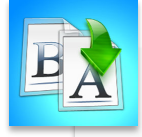


# JAVASCRIPT'TE İLERİ KONULAR



## İÇİNDEKİLER

- Metin İşlemleri
- Döngü Komutları
- Diziler
- Fonksiyonlar
- Olaylar
- DOM ile Programlama
- Window Nesnesi



## HEDEFLER

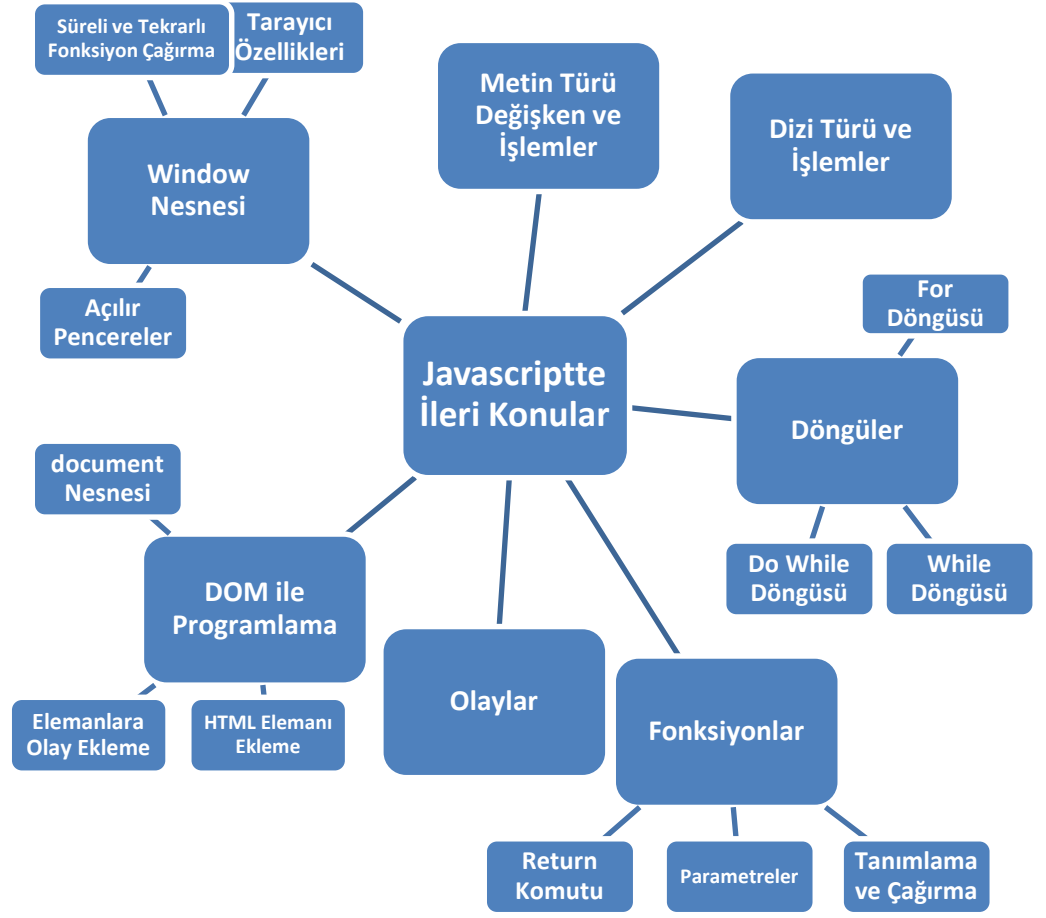
- Bu üniteyi çalıştıktan sonra;
- Javascript'te metinler üzerinde işlem yapabilecek,
- Dizileri ve döngüleri programlarınızda kullanabilecek,
- Farklı türde fonksiyon tanımlayabilecek ve kullanabilecek,
- DOM ile safta elemanlarına erişebilecek,
- Window nesnesi ile tarayıcı üzerinde ileri işlemler yapabileceksiniz.



**Atatürk Üniversitesi**  
Açıköğretim Fakültesi

## İNTERNET PROGRAMCILIĞI I Dr. Öğr. Üyesi Halil ERSOY

## ÜNİTE 11



## GİRİŞ

Önceki iki ünite de Javascript'in temelleri olabilecek konular işlenmişti. Bu ünite de ise Javascript dili ile yapılabilecek ileri düzey işlemler için gerekli olacak diğer konular ele alınacaktır.

Değişken türlerinde kısaca bahsedilen *metin (string)* türü üzerinde yapılabilecek birçok işlem mevcuttur. Metinler ile yapılabilecek işlemler iki açıdan önemlidir. İlki metin türünde girilmiş değerler üzerinde işlem yapmak, ikincisi HTML etiketlerini metin türünde düzenlemek. Her iki durum aşağıdaki bölümde detaylıca ele alınacaktır.

Programın akışını kontrol eden koşul yapıları önceki ünite de ele alınmıştı. Bunlara ek aynı işlemlerin birden fazla defa yapılması istendiğinde komutların *döngüler* içerisinde yazılması konusu, bu ünite de gösterilecektir. For, while ve do döngüleri bu amaçla ünite de ele alınacaktır.

Yine veri türü olarak kısaca önceki ünite de ifade edilen *diziler*, bu ünite de detaylıca anlatılacaktır. Dizilerin döngüler ile bir arada kullanılması çok yaygın bir uygulamadır. Bu nedenle her iki konu birbirini destekleyici niteliktedir.

Yazılan komutların satır satışı arttıkça, programların anlaşılması ve olası hataların ayıklanması güçleşir. Ayrıca bir işi yapan komutların kümeler hâlinde tekrar tekrar kullanılması durumunda, satır sayıları anlamsızca artar. Bu karmaşıklığı ve gereksiz tekrarlamayı engellemek için komutlarımızı *fonksiyonlar* hâlinde yazmaya çalışacağız. Fonksiyonlar konusu tıpkı diğer diller gibi Javascript için de önemli bir konudur ve bu ünite de ele alınacaktır.

Javascript'in HTML etiketleri üzerindeki etkisini göstermek için *DOM* (Document Object Model) modelinin bilinmesi gerekir. Bu model çerçevesinde hangi etikete ne şekilde erişebileceğimiz belirlenir. Aynı zamanda bu modele göre HTML etiketlerinin veya kullanıcı hareketlerinin tetiklediği *olaylar* da bu ünite de ele alınacaktır.

Son olarak DOM içerisindeki *window* nesnesinin tarayıcı özellikleri, tarayıcı geçmiş ve açılı pencereler gibi metotları anlatılmıştır.

## METİN TÜRÜ VE İŞLEMLERİ

Değişken ve veri türü olarak metin (string) türünü daha önceki ünite de görmüştük. Aşağıdaki örnekte tüm değişkenlerin türü metindir. Dikkat edilirse hepsinin çift ya da tek tırnak içerisinde olduğu görülmektedir.

**Tablo 11.1.** Metin Türü Değişken Ve Değerler

Javascript Komutları
<pre>&lt;script&gt; var a="Merhaba dünya"; var b='Merhaba dünya'; var c="3.1417"; var d="Ankara'da hava sıcak";</pre>



Metin türündeki değerleri çift tırnak ya da tek tırnak ile sınırlandırabilirsiniz.

```
var e="<p>Hikaye başlıyor...</p>";
</script>
```

Metin değerlerini sınırlandırmak için başına ve sonuna tek tırnak (ya da kesme) veya çift tırnak karakterleri kullanılır. Ancak hangisi ile başlanırsa yine aynıysa ile sonlandırılmalıdır.

Eğer metin içinde sembol olarak tek tırnak ya da çift tırnağa ihtiyaç varsa önüne ters bölü ( \ ) sembolü eklenerek metin içerisinde yer verilebilir.



Boşluk, noktalama işareti, matematiksel semboller gibi tüm karakterler metin değişkenlerinde yer kaplar.



Örnek

```
•var d="Ankara'da hava sıcak<br>";
•var e='Ankara\'da hava sıcak<br>';
•var f="Ankara \"sıcaklık\" artıyor<br>";
•document.write(d); // Çıktı : Ankara'da hava sıcak
•document.write(e); // Çıktı : Ankara'da hava sıcak
•document.write(f); // Çıktı : Ankara "sıcaklık" artıyor
```

Metin türündeki değişkenler, aktarıldığı değişken içerisinde aslında sıraya dizilmiş karakter kümeleridir. Dolayısı ile her bir karaktere sırasını belirten bir sayı ile aşağıdaki gibi erişilebilir.

var a="Merhaba dünya";												
M	e	r	h	a	b	a		d	ü	n	y	a
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]	a[12]

Şekil 11.1. Metin karakterlerinin indis numaraları

- document.write(a[0]); // → 'M'
- document.write(a[12]); // → 'a'

Sıra (indis) *numarası 0'dan* başlar, bu nedenle metindeki ile karakterin sıra numarası 0'dır. Son karakterin numarası ile *karakter sayısının bir eksiğidir*. Bu sıra numarasına bundan sonra indis denilecektir.

Javascript'te metin üzerinde yapılacak işlemlerde kullanılacak hazır fonksiyonlar aşağıda anlatılmıştır. Bu fonksiyonlarda hangi karakter veya karakterler üzerinde işlem yapılacağı ilgili karakterlerin *indis numaraları* üzerinden anlatılmıştır.

## Metin Uzunluğu

Bir metindeki karakter sayısına o metnin karakter uzunluğu denir. Karakter uzunluğu görünen veya görünmeyen tüm karakterlerin toplamıdır. Bir metnin uzunluğunu bulmak için metin türündeki değişkenin *.length* özelliği kullanılır.

**Tablo 11.2.** Length Özelliği İle Metin Uzunluğu Bulma

JavaScript Komutları
<pre>var a="Merhaba dünya"; document.write(a.length); → 13 var b="Ankara'da"; document.write(b.length); → 9 var c="1350 TL"; document.write(c.length); → 7</pre>

*Length* özelliği metni oluşturan tüm karakterlerin adedini verir. Buna göre metnin *ilk karakterinin indisi 0* iken, *son karakterin indisi a.length-1*'dir.

## Büyük ve Küçük Harf Kontrolü ve Çevirme

Metindeki karakterlerin büyük harfe ya da küçük harfe çevrilmesi için *toUpperCase()* ve *toLowerCase()* özellikleri kullanılır.

**Tablo 11.3.** Büyük Ve Küçük Harfe Çevirme

JavaScript Komutları
<pre>var a="Merhaba dünya"; a=a.toUpperCase(); → MERHABA DÜNYA a=a.toLowerCase(); → merhaba dünya</pre>

Yukarıdaki iki fonksiyonda dikkat edilmesi gereken, fonksiyonların çağırıldıkları metin içerisindeki değerleri değiştirip geriye değişmiş hâlini döndürdükleridir. Aşağıdaki örneği dikkatli inceleyin.

**Tablo 11.4.** Touppercase Fonksiyonu

JavaScript Komutları
<pre>var a="Merhaba dünya"; document.write(a); → Merhaba dünya document.write(a.toUpperCase()); → MERHABA DÜNYA document.write(a); → Merhaba dünya a=a.toUpperCase(); document.write(a); → MERHABA DÜNYA</pre>

*ToUpperCase* ve *toLowerCase* fonksiyonları metin içerisindeki rakam veya noktalama işaretlerinde işlem yapmaz.

Eğer metindeki bir karakterin büyük ya da küçük olduğunu sorgulamak isterseniz aşağıdaki örnekte gösterildiği üzere karakterin, kendisinin büyük ya da küçük hâline eşit olup olmadığı kontrol edilebilir.

**Tablo 11.5.** Küçük Ya Da Büyük Harf Kontrolü Örneği

JavaScript Komutları
<pre>var a="Merhaba dünya"; if(a[0] == a[0].toUpperCase()) document.write("ilk karakter büyük")</pre>



“toUpperCase” ve “toLowerCase” fonksiyonları hem bir metin hem de bir karakter için çalışır.

```
else
    document.write("ilk karakter küçük")
```

**Çıktı:**

İlk karakter büyük

## Metin İçinde Arama Yapma

Metin içerisinde karakter ya da başka metin aramak için *indexOf()* ve *lastIndexOf()* fonksiyonları kullanılır. Bu fonksiyonlar aradıkları ifadenin metin içerisindeki, varsa, *indis numarasını* geri döndürürler.

*IndexOf()* baştan, *lastIndexOf()* ise sona aramaya başlar. Eğer ifade bulunmazsa -1 değerini döndürürler. Arama yine büyük ve küçük harfe duyarlıdır.

**Tablo 11.6.** Indexof Ve Lastindexof Fonksiyonları

### Javascript Komutları

```
var a="Merhaba dünya";
var konum1=a.indexOf("a");    → 4
var konum2=a.lastIndexOf("a"); → 12
var konum3=a.indexOf("A");    → -1
```

## Metnin Karakterleri Üzerinde İşlem Yapma

Metnin karakterlerine, karakter dizileri gibi *indis numarası* ile erişilebilir. Ancak bu metnin bir dizi olduğu anlamına gelmez. İndis numarası ile karakterler ulaşabiliriz ancak onları bu yolla *değiştiremeyiz*.

**Tablo 11.7.** Karakterlere İndisle Erişme

### Javascript Komutları

```
var a="Merhaba dünya";
a[0]="T";    // Hata vermez ancak değişiklik yapmaz!
document.write(a); // Çıktı: Merhaba dünya
```

Eğer karakterler üzerinde işlem yapmak istersek o *zaman metni önce karakter dizisine dönüştürmek* gerekir. Dizi konusu ileride anlatılacaktır ancak metinlerin diziye dönüştürülmesini aşağıdaki örnek ile burada görelim.

**Tablo 11.8.** Karakterleri Diziye Çevirme Ve Birleştirme

### Javascript Komutları

```
var a="Merhaba dünya";
var k=a.split(""); // karakterlere göre parçalama
document.write(a);
k[0]="T"; // İlk karakteri T yapma
document.write(k);
a=k.join(""); // Karakterleri birleştirme ve string yapma
document.write(a);
```

**Çıktı:**

```
Merhaba dünya
T,e,r,h,a,b,a ,d,ü,n,y,a
Terhaba dünya
```



IndexOf ve lastIndexOf metotları sadece ilk buldukları karakterin yerini gösterir, devamını kontrol etmez.



Karakter dizisi ile metin tamamen aynı türde değişken değildir, ancak birbirine dönüştürülebilir.

## DÖNGÜLER

Programlama sürecinde bir ya da birden çok komutun, bir kez yazıldığı hâlde birden çok kez tekrar çalışmasını sağlayan komutlara döngüler denir. Döngüler tıpkı koşul komutları gibi programlama dillerinin temel komutlarıdır ve programların daha az sayıda satır ile yazılabilmelerini sağlar.

Javascript'te üç farklı komut döngü komutu vardır:

- for
- while
- do while

Bu üç döngü aşağıda ayrı ayrı ele alınmıştır. Öte taraftan bu üç döngünün de ek değişkenler ile birbiri yerine kullanılabilecekleri unutulmamalıdır.

### For Döngüsü

Tekrarlanma sayısı belirli olan durumlarda **for** döngüsü kullanmak tavsiye edilir. For döngüsünün yazım şekli aşağıdaki gibidir:

**Tablo 11.9.** For Döngüsü Yapısı

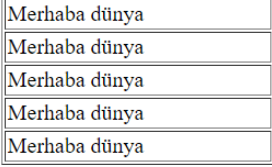
For Döngüsü Yapısı ve Tanımlar	
<pre>for(komut1; koşulKomutu; komut3) {   tekrarlanması istenilen komutlar;   tekrarlanması istenilen komutlar;   tekrarlanması istenilen komutlar; }</pre>	
komut1;  örneğin i=0; yada sayac=50;	Bu komut ilk sırada ve bir defa çalıştırılır. Genellikle döngüde kullanılacak sayaç değişkeninin ilk değerini vermek için kullanılır. Eğer istenirse birden fazla değişkene, aralarına virgül konularak ilk değer verilebilir. Sonunda noktalı virgül yer almalıdır.
koşulKomutu;  örneğin i<100; ya da sayac%5==0;	Komut1'den sonra çalıştırılır, sonucun true (doğru) olması durumunda döngü içerisindeki "tekrarlanması istenilen komutlar" bloku çalıştırılır. Eğer koşulKomutu değeri false (yanlış) olursa for döngüsü sona erer ve program akışı for döngüsü bloku dışından devam eder.
komut3;  örneğin i++; sayac--;	Blok çalıştıktan sonra komut3 çalıştırılır. Genellikle, sayaç olarak kullanılan değişkenin artma ya da azaltma komutu verilir.  Bu komuttan sonra akış yine koşulKomutuna geçer ve koşulun sonucuna göre ya blok çalıştırılır ya da döngü sona erer.



For döngüsü dizilerle birlikte sıklıkla kullanılır.

Aşağıdaki örnekte web sayfasında bir tabloya 5 satır eklenmiş olur. Bu örnekteki *sayac* değişkeni ilk değeri olan 1 ile döngü başlar. Ardında *sayac++* komutu ile değeri 1 artarak sırasıyla 2, 3, 4 ve 5 olur ve for döngüsü altındaki *document.write()* komutu 5 kez çalışır. Artış komutu *sayac* değişkeninin değerini 6 yapar, ancak 6 değeri koşulda *false* sonucunu getirdiği için döngü sona erer.

**Tablo 11.10.** For Döngüsü İle Tabloya Satır Ekleme

JavaScript Komutları
<pre>&lt;table border="1" width="200"&gt; &lt;script&gt; var sayac; for(sayac=1; sayac&lt;=5; sayac++)   document.write("&lt;tr&gt;&lt;td&gt;Merhaba dünya&lt;/td&gt;&lt;/tr&gt;"); &lt;/script&gt; &lt;/table&gt;</pre>
ÇIKTI


Döngü kullanımında yapılan iki tür hata vardır: Döngünün hiç başlamaması ya da döngünün hiç bitmemesi, yani sonsuz döngü durumu. Her iki durum da çoğu zaman programcının mantık hatasından kaynaklanır. Bu iki hata tüm döngülerde yapılabilir. Aşağıdaki iki örnekten ilkinde *for* döngüsünün hiç başlamadığı, ikincisinde ise döngünün sona eremediği görülebilir.



Web sayfalarında oluşacak **sonsuz döngüler**, o noktadan sonra sayfanın yüklenmesini engelleyebilir ya da sayfayı kullanılamaz hâle sokabilir.



Örnek

- Hatalı döngü örneği: Döngü hiç başlamaz!  

```
var sayac;
for(sayac=1; sayac>=5; sayac++)
  document.write("Merhaba");
```
- Hatalı döngü örneği: Döngü başlar ancak hiç bitmez!  

```
var sayac;
for(sayac=1; sayac<=5; sayac--)
  document.write("Merhaba");
```

*For* döngüsünün yazım şeklindeki 3 kısımdan (komut1, koşulKomutu ve komut3) herhangi biri istenirse yazılmayabilir. Bu durumda eksik kısmın işlevini yapacak ek komutlar döngü öncesinde ya da döngü içerisinde yazılmalıdır. Aşağıdaki örneklerde komut1 ve komut3'ün olmadığı *for* döngüleri görülebilir. Bu komutlar geçerli olarak çalışır ancak profesyonel olarak anlaşılması zor olduğu için önerilmez.



Tablo 11.11. For Döngüsü Örnekleri

Javascript Komutları	ÇIKTI
<pre>var sayac=1; for(; sayac&lt;=5; sayac++)   document.write(sayac + " . satır&lt;br&gt;")</pre>	1 . satır 2 . satır 3 . satır 4 . satır 5 . satır
<pre>var sayac=1; for(; sayac&lt;=5;){   document.write(sayac+" . satır&lt;br&gt;");   sayac++; }</pre>	1 . satır 2 . satır 3 . satır 4 . satır 5 . satır



For döngüsünde bazı komutlar parantez içine yazılmasa bile noktalı virgül karakterleri ( ; ) mutlaka yazılmalıdır.

## While Döngüsü

Tekrar etmesini istediğimiz komutların, belirli bir koşul sağlandığı sürece tekrar çalıştırılmasını sağlayan döngü **while** döngüsüdür.

Tablo 11.12. While Döngüsü Yapısı

While Döngüsü Yapısı	
<pre>while(koşulKomutu) {   tekrarlanması istenilen komutlar;   tekrarlanması istenilen komutlar;   tekrarlanması istenilen komutlar; }</pre>	
koşulKomutu	Bu koşul true veya false üreten bir ifade olmalıdır. İfade true (doğru) değerini ürettiği sürece döngü içerisindeki komutlar çalıştırılır.

**While** döngüsünde, daha önce bahsedilen iki hata riski daha fazladır. Döngünün hiç başlamaması veya döngünün hiç bitmemesi için koşul dikkatli yazılmalı ve döngü bloku içerisinde bu koşulu er ya da geç false değerine dönüştürecek işlemler unutulmamalıdır.

Tablo 11.13. While Döngüsü Örneği

Javascript Komutları
<pre>&lt;table border="1" width="200"&gt; &lt;script&gt; var sayac=1; // önemli while(sayac&lt;=5){   document.write("&lt;tr&gt;&lt;td&gt;Merhaba dünya&lt;/td&gt;&lt;/tr&gt;");   sayac++; // önemli } &lt;/script&gt; &lt;/table&gt;</pre>
ÇIKTI

Merhaba dünya
Merhaba dünya
Merhaba dünya
Merhaba dünya
Merhaba dünya

Yukarıdaki komutlarda önemli olarak işaretlenen satırlar, döngünün sağlıklı olarak başlaması ve bitebilmesi için dikkatlice yazılmalıdır.

## Do While Döngüsü

Tekrar edilecek işlemlerin en az bir defa çalıştırılması ve devamında bir koşula göre tekrar çalıştırılması gerekirse, kullanılması gereken döngü *do while* olabilir. *Do while* döngüsünün yapısı aşağıdaki gibidir.

**Tablo 11.14.** Do While Döngüsü

Do while döngüsü yapısı	
<pre>do{   tekrarlanması istenilen komutlar;   tekrarlanması istenilen komutlar;   tekrarlanması istenilen komutlar; }while(koşulKomutu);</pre>	
koşulKomutu	<p>Önce blok içerisindeki komutlar bir kez çalıştırılır.</p> <p>Ardından true veya false üreten koşulKomutu kontrol edilir.</p> <p>Sonuç true (doğru) değerini ürettiği sürece döngü içerisindeki komutlar çalıştırılır.</p>



*While* ve *do..while* döngüleri birbirine dönüştürülebilir.

*Do while* döngüsünde koşulun blok sonunda olmasından anlaşılacağı üzere, döngü içerisindeki komutlar koşuldan *önce* bir defa çalıştırılır. Ardından koşul kontrol edilir. Diğer döngülerdeki gibi döngünün sonsuz döngüye dönüşmemesi veya tekrar etmesini engelleyecek koşulların yazılmamasına özen gösterilmelidir.

## DİZİLER

Aynı türde birden fazla değeri ayrı ayrı tutabilen değişkenlere *dizi* (*array*) denir. Kimi programlama dillerinde diziler bir değişken türü, kimilerinde ise veri tutan nesne türü olarak tanımlanmıştır. Javascript'te bu yapılar tür olarak *nesne* (*object*) türündedir.

Birçok dilden farklı olarak Javascript'te dizilerin her bir elemanı farklı türlerde değer tutabilir. Ancak bu ünite sadece tüm değerleri aynı türde değer tutan diziler ele alınmıştır.

## Dizi Tanımlama

Diziler aşağıdaki gibi tanımlanabilir.

Tablo 11.15. Dizi Tanımlama Örnekleri

JavaScript Komutları
<pre>var dizi1=[2,4,6,8,10]; document.write("dizi1 elemanları=" + dizi1 + "&lt;br&gt;");  var dizi2=["TL", "USD", "EU"]; document.write("dizi2 elemanları=" + dizi2 + "&lt;br&gt;");  var dizi3=new Array(5,10,15,20); document.write("dizi3 elemanları=" + dizi3 + "&lt;br&gt;");</pre>
ÇIKTI
<pre>dizi1 elemanları=2,4,6,8,10 dizi2 elemanları=TL,USD,EU dizi3 elemanları=5,10,15,20</pre>



Metin türündeki değişkenler dizi gibi indisli bir yapıya sahiptir ancak tam anlamı ile dizi değildir.

Yukarıdaki örnekten anlaşılacağı üzere;

- Diziler *farklı* türlerde değerleri saklayabilir.
- Dizilere ilk değerler, *köşeli parantez* içerisinde ve aralarına virgül yazılarak verilebilir.
- Dizilere ilk değerler, *new Array()* komutunun parantezleri içerisinde aralarına virgül yazılarak verilebilir.
- *document.write()* komutu dizinin tüm elemanlarını, ekrana aralarına *virgül* koyarak yazabilir.

Dizileri yaratırken *new Array()* komutunun kullanılmasına gerek yoktur. Yukarıdaki örnekte dizi1 ve dizi3 aynı özelliklere sahip dizilerdir. *New Array()* komutunun kullanılmasının tek avantajı, eleman sayısı belli ancak değerleri henüz belirlenmemiş dizi yaratabilmektir.



Örnek

- var d1=[]; // 0 elemanlı bir dizi
- var d2=new Array(); // 0 elemanlı bir dizi
- var d3=new Array(5); // 5 elemanlı bir dizi

Öte taraftan *new Array()* komutundaki parantezlerin içerisine yazılan değerlere dikkat etmek gerekir. Eğer tek bir tam sayı yazılırsa, o zaman bu sayı dizinin *eleman sayısı* olarak algılanır. Eğer birden fazla değer yazılırsa, bunlar dizinin elemanlarının *ilk değerleri* olarak algılanır.

Tablo 11.16. New Array Komutu Örnekleri

JavaScript Komutları ve Çıktılar
<pre>var d3=new Array(5); → 5 elemanlı bir dizi var d3=new Array(5,6,7); → 3 elemanlı bir dizi</pre>

## Dizilerin Elemanlarına Erişim ve İndis Numaraları

Dizilere atanan değerlere o değerın sırasına göre indis numarası ile teker teker erişilebilir. *İndis numarası* her zaman 0'dan başlar.

**Tablo 11.17.** Dizi Ve Elemanlarına Erişim Örneği



Dizilerde ilk elemanın indis numarası her zaman sıfırdır.

### Javascript Komutları

```
var sembol=["TL", "USD", "EU"];
document.write("sembol elemanları=" + sembol + "<br>");
document.write("sembol[0]=" + sembol[0] + "<br>");
document.write("sembol[1]=" + sembol[1] + "<br>");
document.write("sembol[2]=" + sembol[2] + "<br>");
document.write("<hr>");
```

```
sembol[2]="gbp";
document.write("sembol[2]=" + sembol[2] + "<br>");
```

```
sembol[2]=sembol[2].toUpperCase(sembol[2]);
document.write("sembol[2]=" + sembol[2] + "<br>");
```

### ÇIKTI

```
sembol elemanları=TL,USD,EU
sembol[0]=TL
sembol[1]=USD
sembol[2]=EU
```

```
sembol[2]=gbp
sembol[2]=GBP
```

Dizilerin elemanlarının her biri kendi türlerine göre (number/sayı, metin/string, vb.) farklı işlemlere tabi tutulabilir. Yukarıdaki örnekte sembol[2] ifadesi *string* bir değerdir ve *toUpperCase()* fonksiyonuna parametre olarak gönderilebilir.

Dizilerin her bir elemanı *indis numarası* kullanılarak erişilebilir ve değiştirilebilir. Yukarıdaki örnekte sembol[2]= "gbp"; satırında üçüncü değer değiştirilmiştir.

## Dizinin Eleman Sayısı

Dizilerin eleman sayısına “dizinin uzunluğu” da denilebilir ve *length* ifadesi ile tamsayı olarak elde edilebilir.

**Tablo 11.18.** Farklı Dizilerin Length Özelliği

### Javascript Komutları

```
var sembol=["TL", "USD", "EU"];document.write("sembol elemanları=" + sembol + "<br>");
document.write("eleman sayısı=" + sembol.length + "<hr>");
```

```
var d1=[];
```



Hiç elemanı olmayan dizilerin eleman sayısı (*length* değeri) 0'dır.

```
document.write("d1 elemanları=" + d1 + "<br>");
document.write("eleman sayısı=" + d1.length + "<hr>");

var d2=new Array(4);
document.write("d2 elemanları=" + d2 + "<br>");
document.write("eleman sayısı=" + d2.length + "<hr>");

var d3=[5.0, 3.14, 7.65];
document.write("d3 elemanları=" + d3 + "<br>");
document.write("eleman sayısı=" + d3.length + "<hr>");
```

#### ÇIKTI

sembol elemanları=TL,USD,EU  
eleman sayısı=3

d1 elemanları=  
eleman sayısı=0

d2 elemanları=,,,  
eleman sayısı=4

d3 elemanları=5,3.14,7.65  
eleman sayısı=3

## Dizilerin Elemanlarına Döngüler ile Erişim

Dizilerin elemanlarının her biri için yapılacak işlemler genellikle aynı olduğu için bu işlemlerin döngüler içerisinde yapılması tercih edilir. Dizilerin eleman sayıları belirli olduğu için (*length* özelliği ile) bu işlemlerde en çok *for* döngüsü tercih edilir.

**Tablo 11.19.** For Döngüsü İle Elemanlara Erişim

#### Javascript Komutları

```
var sayilar=[4,6,2,8, 4,5,12];
var sayac, toplam = 0;
var ortalama;
for(sayac= 0; sayac < sayilar.length; sayac++){
    toplam += sayilar[sayac];
}
ortalama = toplam / sayilar.length;
document.write("Toplam=" + toplam + "<br>");
document.write("Eleman sayısı=" + sayilar.length + "<br>");
document.write("Ortalama=" + ortalama + "<br>");
```

#### ÇIKTI

Toplam=41  
Eleman sayısı=7  
Ortalama=5.857142857142857



Dizilerin ilk elemanının indisi *sıfır* ( 0 ), son elemanın indisi ise *dizi.length-1*'dir .

Yukarıdaki komutlardan *for* satırında, *koşul ifadesine* dikkat ediniz. Eğer “ < ” sembolü yerine “ <= ” sembolü kullanılırsa, sayaç 7 olacak ve 7. eleman için döngü komutları çalışıp hata oluşacaktır. Bunun sebebi, dizideki son elemanın indisinin 6 olmasıdır. Dizinin sahip olmadığı bir elemana ulaşmak *hata* oluşturur.

## Dizi Elemanları İçinde Arama Yapma

Dizilerin elemanları arasında arama yapmak için *indexOf* ya da *lastIndexOf* fonksiyonları kullanılır. *indexOf* aranan değerin dizinin *başından* itibaren ilk bulunduğu konumun indis numarasını, *lastIndexOf* ise *sondan* itibaren ilk bulunduğu konumun indis numarasını döndürür. Eğer değer *bulunmazsa* -1 değeri döndürülür. Metin türündeki dizilerde arama işlemlerinde *büyük ve küçük harf* fark eder.

**Tablo 11.20.** Indexof Ve Lastindexof Fonksiyonları İle Arama

Javascript Komutları ve Çıktıları	
var sayilar=[10,5,2,3,8,1,5];	
var sehirler=["İstanbul", "Ankara", "İzmir"];	
var sonuc;	
sonuc=sayilar.indexOf(5);	→ 1 döndürür
sonuc=sayilar.indexOf(4);	→ -1 döndürür
sonuc=sayilar.lastIndexOf(5);	→ 6 döndür
sonuc=sehirler.lastIndexOf(İzmir);	→ 2 döndür
sonuc=sehirler.lastIndexOf(izmir);	→ -1 döndür, i küçük!

## FONKSİYONLAR

Birden fazla komutu gruplayarak tek bir isim ile çağırabildiğimiz yapılara *fonksiyon* denir. Fonksiyonlar tüm programlama dillerinde benzer şekilde kullanılırlar. Sıklıkla yapılan işlerin tekrarlanması durumunda gereksiz yere yer işgal etmemeleri ve ya farklı değerler için aynı işlemin yapılması gerektiğinde yeniden aynı kodları yazmamak adına bu işler fonksiyon hâline getirilir.

**Tablo 11.21.** Indexof Ve Lastindexof Fonksiyonları İle Arama

Javascript Komutları ve Çıktıları	
function topla(a,b){ return a + b; }	
var sayi1=10, sayi2=30;	
var sonuc=topla(sayi1, sayi2);	
document.write("Sonuc=" + sonuc);	→ Sonuc=40
document.write("Sonuc=" + topla(sayi1,sayi2));	→ Sonuc=40
document.write("Sonuc=" + topla(5,3));	→ Sonuc=8
document.write("Sonuc=" + topla(3.14 , 7.5));	→ Sonuc=10.64



Fonksiyonlar değişken adlandırma kurallarına göre adlandırılır.

```
document.write("Sonuc=" + topla("merhaba","dünya"));
→ Sonuc=merhabadünya
```

Yukarıdaki örnekte görüleceği üzere, **topla()** adındaki bir fonksiyon ilk 3 satırda **tanımlanmış**, ardından 5 farklı satırda kullanılmış, yani **çağırılmıştır**. Fonksiyonların isimleri tıpkı değişkenler isimleri gibi amacına uygun biçimde ve isimlendirme kuralları uygulanarak geliştirici tarafından belirlenir.

Fonksiyonlar ancak **tanımlandıktan** sonra **çağırılabilirler**. Tanımlama kısmı ve çağırma kısmı genellikle aşağıdaki kalıba uygun biçimde yapılır.

**Tablo 11.22.** Fonksiyon Çağırma Yolları

JavaScript'te Fonksiyon Tanımlama
<pre>function fonksiyonAdi (varsaParametre1, varsaParametre2, ... ) {     Fonksiyon içinde çalışması istenen komutlar....     return varsaGeriDönüşDeğeri; }</pre>
ÇAĞIRMANIN İKİ YOLU
<pre>// 1. Geri dönüş değeri varsa: var değişken= fonksiyonAdi(argüman1, argüman2...) // 2. Geri dönüş değeri yoksa fonksiyonAdi(argüman1, argüman2);</pre>



Bir fonksiyon bir defa tanımlanır ancak birden çok defa çağırılabilir.

## Parametreler ve Argümanlar

Parametreler aslında değişkenlerdir. Fonksiyonun eğer istenirse çağırıldığı noktadan değişkenler ya da değerler almasını sağlar. Tanımlama başlığında parantez içerisinde bir ya da birden çok parametre alabilir, eğer parametre istenmezse o zaman parantezlerin içi boş bırakılır.

Tanımlama kısmında eğer **parametre** eklenirse, o zaman o fonksiyon çağırılma kısmında da aynı sayıda **argüman** ile çağırılmalıdır. Argümanlar değişken ya da açık bir değer olabilir. Aşağıdaki örnekte tanımlamaya, çağırmaya ve açıklamalara dikkat ediniz:

**Tablo 11.23.** Fonksiyon Tanımlamaları Ve Çağırılmalar

JavaScript Komutları
<pre>// Dört fonksiyon tanımlaması function paragrafEkle(metin){ // bir parametrelilik fonk.     document.write("&lt;p&gt;" + metin + "&lt;/p&gt;"); } function baslikEkle(metin){ // bir parametrelilik fonk.     document.write("&lt;h1&gt;" + metin + "&lt;/h1&gt;"); }  function çizgiEkle(){ // parametresiz fonk.     document.write("&lt;hr&gt;");</pre>



Hiç argümanı olmayan fonksiyonları çağırmak için içi boş parantezler yine de yazılmalıdır.

```

}

function ussunuBul(sayi, us){ // iki parametrelili ve
    var carpim=1, i; // geri dönüş değerli fonk.
    for(i=0; i<us; i++)
        carpim *= sayi;
    return carpim;
}

// Fonksiyonları çağırılması
var a=2, b=3;
document.write("a^b=" + ussunuBul(a,b) + "<br>"); //iki argüman
document.write("3^3=" + ussunuBul(3,3) + "<br>"); //iki argüman
cizgiEkle(); //argüman yok
baslikEkle("Fonksiyonlar Konusu"); //bir argüman
paragrafEkle("Merhaba, bu bir paragraftır"); //bir argüman
paragrafEkle("Bu da ikinci paragraf olsun"); //bir argüman
cizgiEkle(); //argüman yok
    
```

#### ÇIKTI:

a^b=16  
3^3=81

## Fonksiyonlar Konusu

Merhaba, bu bir paragraftır

Bu da ikinci paragraf olsun

#### HTML

```

<script>...</script>
"a^b=8"
<br>
"3^3=27"
<br>
<hr>
<h1>Fonksiyonlar Konusu</h1>
<p>Merhaba, bu bir paragraftır</p>
<p>Bu da ikinci paragraf olsun</p>
<hr>
    
```

Parametrelerin değeri fonksiyon içerisinde değişebilir, ancak bu değişiklik temsil ettikleri argümanlara *her zaman yansımaz*. Eğer argümanlar sayısal veya metin türünde değişkenler ise değişiklik yansımaz, ancak parametreler nesne ya da dizi ise değişiklik yansır. Aşağıda buna yönelik iki örnek verilmiştir.

**Tablo 11.24.** Değişen Parametreler Ve Argümanlara Etkisi

#### Javascript Komutları

```

function fonksiyon1(x,y){ // Değerleri değiştiren fonksiyon
    var yedek;
    yedek=x;
    x=y;
    y=yedek;
}

function fonksiyon2(z){ // dizinin her bir elemanına 100
    var i; // ekleyen fonksiyon
    for(i=0; i<z.length; i++)
        z[i] += 100;
    
```



```

}

var a=10, b=20;
var d=[5,10,15,20];

document.write("Fonksiyondan önce a=" + a + " b=" + b + "<br>");
document.write("Fonksiyondan önce d=" + d + "<br>");
fonksiyon1(a,b);
fonksiyon2(d);
document.write("Fonksiyondan sonra a=" + a + " b=" + b + "<br>");
document.write("Fonksiyondan sonra d=" + d + "<br>");

```

**ÇIKTI**

Fonksiyondan önce a=10 b=20	
Fonksiyondan önce d=5,10,15,20	
Fonksiyondan sonra a=10 b=20	→ Değişim yok
Fonksiyondan sonra d=105,110,115,120	→ Değişim var



Return komutu,  
fonksiyonu bitiren  
komuttur.

## Return Komutu

Eğer fonksiyon içerisindeki işlemlerin sonucunda, fonksiyonun çağırıldığı noktaya bir *değer göndermek istenirse*, o zaman fonksiyonun uygun yerine *return* kelimesi ve *yanına* uygun bir değer ya da değişken yazılır. Eğer farklı durumlarda farklı değerler geri döndürülecek ise *birden fazla return yazılabilir*. Fonksiyonların çalışması, eğer varsa, kendi içerisindeki *return* kelimesine ulaşıldığında sona erer. *Return* kelimesinden sonraki komutlar çalıştırılmaz. Öte taraftan eğer *return* kelimesi *yoksa* fonksiyonun tüm komutları çalıştırılır.

Fonksiyonlar, çağırıldıkları anda programın akışını devralır ve kendi içerisindeki komutların çalışması bitene kadar ya da varsa *return* komutuna ulaşana kadar akışı elinde tutar.

**Tablo 11.25.** Return Kelimesi Örneği

Javascript Komutu	ÇIKTI:
<pre> function kucukOlan(x,y){   if(x&lt;y)     return x;   return y; } var a=10, b=20, c; c=kucukOlan(a,b); document.write("Küçük olan=" + c); </pre>	Küçük olan=10

Yukarıdaki örnekte, 10 ve 20 değerleri ile çağrılan fonksiyon, *return x* komutuna gelince x'in değerini (10) *geri* döndürür ve fonksiyon *bu satırda durur*.

## Fonksiyonların Çalışma Zamanı

Fonksiyonlar 3 farklı zamanda çalışabilir:



- Komutlar içerisinde *çağırıldıklarında*,

- HTML sayfasındaki bir elemanın herhangi bir *olayı* tetiklendiğinde,
- Kendi kendine.

Bu üç durumdan ilki yukarıdaki örneklerde gösterilmiştir. Fonksiyon tanımlandıktan sonra adı ve argümanlar ile fonksiyon çağrılabilir.

İkinci durumda fonksiyon çağırmak için HTML etiketlerinin içerisine, o etikete özgü olaylardan herhangi birinin adı ve tanımlanmış olan bir fonksiyonun adı ve argümanlarını uygun biçimde yazmak gerekir. *Olaylar (events)* konusu bu ünitenin devamında ele alınmıştır. Ancak örnek olması açısından aşağıdaki kodlara bakabilirsiniz.

**Tablo 11.26.** Tıklanan Paragrafın Metninin Değişmesi

Javascript Komutları	
<pre>&lt;script&gt; function yaziDegistir(yeniMetin){     document.getElementById("p1").innerText=yeniMetin; } &lt;/script&gt;  &lt;p id="p1" onClick="yaziDegistir('Merhaba');"&gt;Bu bir paragraftır&lt;/p&gt;</pre>	
İLK ÇIKTI:	TIKLAMA SONRASI ÇIKTI
Bu bir paragraftır 	Merhaba 

Yukarıdaki örnekte, *paragraf* etiketine *tıklanma* olayında (*onClick*) çağırılması için *yaziDegistir()* fonksiyonu yazılmıştır. Bu durumda kullanıcı fare ile bu paragrafın üzerine her tıkladığında *yaziDegistir()* fonksiyonu çalışacaktır.



Üçüncü durum olan *kendi kendine* çalışmada yönteminde ise fonksiyona bir isim verilmeden aşağıdaki gibi yazılır. Burada amaç aslında yeni bir fonksiyon yaratmak değil, bazı komutları ve değişkenleri sanki bir fonksiyonun içerisindeymiş gibi kullanıp, lokal oldukları için fonksiyon sonunda hafızadan silinmelerini sağlamaktır.

**Tablo 11.27.** Kendi Kendine Çalışan Fonksiyon

Javascript Komutları
<pre>&lt;p id="bilgi1"&gt;&lt;/p&gt; &lt;p id="bilgi2"&gt;&lt;/p&gt; &lt;script&gt; (function(){     document.getElementById("bilgi1").innerText="İlk fonksiyon çalıştı"; })();  function yap(){     document.getElementById("bilgi2").innerText="İkinci fonksiyon çalıştı";</pre>

```
}
</script>
<button onclick="yap()">Dene</button>
```

#### ÇIKTILAR


İlk fonksiyon çalıştı 	Sayfa yüklenince ilk çıktı
İlk fonksiyon çalıştı İkinci fonksiyon çalıştı 	"Dene" butonu tıklandıktan sonra çıktı

Yukarıdaki örnekte ilk fonksiyon yazıldığı yerde otomatik olarak çalışır. Sanki bir komutmuş gibi ilk fonksiyonun bloku kapandıktan sonra (); sembolleri konur.

## Fonksiyonların JS Dosyalarına Yazılması

Fonksiyonlar genellikle birden çok HTML sayfasında benzer biçimde çalışmaları için yazılırlar. Bu durumda fonksiyonun her HTML sayfasında tekrar tanımlanması gerekir. Bunun yerine fonksiyonlar *JS uzantılı Javascript dosyaları* içerisinde yazılıp, HTML sayfalarına *eklenebilirler*.

**Tablo 11.28.** Fonksiyonların Javascript Dosyalarında Saklanması

Javascript Komutları	
<pre>&lt;!-- index.html dosyası --&gt; &lt;html&gt; &lt;head&gt; &lt;meta charset="utf-8"&gt; &lt;title&gt;Untitled Document&lt;/title&gt; &lt;script language="JavaScript" src="scripts.js"&gt;&lt;/script&gt; &lt;/head&gt; &lt;body&gt; &lt;p id="p1" onclick="yaziDegistir('Merhaba', 'p1');"&gt;Bu bir paragraftır&lt;/p&gt; &lt;/body&gt; &lt;/html&gt;</pre>	
ÇIKTI	
<pre>// scripts.js dosyası  function yaziDegistir(yeniMetin, paragrafID){   document.getElementById(paragrafID).innerText=yeniMetin; }</pre>	
İLK ÇIKTI:	TIKLAMA SONRASI ÇIKTI
Bu bir paragraftır 	Merhaba



Verimliliği ve güvenliği artırmak için Javascript fonksiyonları harici JS dosyaları içerisinde tanımlanır.

Yukarıdaki örnekte, *index.html* dosyasına, aynı klasörde bulunan *scripts.js* dosyasının eklendiği satırı görebilirsiniz. Scripts.js dosyası içerisindeki fonksiyon bu sayede farklı HTML dosyalarında çalışabilir.

Yukarıdaki son örnekte dikkat edilmesi gereken bir başka durum ise fonksiyonun iki parametrelili olmasıdır. İkinci parametre, *ID'si* verilen her türlü HTML etiketi ile çalışabilir.

## OLAYLAR



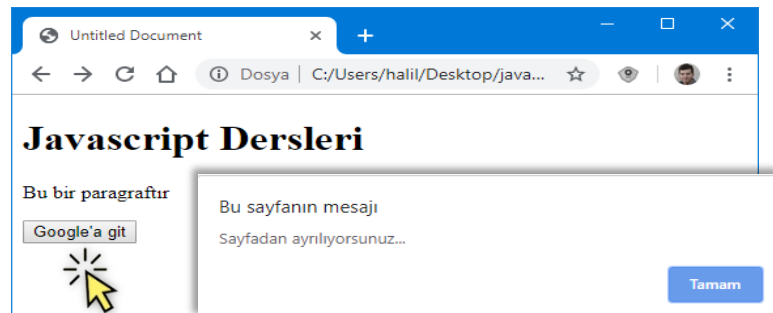
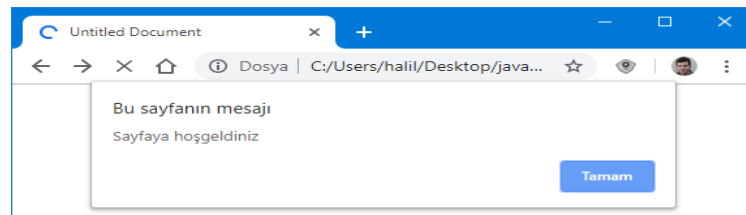
Olaylar, Javascript'in HTML ile iletişimde çok önemlidir.

HTML etiketleri sayfaya yüklenme anında ya da yüklendikten sonra kullanıcının hareketlerini algılayabilirler. Algılayabildikleri bu durumlara *olay* (event) denir. Algılanan bir *olay* sonucunda Javascript komutları ya da fonksiyonları çalıştırılabilir. Aşağıdaki kodlara ve devamındaki çıktı ekranlarına bakınız.

**Tablo 11.29.** Farklı Etiketlerin Farklı Olayları

Javascript Komutları
<pre> &lt;script&gt; function git(adres){   alert("Sayfadan ayrılıyorsunuz...");   document.location.href=adres; } &lt;/script&gt; &lt;/head&gt; &lt;body onload="alert('Sayfaya hoşgeldiniz');"&gt; &lt;h1 onclick="alert('başlığa tıkladınız');"&gt;Javascript Dersleri&lt;/h1&gt; &lt;p onclick="alert('başlığa tıkladınız');"&gt;Bu bir paragraftır&lt;/p&gt; &lt;button onclick="git('http://www.google.com.tr')"&gt;Google'a git&lt;/button&gt; &lt;/body&gt; </pre>

Çıktı aşağıdaki gibi olacaktır.



**Şekil 11. 2.** Sayfanın Yüklenme Ve Butonun Tıklanma Sonrasındaki Görüntüsü

Yukarıdaki kodlara ve ekran görüntülerine bakıldığında, bazı olayların (*onload*) etiketin yüklenmesi sırasında otomatik olarak, bazı olayların ise *fare ile tıklanma* (*onclick*) olayı gerçekleştiğinde algılama yapıldığı anlaşılabilir. Algılanan olayın sonuna *eşittir* ifadesi ile Javascript komutları ya da fonksiyonları yazılabilir.

Olaylar *HTML* dilinin bir parçasıdır, bu nedenle yazım kuralları Javascript'ten farklı biçimde *büyük küçük harf ayrımı* olmadan yazılabilir. Ancak algılanan olayın sonucunda çalıştırılacak Javascript komut ya da fonksiyonları küçük ve büyük harfe duyarlıdır.



Olaylar HTML içerisinde büyük/küçük harfle yazılabilir.

```
<button onclick="git('http://www.google.com.tr')">Google'a git</button>
<button ONCLICK="git('http://www.google.com.tr')">Google'a git</button>
```

Şekil 11. 3. Olayların Farklı Biçimde Yazımı Geçerlidir.

## En Sık Kullanılan Olaylar

HTML etiketlerinin hepsinin kendisine has algılayabildikleri olay listesi vardır. Bu çok uzun bir listedir. Bunun yerine sıklıkla kullanılan aşağıdaki olayları bilmek çoğu durumda geliştiricilere yeterli olacaktır.

Tablo 11.30. En Çok Kullanılan Olaylar

Olay Adı	Açıklaması
<b>onClick</b>	HTML elemanına kullanıcının fare ile tıklanması
<b>onDbIcClick</b>	Elemana fare ile çift tıklanması
<b>onLoad</b>	Body, image veya frame gibi elemanların yüklenmesinin tamamlanması
<b>onUnLoad</b>	Web sayfasından çıkıldığında (tarayıcıyı kapatarak ya da yeni bir sayfaya giderek)
<b>onFocus</b>	Bir elemana odak (focus) yapıldığında
<b>onBlur</b>	Odağa sahip elemanın odağı kaybetmesi sırasında
<b>onMouseOver</b>	Fare işaretçisinin elemanın üzerine geldiğinde
<b>onMouseOut</b>	Fare işaretçisinin elemanın üzerinden çekilmesi sırasında
<b>onMouseMove</b>	Fare işaretçisiyle elemanın üzerinde hareket edilmesinde (elemanın dışına çıkılmadan)
<b>onMouseDown</b>	Fare işaretçisinin elemanın üzerindeyken farenin sol tuşunun basılması
<b>onMouseUp</b>	Fare işaretçisinin elemanın üzerindeyken farenin sol tuşunun bırakılması
<b>onKeyDown</b>	Klavyeden bir tuşa basılması
<b>onSelect</b>	Bir metin kutusu (textarea ya da input) içindeki metnin fare ile seçilmesi
<b>onChange</b>	Bir listeden seçim yapıldığında
<b>onResize</b>	Tarayıcı penceresinin boyutunun değiştirilmesi



Farklı HTML elemanları farklı olayları destekler.

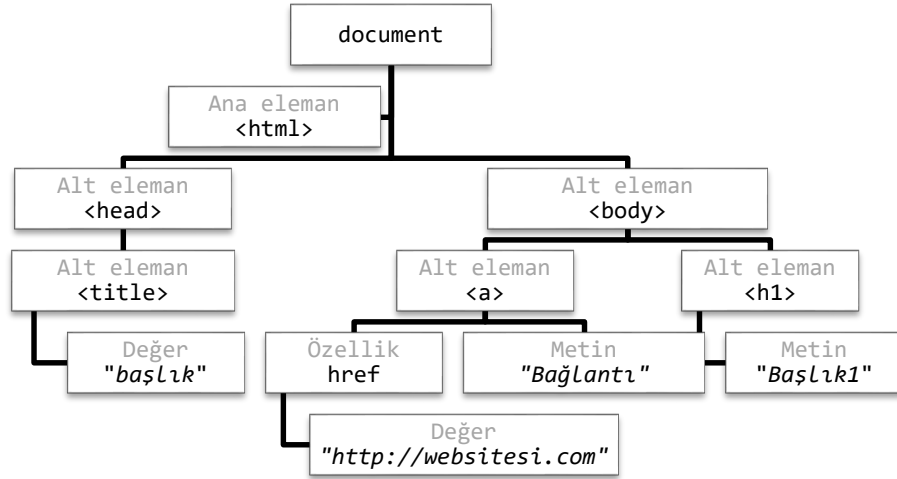


Bireysel Etkinlik

- Yukarıdaki olaylara ilaveten farklı HTML elemanlarına ait onlarca olay vardır. Bu kitap kapsamında olmayan bu olaylara şu adresten ulaşabilirsiniz:  
[https://www.w3schools.com/jsref/dom\\_obj\\_event.asp](https://www.w3schools.com/jsref/dom_obj_event.asp)

## DOM MODELİ

HTML sayfaları yüklendiğinde, tarayıcılar sayfadaki tüm elemanları (HTML etiketlerini) *document* ismindeki bir nesnenin alt nesneleri olarak belirler. Bu hiyerarşi sayesinde HTML sayfasındaki tüm etiketlerin içeriği, özellikleri, CSS stilleri ve alt etiketleri değiştirilebilir. Bu hiyerarşik yapıya *Document Object Model* (DOM) denir.



Şekil 11.4. Örnek Dom Hiyerarşisi

DOM *tarayıcı* tarafından yaratılır ve farklı betik dilleri ile (örneğin Javascript, VBScript vb.) programlanabilir. Bu kitapta standart hâline gelmiş olan Javascript ile DOM programlama gösterilmektedir.

Yukarıdaki şekilde gösterilen hiyerarşik yapı, aşağıdaki HTML kodlarından oluşan bir sayfaya ait olabilir.

Tablo 11.31. Basit Bir HTML Sayfası Kodları Ve Görüntüsü

Javascript Komutu	Çıktı
<pre> &lt;html&gt; &lt;head&gt;   &lt;title&gt;başlık&lt;/title&gt; &lt;/head&gt; &lt;body&gt;   &lt;a href="http://websitesi.com"&gt;     Bağlantı&lt;/a&gt;   &lt;h1&gt;Başlık1&lt;/h1&gt; &lt;/body&gt; </pre>	



DOM, farklı betik (script) dilleri ile kullanılabilir, ancak HTML5 standartlarında Javascript kabul edilmiştir.

</html>

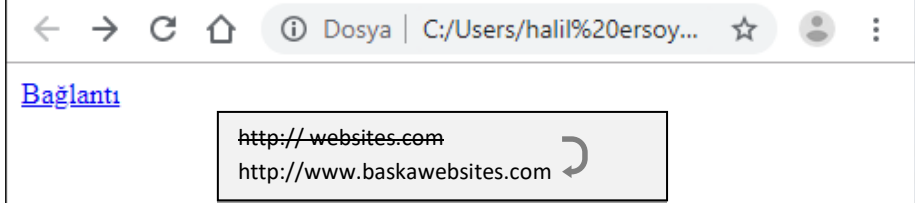
Javascript ile DOM kullanılarak aşağıdaki işlemler yapılabilir:

- Herhangi bir HTML elemanı sayfaya eklenebilir, var olanlar silinebilir.
- Herhangi bir HTML etiketinin herhangi bir özelliği değiştirilebilir, eklenebilir, silinebilir.
- Herhangi bir HTML etiketinin CSS stilleri değiştirilebilir.
- Herhangi bir HTML elemanının olayına yönelik kod yazılabilir.
- Herhangi bir HTML etiketine yeni olay eklenebilir.

DOM, Javascript veya farklı betik dilleri ile dinamik içerik oluşturulabilmesi için;

- HTML etiketlerini *nesne* olarak,
- Bu nesnelerin *özelliklerini*,
- Nesne ve özelliklere ulaşmak için gerekli *metotları* ve
- Nesnelerin *olaylarını* tanımlayan modeldir.

**Tablo 11.32.** Javascript İle Bir Bağlantının Adresinin Değiştirilmesi

Javascript Komutları
<pre>&lt;body&gt; &lt;a href="http://websitesi.com" id="link1" &gt;Bağlantı&lt;/a&gt; &lt;script&gt; document.getElementById("link1").href="http://www.baskawebsites.com"; &lt;/script&gt; &lt;/body&gt;</pre>
ÇIKTI


Yukarıdaki *<a>* etiketinin adresi Javascript ile değiştirilmiştir. Kodlarda yer alan;

- *document*, tüm sayfayı temsil eden *nesne*;
- *getElementById("link1")*, id özelliği "link1" olan alt eleman nesnesine ulaşmak için kullanılan *metot*;
- *href*, erişilen nesnenin bir *özelligi*;
- "http://www.baskawebsites.com" ise bu özelliğe verilen yeni *değerdir*.

## DOM ile Elemanlara Erişim

DOM'a göre oluşturulan elemanlara erişmek için 4 farklı metot vardır. Bunlar aşağıdaki tabloda gösterilmiştir.



Javascript komutu olan *getElementById* komutundaki büyük ve küçük harflere dikkat edilmeli ve aynen yukarıdaki gibi yazılmalıdır.

Tablo 11.33. Elemanlara Erişim Yolları

Javascript Komutu	Açıklama
document.getElementById("idBilgisi")	Sayfa içerisinde <i>id</i> özelliğine göre elemana erişme
document.getElementsByName("nameBilgisi")	Sayfa içerisinde <i>name</i> özelliğine göre elemanlara erişme
document.getElementsByTagName("EtiketAdı")	Sayfa içerisinde <i>etiket</i> in <i>adına</i> göre elemanlara erişme
document.getElementsByClassName("classAdı")	Sayfa içerisinde <i>CSS sınıfı</i> özelliğine göre elemanlara erişme



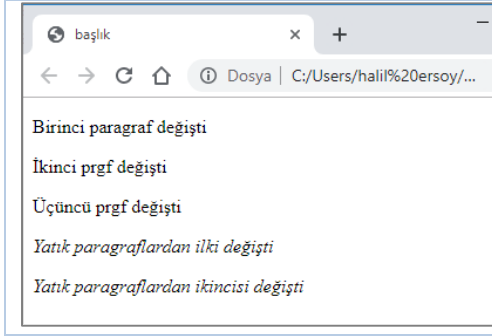
Elemanlara erişim için kullanılan metotlar, eğer kritere uyan eleman bulunamazsa *undefined* sonucunu geri döndürür.

Yukarıdaki dört erişim yolundan *getElementById()*, sayfadaki *id* bilgisi bilinen *bir elemana* erişir. Diğer metotlar ise, *name özelliği*, *sınıf özelliği* veya *etiket adı* verilen *birden çok elemana* erişmeyi sağlar. Bu son üç metot birden fazla eleman bulunması ihtimaline karşın, elemanları 0'dan başlayan *dizinin elemanları* olarak erişir. Aşağıdaki örnekte, *getElementsByName()* ve *getElementsByClassName()* metotlarından sonra dizinin 0'dan başlayan indis numarası bu nedenle kullanılmıştır.

Tablo 11.34. Elemanlara Erişim Ve Çıktı

Javascript Komutları
<pre> &lt;head&gt;   &lt;style&gt;     .yaziStili{       font-style: italic;     }   &lt;/style&gt; &lt;/head&gt; &lt;body&gt; &lt;p id="yazi1"&gt;Bu birinci paragraftır.&lt;/p&gt; &lt;p name="yazi2"&gt;Bu ikinci paragraftır.&lt;/p&gt; &lt;p name="yazi2"&gt;Bu üçüncü paragraftır.&lt;/p&gt; &lt;p class="yaziStili"&gt;Bu dördüncü paragraftır.&lt;/p&gt; &lt;p class="yaziStili"&gt;Bu beşinci paragraftır.&lt;/p&gt;  &lt;script&gt; document.getElementById("yazi1").innerText="Birinci paragraf değişti"; document.getElementsByName("yazi2")[0].innerText="İkinci prgf değişti"; document.getElementsByName("yazi2")[0].innerText="Üçüncü prgf değişti"; document.getElementsByClassName("yaziStili")[0].innerText="Yatık       paragraflardan ilki değişti"; document.getElementsByClassName("yaziStili")[1].innerText="Yatık       paragraflardan ikincisi değişti"; &lt;/script&gt; &lt;/body&gt; </pre>
<b>ÇIKTI</b>





## InnerHTML ve InnerText Özellikleri

Çoğu HTML etiketi kendi içeriden başka etiketleri ya da metni barındırır. Eğer bir HTML elemanının açılış ve kapanış etiketlerinin içine *yeni HTML etiketleri* eklemek isterseniz ya da var olan HTML etiketlerini değiştirmek isterseniz, *innerHTML* özelliğini kullanınız. Öte taraftan eğer bir HTML elemanının açılış ve kapanış etiketleri arasına *metin* eklemek ya da var olan metni değiştirmek isterseniz, *innerText* özelliğini kullanınız.

Tablo 11.35. Innerhtml Ve Inntertext Özelliği



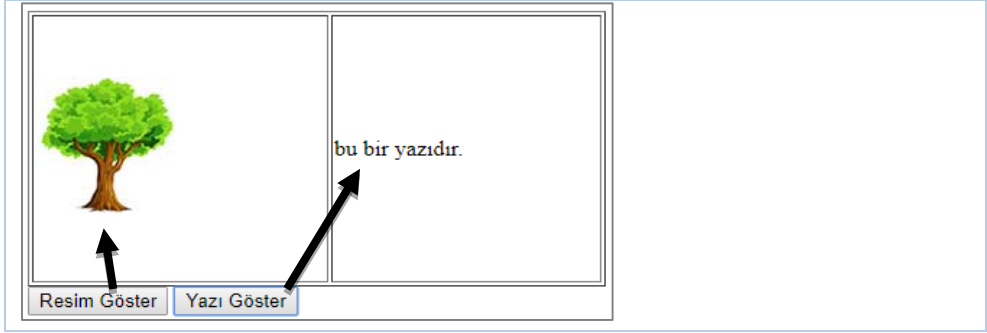
Bir eleman içerisine yeni eleman eklenmek istenirse *innerHTML* özelliği kullanılmalıdır.

### Javascript Komutları

```
<script>
function resimYukle(resimDosyasiAdi){
    var resimHTMLKodlari="<img src=\"\" + resimDosyasiAdi + \"\"/>";
    document.getElementById("hucre1").innerHTML=resimHTMLKodlari;
}
function yaziYukle(metin){
    document.getElementById("hucre2").innerText=metin;
}

</script>
</head>
<body>
    <table border="1" width="400" height="200">
        <tr>
            <td id="hucre1"> </td>
            <td id="hucre2"> </td>
        </tr>
    </table>
    <button onclick="resimYukle('resim1.jpg');">Resim Göster</button>
    <button onclick="yaziYukle('bu bir yazıdır.');">Yazı Göster</button>
</body>
```

### ÇIKTI



## DOM Elemanlarının Özellik ve Stillerini Değiştirmek

HTML etiketlerinin özellik (attribute) ve stil öğelerini (style) Javascript ile aşağıdaki gibi değiştirebilirsiniz. Değiştirebileceğiniz tüm CSS kurallarının listesine [https://www.w3schools.com/jsref/dom\\_obj\\_style.asp](https://www.w3schools.com/jsref/dom_obj_style.asp) adresinden ulaşabilirsiniz.

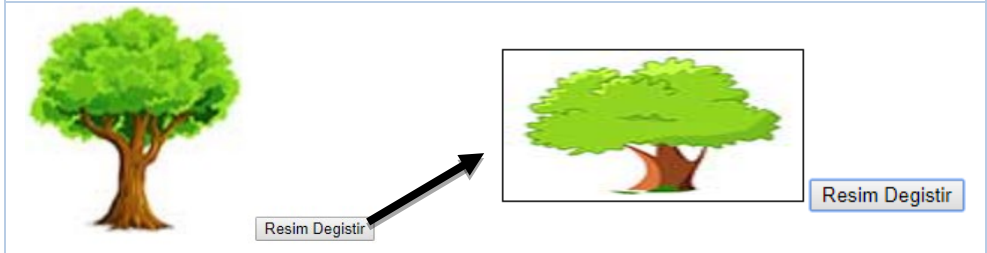
**Tablo 11.36.** HTML Etiketlerinin Özellik Ve Stillerinin Değiştirilmesi

### Javascript Komutları

```
<script>
function resimDegistir(resimDosyasiAdi){
document.getElementById("resimKutusu").src=resimDosyasiAdi;
document.getElementById("resimKutusu").height=100;
// Üstteki satır VE YA alttaki satır
document.getElementById("resimKutusu").setAttribute("height",100);
document.getElementById("resimKutusu").style.border="1px solid black";
}
</script>
</head>
<body>

<button onclick="resimDegistir('resim2.jpg');">Resim Degistir</button>
</body>
```

### ÇIKTI



Yukarıdaki gibi herhangi bir stil öğesini değiştirmenin ötesinde, elemanın varsa CSS sınıfını **className** özelliğine yeni sınıf atayarak aşağıdaki gibi değiştirebilirsiniz:



Örnek

```
•document.getElementById("resim").className="kucukResimSinifi";
```



Bir elemanın stil özelliğini değiştirmek için **setAttribute** metodu ya da o özelliğin adının açıkça yazılmış şekli kullanılabilir.



Javascript ile değiştirilecek olan stil sınıfları önceden tanımlanmış olmalıdır.

## WINDOW NESNESİ

**Window** nesnesi, kullanıcının **tarayıcısının** özelliklerinin toplandığı ve Javascript ile programlanabilir alt özellik ve metotlara sahip nesnedir. Bu bölümde Javascript ile sıklıkla yapılan bazı tarayıcı işlemlerini, **window** nesnesinin metot ve özelliklerini kodlayarak ele alacağız.

### Tarayıcı Ebatları

Kullanıcıların **tarayıcıları** farklı **ekranlarda** ve farklı en ve boy değerlerinde olabilir. Bu durum Javascript ile algılanabilir.

**Tablo 11.37.** Tarayıcının Ve Ekranın Ebatlarını Öğrenme

Javascript Komutu	Açıklama
window.innerWidth	Tarayıcının genişliğini piksel cinsinden verir.
window.innerHeight	Tarayıcının yüksekliğini piksel cinsinden verir.
window.screen.width	Ekranın genişliğini piksel cinsinden verir.
window.screen.height	Ekranın yüksekliğini piksel cinsinden verir.
window.screen.pixelDepth	Ekranın renk derinliğini (16,24 veya 32) verir.

Yukarıdaki özellikler farklı ekran büyüklüklerinde farklı görünümlü HTML sayfalarını göstermek için kullanılabilir.

**Tablo 11.38.** Tarayıcı Ve Ekran Nesnelerinin Özellikleri

Javascript Komutları
<pre> &lt;p id="bilgi"&gt;&lt;/p&gt; &lt;script&gt; var eleman=document.getElementById("bilgi"); bilgi.innerHTML="Tarayıcı Genişliği = " + window.innerWidth + "px&lt;br&gt;"; bilgi.innerHTML=bilgi.innerHTML+"Tarayıcı Yüksekliği=" + window.innerHeight+"px"; bilgi.innerHTML=bilgi.innerHTML+"&lt;hr&gt;"; bilgi.innerHTML=bilgi.innerHTML+"Ekran Genişliği=" + window.screen.width + "px&lt;br&gt;"; bilgi.innerHTML=bilgi.innerHTML+"Ekran Yüksekliği=" + window.screen.height+"px"; bilgi.innerHTML=bilgi.innerHTML+"&lt;hr&gt;"; bilgi.innerHTML=bilgi.innerHTML+"Ekran Çözünürlüğü= " + window.screen.pixelDepth+ "bit&lt;br&gt;"; bilgi.innerHTML=bilgi.innerHTML+"&lt;hr&gt;"; &lt;/script&gt; </pre>
<b>ÇIKTI</b>



Tarayıcının ebatları kullanıcı tarafından değiştirilirse, **body** elemanının **onresize** olayı tetiklenmiş olur.

Tarayıcı Genişliği = 500px  
 Tarayıcı Yüksekliği = 189px  
 Ekran Genişliği = 1920px  
 Ekran Yüksekliği = 1080px  
 Ekran Çözünürlüğü = 24bit

## Tarayıcı Adresi

*Window.location* nesnesi, ziyaret edilen *web sitesinin* ve *sayfasının* tam adresini, sayfanın adını, bağlantı protokolünü verir. İstenirse başka bir web sayfası adresi verilerek yeni sayfanın *yüklenmesi* sağlanır.

**Tablo 11.39.** Window.Location Nesnesinin Özellikleri Ve Çıktıları

Javascript Komutu	Açıklama
window.location.href	Aktif sayfanın tam adresini verir. Örnek çıktı: <a href="http://www.websites.com.tr/javascript/konu1.html">http://www.websites.com.tr/javascript/konu1.html</a>
window.location.hostname	Aktif sayfanın sunucu ismini verir. Örnek çıktı: <a href="http://www.websites.com.tr">http://www.websites.com.tr</a>
window.location.pathname	Aktif sayfanın yolunu ve dosya adını verir. Örnek çıktı: <a href="/javascript/konu1.html">/javascript/konu1.html</a>
window.location.protocol	Aktif sayfanın bağlantı protokolünü verir (http: ya da https: şeklinde). Örnek çıktı: <a href="http://">http:</a>
window.location.assign()	Yeni sayfanın yüklenmesini sağlar. Örnek kullanımı: window.location.assign("http://www.site2.com");



Eğer tarayıcının geçmişinde herhangi adres yoksa *back()* ve *forward()* komutları işlem yapmaz.

## Tarayıcı Geçmişi Nesnesi

Tarayıcılar ziyaret edilen tüm sayfaları “geçmiş” adı altında bir liste şeklinde saklar. Bu listeye Javascript ile ulaşabilirsiniz. Örneğin kullanıcınızın herhangi bir işlemi yanlış yaptığında bir önceki sayfaya gitmesini sağlayabilirsiniz. Bunun için *window.history* nesnesinin *back()* ve *forward()* metotları kullanılır.

**Tablo 11.40.** Tarayıcı Geçmişi Nesnesi

Javascript Komutları
<button onclick="window.history.back();">Geri git</button> <button onclick="window.history.forward();">İleri git</button>
ÇIKTI
<input type="button" value="Geri git"/> <input type="button" value="İleri git"/>

## Tarayıcı Adı ve Sürümünü Belirleme

Javascript'in en çok kullanıldığı yer, tarayıcı algılamaktır. Bunun nedeni, farklı tarayıcıların farklı kodları (Javascript, CSS ve hatta HTML) farklı biçimde desteklemesidir. Geliştiriciler bu probleme karşın, farklı tarayıcılar için alternatif kodlar yazarak yapılması gereken işlemlerin farklı tarayıcıya sahip kullanıcıları için




Farklı tarayıcıların algılanmasında farklı yollar vardır.

yapılmasını sağlamaya çalışırlar. Bunun için öncelikle tarayıcının adının ve sürüm numarasının algılanması gerekir.

*Window.navigator* nesnesi, tarayıcı hakkında bilgi veren bir nesnedir. Aşağıdaki örnekte tarayıcı adı ve sürümünü algılayan kodlar yer almaktadır.

**Tablo 11.41.** Tarayıcı Algılama

Javascript Komutları	
<pre>&lt;p id="bilgi"&gt;&lt;/p&gt; &lt;script&gt; var eleman=document.getElementById("bilgi"); bilgi.innerHTML="navigator.appName = "+window.navigator.appName + "&lt;hr&gt;"; bilgi.innerHTML+="navigator.appVersion= " +     window.navigator.appVersion+"&lt;hr&gt;"; bilgi.innerHTML+="navigator.platform="+window.navigator.platform+"&lt;hr&gt;"; var strVersion=window.navigator.appVersion; var sonuc=""; if(strVersion.indexOf("OPR") &gt; -1) sonuc="Opera"; else if(strVersion.indexOf("Edge") &gt; -1) sonuc="Edge"; else if(strVersion.indexOf("Chrome") &gt; -1) sonuc="Chrome"; else if(strVersion.indexOf("rv:11") &gt; -1) sonuc="Internet Explorer"; bilgi.innerHTML += "Sonuç = " + sonuc + "&lt;hr&gt;"; &lt;/script&gt;</pre>	
ÇIKTI	
<pre>navigator.appName = Netscape  navigator.appVersion = 5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.3809.100 Safari/537.36  navigator.platform= Win32  Sonuç = Chrome</pre>	 <p>Google Chrome Çıktısı</p>



**Bireysel Etkinlik**

- Yukarıdaki kodu yazıp, farklı tarayıcılarda deneyiniz.
- Sadece belirli bir tarayıcıda çalışacak kodlar ekleyiniz.

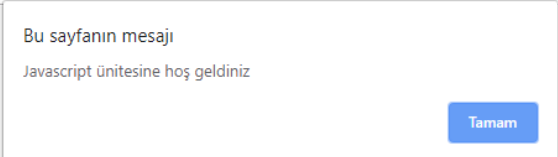
Yukarıdaki örneğe baktığımızda *navigator* nesnesinin *.appName*, *.appVersion* ve *.platform* özelliklerinin çıktılarının farklı tarayıcılarda da olsa benzer ya da aynı çıktıları verdiğini görüyoruz. Bu durum, tarayıcının tam adını ve sürümünü algılamak için ek adımlar atmamızı gerektirir. Örneğin tarayıcının versiyon bilgisinde farklı ifadeleri *arayarak* yukarıdaki örnekteki gibi kesin sonuç elde edebiliriz.

## Mesaj Kutuları

Kullanıcıya tarayıcının içerisinde açılarak kısa mesajlar vermeyi veya kullanıcından kısa bilgiler almayı sağlayan birkaç mesaj kutusu alternatifi vardır. Bunlar *window* nesnesinin;

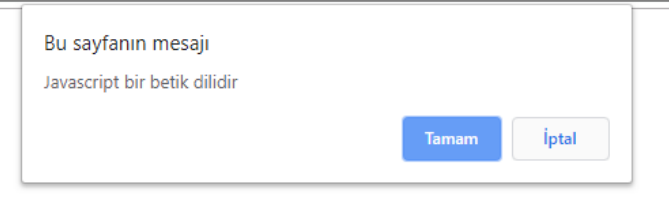
- *window.alert()* komutu
- *window.confirm()* komutu ve
- *window.prompt()* komutudur.

**Tablo 11.42.** Alert Metodu Örneği

JavaScript Komutları
<pre>&lt;script&gt; alert("Javascript ünitesine hoş geldiniz."); &lt;/script&gt;</pre>
ÇIKTI


*Alert()* metodu ile ister örnekteki gibi açık ifadeler, istenirse de farklı değişkenlerin değerleri ekrana yazdırılabilir. Mesaj kutusunda görülen “Tamam” butonunun üzerindeki yazı, renk ve büyüklüğü değiştirilemez. Farklı tarayıcılarda farklı görülebilir.

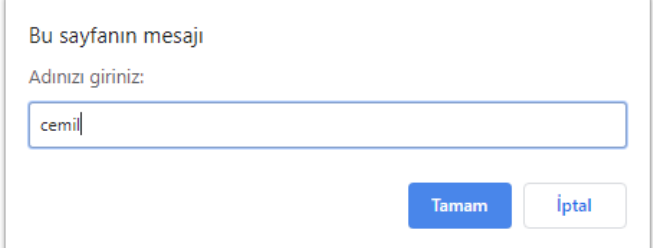
**Tablo 11.43.** Confirm Metodu Örneği

JavaScript Komutları
<pre>p id="bilgi"&gt;&lt;/p&gt; &lt;script&gt; if(confirm("Javascript bir betik dilidir"))     metin="Tamama basıldı"; else     metin="İptale basıldı"; document.getElementById("bilgi").innerText=metin; &lt;/script&gt;</pre>
ÇIKTI


Yukarıdaki örnekte görülen *confirm()* metodunda, ekrana yazılan mesajdan başka ayrıca “Tamam” ve “İptal” butonları da vardır. Kullanıcının bu iki butondan herhangi birine tıklamasıyla pencere kapanır. Eğer “Tamam” butonuna tıklanırsa *confirm* metodu *true*, “İptal” butonuna tıklanırsa *false* değerini döndürür.

Örnekteki gibi bir *if* ile bu değer kontrol ettirilip, farklı işlemler kullanıcı cevabına göre yapılabilir.

**Tablo 11.44.** Prompt Metodu Örneği

JavaScript Komutları
<pre> &lt;p id="bilgi"&gt;&lt;/p&gt; &lt;script&gt; var isim=prompt("Adınızı giriniz:", "adınız"); if(isim!= null)     metin="Merhaba sayın " + isim; else     metin="İptale basıldı"; document.getElementById("bilgi").innerText=metin; &lt;/script&gt; </pre>
ÇIKTI


*Prompt* komutunda ilk parametre olarak ekranda görünmesini istediğiniz ifadeyi, ardından ikinci parametre olarak da metin kutusunda varsayılan değer olarak yer almasını istediğimiz değeri yazılır. Kullanıcı bilgi girip “Tamam” butonuna tıklarsa, bu metod *girilen değeri*; eğer “İptal” butonuna tıklarsa *null* değerini döndürür.

## Açılır Pencere Açma

Birçok web sitesinde var olan tarayıcının içerisinde yeni bir sayfa ya da sekme açmak için *window.open()* metodu kullanılır. Bu metod toplam dört parametre alabilir.

<pre> window.open("http://yenisite.com", "_blank", "width=200", true); </pre>
---

**Şekil 11.5.** Window.Open Metodu

Yukarıdaki örnek gösterimde, ilk parametre açılmasını istediğimiz pencerenin içerisinden görüntülenecek olan web sayfasının adresidir.

İkinci parametre ise açılacak olan pencerenin, var olan pencere olmayıp yeni bir pencere olacağıdır. Bu parametrenin alternatifleri şunlardır:

- *\_blank* → yeni tarayıcı penceresi veya yeni tarayıcı sekmesi
- *\_parent* → varsa ana frame içerişi
- *\_self* → şu anda içinde bulunulan pencere ya da frame
- *\_top* → Varsa tüm frameset’lerin üstündeki ana pencere



Açılır pencereler (*pop-up windows*) bazı tarayıcılar tarafından engellenebilir.



*true* ve *false* ifadeleri tek başına bir değerdir, metin türü değildir!

- pencereAdi → Varsa adı pencereAdi olan frame içerişi

Üçüncü parametre, açılacak pencerenin genişlik, yükseklik, ekrandaki konumu gibi özellikleri parametrik olarak belirlemek için kullanılır. Bununla ilgili detaylı ayarlara [https://www.w3schools.com/jsref/met\\_win\\_open.asp](https://www.w3schools.com/jsref/met_win_open.asp) sayfasından ulaşabilirsiniz.

Dördüncü parametre ise true ya da false olabilir. **True** olduğunda açılan pencerenin tarayıcı geçmişine eklendikten sonra var olan sayfanın geçmişten çıkarılmasını sağlar. **False** değeri ise tarayıcı geçmişine hem kendisini ekler hem de var olan sayfanın kalmasını sağlar.





## Özet

- Bu ünite de Javascript dili ile yapılabilecek ileri düzey işlemler için gerekli konular ele alınmıştır. Bunlar metin (string) işlemler, döngüler, fonksiyonlar, olaylar, DOM hiyerarşisi ve Window nesnesinin özellikleridir.
- **METİN İŞLEMLERİ**
  - Metin tür değerler (string) karakter dizileri şeklinde hafızada tutulur. Metinlerin her bir karakterine o karakterin indis numarası ile ulaşılabilir. İndis numarası 0'dan başlar. Metnin karakter sayısı ise length özelliği ile elde edilir. Bunun dışında metin üzerinde yapılabilecek işlemler hazır fonksiyonlar ile yapılır.
- **DÖNGÜLER**
  - Döngüler aynı işlemin birden fazla yapılması gerektiğinde komutların blok halinde yazılıp döngü komutu ile kaç kez tekrarlanmasının belirlendiği yapılardır. For, while ve do-while olmak üzere 3 farklı döngü komutu ile işlemler yapılabilir. Dizilerin elemanlarına ulaşmak için genellikler for döngüsü kullanılır.
- **FONKSİYONLAR**
  - Fonksiyonlar, birbiri ile ilgili olduğu düşünülen komutların gruplanarak tek bir komut benzeri yapılar içerisinde saklanmasını sağlayan yapılardır. Fonksiyonlar öncelikle tanımlanır. Tanımlamada fonksiyon ismi ve istenirse parametreleri yazılır. Küme parantezleri içerisinde ise çalıştırılmak istenen komutlar yazılır. Fonksiyonlar bu yolla tanımlandıkları yerde ve zamanda değil, daha sonra çağırıldıkları zaman çalışır. Çağırılırken fonksiyon adı ve varsa parametreleri (argümanı) yazılır.
  - Eğer fonksiyon işlem sonucunda bir değer üretip bunu çağırıldığı noktaya geri döndürmek üzere tanımlanırsa, fonksiyon bloğu içinde bu değer return kelimesi ile yazılmalıdır.
  - Fonksiyonlar ya çağırıldıkları zaman, ya bir olay gerçekleştiğinde, ya da özel biçimde tanımlanarak tanımlandıkları anda çalışabilirler.
  - Parametreler fonksiyon içinde değişirse bu çağırıldıkları yere yansımaz. Dizi veya nesne türündeki parametreler ise buna istisna olarak fonksiyon içinde değişirlerse çağırıldıkları yerde de değişir.
- **OLAYLAR**
  - Tüm HTML etiketlerinin kullanıcının bir takım hareketlerini veya sayfanın yüklenme zamanı algılayıp cevap verebileceği olayları (events) vardır. Bu olaylar farklı etiketler için farklı sayıda ve isimdedir. Bu olaylara eğer bir Javascript komutu ya da fonksiyonu yazılırsa, olay gerçekleştiğinde bu komut ya da fonksiyon çalıştırılır.
  - En çok kullanılan olaylar onclick, onmouseover, onload olaylarıdır.
- **DOM MODELİ**
  - HTML sayfaları yüklendiğinde, tarayıcılar sayfadaki tüm elemanları (HTML etiketlerini) *document* ismindeki bir nesnenin alt nesneleri olarak belirler. Bu *hiyerarşi* sayesinde HTML sayfasındaki tüm etiketlerin içeriği, özellikleri, CSS stilleri ve alt etiketleri değiştirilebilir. Bu hiyerarşik yapıya *Document Object Model* (DOM) denir.



## Özet(devamı)

### •DOM ile Elemanlara Erişim

•Tarayıcı tarafından yaratılan ilk nesne **document** nesnesidir. HTML sayfasındaki diğer etiketler (elemanlar) bu nesnenin alt elemanları olarak hafızada tutulur. Bu hiyerarşide herhangi bir etikete erişmek istenirse öncelikle o etiketin "id" özelliğine bir isim verilmeli, sonrasında **document.getElementById()** metodu ile o elemana erişim sağlanabilir.

•getElementById() metodu ile erişilen elemanın kendine has özellik ve metotları vardır. Bu metot ve özellikler Javascript ile değiştirilebilir.

### •InnerHTML ve InnerText Özellikleri

Bir HTML elemanının açılış ve kapanış etiketlerinin *arasına yeni HTML etiketleri* eklemek isterseniz ya da var olan HTML etiketlerini değiştirmek isterseniz, *innerHTML* özelliğini kullanınız. Öte taraftan eğer bir HTML elemanının açılış ve kapanış etiketleri *arasına metin* eklemek ya da var olan metni değiştirmek isterseniz, *innerText* özelliğini kullanınız.

### •DOM Elemanlarının Özellik ve Stillerini Değiştirmek

HTML etiketlerinin özellik (attribute) ve stil öğelerini (style) Javascript ile aşağıdaki gibi değiştirebilirsiniz.

```
document.getElementById("resim").style.border="1px solid black";
document.getElementById("resim").className="kucukResimSinifi";
```

### •WINDOW NESNESİ

•Tarayıcı ve var olan web sayfası hakkında bilgi almak, kullanıcıya tarayıcı içerisinde mesajlar vermek ve kimi zamanda kullanıcıdan kısa bilgiler almak için window nesnesinin alt nesneleri ve metotları kullanılır.

•Window.open() metodu ile alırlar pencereler açılabilir.

## DEĞERLENDİRME SORULARI

1. Aşağıdakilerden hangisi metnin içindeki harfin küçük hâlini geri döndüren fonksiyondur?
  - a) toSmallLetter()
  - b) toSmallString()
  - c) toLowerString()
  - d) toLowerCase()
  - e) toCaseMin()
2. Aşağıdaki iki satır sonucunda, b değişkeninin değeri ne olur?  
a="merhaba";  
b=a.indexOf("a");
  - a) true
  - b) 7
  - c) 4
  - d) 2
  - e) 6
3. Aşağıdaki for döngülerinden hangisinde yazım hatası yoktur?
  - a) for(s < 20; s++) { a += s; }
  - b) for(s = 1 , s < 20 , s++) { a += s; }
  - c) for(s = 1) ; (s < 20) ; (s++); { a += s; }
  - d) for(1<s<20; s++) { a += s; }
  - e) for(s = 1 ; s < 20; s++) { a += s; }
4. Aşağıdaki while döngülerinden hangisi sonsuz döngüdür?
  - a) t=0; s=1; while(s>10){ t += s;}
  - b) t=0; s=1; while(s<10){ t += s;}
  - c) t=0; s=2; while(s<5){ t += s; s++;}
  - d) t=0; s=20; while(s>10){ t += s; s--;}
  - e) t=0; s=20; while(s>5){ t += s; s=s-2;}
5. Aşağıdaki dizi tanımlamasından sonra, dizinin uzunluğunu ekrana yazan komut hangisidir?  
var dizi=["TL", "USD", "EU"];
  - a) document.write(dizi.items.count);
  - b) document.write(dizi.length);
  - c) document.write(dizi[].values);
  - d) document.write(count(dizi));
  - e) document.write(dizi.index);

6. Aşağıdakilerden hangisi indexOf komutunu doğru biçimde tanımlar?
  - a) Dizi indislerini değiştirir.
  - b) Diziyi sıralar.
  - c) Diziyi ters çevirir.
  - d) Dizi içinde arama yapar.
  - e) Dizi eleman sayısını verir.
  
7. Kendisine gönderilen 2 sayının farkını bulup geri döndüren fonksiyon hangi seçenekte doğru olarak tanımlanmıştır?
  - a) `function f(a,b){return a-b;}`
  - b) `function f(a,b){f=a-b;}`
  - c) `function f(a,b){a-b; return;}`
  - d) `function f(){ a-b;}`
  - e) `function f(){return a-b;}`
  
8. Bir web sayfası yüklendiğinde çalışan olay aşağıdakilerden hangisidir?
  - a) `onLoad`
  - b) `onClick`
  - c) `onMouseOver`
  - d) `onVisit`
  - e) `onFocus`
  
9. Aşağıdakilerden hangisi CSS sınıfı "uyari" olan elemanların tümünü getirir?
  - a) `document.getElementById("uyari")`
  - b) `document.getElementsByTagName("uyari")`
  - c) `document.getElementsByClassName("uyari")`
  - d) `document.getElementsByName("uyari")`
  - e) `document.getElementByIdStyleName("uyari")`
  
10. Aşağıdakilerden hangisi ziyaret edilen web sayfasının tam adresini verir?
  - a) `window.location.adress`
  - b) `window.location`
  - c) `window.location.port`
  - d) `window.location.hostname`
  - e) `window.location.href`

**Cevap Anahtarı**

1.d, 2.c, 3.e, 4.b, 5.b, 6.d, 7.a, 8.a, 9.c, 10.e

## **YARARLANILAN KAYNAKLAR**

Javascript Dersleri. 7 Temmuz 2019 tarihinde <http://javascript.sitesi.web.tr/> adresinden erişildi.

Javascript Tutorial. 23 Temmuz 2019 tarihinde <https://www.w3schools.com/js/> adresinden erişildi.

Javascript HTML DOM Nesneleri. 29 Temmuz 2019 tarihinde <https://www.yazilimbilisim.net/javascript/javascript-html-dom/> adresinden erişildi