Web Projesi Yönetimi

Web Projesi Yönetim Aşamaları

Web Projesi Yönetim Aşamaları

- Web Projesinin Başlangıç Aşaması
- Web Projesinin Tanımlanması
- Web Projesinin Yapısı
- Tasarım Aşaması ve Prototip Web Sitesi Üretimi
- Web Sitesinin Üretim ve Test Aşaması
- Web Sitesinin Sunumu
- Değerlendirme ve Sürdürülebilirliktir

1. Web Projesinin Başlangıç Aşaması

- Endüstrinin Analizi
- Sektör Rekabet Analizi
- Endüstri ve Pazar Araştırması
- Hedef Kullanıcıların Analizi
- Kullanıcı Profillerinin Çıkarılması
- Ön Testler ve Araştırmalar
- Tüketici Pazar Araştırması

2. Web Projesinin Tanımlanması

- Genel Amaç ve Hedeflerin Tanımlanması
- Öncelikli Hedeflerin Belirlenmesi
- Amaçların Belirlenmesi
- Müşteri Anketlerinin Dağıtılması
- Proje Planının Yaratılması
- Proje Takımının Oluşturulması
- Proje Programının Belirlenmesi
- Bütçe Planı ve Ayarlamalar
- Nihai Site Yapısına Karar Verilmesi
- İşlevsellik Test Planı
- Kalite Güvenliği (QA) Planı

3. Web Projesinin Yapısı

- Web Projesinin Yapısı
- Kullanıcılarla Test Etme
- Web Sitesi İçeriği (Kapsamı)
- İçeriğe Ait Ana Hatların Belirlenmesi
- İçerik Teslim Planının Yapılması
- Web Sitesi Görünümü
- Site Haritasının Oluşturulması
- İsimlendirme Konvansiyonları
- Fonksiyonel Alan Adresleri
- Ekran Görünümü
- Ekran Şemalarının Yaratılması
- Yönlendirmenin Belirlenmesi
- İçerik ve İşlevsellik Çerçevesi
- Kullanıcı Görünümü
- Kullanıcı Senaryolarının Yaratılması
- Kullanıcı Yollarının Belirlenmesi
- Kullanıcılarla Test Etme

4. Tasarım Aşaması ve Prototip Web Sitesi Üretimi

- Görsel Tasarım Yaklaşımı
- Kavramsal Beyin Fırtınası
- Grafik Şablonların Yaratılması
- Tasarım Rehberinin Hazırlanması
- HTML Prototipinin Yaratılması
- HTML Prototip Geliştirme
- Test Etme ve Doğrulama
- İşlevsellik Testi

5. Web Sitesinin Üretimi ve Test Aşaması

- Web Sitesi Üretimi Öncesi Planlama
- Faaliyet Alanı ve Bütçenin Yeniden Değerlendirilmesi
- Teknik İhtiyaçların Belirlenmesi
- Versiyon Kontrol Yöntemleri
- Dosya Yapılarının Düzenlenmesi
- Site Bakım Planının Hazırlanması
- Web Sitesinin Yapılandırılması
- Grafiklerin Düzenlenmesi
- HTML Şablonlarının Oluşturulması
- Yazılı Metinlerin Adreslendirilmesi
- Sayfa Düzenlemeleri
- İçerik ve Üretimin Sonlandırılması
- Test Etme
- Kalite Güvenlik (QA) Test Planı
- Kalite Güvenlik Test Yönetimi
- Güvenlik Yönetimi (Virüs takibi ve önlenmesi)

6. Web Sitesinin Sunumu

- Son Değerlendirme ve Kalite Güvenliği
- Sitenin İnternette Sunumu
- Site Güvenliğinin Kontrol Edilmesi
- HTML Biçim Kılavuzu Hazırlanması
- Bakım Uygulamaları
- Dokümantasyon Takibi
- Site Tutundurma Faaliyetleri
- Tutundurma Planı
- Arama Motorlarına Katılımın Sağlanması

7. Değerlendirme ve Sürdürülebilirlik

- Başarı Aşamalarının Gözden Geçirilmesi
- Kullanıcı Geri Beslemesi Sağlanması
- Kullanıcıların İzlenmesi
- Ölçülebilir Sonuçların Elde Edilmesi
- Proje Sonrasının Değerlendirilmesi
- Proje Aşamalarının Yeniden Kontrolü

Javascript

Javascript

- Netscape firmasının 1995 yılında Netscape Navigator 2.0 ile birlikte geliştirdiği C dilinde yazılmış bir script dilidir.
- Pascal, C++, Java, C#, Basic, Visual C, Visual Basic, Visual C#, Delphi vb gibi bir programlama dili değildir.
- Kodlama sonunda com veya exe gibi derlenmiş dosya oluşturulmaz.
- Script dilleri, web sayfalarının dinamik ve etkileşimli olarak hazırlanmasını sağlar.
- Javascript ile HTML kaynak kodları değiştirilebilir. Böylece dinamik web sayfaları hazırlamak mümkün olur.
- Kullanıcı taraflı (client) olduğu için javascript kodları çok hızlıdır.
- Javascript ile hazırlanmış web sayfaları sadece javascript desteği veren web tarayıcılarında izlenebilir.
- Jquery kodları javascript ile yazılmaktadır. Bu nedenle jquery kullanabilmek için javascript kodları hakkında bilgi sahibi olunmalıdır.

Javascript'in Özellikleri

- Kullanıcı taraflı çalışan bir script dildir.
- Javascript kodları ile bilgisayardan herhangi bir dosya okunamaz, dosya üzerine yazılamaz veya dosya oluşturulamaz. (Çerezler (cookies) hariç)
- HTML sayfalarının bir parçasıdır. Tek başına bir anlam ifade etmez veya çalışmaz.
- Yorumlanabilmeleri için tarayıcıya ihtiyaç duyulur.
- JS ile karşı sunucuya dosya gönderilemez.
- JS kodları <script>...</script> etiketleri arasında yazılır.
- Kodların çoğalması durumunda harici dosya (.js uzantılı) kullanılır ve HTML sayfasından çağrılır.
- JS kodları herhangi bir metin editöründe yazılabilir.
- Her JS kodu farklı tarayıcılarda aynı sonucu vermeyebilir. Bazı kullanımlar bazı tarayıcılarda çalışmayabilir.

Sayfa İçerisinde Javascript Kodlarının Kullanımı

- Genellikle <head>...</head> etiketleri içerisinde <script>...</script> etiketleri arasında yazılır.
- HTML sayfasının diğer bölümlerinde de JS kodları yazılabilir.

```
<html>
<head>
<meta ..../>
<title>...</title>
<script type="text/javascript" language="javascript">
//JS kodları
</script>
<style>...</style>
</head>
<body>
</body>
</html>
```

Harici Javascript Dosyasını Çağırma

- Harici javascript kodlarının yazıldığı dosya «.js» uzantısı ile kaydedilir.
- JS dosyası (ornek.js)

```
// JavaScript Document
alert("Message...");
```

- HTML sayfası (ornek.html)

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>JS Örnek</title>
<script type="text/javascript" src="ornek.js" language="javascript"></script>
</head>
<body>
</body>
</html>
```

Yazım Kuralları

- Bütün ifadeler harf veya "_" ile başlar. İfadelerin ikinci veya sonraki karakterleri rakam olabilir.
- Değişken isimlerinin uzunluğu 255 karakter olabilir.
- Büyük-küçük harf ayrımı vardır. "XYZ" ile "xyz" aynı ifade değildir.
- Türkçe karakterler tercih edilmez. (çğıöşüÇĞİÖŞÜ)
- Anahtar kelimeler (if, for gibi) diğer tanımlamalarda kullanılamaz.
- Satır sonlarında «;» karakteri kullanılır.
- Kod blokları «{...}» arasında yazılır. Açılan blok işaretinin karşılığı mutlaka olmalıdır. Kod blokları kapatılmalıdır.
- «'» ile «"» arasında fark yoktur. İç içe kullanımlarda tercih edilebilir. Özellikle HTML veya CSS özelliklerini JS içerisinde düzenlerken kullanılabilir.
- Metinlerde özel karakterlerin yer alması gerekiyorsa, öncesinde «\» işareti eklenir.
  - "Javascript \"SCRIPT\" bir dildir." → Javascript "SCRIPT" bir dildir.

Yazım Kuralları

- **Açıklama satırları**, "//" veya "/* ... */" işaretleri ile tanımlanır.
  - Her satırın başında "//" işareti konulabilir
  - "/* ... */" işaretleri arasında tüm açıklamalar tanımlanabilir.
  - Aynı zamanda bu işaretler, bazı JS kodlarının kapatılması için kullanılır.
- **Kod satırlarını bölmek için**, "\" işareti kullanılır.
  - Bölme işlemi sadece metinlerde kullanılır. JS kodları bölünemez.
- **Eski tarayıcılarda kod gizlemek için**, "<!-- .... -->" işaretleri kullanılır.
  - Tarayıcıların JS kodlarını açıklama satırları gibi algılamasını sağlar.

Değişkenler

- Javascript'te bilgi saklama ve bilgiyi kullanma amaçlı kullanılır.
- Değişkenler "var" ifadesi ile tanımlanır.
- Değişken veri tipi, değer atandıktan sonra JS tarafından belirlenir.
- Örnek tanımlamalar:

```
var x;
var y=5;
var a="ali";
var b=true;
var c='veli';
```

- Veri Türleri:
  - **String :** Tek veya çift tırnak arasında yazılan ifadeler metin olarak değerlendirilir.
  - **Number:** Tamsayılar, ondalık sayılar vs. Ondalık basamaklar "." işareti ile ayrılır.
  - **Boolean:** Mantıksal "true/false" değerler için kullanılır.
  - **Null:** Boş değer atamak için kullanılır. Genellikle herhangi bir değişkenin içeriğini boşaltmak için kullanılabilir.

Genel ve Yerel Değişkenler, Sabitler

- **Genel değişkenler**, tüm script içerisinde kullanılabilir.
- **Yerel değişkenler**, sadece tanımlandığı fonksiyon veya kod bloğu içerisinde kullanılabilir.
- Örnek:

```
var x=12.5;
function islem(){
 var y=4;
 alert(x*y);
}

islem();
alert(y); //çalışmaz
```

- **Sabitler:** Değeri değiştirilemeyen ifadelerdir. "const" ile tanımlanır.
  - const pi=3.14;

Operatörler-1

- Sayısal veya mantıksal bir veya birden fazla değer arasında atama, karşılaştırma ve hesaplama gibi işlemlerin yapılmasını sağlar.
- Operatörlerin tamamı ya bir sonuç döndürür ya da kullanıldığı değeri etkiler.
- **Aritmetik:** Toplama (+), Çıkarma (-), Çarpma (*), Bölme (/), Kalan (%)
- **Atama:** Atama (=), Topla Ata(+=), Çıkar ata(-=), Çarp ata(*=), Böl ata(/=)
- **Artırma ve Azaltma:** Artırma (++), Azaltma(--), a++:a=a+1, a--:a=a-1
- **Karşılaştırma:** Eşitse(==), Kesin eşitse (===), Eşit değilse (!=), Kesin eşit değilse (!==), Büyükse (>), Küçükse (<), Büyük veya eşitse (>=), Küçük veya eşitse (<=)
  - Kesin eşit ve kesin eşit değil operatörlerinde değerler ile birlikte veri türleri dikkate alınır.
- **Mantıksal:** Ve (&&), Veya (||), Değil (!)
- **String birleştirme:** Sadece metinlerde iki veya daha fazla metin "+" işareti ile birleştirilir.
- **Koşullu atama:** Koşullu değer atama işlemlerinde "?:" kullanılır.

```
var değişken= x==y? "Eşitse": "Eşit değilse";
```

Operatörler-2

- **IN:** Belirtilen değer veya indis, belirtilen dizi içerisinde varsa "true", yoksa "false" değerini döndürür. Ayrıca belirtilen özellik belirtilen nesne içerisinde varsa "true", yoksa "false" döndürür.
  var dizi=new Array(4, 6, 3, 8); 2 in dizi --> true,
- **INSTANCEOF:** Nesne belirtilen türde ise "true" değeri döndürür.
  var dizi=new Array("Ali", "Veli"); dizi instanceof Array --> true
- **NEW:** Kullanıcı tanımlı bir nesne veya ön tanımlı nesne tiplerinden birinin (Array, Boolean, Date, Function, Image, Number, Object, Option, RegExp, String) örneğini tanımlamak için kullanılır.
  var dizi=new Array(1,2,3);
- **THIS:** Geçerli nesneyi ifade eder. Genel olarak metotlar içerisinde kullanılır.
  function fonk(nesne){....}
  <input type="text" name="txt" onChange="fonk(this);"/>
- **TYPEOF:** Değişkenin veri türünü döndürür.
  var x=5; typeof(x) --> number

Karar Yapıları (if, Switch)

- Kodlar, JS tarafından işlenirken, akış yönünün belirli şartlara göre değiştirilmesini sağlayan ifadelerdir.
- **IF:** Belirtilen koşul/koşullar doğru ise kod bloğu çalıştırılır.

```
var x=5; if(x==5) document.write("Eşit");
```

- **IF...ELSE:** Belirtilen kod bloğu doğru ise IF'ten sonra yazılan kodlar, değilse ELSE'den sonra yazılan kodlar çalıştırılır.

```
var x=5; if(x==5) document.write("Eşit"); else document.write("Eşit değil");
```

- **SWITCH:** Koşullar belirli ise kullanılan karar yapısıdır.

```
var x=1;
switch (x)
{
case 1: document.write("bir"); break;
case 2: document.write("iki"); break;
case 3: document.write("üç"); break;
default: document.write("yanlış"); break;
}
```

Döngüler (For, While, Do-While, For..in)

- **FOR:** Herhangi bir kod bloğunu birden fazla çalıştırmak için kullanılan döngü yapısıdır. Belirtilen koşul doğru olduğu sürece döngü devam eder.

```
for(var i=1;i<=10;i++){document.write(i);}
```

- **WHILE:** Herhangi bir kod bloğunu birden fazla çalıştırmak için kullanılan döngü yapısıdır. For'dan farklı olarak başlangıç değeri ve artış değeri belirtilmez. Koşul doğru olduğu sürece döngü devam eder.

```
var i=0; while(i<10) {document.write(++i);}
```

- **DO...WHILE:** While döngüsüne benzer. While döngüsünden farklı olarak, kod bloğu en az bir defa çalıştırılır. Kod bloğu çalıştırıldıktan sonra koşulun doğru olup olmadığı kontrol edilir.

```
var i=0; do {document.write(++i);} while(i<10);
```

- **FOR...IN:** Bir nesnenin özelliklerine ya da bir dizinin öğelerine erişmek için kullanılır.

```
var dizi=new Array("Ali", "Veli");
for(indis in dizi) document.write(dizi[indis]);
var nesne={ad:"Ali", soyad:"Koş"};
for(ozellik in nesne) document.write(nesne[ozellik]);
```

İç içe Döngüler

```
tablo="<table border=1 cellspacing=0 cellpadding=1>";
for(i=1; i<=5; i++)
{
      tablo+="<tr>";
      for(j=1; j<=5; j++)
            tablo+="<td>"+i*j+"</td>";
      tablo+="</tr>";
}
tablo+="</table>";
document.writeln(tablo);
```

1	2	3	4	5
2	4	6	8	10
3	6	9	12	15
4	8	12	16	20
5	10	15	20	25

```
for(var i=1, j=1; i<=5 && j<=5; i++, j++)
      document.writeln(i*j);
```

1 4 9 16 25

Break, Continue

Break

- Herhangi bir döngüyü belirli bir yerde sonlandırmak için kullanılır.

```
for(i=1; i<=5; i++)
{
      document.writeln(i);
      if(i==3) break;
}
```

Continue

- Döngü içerisinde kullanıldığında döngü bir sonraki adımdan işlemine devam eder.

```
for(i=1; i<=5; i++)
{
      if(i==3) continue;
      document.writeln(i);
}
```

Try-Catch-Finally (Hata Yönetimi)

- Kodlarda sıfıra bölünme, sonsuz döngüye girme, tanımlanmamış değişkenlerle çalışma vs hatalara sebep olur.
- Hata oluşabilecek kodlar "TRY" bloğuna, hata oluşursa yapılacak işlemler "CATCH" bloğuna yazılır.
- Hata olsa da, olmasa da çalıştırılması gereken kodlar "FINALLY" bloğuna yazılır.

```
var x=5, y=0;
try
{
    document.write("Sonuç:"+x/y);
}
catch(error)
{
    document.write(error.message);
}
finally
{
    document.write("sürekli çalışan kodlar");
}
```

Try-Throw (Hata Yönetimi)

- Try bloğunda, özel hata blokları oluşturulabilir.
- Throw ile hata mesajları özelleştirilebilir.

```
var x=5, y=0;
try{
      if(y==0) throw "sıfıra bölme hatası";
      document.write("Sonuç:"+x/y);
}catch(err){
      document.write(err);
}
```

Diziler

- Aynı tipte birden fazla veriyi saklamak için kullanılan veri yapısıdır.
- Dizideki her değerin bir indeksi vardır. bu indekslerle değerlere ulaşılabilir.
- Dizi indeksleri 0'dan başlar.
- Dizi indeks numaraları dizi adından sonra köşeli parantezler içerisinde belirtilir. Dizi[indeks]

```
var dizi=new Array(); dizi[0]="Ali"; dizi[1]="Veli";
var dizi=new Array("Ali", "Veli");
var dizi=["Ali","Veli"];

var dizi=new Array(2);
dizi[0]=new Array("Ali","Veli","Selami"); dizi[1]=new Array("Ayşe","Oya","İpek");

var dizi=new Array(new Array("Ali","Veli","Selami"), newArray("Ayşe","Oya","İpek"));

var dizi=[ ["Ali","Veli","Selami"], ["Ayşe","Oya","İpek"] ];

for(i=0;i<2;i++)
      for(j=0;j<3;j++)
            document.write(dizi[i][j]);
```

Dizi Metotları

- **LENGTH:** Dizinin eleman sayısını verir.

  ```
  Dizi.length
  ```

- **CONCAT:** Dizileri birleştirir.

  ```
  Dizi1.concat(dizi2)
  ```

- **JOIN:** Dizi içindeki tüm değerleri belirtilen özel karakter ile birleştirir ve string olarak geri döndürür. Özel karakter belirtilmezse "," işareti ile birleştirilir.

  ```
  Dizi.join("-");
  ```

- **SHIFT:** Dizinin ilk elemanını verir ve diziden siler.

  ```
  Dizi.shift();
  ```

- **POP:** Dizinin son elemanını verir ve diziden siler.

  ```
  Dizi.pop();
  ```

- **PUSH:** Dizinin sonuna bir veya daha fazla değer ekler.

  ```
  Dizi.push("1","2",...);
  ```

- **UNSHIFT:** Dizinin başına bir veya daha fazla değer ekler.

  ```
  Dizi.unshift("1","2",...);
  ```

Dizi Metotları

- **REVERSE:** Diziyi ters çevirir.

```
Dizi.reverse();
```

- **SORT:** Diziyi alfabetik olarak sıralar.
  - ASCII kodlarına göre sıralama yapar.
  - Sayılar alfabetik olarak sıralanır.
  - Sayıların gerçek değerlerine göre sıralanması için fonksiyon yazılmalıdır.

```
var dizi=new Array(245, 56, 25, 89, 123, 763);
dizi=dizi.sort();
dizi=dizi.sort(function(a,b){return b-a}); //sayıları sıralar.
for(i=0; i<dizi.length;i++) document.write(dizi[i]+"<br>");
```

- **SLICE:** Dizinin bir kısmını almak için kullanılır.
  - Slice(başlangıç indeksi, bitiş indeksi)
  - Bitiş indeksi belirtilmezse, dizinin sonuna kadar seçilir.
  - Bitiş indeksi negatif değer olursa, dizinin sonundan belirtilen kadar değer alınmaz.
  - Bitiş indeksinde bulunan değer alınmaz.

```
Dizi.slice(2); //2.indeksten başla, sonuna kadar al
Dizi.slice(2,5); //2.indeksten başla, 5.indeks hariç diğerlerini al
Dizi.slice(2,-1); //2.indeksten başla, sonuna kadar al, sondan 1 değeri alma
```

Dizi Metotları

- **SPLICE:** Dizi elemanlarından bir kısmını sildikten sonra, belirtilen değerleri diziye ekler.
  - Dizi.splice(başlangıç indeks, silinecek eleman sayısı, eklenecek değerler)

```
Dizi.splice(1,4,"x","y","z");
```

- **EVERY:** Dizi elemanlarının belirli bir niteliğe sahip olup olmadığını kontrol eder. true/false değeri döndürür.
- **FILTER:** Dizi elemanlarını fonksiyon aracılığıyla filtreler. Fonksiyonun true/false değerlerini döndürmesi gerekir.

```
function kontrol(deger){if(!isNaN(deger)) return true; else return false;}
var dizi=new Array(245, 56, 25, 89, 123, 763, "X");
document.write(dizi.every(kontrol)); --> false
var degerler=dizi.filter(kontrol);
document.write(degerler.join()); --> 245, 56, 25, 89, 123, 763
```

- **FOREACH:** Dizi elemanları üzerinde belirtilen fonksiyon ile işlem yapar.

```
var dizi=new Array(245, 56, 25, 89, 123, 763,"X");
function kontrol(deger){if(deger%2==0) document.write("çift"); else document.write("tek");}
dizi.forEach(kontrol);
```

Dizi Metotları

- **INDEXOF:** Belirtilen değerin, dizi içindeki konumunu verir.

```
Dizi.indexOf("X");
```

- **LASTINDEXOF:** Belirtilen değerin, dizi içindeki en son konumunu verir.

```
Dizi.lastIndexOf("X");
```

- **MAP:** Dizinin elemanlarını belirtilen bir fonksiyondan geçirir. İşlem sonuçları dizi olarak geri döner.

```
var dizi=new Array(3, 6, 8);
function kare(n){ return n*n}
var s=dizi.map(kare);
document.write(s.join());
```

- **REDUCE:** Dizinin elemanları üzerinde fonksiyon ile işlem yapar.

```
var dizi=new Array(3, 6, 8, 9);
function topla(a,b){ return a+b;}
document.write(dizi.reduce(topla)); --> 26
```

- **SOME:** Dizi elemanlarını kontrol eder. true/false

```
var dizi=new Array(3, 6, 8, 9);
function kontrol(x){ return (x>7);}
document.write(dizi.some(kontrol)); -->true
```

Fonksiyonlar

- Bağımsız çalışan kod bloklarıdır.
- Kodların tekrar yazılmasına gerek kalmadan birden fazla çalıştırılması sağlanır.
- Fonksiyonlar değer döndürebilir veya sadece işlem yapılabilir.
- Fonksiyonlar parametresiz olabilir.
  - function fonksiyon_adı(){....}
  - var fonksiyon_adı=function(){....}
  - function fonksiyon_adı(parametreler){....}
  - var fonksiyon_adı=function(parametreler){....}
  - function fonksiyon_adı(){return değer;}
  - var fonksiyon_adı=function(){return değer;}

```
function topla(a,b){return a+b;} //var topla=function(a,b){return a+b;}
document.write(topla(4,5));
```

Fonksiyonlar

- Fonksiyon parametreleri dizi olarak elde edilebilir.

```
<script>
    function Degerler(par)
    {
        var toplam=0;
        for(var i in Degerler.arguments)
            toplam+=Degerler.arguments[i];
        return toplam;
    }
    document.write(Degerler(4,3,8));
</script>
```

Kendini Çağıran Fonksiyonlar (Recursive)

- Kendini çağırabilen fonksiyonlara özyinelemeli (recursive) fonksiyon denir.

```
function faktoryel(n)
{
      if(n==1)
            return 1;
      else
            return n*faktoryel(n-1);
}

document.write(faktoryel(5));
```

Hazır Fonksiyonlar

- **parseInt:** String olarak belirtilen sayıları tamsayıya çevirir.
  - parseInt("100", 2) --> 4
  - parseInt("100", 8) --> 64
  - parseInt("100", 16) --> 256
  - parseInt("100", 10) --> 100
- **parseFloat:** String veriyi float sayı tipine çevirir.
  - parseFloat("3.14") --> 3.14
- **String:** Object değeri stringe çevirir.

```
document.write(String(23));
```

- **Number:** Object değeri sayıya çevirir.

```
document.write(Number("23")*5);
```

- **isNaN:** Değerin sayı olup olmadığını kontrol eder.

```
document.write(isNaN("A"));
```

- **toString:** Nesneleri string ifadeye çevirir. Nesne tanımlamasında özelleştirilebilir.

Nesneler

- Javascript, nesneye yönelik programlama yapılabilen bir dildir.
- JS'te sayfadaki her öğe bir nesnedir.
- Örnek olarak "document" bir nesne, "write" ise document sınıfındaki bir metottur.
- Oluşturulan diziler birer nesnedir.
  - var dizi=new Array();
- JS'te özel nesneler oluşturulabilir.
  - 1.Yöntem: Object ifadesi ile oluşturma
  - 2.Yöntem: "{...}" işaretleri arasında oluşturma
  - 3.Yöntem: fonksiyon kullanarak oluşturma

I.Yöntem: new Object() ile Nesne Oluşturma

- var nesne_adı=new Object(); ile tanımlama yapılır.
- Tanımlama yapıldıktan sonra, özellikler değer atanarak belirlenir.
  - Özellikler, nesne_adı.özellik_adı=değer; şeklinde tanımlanabilir.
  - Özellikler, nesne_adı["özellik_adı"]=değer; şeklinde tanımlanabilir.
- Nesne içerisinde fonksiyon tanımlaması yapılabilir.

```
var nesne1=new Object();
nesne1.özellik1="abc";
nesne1["özellik2"]="123";
nesne1.toString=function(){return this.özellik1+this.özellik2;}
document.write(nesne1);
```

II.Yöntem: "{...}" ile Nesne Oluşturma

- var nesne_adı={özellikler ve fonksiyonlar} ile tanımlama yapılır.
- Tanımlama yapılırken özellikler ve fonksiyonlar "{...}" işaretleri arasında belirtilir.
- Özellikler, özellik_adı:değer, ... şeklinde tanımlanır.
- Nesne içerisinde fonksiyon tanımlaması yapılabilir.

```
var nesne1={
        özellik1:"abc",
        özellik2:"123",
        toString:function(){return this.özellik1+this.özellik2;}
    };

nesne1.özellik1="def";
nesne1["özellik2"]="456";
document.write(nesne1);
```

III.Yöntem: "function" ile Nesne Oluşturma

- function nesne_adı(özellik_değer_parametreleri){ özellikler...} ile tanımlama yapılır.
- Tanımlama yapılırken özellikler ve fonksiyonlar "{...}" işaretleri arasında belirtilir.
- Özellikler, özellik_adı=değer, ... şeklinde tanımlanır.
- Nesne içerisinde fonksiyon tanımlaması yapılabilir.

```
function Nesne() //function Nesne(x,y)
{
    this.özellik1=""; //this.özellik1=x;
    this.özellik2=""; //this.özellik2=y;
    this.toString=function(){return this.özellik1+this.özellik2;}
}

var nesne1=new Nesne();
nesne1.özellik1="abc";
nesne1.özellik2="123";
document.write(nesne1);
```

Özellik veya Fonksiyon Ekleme ve Silme

- **Özellik ekleme:**

```
nesne1.özellik3="xyz";
```

- **Fonksiyon ekleme:**

```
nesne1.toString=function(){return this.özellik1+this.özellik2+this.özellik3;}
document.write(nesne1);
```

- **Özellik veya fonksiyon silme:**

```
delete nesne1.özellik2;
delete nesne1.toString;
```

- **Özelliğin veya fonksiyonun varlığını kontrol etme:**

```
document.write("özellik1" in nesne1);
```

- Örnek:

```
function Nesne()
{
        this.özellik1="";
        this.özellik2="";
//      this.toString=function(){return this.özellik1+this.özellik2;}
}

var nesne1=new Nesne();
nesne1.özellik1="abc";
nesne1.özellik2="123";
nesne1.özellik3="xyz";

delete nesne1.özellik2;
delete nesne1.toString;

document.write(nesne1);

document.write("özellik1" in nesne1);
```

Nesne Özelliklerini Listeleme

```
function Nesne(x,y,z,t)
{
 this.özellik1="";
 this.özellik2=0;
 this.özellik3=true;
 this.fonksiyon1=function(){document.write("mesaj");}
}

var nesne1=new Nesne();
for(var eleman in nesne1)
{
 document.write(eleman+":"+nesne1[eleman]);
}
```

Prototype

- JS nesnelerine yeni bir özellik veya metot tanımlamak amacıyla kullanılır.
- JS nesnelerinin mevcut özellik veya metotlarının yeniden tanımlanması için kullanılır.

```
function ilkHarfler(x)
{
 var s=this.split(x);
 var s2="";
 for(var i in s) s2+=s[i][0];
 return s2;
}

String.prototype.ilkHarfler=ilkHarfler;
String.prototype.toUpperCase=function(){ return "XYZ";}

var ss="Celal Bayar Üniversitesi";
document.write(ss.ilkHarfler(" ")); //CBÜ
document.write(ss.toUpperCase()); //XYZ
```

String

- **length:** String ifadenin uzunluğunu verir.

```
var s="Akhisar"; document.write(s.length); //7
```

- **charAt:** String ifade içerisinde belirtilen konumdaki karakteri verir.

```
var s="Akhisar"; document.write(s.charAt(2)); //h
```

- **charCodeAt:** String ifade içerisinde belirtilen konumdaki karakterin UNICODE değerini verir.

```
var s="Akhisar"; document.write(s.charCodeAt(0)); //65
```

- **fromCharCode:** Belirtilen ASCII kodunun karakter karşılığını verir.

```
document.write(String.fromCharCode(65,66,67)); //ABC
```

- **indexOf:** Belirtilen metin parçasının konumunu belirtir.

```
var s="akhisar"; document.write(s.indexOf("a")); //0
```

- **lastIndexOf:** Belirtilen metin parçasının son konumunu belirtir.

```
var s="akhisar"; document.write(s.lastIndexOf("a")); //5
```

- **link:** HTML bağlantısı oluşturur.

```
var s="akhisar";
document.write(s.link("http://www.akhisar.gov.tr"));
```

String

- **replace:** String ifadenin bir kısmını değiştirir.

```
document.write(s.replace("hi","xy")); //akxysar
```

- **search:** String ifade içerisinde aranan metnin konumunu verir.

```
document.write(s.search("hi"));//2
```

- **slice:** String ifadenin bir kısmını keser.

```
document.write(s.slice(2,4));//hi
```

- **split:** String ifadeyi belirtilen karakterlerden parçalara ayırır. Dizi olarak parçaları verir.

```
var s="Celal Bayar Üniversitesi";
var s2=s.split(" ");
for(var x in s2) document.write(s2[x][0]); //CBÜ
```

- **concat:** String ifadeleri birleştirir.

```
var s1="Akhisar";
var s2="MYO";
document.write(s1.concat(s2)); //Akhisar MYO
```

String

- **sub:** String ifadeyi alt simge olarak gösterir.

```
var s1="2", s2="4"; document.write("H"+s1.sub()+"SO"+s2.sub()); //H2SO4
```

- **sup:** String ifadeyi üst simge olarak gösterir.

```
var s1="2", s2="4"; document.write("H"+s1.sup()+"SO"+s2.sup()); // H²SO⁴
```

- **substr:** Belirtilen konumdan itibaren, belirtilen miktarda karakteri verir.

```
var s="Akhisar"; document.write(s.substr(2,4)); //hisa
```

- **substring:** Belirtilen iki konum arasındaki karakterleri verir

```
var s="Akhisar"; document.write(s.substring(2,4)); //hi
```

- **toLowerCase:** String ifadeyi küçük harflere dönüştürür.

```
var s="AKHİSAR"; document.write(s.toLowerCase()); //akhisar
```

- **toUpperCase:** String ifadeyi büyük harflere dönüştürür.

```
var s=" akhisar "; document.write(s.toUpperCase()); // AKHISAR
```

- **match**: Düzenli ifade eşleşen string parçalarını dizi olarak geri döndürür.

Math

Özellik	Açıklama
E	2.718 değerini verir
LN2	2'nin logaritmasını verir
LN10	10'nun logaritmasını verir
LOG2E	E'nin logaritmasını verir
LOG10E	10'nun logaritmasını verir
PI	Pi sayısını verir
SQRT1_2	½'nin karekökünü verir
SQRT2	2'nin karekökünü verir

Math

Metot	Açıklama
abs	Sayının mutlak değeri verir
ceil	Ondalık sayıyı en yakın üst sayıya yuvarlar
floor	Ondalık sayının tam kısmını verir.
round	Ondalık sayıyı üste veya alta yuvarlar
max	2 sayıdan büyük olanı verir
min	2 sayıdan küçük olanı verir
pow	A sayısının B. kuvvetini hesaplar
random	0 ile 1 arasında rastgele sayı üretir
sqrt	Sayının karekökünü hesaplar

Date

date	Geçerli tarih veya belirtilen tarih üzerinde işlem yapılmasını sağlayan nesne tanımlanır.
getDate	Ay içerisindeki gün (1-31)
getDay	Hafta içerisindeki gün (0-6)
getFulYear	Yıl (0-2013)
getMonth	Ay (0-11)
getYear	Yıl (1900-2013)
getHours	Saat (0-23)
getMinutes	Dakika (0-59)
getSeconds	Saniye (0-59)
getMilliseconds	Mili Saniye (0-999)
getTime	1 Ocak 1970 tarihinden itibaren geçen süre (ms)
toLocalestring	Tarih ve Saati yerel formata göre gösterir
toDateString	Tarihi genel formata göre gösterir
toLocaleDateString	Tarihi yerel formata göre gösterir
toTimeString	Saati genel formata göre gösterir
toLocaleTimeString	Saati yerel formata göre gösterir

Date

```
var tarih=new Date();
document.write("<br>getDate:"+tarih.getDate());
document.write("<br>getDay:"+tarih.getDay());
document.write("<br>getFullYear:"+tarih.getFullYear());
document.write("<br>getMonth:"+tarih.getMonth());
document.write("<br>getYear:"+tarih.getYear());

document.write("<br>getHours:"+ tarih.getHours());
document.write("<br>getMinutes:"+ tarih.getMinutes());
document.write("<br>getSeconds:"+ tarih.getSeconds());
document.write("<br>getMilliseconds:"+ tarih.getMilliseconds());
document.write("<br>getTime:"+ tarih.getTime());

document.write("<br>toLocaleString:"+ tarih.toLocaleString());
document.write("<br>toDateString:"+ tarih.toDateString());
document.write("<br>toTimeString:"+ tarih.toTimeString());
document.write("<br>toLocaleTimeString:"+ tarih.toLocaleTimeString());
```

Number

Özellik/Metot	Açıklama
MAX_VALUE	En büyük sayıyı verir. $1.79x10^{308}$
MIN_VALUE	En küçük sayıyı verir. $(5x10^{-324})$
toFixed	Ondalık sayılarda virgülden sonraki basamak sayısını düzenler. x=123.456789; x.toFixed(2); --> 123.45
toPrecision	Ondalık sayılarda toplam basamak sayısını düzenler. x=123.456789; x.toPrecision(4); --> 123.4

RegExp (Düzenli İfade)

- Düzenli ifadeler olarak çevrilen (Regular Expression); verilen string ifadelerin bölünmesi, karşılaştırılması, ayrılması gibi değişik formatların uygulanmasını sağlayan desenlerdir.
- Email vb. bilgilerin girişinin kontrol edilmesi sağlanır.
- Kullanım alanları:
  - Arama motorları
  - Veri doğrulama
  - Sözdizimi doğrulama
  - Kelime işleme
  - Bilgi edinme
- RegExp nesnesi iki kısımdan oluşur.
  - Kullanıcı tarafından girilecek bilgiye uygulanacak desen
  - Oluşturulan desene yapılacak düzenlemeler
  - Düzenli ifade = / desen / ayarlar
  - Ayarlar:
    - g: Düzenli ifadenin her tarafında eşleme aranır.
    - i: Eşleşmelerde büyük-küçük harf duyarlılığının olmaması için kullanılır.
    - m: Çok satırlı metinlerde düzenli ifade eşleşmesi aramak için kullanılır.

RegExp (Düzenli İfade)

- Desen ve ayarlar sınırlayıcı bir karakter içerisine alınması gerekir.
- Genellikler "/" işareti kullanılır.
- **ESCAPE:** String ifade de özel karakter aranıyorsa, karakterin önüne "\" işareti eklenir.

\n	Yeni satır (newline)
\r	Satırbaşı
\t	Sekme (Tab) karakteri
\v	Dikey tab karakteri
\0	Null karakteri

RegExp (Düzenli İfade) – Özellikler ve Metotlar

Özellik/Metot	Açıklama
global	String ifadenin tamamının karşılaştırılması durumunu kontrol eder.
ignoreCase	String ifadedeki küçük-büyük harf ayrımının dikkate alınıp alınmayacağını kontrol eder.
exec	Düzenli ifade ile eşleşmeyi kontrol eder, eşleşenler dizi olarak geri döndürülür
test	Düzenli ifade ile eşleşmeyi kontrol eder. (true/false)

RegExp (Düzenli İfade) – Özel Karakterler

\	Belirtilen karakteri ifade içinde arar. s="Javascript!"; x=/!/g; document.write(x.test(s)); --> true
^	Verilen ifadenin satırın başında yer alıp almadığını kontrol eder s="Javascript!"; x=/^J/g; document.write(x.test(s)); --> true
$	Verilen ifadenin satırın sonunda yer alıp almadığını kontrol eder. s="Javascript!"; x=/!$/g; document.write(x.test(s)); --> true
.	Herhangi bir karakteri temsil eder. s="Javascript!"; x=/./g; document.write(x.test(s));
\b	Sonrasında gelen karakterleri kelime içerisinde arar. s="!Javascript!"; x=/\b!/g; document.write(x.test(s)); -->true
\B	Sonrasında gelen karakterleri kelime başında ve sonunda arar. s="!Javascript!"; x=/\B!/g; document.write(x.test(s)); -->true
?=	Aranacak ifade ile düzenli ifadenin birlikte geçenlerini döndürür. s="Javascript dili"; x=/pt(?= dil)/g;
?!	Aranacak ifade ile düzenli ifadenin birlikte geçmeyenlerini döndürür. s="Javascript dili"; x=/ja(?! dil)/g;

RegExp (Düzenli İfade) – Özel Karakterler

X|Y	X veya Y karakterleri ile karşılaştırma s="mehmetdikmen.net"; x=/com|net/g; document.write(x.test(s));
\d	Rakamları temsil eder. s="a1"; x=/\d/g; document.write(x.test(s));
\D	Rakamların haricindeki karakterleri temsil eder. s="a1"; x=/\D/g; document.write(x.test(s));
\s	Boşlukları temsil eder. s="a b"; x=/\s/g; document.write(x.test(s));
\S	Boşluklar hariç diğer karakterleri temsil eder. s="a b"; x=/\s/g; document.write(x.test(s));
\w	Harf, rakam, alt çizgi karakterleri temsil eder. s="a b"; x=/\w/g; document.write(x.test(s));
\w	Harf, rakam, alt çizgi dışındaki karakterleri temsil eder. s="a%b"; x=/\W/g; document.write(x.test(s));

RegExp (Düzenli İfade) – Özel Karakterler

[ABC]	Köşeli parantezler içerisindeki karakterlerin varlığını test eder. s="Javascript123"; x=/[as3]/g; document.write(x.test(s));
[A-Z]	Belirtilen karakterlerin arasındakileri test eder. s="Javascript123"; x=/[as3]/g; document.write(x.test(s));
[^ABC]	Köşeli parantezler içerisindeki karakterlerin dışındakileri test eder. s="Javascript123"; x=/[as3]/g; document.write(x.test(s));
[^A-Z]	Belirtilen karakterler aralığının dışındakileri test eder. s="Javascript123"; x=/[as3]/g; document.write(x.test(s));
{x}	Öncesinde gelen karakterlerin x kadar tekrarlanmasını kontrol eder. s="Jaavascript123"; x=/a{2}/g; document.write(x.test(s));
{x,}	Öncesinde gelen karakterlerin en az x kadar tekrarlanmasını kontrol eder. s="Jaavaaascript123"; x=/a{2,}/g; document.write(x.test(s));
{x,y}	Öncesinde gelen karakterlerin en az x, en fazla y kadar tekrarlanmasını kontrol eder. s="Jaavaaascript123"; x=/a{2,3}/g; document.write(x.test(s));

RegExp (Düzenli İfade) – Özel Karakterler

+	Öncesindeki karakterin 1 veya daha fazla tekrarlanacağını s="html,html,html"; x=/ml+/g; document.write(s.match(x));
?	Öncesindeki karakterin 0 veya 1 defa tekrarlanacağını belirtir. s="html,html,html"; x=/ml?/g; document.write(s.match(x));
*	Öncesindeki karakterin 0 veya daha fazla tekrarlanacağını belirtir. s="html,html,html"; x=/ml*/g; document.write(s.match(x));
()	Parantez içindeki ifade aranır. s="html,html,html"; x=/(ml)/g; document.write(s.match(x));

RegExp (Düzenli İfade) – Örnek

- var s1="_a1";
- var s2="0(555) 555 55 55";
- var s3="abc@xyz.com.tr";
- var s4="11-03-2013";

- var x1=/\w{0,}\d{1,}/g;
- var x2=/0\(5[0-9]{2}\)\s[0-9]{3}\s[0-9]{2}\s[0-9]{2}/g;
- var x3=/^[^0-9][a-zA-Z0-9_]+([.][a-zA-Z0-9_]+)*[@][a-zA-Z0-9_]+([.][a-zA-Z0-9_]+)*[.][a-zA-Z]{2,4}$/g;
- var x4=/^\d{1,2}[\/.-]\d{1,2}[\/.-]\d{2,4}$/g;

- document.write(x1.test(s1));
- document.write(x2.test(s2));
- document.write(x3.test(s3));
- document.write(x4.test(s4));

Tarayıcı Nesneleri – Window

- DOM'un en üstteki nesnesidir. Web sayfasının bulunduğu penceredir.
- En üstte olduğu için document, navigatör, screen, location, history gibi nesneleri içine alır.
- Metotlarda "window" yazılmasına gerek yoktur.

- screenX: Ekran ile tarayıcının sol kenarı arasındaki değeri piksel cinsinden verir.
- screenY: Ekran ile tarayıcının üst kenarı arasındaki değeri piksel cinsinden verir.

```
function goster1(){ document.title=window.screenX+","+window.screenY; }
<input type="button" name="göster1" value="göster1" onclick="goster1()" />
```

- print: Web sayfasının içeriğini yazıcıya döken arayüzü açar. Gerekli ayarlar yapıldıktan sonra sayfa yazdırılabilir.

```
window.print();
```

Window

- resizeBY: Pencerenin yüksekliğinin ve genişliğinin belirtilen miktarda değiştirilmesi.
  window.resizeBy(250, 200);
- resizeTo: Pencerenin genişlik ve yüksekliğinin belirtilen miktarda boyutlandırılması.
  window.resizeTo(500, 450);
- moveBy: Pencerenin yatay ve dikey eksende taşınması.
  window.moveBy(50, 150);
- moveTo: Pencerenin belirtilen konuma taşınması.
  window.moveTo(50, 150);
- scrollBy: Kaydırma çubuklarının mevcut konumlarından belirtilen miktar kadar kaydırılmasını sağlar. Yatay veya dikey olarak artırma/azaltma yapılabilir.
  window.scrollBy(0, 100);
- scrollTo: Kaydırma çubuklarının belirtilen koordinantlara konumlandırılmasını sağlar.
  - window.scrollTo(50, 150);

Window

- **alert**: Bilgi/uyarı vermek amacıyla kullanılan mesaj kutularıdır.

```
alert("Mesaj kutusu...");
```

- **prompt**: Kullanıcının bilgi girişi yapması istenilen penceredir.

```
var x=prompt("Bilgi gir:","varsayılan değer");
alert("Girilen değer:"+x);
```

- **confirm**: Evet/Hayır, True/False gibi alternatifli sorular sorularak cevaplar alınmasını sağlar.

```
var x=confirm("Soru");
alert("Cevap:"+x);
```

Window

- open: Belirtilen URL adresini girilen özelliklere göre yeni pencerede açar.
  var pencere=window.open("http://localhost");
- window.open("URL","Konum","Özellikler");
  - URL: açılacak bağlantı belirtilir.
  - Konum: pencerenin nasıl açılacağı belirtilir.
    - _blank: Yeni pencerede açar.
    - _self: Geçerli pencerede açar.
  - Özellikler:
    - directories: yer imleri çubuğu durumu. (yes/no, 1/0)
    - fullscreen: Tam ekran modu. (yes/no, 1/0)
    - height: Pencere yüksekliği. (piksel)
    - width: Pencere genişliği
    - left: Pencerenin soldan konumu
    - top: Pencerenin üstten konumu
    - location: Adres satırının durumu (yes/no, 1/0)
    - menubar: Tarayıcı menü çubuğu durumu
    - resizable: Boyutlandırılabilme durumu
    - scrollbars: Kaydırma çubuklarının gösterim durumu
    - titlebar: Başlık çubuğunun gösterim durumu
    - toolbar: İleri-geri düğmelerinin durumu
    - status: Durum çubuğunun gösterim durumu
- close: Open ile açılan pencereyi kapatır.

Window

- setInterval: Belirtilen fonksiyonun veya kodun belirtilen süre sonunda tekrar çalıştırılmasını sağlar.
- clearInterval: SETINTERVAL ile başlatılan işlemi sonlandırır.

```
<script>
var x;
var i=-1;
var resimler=new Array("1.jpg","2.jpg","3.jpg","4.jpg");
function islem(){
      i++;
      i%=resimler.length;
      document.getElementById("img1").src=resimler[i];
      document.getElementById("span1").innerHTML=resimler[i];
}
function basla(){ x=window.setInterval("islem()", 1000); }
function dur(){ window.clearInterval(x);}
</script>

<input type="button" value="Basla" onclick="basla();" />
<input type="button" value="Dur" onclick="dur();" />
<br><img id="img1"  width="320" height="240"/>
<br><span id="span1"></span>
```

Window

- setTimeout: Belirtilen fonksiyonun, belirtilen süre sonunda bir defa çalıştırılmasını sağlar.
- clearTimeout: SETTIMEOUT ile başlatılan işlemi sonlandırır.

```
<script>
var x;
var i=-1;
var resimler=new Array("1.jpg","2.jpg","3.jpg","4.jpg");
function islem(){
	i++;
	i%=resimler.length;
	document.getElementById("img1").src=resimler[i];
	document.getElementById("span1").innerHTML=resimler[i];
	x=window.setTimeout("islem()", 1000);
}
function basla(){ islem();}
function dur(){ window.clearTimeout(x);}
</script>

<input type="button" value="Basla" onclick="basla();" />
<input type="button" value="Dur" onclick="dur();" />
<br><img id="img1"  width="320" height="240"/>
<br><span id="span1"></span>
```

Document

- HTML sayfasında bulunan bütün nesnelerin temelini oluşturur.
- Sayfa özelliklerine, sayfada bulunan diğer nesnelere erişmek için kullanılır.
  - write/writeline: Sayfa üzerine yazdırma metodudur.
  - characterSet: Sayfanın karakter setini gösterir.
  - domain: Alan adını gösterir.
  - title: Sayfa başlık bilgisini gösterir.
  - referrer: Önceki sayfanın adresini gösterir.
  - URL: Mevcut sayfanın adresini gösterir.
  - width-height: Genişlik ve yükseklik
  - lastModified: Son değiştirilme tarihini gösterir.

```
document.write("<br>characterSet:"+document.characterSet);
document.write("<br>domain:"+document.domain);
document.write("<br>title:"+document.title);
document.write("<br>referrer:"+document.referrer);
document.write("<br>URL:"+document.URL);
document.write("<br>width:"+document.width);
document.write("<br>height:"+document.height);
document.write("<br>lastModified:"+document.lastModified);
```

Document

```
<form name="form1">
    <input type="text" name="txt1" />
    <input type="text" name="txt2" />
</form>

<a href="#" name="link1">link1</a>
<a href="#" name="link2">link2</a>

<img src="#" />
<img src="#" />
<img src="#" />

<script>
    document.write("<br>Form Sayısı: "+document.forms.length);
    document.write("<br>Link Sayısı: "+document.links.length);
    document.write("<br>Resim Sayısı: "+document.images.length);

    document.write("<br>Form1 eleman sayısı : "+document.forms[0].length);
    document.write("<br>Form1 1.alt eleman : "+document.forms[0][1].name);
</script>
```

Document

- getElementById: Verilen id numarasına göre ilgili nesneye ve özelliklere erişilir.
- getElementsByName: Verilen «name» özelliğine göre ilgili nesneye ve özelliklerine ulaşılır. Nesne özelliklerine doğrudan ulaşılamaz, item özelliğinde indeks numarası belirtilmelidir.
- getElementsByTagName: HTML içerisinde belirtilen etikette olan tüm nesnelerin listesini verir.

```
<input type="text" name="txt1" id="txt1" value="metin kutusu1" />
<input type="text" name="txt1" id="txt2" value="metin kutusu2" />
<input type="text" name="txt2" id="txt3" value="metin kutusu3" />
<input type="text" name="txt2" id="txt4" value="metin kutusu4" />

<script>
	document.getElementById("txt1").style.backgroundColor="#008";
	var etiketler1=document.getElementsByName("txt1");
	for(var i=0; i<etiketler1.length;i++)
		etiketler1.item(i).style.color="#ff0";
	var etiketler2=document.getElementsByTagName("input");
	for(var i=0; i<etiketler2.length;i++)
		document.write("<br>"+etiketler2.item(i).value);
</script>
```

Document

- createElement: HTML elementinin oluşturulmasını sağlar.
- createTextNode: Elementlerin içerisine metin ekler.

```
var table=document.createElement("table");
var tbody=document.createElement("tbody");
for(var i=1; i<5;i++)
{
	var tr=document.createElement("tr");
	for(var j=1; j<4; j++)
	{
		var td=document.createElement("td");
		var text=document.createTextNode(i+".satır, "+j+".sütun");
		td.appendChild(text);
		tr.appendChild(td);
	}
	tbody.appendChild(tr);
}
table.appendChild(tbody);
table.setAttribute("border","1");
document.getElementsByTagName("body")[0].appendChild(table);
```