o divlər id-lərə görə fərqlənir. Bu divlərin hər birini bir sinifin obyekti kimi təsəvvür edib ona xüsusiyyətlər ötürə bilərik. Nümunəyə baxaq:

** **

** **

** **

<script>

class duzbucaqli {

constructor (id,width,height,color) {

this.id=id;

this.width=width;

this.height=height;

this.color=color;

```
}

change(){

var
d=document.getElementById(this.id).style
;

d.width=this.width;

d.height=this.height;

d.backgroundColor=this.color;

}

}

var div1=new
duzbucaqli("div1",300,300,"blue");

div1.change();

var div2=new
duzbucaqli("div2",300,300,"red");

div2.change();

var div3=new
duzbucaqli("div3",300,300,"green");

div3.change();
```

</script>

Burada ilk olaraq 3 ədəd div yaradırıq. Sonra isə javascript daxilində sinifimizi yaradırıq. Constructor daxilində obyektə id, uzunluq, en və rəng qiymətlərini veririk. Sonra isə change() funksiyasında elementlərimizə xüsusiyyətləri ötürürük. Beləliklə 3 ədəd divə xüsusiyyətlər vermiş oluruq. Burada **document.getElementById(this.id)** yazaraq elementin id-nin də obyekt yaradılarkən seçilməsini təmin edirik. Beləliklə istənilən id-li divin xüsusiyyətlərini bu siniflə dəyişə bilərik.

Obyektyönümlü yanaşmada inheritance, yəni miras alma anlayışı mövcuddur. Bu anlayışda, bir sinif digər ana sinifin funksiyalarını işlədə bilir. Nümunəyə baxaq:

<script>

class telefon {

constructor(yaddas) {

document.write("Telefonun yaddasi: "+ yaddas);

}

```
}
class model extends telefon {
constructor(model,yaddas) {
super(yaddas);
}
}
var t1=new model("xiaomi",8);
</script>
```

Burada bir ədəd telefon, bir ədəd də model sinifimiz var. Miras alma extends açar sözü ilə həyat keçirilir. Burada model extends telefon yazaraq telefondakı funksiyaları modelin içərisində də işlədə bilirik. Gördüyümüz kimi, modelin constructorunun içərisində super(yaddas) yazmışıq. Bu funksiya ana sinifin, yəni telefon sinifinin constructorunu çağırır və ona parametr ötürür. Beləliklə telefon sinifinin constructor funksiyası işə düşür. Başqa bir nümunəyə də baxaq:

```
<script>
```

```
class telefon {
constructor(yaddas) {
this.yaddas=yaddas;
}
show(){
document.write(this.yaddas);
}
}
class model extends telefon {
constructor(model,yaddas) {
super(yaddas);
}
}
var t1=new model("xiaomi",8);
t1.show();
</script>
```

Burada gördüyümüz kimi, mirasalmadan istifadə etdiyimiz üçün, model sinifli obyekt vasitəsilə telefon sinifindəki funksiyanı istifadə edə bilirik.

Bunu kvadrat formalı div elementləri üçün tətbiq edə bilərik. Məsələn, düzbucaqlılarla kvadratların sahəsi eyni qaydada hesablanır. Buna görə də, sahə funksiyasını bir sinifdə yaradıb digərində də istifadə edə bilərik. Nümunəyə baxaq:

```
<div id="div"></div>

<script>

class duzbucaqli{

constructor(width,height){

this.width=width;

this.height=height;

}

sahe(){

return this.width*this.height;

}
```

```
}
class kvadrat extends duzbucaqli{
constructor(id,width,height,color){
super(width,height);
var d=document.getElementById(id).style;
d.width=width;
d.height=height;
d.backgroundColor=color;
}
}
var div1=new
kvadrat("div",200,200,"red");
alert(div1.sahe());
</script>
```

Beləliklə funksiyanı aşağıda da işlətmiş oluruq.

Sinif daxilindəki funksiyaları kənardan çağırıla bilməsi üçün qarşısına static sözünü yazmağımız kifayətdir. Nümunəyə baxaq:

```
<script>
```

```
class sinif1{

constructor(){

}

static show(){

alert("Salam");

}

}

sinif1.show();

</script>
```

Beləliklə sinif daxilindəki funksiyaya kənardan obyekt yaratmadan müraciət edə bilirik.

Bunlardan əlavə olaraq get və set ifadələri də mövcuddur ki, onlardan istifadə edərək, sinif daxilindəki parametrə müraciət edə və dəyişiklik edərkən əməliyyatlar apara bilərik. Nümunəyə baxaq:

```
<script>

class User {
```

```
constructor(name) {
this.name = name;
}
get name() {
return this._name;
}
set name(value) {
if(value.length<5) alert("ad cox qisadir");
else this._name=value;
}
}
var user = new User("Birinci ad");
alert(user.name);
user = new User("ad");
</script>
```

Beləliklə ad qısa olduqda qısa olduğu bildirilir və dəyişdirilmir.

Alqoritmik məsələlər və onların həlli

Alqoritm qarşıya çıxan hər-hansı bir problemin həll yoludur. Qarşınıza bir problem çıxdıqda onu həll etmək üçün ən azı beyninizdə onun alqoritmini qurmalı, həll yolunu tapmalısınız. Məsələn, qarşıya belə bir məsələ qoyaq. Tələbə imtahanda cavab verdiyi doğru cavabların sayını daxil edir. Doğru cavab sayı 50-dən yuxarı olarsa, imtahandan keçmiş olur. Bunun üçün aşağıdakı kimi bir ardıcıllıq qura bilərik:

1) Tələbədən input vasitəsilə doğru cavab sayını götür;

2) Həmin qiymətin 50-dən böyük olub-olmadığını yoxla;

3) Qiymət 50-dən böyük olarsa imtahandan keçdiyini bildir. Əks halda isə imtahandan keçmədiyini bildir.

Bunu kod vasitəsilə yazaq.

<input type="text" id="input">

<button type="button" onclick="funksiya();">Yoxla</button>

```
<script>

function funksiya(){

var
x=document.getElementById("input").valu
e*1;

if(x>50){

alert("Imtahani kecdiniz.");

}

else{

alert("Imtahani kece bilmediniz.");

}

}

</script>
```

Beləliklə tələbənin daxil etdiyi qiymət 50-dən böyük olarsa, o zaman, imtahandan keçdiyini bildiririk. Əks halda isə keçmədiyini bildiririk.

Qarşımıza çıxacaq digər problemlərdə də bu cür ardıcıllıqlar quraraq problemləri rahatlıqla həll edə bilərik. Alqoritmik düşüncəniz nə qədər güclü olarsa, problemlərin həll ardıcıllığını da o

qədər rahat quracaqsınız. Alqoritmik düşüncəni gücləndirmək üçün də çoxlu sayda məsələ həll etmək lazımdır. Həmçinin, oyun yazmaq alqoritmik düşüncəni gücləndirə bilir. İndi isə sıra ilə məsələləri həll edək. Məsələlərin həll yoluna baxmazdan öncə özünüz həll etməyə çalışın. Həll edə bilmədiyiniz təqdirdə həll yoluna baxın.

Məsələ 1. 1-dən 100-ə qədər ədədlərin cəmini tapın. Bunun üçün for istifadə edin.

İlk olaraq necə edəcəyimiz düşünək. For vasitəsilə ədədləri 1-dən 100-ədək sıralaya biləcəyimizə görə, deməli həmin ədədləri toplaya da bilərik. İlk olaraq bir ədəd x adlı dəyişən yaradaq və qiymətini 0 edək. Ədədlər 1-dən 100-ədək sıralandıqca hər birini x-in üzərinə gələk. Beləliklə 1-dən 100-ə qədər ədədlərin cəmi x-ə ötürülmüş olsun. Koda baxaq.

```
<script>

var x=0;

for(i=1;i<=100;i++){

x=x+i;
```

}

document.write(x);

</script>

Kodu işlətdikdə 1-dən 100-ə qədər ədədlərin cəminin çap olunduğunu görəcəyik. İndi isə addım-addım kodu analiz edək.

İlk olaraq x dəyişəni yaratdıq. Bundan sonra 1-dən 100-ə qədər ədədləri sıralamaq üçün for işlətdik.

1-ci dövrə. İlk dövrədə i dəyişənin qiyməti 1-dir. İlk dövrədə x=x+i yazaraq x-in köhnə qiymətinin üzərinə I-nin hal-hazırki qiymətini gəldik. Yəni, x=0+1=1 oldu. Beləliklə x dəyişəninin yeni qiyməti 1 oldu.

2-ci dövrə. Birinci dövrədə x-in qiyməti 1 olmuş idi. İkinci dövrədə x=x+i yazaraq x-in köhnə qiymətinin üzərinə I-nin ikinci dövrədəki qiymətini gəlmiş oluruq. Yəni, x=1+2=3 olmuş olur. Beləliklə x-in yeni qiyməti 3 olur.

3-cü dövrə. İkinci dövrənin sonunda x-in qiyməti 3 idi. Üçüncü dövrədə x=x+i yazaraq x-in köhnə qiymətinin üzərinə I-nin hal-hazırki qiymətini gəlmiş oluruq. Yəni, x=3+3=6 olur. Beləliklə x-in yeni qiyməti 6 olmuş olur.

4-cü dövrə. Üçüncü dövrənin sonunda x-in qiyməti 6 olmuşdu. Bu dövrədə də x=x+i yazaraq x-in köhnə qiymətinin üzərinə I-nin hal-hazırki qiymətini gəlmiş olur. Yəni, x=6+4=10. Beləliklə bu dövrənin sonunda x-in qiyməti 10 olmuş olur.

5-ci dövrə. Dördüncü dövrənin sonunda x-in qiyməti 10 olmuşdu. Bu dövrədə də x=x+i yazaraq x-in köhnə qiymətinin üzərinə I-nin hal-hazırki qiymətini gəlirik. Yəni, x=10+5=15 olur. Beləliklə, x-in yeni qiyməti 15 olur.

Bu ardıcıllıqla 100 dövrə davam edir.

100-cü dövrə. 99-cu dövrənin sonunda x-in qiyməti 4950 olmuş olur. 100-cü dövrədə x=x+i yazaraq x-in köhnə qiymətinin üzərinə I-nin hal-hazırki qiymətini gəlmiş oluruq. Yəni, x=4950+100=5050. Beləliklə x-in qiyməti 5050 olmuş olur və bütün dövrələr bitir.

Nəticədə 1-dən 100-ə qədər ədədlərin cəminin 5050 olmuş olduğunu görürük.

Məsələ 2. 1-dən 5-a qədər ədədlərin hasilini tapın. Bu məsələ də əvvəlki məsələ ilə oxşardır. Sadəcə, burada toplama əvəzinə vurma istifadə

olunacaq və x dəyişəninin ilk qiyməti 0 yox 1 olacaq. Çünki 0 olarsa bütün ədədləri 0-a vurar və sonda hasil 0 olar. 1 olarsa nəticə düzgün olar. İndi isə kodu yazaq və addım-addım izah edək.

```
<script>

var x=1;

for(i=1;i<=5;i++){

x=x*i;

}

document.write(x);

</script>
```

1-ci dövrə. İlk olaraq x-in qiyməti 1-dir. Burada x=x*i yazmaqla x-in köhnə qiymətini i-nin dövrədəki qiymətinə vurmuş oluruq. Elə yeni qiyməti də x=1*1=1 olur.

2-ci dövrə. X-in əvvəlki qiyməti 1-ə bərabərdir. Burada x-in yeni qiyməti x=x*i=1*2=2 olur. 2-ci dövrədə i-nin qiyməti 2 olduğu üçün 2-ə vurulur.

3-cü dövrə. X dəyişəninin son qiyməti 2 idi. Bu dövrədə i dəyişəninin qiyməti 3-dür. Ona görə də. bu dövrədə x=x*i=2*3=6 olur.

4-cü dövrə. X dəyişəninin son qiyməti 4 idi. Bu dövrədə isə i dəyişəninin qiyməti 4-dür. Ona görə də, x=x*i=6*4=24 olur.

5-ci dövrə. Bu dövrə sonuncu dövrədir. Bundan əvvəl x dəyişəninin son qiyməti 24 idi. Bu dövrədə i dəyişənin qiyməti isə 5-dir. Buna görə də, x=x*i=24*5=120 olmuş olur.

Beləliklə, 1-dən 5-ə qədər ədədlərin hasili 120 olmuş olur. Qeyd edək ki, 1-dən 5-ə qədər ədədlərin hasili həm də 5 faktorial deməkdir.

Məsələ 3. Ədədin tək ədəd yoxsa cüt ədəd olub-olmadığını yoxlayın.

Tək ədədlərin 2-ə bölünməsindən alınaq qalıq 1 olur. Cüt ədədlərin isə 2-ə bölünməsindən alınan qalıq 0 olur. Ona görə də, bu deyilənlərdən istifadə edərək ədədin tək yoxsa cüt olub-olmamasını yoxlaya bilirik. Javascriptdə və bir sıra dillərdə ədədin 2-ə bölünməsindən alınan qalığı x%2 yazaraq tapa bilirik. Aşağıdakıı koda baxaq.

<script>

var x=7;

document.write(x%2);

</script>

Kodu işlətdikdə ekrana 1 yazıldığını görəcəksiniz. Çünki 7 tək ədəd olduğu üçün 2-ə bölünməsindən alınan qalıq 1-ə bərabərdir. Əgər, ədəd cüt olarsa, o zaman 2-ə bölünməsindən alınan qalıq 0 olardı. Bu dediyimizi şərtlə yoxlayaraq ədədin cüt yoxsa tək olduğunu ekrana yaza bilərik. Koda baxaq:

<script>

var x=prompt("Ededi daxil edin:");

if(x%2==1){

document.write("Eded tekdir.");

}

else{

document.write("Eded cutdur.");

}

</script>

İstifadəçi x dəyişəninin qiymətini daxil etdikdən sonra onu 2-ə böldükdə alınan qalığın 1 olub-olmaması yoxlanılacaq. Əgər, 1 olarsa ədədin tək olduğu yazılacaq. Əks halda isə tək deyilsə cüt olduğu yazılacaq. Beləliklə tək ədəd daxil etdikdə bizə tək olduğunu, cüt ədəd daxil etdikdə isə cüt olduğunu bildirəcək. Qeyd edək ki, bu şəkildə ədədin istənilən ədədə bölünməsindən alınan qalığı görə bilərik.

Məsələ 4. 1-dən 10-a qədər olan tək ədədləri çap edin.

Bunun üçün sadəcə 1-dən 10-a qədər ədədləri sıralamalı və onların tək olub-olmadığını yoxlamalı, tək olanları çap etməliyik. Koda baxaq:

```
<script>

for(i=1;i<=10;i++){

if(i%2==1){

document.write(i+"<br>");

}

}

</script>
```

1-ci dövrə. Bu dövrədə I-nin qiyməti 1-dir. Şərtdə I-nin, yəni 1-in 2-ə bölünməsindən alınan qalığın 1 olub-olmadığı yoxlanılır. Şərt doğrudur. 1 tək ədəd olduğu üçün 2-ə bölünməsindən alınan qalıq 1 olur və ədəd ekrana yazılır. Burada i+”
” yazmağımızın səbəbi ədədləri alt-alta yazmaq istəməyimizdir.

2-ci dövrə. Bu dövrədə I-nin qiyməti 2-dir. Şərtdə I-nin, yəni 2-nin 2-ə bölünməsindən alınan qalığın 1 olub-olmadığı yoxlanılır. 2 cüt ədəd olduğu üçün şərt ödənmir və heç bir şey çap olunmur.

3-cü dövrə. Bu dövrədə I-nin qiyməti 3-dür. Şərtdə 3-ün 2-ə bölünməsindən alınan qalığın 1 olub-olmaması yoxlanılır. 3 tək ədəd olduğu üçün alınan qalıq 1 olur və şərt ödənir. Şərt ödəndiyi üçün də ədəd çap olunur.

Eyni qaydada 10 dövrə davam edir.

10-cu və sonuncu dövrə. Bu dövrə sonuncudur. Burada I dəyişənin qiyməti 10-dur və 2-ə bölünməsindən alınan qalığın 1 olub-olmaması yoxlanılır. 10 cüt ədəddir və 2-ə bölünməsindən alınan qalıq 0-dır, 1 deyil. Buna görə də şərt ödənmir və heç bir şey çap olunmur.

Beləliklə 10 dövrənin hamısı tamamlanır və 1-dən 10-a qədər tək ədədlər çap olunmuş olur.

Məsələ 5. 1-dən 10-a qədər tək ədədlərin cəmini tapın. Daha əvvəl dövrədəki ədədlərin cəmini tapmışdıq. Burada isə sadəcə dövrədəki ədədləri yoxlayacağıq və tək olanların cəmini tapacağıq. Koda baxaq.

```
<script>

var x=0;

for(i=1;i<=10;i++){

if(i%2==1){

x=x+i;

}

}

document.write(x);

</script>
```

Beləliklə 1-dən 10-a dək olan tək ədədlərin cəmini tapırıq.

Məsələ 6. Çoxluqdakı ədədlərin cəmini tapın.

Bir ədəd çoxluğumuz var və o çoxluqdakı ədədlərin cəmini tapmaq istəyirik. Bunun üçün sadəcə o çoxluqdakı ədədləri toplayarıq. Koda baxaq:

```
<script>

var array=[5,8,3];

var cem=0;

cem=cem+array[0];

cem=cem+array[1];

cem=cem+array[2];

document.write(cem);

</script>
```

Gördüyümüz kimi, çoxluqdakı elementləri bir-bir dəyişənin üzərinə gəlirik və toplayırıq. Ancaq çoxluqda element çox sayda olarsa, o zaman bu şəkildə yaza bilmərik. O halda for operatoru ilə çoxluğun bütün elementlərini sıralaya bilərik. Koda baxaq:

```
<script>

var array=[5,8,3];
```

```
var cem=0;

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

cem=cem+array[i];

}

document.write(cem);

</script>
```

Beləliklə for vasitəsilə çoxluq elementlərini toplamış oluruq. Burada array.length çoxluğun element sayını bildirir və 3-ə bərabərdir. Çoxluqda saymaq 0-dan başlayır və 0-cı, 1-ci və 2-ci elementlər mövcuddur. 3-cü element mövcud olmadığına görə for içərisində şərtdə kiçik bərabər yox, kiçikdir yazılır. Çünki 3 nömrəsi sıralanmayacaq.

Məsələ 7. Math.sqrt() işlətmədən daxil edilmiş tam ədədin kök altısını tapın.

Bu məsələ üçün yəqin ki bir çox alqoritmlər mövcuddur və araşdırılmasına ehtiyac var. Biz istifadə edəcəyimiz üsulda 1-dən daxil edilmiş ədədə qədər bütün ədədləri sıralayacağıq və onlardan hansınn kvadratının daxil edilmiş ədədə bərabər olduğunu tapacağıq. Beləliklə

tapdığımız ədəd daxil edilmiş ədədin kök altısı olmuş olacaq. Koda baxaq:

```
<script>

var x=prompt("daxil edin:")*1;

var kok;

for(i=1;i<=x;i++){

if(i*i==x){

kok=i;

break;

}

}

document.write(kok);

</script>
```

Sıralanan ədədin kvadratı daxil edilmiş ədədə bərabərdirsə, deməli kök altısı həmin ədəddir. Şərt ödəndiyi zaman kök dəyişənini həmin ədədə bərabər edirik və break vasitəsilə dövrəni dayandırırq. Çünki kök altını tapdığımız üçün artıq növbəti ədədlərin sıralanmasına ehtiyac yoxdur. Sonda isə həmin ədədi çap edirik. Qeyd

edək ki, bu üsul sadəcə tam ədədlər üçündür və araşdırma ilə daha yaxşı alqoritmlər tapmaq mümkündür.

Məsələ 8. For operatorundan istifadə edərək sözün içərisində “@” işarəsi olub-olmadığını tapın.

Bunun üçün sadəcə sözdəki simvolları tək-tək sıralamalı və hər-hansı simvolun “@” işarəsinə bərabər olub-olmadığını yoxlamalıyıq. Əgər, şərt doğru olarsa, bu o deməkdir ki, söz daxilində “@” işarəsi var. Koda baxaq:

```
<script>

var x="Java@script";

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

if(x[i]=="@"){

document.write("Var.");

break;

}

}

</script>
```

Beləliklə sıralanan simvol “@” işarəsinə bərabər olarsa var olduğu yazılır və dövrə dayandırılır. Çünki artıq söz daxilində “@” işarəsinin var olduğu bilinir.

Məsələ 9. For operatorundan istifadə edərək cümlədə neçə ədəd söz olduğunu tapın.

Cümlə daxilində sözlər boşluqlara görə ayrılır. Aşağıdakı cümləyə baxaq:

“Javascript çox maraqlı dildir.”

Gördüyümüz kimi burada 3 ədəd boşluq simvolu və 4 ədəd söz var. Yənu, boşluq simvollarının sayını tapıb üzərinə 1 gəlsək söz sayını tapmış oluruq. Bunun üçün də sadəcə cümlədəki hərfləri tək-tək sıralayıb boşluqları saymaq kifayətdir. Koda baxaq.

```
<script>

var say=0;

var x="Javascript cox maraqli dildir.";

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

if(x[i]==" "){

say++;
```

```
}

}

document.write(say+1);

</script>
```

Beləliklə boşluq sayını taparaq üzərinə 1 gələrək söz sayını tapmış oluruq.

Məsələ 10. 16 və 165 ədədlərini parçalayıb rəqəmlərini alt-alta yazın.

Əgər, bu ədədin 10-a bölünməsindən alınan qalığı tapsaq, ədədin ikinci rəqəmini alarıq. Ədədin 10-a bölünməsindən alınan tam hissəni tapsaq ədədin birinci rəqəmini alarıq. Nümunə koda baxaq:

```
<script>

var eded=16;

var reqem1=Math.floor(eded/10);

var reqem2=eded%10;

document.write("Birinci reqem: " +
reqem1 + "<br>Ikinci reqem: " + reqem2);

</script>
```

Burada ededi 10-a bölərək 1.6 ədədini alırıq. Sonra Math.floor funksiyası ilə onu aşağı yuvarlaqlaşdırırq və 1 alırıq. Beləliklə, ədədimizin ilk rəqəmini götürmüş oluruq. Növbəti hissədə eded%10 yazaraq ədədin 10-a bölünməsindən alınan qalığı, yəni 6-ı alırıq. Beləliklə ədədi parçalamış oluruq.

İndi isə 165 ədədini parçalayaq. Son rəqəmi tapmaq üçün yuxarıdakı qaydada ədədin 10-a bölünməsindən alınan qalığı tapırıq. Ondan sonra isə ortadakı 6 rəqəmini tapmalıyıq. Bunun üçün ilk olaraq 165-i 10-a bölüb yuvarlaqlaşdıraraq 16-a çeviririk. Sonra isə 16-nın 10-a bölünməsindən alınan qalığı taparaq 6 rəqəmini də götürürük. Sonra yenə 16-ı 10-a bölüb yuvarlaqlaşdırırıq və 1 rəqəmini alırıq. Nümunə koda baxaq:

```
<script>

var eded=165;

var reqem3=eded%10; // 5 rəqəmini götürdük

eded=Math.floor(eded/10); // 165-i 16-a çeviririk
```

reqem2=eded%10; // 6 rəqəmini də götürdük

var reqem1=Math.floor(eded/10); // Ədədin birinci rəqəmini də götürdük

**document.write("Birinci reqem: " + reqem1 + "
Ikinci reqem: " + reqem2 + "
Üçüncü reqem: " + reqem3);**

</script>

Beləliklə ədədi rəqəmlərinə parçalamış olduq.

Məsələ 11. 1653 ədədini rəqəmlərinə parçalayın.

Ədəddəki rəqəmlərin sayı az olduqda onları tək-tək parçalaya bilirdik. Ancaq rəqəmlərin sayı çox olduqda artıq onları tək-tək parçalamağımız çətinləşir. Bu zaman ədədi dövrəyə salıb rəqəmlərini dövrə daxilində sıralaya bilərik. Nümunəyə baxaq:

<script>

var x=1653;

while(x>0){

qaliq=x%10;

**document.write(qaliq + "
");**

x=Math.floor(x/10);

}

</script>

Nəticədə ədəd daxilindəki rəqəmlərin alt-alta yazıldığını görəcəyik. İndi isə dövrlədə bir-bir baxaq.

1-ci dövrə. İlk olaraq x-in 0-dan böyük olub-olmadığı yoxlanılır. X dəyişəninin qiyməti 0-dan böyük olduğu üçün dövrə başlayır. Növbəti olaraq qalıq dəyişəninə ədədin 10-a bölünməsindən alınan qalıq, yəni, ədədin son rəqəmi köçürülür və çap olunur. Beləliklə, ədədin bir rəqəmi artıq hazıırdır. Ondan sonra isə növbəti dövrə üçün ədədi 10-a bölünməsindən alınan tam hissəyə bərabər edirik ki, bu ədəd də 165 olur. Beləliklə, növbəti dövrədə ədədin 10-a bölünməsindən alınan qalığı tapanda sondan ikinci rəqəmi tapmış olacağıq.

2-ci dövrə. X-in 0-dan böyük olması yoxlanılır və dövrə başlanılır. Ədədin qiyməti 165 idi. Onun 10-a bölünməsindən alınan qalığı, yəni 5-i taparaq son rəqəmini götürmüş oluruq. Beləliklə

ədədimizin sondan ikinci rəqəmini də götürüb çap etmiş oluruq. Ondan sonra növbəti dövrə üçün ədədin son rəqəmini itirməliyik. Yəni, 165-i 16-a çevirməliyik. Bunun üçün onu 10-a bölüb alınan tam hissəni götürürük. Beləliklə artıq ədədimiz 16 olmuş olur.

3-cü dövrə. X-in 0-dan böyük olması yoxlanılır və dövrə başlanır. Ədədin qiyməti 16-dır. Onun 10-a bölünməsindən alınan qalığı taparaq 6 rəqəmini götürürük və çap edirik. Bundan sonra ədədi 10-a bölünməsindən alınan tam hissəyə bərabərləşdiririk. Artıq ədədin qiyməti 1 olur.

4-cü dövrə. Bu dövrə sonuncu dövrədir və X-in qiyməti 1-dir. X-in qiymətinin 0-dan böyük olması yoxlanılır və dövrə başlanır. Növbəti olaraq x-in, yəni 1-in 10-a bölünməsindən alınan qalır tapılır. 1-in 10-a bölünməsindən alınan qalır elə 1-dir və beləliklə ədədimizin ilk rəqəmini də götürüb çap edirik. Artıq ədədimizin bütün rəqəmlərini götürdük və çap etdik. Bundan sonra ədədi 10-a bölüb tam hissəsini götürürük və bu ədəd 0 edir. Növbəti olaraq dövrə bitir. Dövrədə yenidən şərt yoxlanılır və 0 rəqəmi 0-dan böyük olmadığı üçün dövrə başlamır və tamamilə bitir. Beləliklə ədədi parçalayıb bütün rəqəmlərini alt-alta çap etmiş oluruq.

Məsələ 12. Ədəd daxilindəki rəqəmlərin cəmini hesablayın.

Burada sadəcə yuxarıdakı qaydada ədədin parçalayıb rəqəmlərini başqa bir dəyişənin üzərinə gələrək topalamalıyıq. Koda baxaq:

```
<script>

var x=1653;

var cem=0;

while(x>0){

qaliq=x%10;

cem=cem+qaliq;

x=Math.floor(x/10);

}

document.write(cem);

</script>
```

Beləliklə ədədin rəqəmlərinin cəmini tapmış oluruq.

Məsələ 13. Ədəd daxilindəki 0-ların sayını tapın.

Bunun üçün sadəcə əvvəlki qaydada ədədin rəqəmlərini sıralamalıyıq. Sıralayarkən 0-a bərabər olan rəqəmləri şərtlə yoxlayıb saymalıyıq. Koda baxaq:

```
<script>

var x=1653;

var say=0;

while(x>0){

qaliq=x%10;

say++;

x=Math.floor(x/10);

}

document.write(say);

</script>
```

Beləliklə, ədədin rəqəmlərinin sayını tapmış oluruq.

Məsələ 14. Ədədin tərs yazılışını tapın. Məsələn, 165-i 561 şəklində edin.

Bunun üçün əvvəlcə işlədəcəyimiz üsula baxaq. Ədədin rəqəmlərini sondan başlayaraq 10-a

vurub üzərinə əvvəlki ədədi gələk. Bütün rəqəmlər üçün bu qaydanı tətbiq etsək ədədin tərs yazılışını alacağıq.

5*10+6=56; 56*10+1=561;

Bu qayda ilə ədədin tərs yazılışı almış oluruq. İndi isə koda baxaq:

```
<script>

var x=1653;

var ters=0;

while(x>0){

qaliq=x%10;

ters=ters*10+qaliq;

x=Math.floor(x/10);

}

document.write(ters);

</script>
```

Burada tərs dəyişəninin qiyməti 0-a bərabərdir. İlk dövrədə 10-a vurulur və üzərinə qalıq gəlinir. Beləliklə, qiyməti son rəqəmə bərabər olur. Sonra isə ədəd 10-a bölünüb tam hissəsi

tapılaraq 165-ə çevrilir. Ondan sonraki hər dövrədə tərs dəyişəni 10-a vurularaq üzərinə əvvəlki rəqəm gəlinir. Beləliklə, ədədin tərs yazılışı tapılmış olur.

Məsələ 15. 100 və 1000 aralığında tərs yazılşı özünə bərabər olan ədələrin siyahısını çıxaın. Məsələn, 111, 121, 222 və s.

Bunun üçün sadəcə 1-dən 100-ə qədər ədədləri sıralamalı və yuxarıdakı qaydada tərsini tapmalıyıq. Bundan sonra şərtlə yoxlayaraq tərsinə bərabər olan ədədləri çap etməliyik. Koda baxaq:

```
<script>

for(i=100;i<1000;i++){

var ters=0;

var m=i;

while(m>0){

qaliq=m%10;

ters=ters*10+qaliq;

m=Math.floor(m/10);

}
```

**if(i==ters) document.write("
"+i);**

}

</script>

Burada ədədləri 1-dən 100-ə qədər sıralayırıq. Parçalanacaq ədəd dəyişəcəyi üçün bir ədəd m dəyişəni yaradırıq və sıralanan i ədədini m dəyişəninə köçürürük. Ondan sonra əvvəlki qaydada ədədin tərsini tapırıq. Əgər, sıralanan ədədin tərsi özünə bərabər olarsa, o zaman, ədədi çap edirik. Sayını tapmaq istəsə idik bir ədəd say dəyişəni yaradıb, ilk qiymətini 0 edib hər şərt ödəndikcə bir vahid artırmalı idik.

<script>

var say=0;

for(i=100;i<1000;i++){

var ters=0;

var m=i;

while(m>0){

qaliq=m%10;

ters=ters*10+qaliq;

```
m=Math.floor(m/10);
}
if(i==ters) say++;
}
document.write(say);
</script>
```

Məsləhətlər

Hal-hazırki nəşrdə yer almayan bir sıra mövzular var ki, onları internet üzərindən araşdırmağınız məsləhət görülür:

Ajax, Asinxron əməliyyatlar, Fetch api, WeakSet və WeakMap obyektləri, apilərlə işləmək və s. mövzuları da araşdırıb öyrənməyiniz məsləhət görülür.

Mövzuları öyrənərkən kodları tam sərbəst anlayana kimi yazmağınız mütləqdir. Bir kodu 10 dəfə yazsanız belə problem deyil. Ən əsas olan mövzunu öyrənməyinizdir.

Öz fantaziyanıza və ya internetdə tapdığınız mənbələrə uyğun proyektlər hazırlamağa çalışın. Proyektlər sizin təcrübənizi artırmaq baxımından önəmlidir.

Çox sayda alqoritmik məsələ həll etməyə çalışın. Alqoritmik məsələlər sizin problem həll etmə bacarığınızı artıracaqdır. Həlləri mümkün qədər özünüz düşünün. Bu müddət 1 gün, hətta 1 həftə də ola bilər. Ancaq özünüz həll edəcəyiniz problemlər sizin proqramlaşdırma məntiqinizi gücləndirəcəkdir.

Oyunlar hazırlayın. Kiçik də olsa oyunlar hazırlamaq sizin proqramlaşdırma məntiqinizi gücləndirəcəkdir.

Kitabın sonu

Əziz oxucu, həqiqətən Javascript o qədər genişdir ki, yaz-yaz bitmir desək, əsl yerinə düşər. Bu kitabda Javascriptdən müəyyən qədər yazmağa çalışdıq. Kitabı yazarkən maksimum çalışdıq ki, kitab heç proqramlaşdırma ilə məşğul olmayan biri üçün rahat olsun. Ancaq yenə də, Javascriptə başlamaq üçün Html və Css biliklərinin olması mütləqdir. Bu kitabda Html və Css bilən biri üçün müxtəlif Javascript mövzularını izah etməyə çalışdıq.

Dediyimiz kimi, Javascript o qədər genişdir ki, tək bu kitabda Javascriptin bütün imkanlarını yazmağımız mümkünsüz kimi bir şeydir. Ona görə kitabın növbəti buraxılışlarını da yazmağı nəzərdə tuturuq.

Günümüzdə Javascriptin istifadəsi hədsiz genişlənməkdədir. Javascriptlə mobil tətbiqlər, oyunlar da hazırlanmaqdadır. Növbəti buraxılış-larımızda bunları da nəzərə alacağıq. Hələlik bu qədər. Növbəti kitabda görüşənə qədər :)