

JavaScript

JavaScript



```
1 <script>
2 if(age > 19){
3   alert("Adult");
4 } else{
5   alert("Teenager");
6 }
7 </script>
```



Döküman İçeriği

- 3 : javascript nedir?
- 8 : projeye dahil etme
- 10 : veri türleri nelerdir?
- 12: değişken tanımlama
- 15: operatörler
- 26: fonksiyonlar
- 36 : object(nesne)
- 38: events(olaylar)
- 42: string(metin)
- 45: template literals(back tick)
- 47: sayı metodları
- 49: array ve metodları
- 60: tarih nesnesi
- 64: math nesnesi
- 68: boolean türü
- 70: koşul ve döngüler
- 81: try catch(hata yakalama)
- 83: scope kavramı
- 89: use strict(katı mod)
- 91: arrow function
- 93: JSON
- 95: asenkron işlemler
- 101: DOM
- 111: BOM
- 120: yerleşik API'ler

JavaScript Nedir?

JavaScript, web geliştiricilerinin web sayfaları, uygulamalar, sunucular ve hatta oyunlar geliştirdikten daha dinamik etkileşimler oluşturmak için yaygın olarak kullandıkları **hafif bir programlama dilidir.**

JavaScript olmadan web siteleri yalnızca düz metin ve resimler şeklinde kalır: Google Haritalar yalnızca basit bir harita şeklinde ve Facebook bir gazete gibi görünür.

JavaScript, web'in dili olarak kabul edilir çünkü karmaşık özellikleri, web'de kolaylıkla uygulamanızı sağlar, sunucu etkileşimi ve yanıt sürelerini azaltır, kullanıcı deneyimini iyileştirir. Bu da onu herhangi bir web uygulaması için önemli bir dil yapar. HTML ve CSS ile birlikte eksiksiz bir web sitesi oluşturmak için kullanılabilir.

JavaScript Ne İçin Kullanılır?

Tarihsel olarak web sayfaları bir kitaptaki sayfalara benzer şekilde statikti. Statik bir sayfa bilgileri çoğunuyla sabit bir düzende görüntüyordu ve şimdiden modern bir web sitesinden beklediğimiz her şeyi yapmıyordu. JavaScript, web uygulamalarını daha dinamik hale getirmek için tarayıcı bazı teknoloji olarak ortaya çıktı. Tarayıcılar JavaScript kullanarak kullanıcı etkileşimi etkileşimine yanıt verebilir ve web sayfasındaki içerik düzenini değiştirebilir.

Dil olgunlaşıkça JavaScript geliştiricileri kitaplıklar, çerçeveler ve programlama uygulamaları oluşturdu ve bunları web tarayıcılarının dışında kullanmaya başladı. Bugün JavaScript'i hem istemci tarafı hem de sunucu tarafı geliştirmeye için kullanabiliyorsınız.

Günümüzde JavaScript; **AngularJS**, **jQuery** ve **ReactJS** gibi karmaşık projeleri basitleştirmek için birçok çerçeve ve kütüphaneye sahiptir.

Javascript ile Neler Yapılabilir?

JavaScript, web sayfalarının işlevsellliğini ve interaktivitesini artırmak için kullanılır. Burada bazı nedenleri sıralayabiliriz:

- **Dinamik içerik oluşturma:** JavaScript kullanarak web sayfalarında dinamik içerik oluşturabilirsiniz. Örneğin, bir formun doldurulmasını kontrol etmek, bir pop-up penceresi açmak veya bir harita gibi interaktif nesneler oluşturmak mümkündür.
- **Kullanıcı deneyimi:** JavaScript kullanarak kullanıcının web sayfasındaki deneyimini artırlırsınız. Örneğin, bir butona tiklandığında bir efekt oluşturmak veya bir formun doğrulanmasını sağlamak mümkün kılınır.

- **Asenkron İşlemler:** JavaScript ile web sayfasının arka plan işlemleri yapabilirsiniz. Örneğin, verileri sunucudan çekmek veya bir veri tablosunu güncellemek gibi işlemleri gerçekleştirebilirsiniz.

- **Platformlar arası:** JavaScript, çok çeşitli platformlar arasında çalışır. Örneğin, JavaScript'i web tarayıcılarında, sunucuda ve mobil uygulamalarda kullanabilirsiniz.

Sonuç olarak, JavaScript, web sayfalarının işlevsellliğini ve interaktivitesini artırmak için kullanılan önemli bir programlama dili olması sebebi ile web geliştirmeye dünyasında önemli bir yere sahiptir.

Kitaplıklar ve Çerçeveler

Geliştiricilerin kodlarını yazarken kullanabilecekleri, önceden yazılmış kod parçacıkları veya kütüphaneler olarak tanımlanabilir. Bu kod parçacıkları, web sayfalarında yapılması zor olan veya yazmak için çok zaman alan işlemleri kolaylaştırmak için tasarlanmıştır.

JavaScript Kitaplıkları(Libraries): JavaScript kitaplıkları, önceden yazılmış bir kod parçacığıdır. Bu kod parçacıkları, web sayfasında yapılması zor olan veya yazmak için çok zaman alan işlemleri kolaylaştırmak için tasarlanmıştır. Örneğin, jQuery, lodash gibi.

JavaScript Çerçeveler(Frameworks): JavaScript çerçeveleri, geliştiricilerin web uygulamalarını oluşturmaya yardımcı olan, daha büyyük bir kod yapısıdır. Bu çerçeveler, genellikle kullanıcı arayüzleri ve veri yönetimi gibi işlemleri kolaylaştırmak için tasarlanmıştır. Örneğin, **React**, **Angular**, **Vue** gibi.

Bir Web Sitesine Nasıl JavaScript Eklenir?

JavaScript, web sayfasına eklenmesi için iki yol vardır:

<script> Etiketi ile: JavaScript kodları, HTML sayfasının <head> veya

<body> bölümünden <script> etiketi içine yazılabılır. Örnek:

```
<head>
  <script>
    function selamla() {
      console.log("Merhab, Dünya!");
    }
  </script>
</head>
<body>
  <button onclick="selamla()"> Bana Tıkla </button>
</body>
```

Dosya içinde: JavaScript kodları, aynı bir dosya içinde saklanabilir ve HTML

sayfasına bir `<script>` etiketi kullanarak başvurulabilir. Örnek:

```
<head>
  <script src="myscript.js"></script>
</head>
<body>
  <button onclick="selamla()"> Bana Tıkla </button>
</body>
//myscript.js
function selamla() {
  console.log("Hello, World!");
}
```

Not: javascript kodlarının yerleşim yeri önemlidir, özellikle sayfa yüklenirken çalışmasını istediğiniz kodların `<head>` kısmında yer alması önerilir.

JavaScript Veri Türleri

JavaScript dilinde çeşitli veri türleri mevcuttur:

Number: Rakamlar (örneğin, [42](#), [3.14](#)).

String: Tıpkı içerisinde gösterilen metin dizeleri (örneğin, "Hello, World").

Boolean: [true](#) veya [false](#) değerleri.

Array: Bir diziye benzer veri yapıları (örneğin, [\[1, 2, 3, 4, 5\]](#)).

Object: Nesne yapıları (örneğin, [{isim: "Mehmet", yaş: 24, boy: 180}](#)).

Function: Bir fonksiyon yapısı (örneğin, [function myFunction\(\) { // code }](#)).

Symbol: JavaScript ECMAScript 6 ile birlikte geldi Symbol veri tipi, benzersiz bir anahtar oluşturmak için kullanılır.

undefined: Atanmış değer.

null: Hiçbir değer olmadığı anlamına gelir.

```
let x = 5;           => Number
let y = "Hello";    => String
let z = true;        => Boolean
let a = [1, 2, 3, 4, 5]; => Array(Dizi)
let b = {name: "John", age: 30}; => Object
let c = function () {console.log("Merhaba")}; => Function
let sym = Symbol();   => Symbol
let un;              => undefined
let n = null;         => null
```

JavaScript Değişkenleri

JavaScript'de, **değişkenler**, bellekte belirli bir değerin saklanması için kullanılan adlar olarak tanımlanabilir. Değişkenler, diğer programlama dillerinde olduğu gibi, **veri türlerini** depolamak veya kullanmak için kullanılırlar.

JavaScript'te değişkenleri tanımlamak için **var**, **let** veya **const** anahtar kelimelerinden birini kullanabilirsiniz.

var: JavaScript'in eski sürümlerinde kullanılan ve global(fonksiyon dışarısında) veya yerel (fonksiyon) düzeyinde tanımlanabilen değişkenleri tanımlar. Ön tanımlı değerleri yoktur ve istenirse atanabilir.

```
var x = 5;  
var y; //undefined değeri
```

let: ECMAScript 6'da önerilen ve yerel (fonksiyon) düzeyinde tanımlanabilen değişkenleri tanımlar. Bu değişkenlerin ön tanımlı değerleri yok ve istenirse bir değer atanabilir.

```
let isim; //bir değer vermedik
```

```
isim = "Furkan" //değeri şimdi tanımladık
```

```
let isim = "Furkan" //direkt değer vererek tanımladık
```

const: Değerleri değiştirilemeyen yerel (fonksiyon) düzeyinde tanımlanabilen değişkenler tanımlar. Bu değişkenlere başlangıç değeri atanması zorunludur.

```
const pi = 3.14;  
console.log(pi); // 3.14  
  
pi = 2.72 // Error: Const ile daha önce tanımlanan bir değeri değiştiremezsiniz.
```

let ve **var** arasındaki **farkın** en önemli yanı, let değişkenleri **sadece** tanımlandıkları **blokta** kullanılabilirken(örneğin bir fonksiyon içinde) var değişkenleri tanımlandıkları **blok** veya daha yukarıda tanımlanmış farklı fonksiyonlarda kullanılabilir.

JavaScript Operatörleri

JavaScript operatörleri, JavaScript programlarında veriler arasında matematiksel, mantıksal ve diğer işlemler gerçekleştirmek için kullanılan simgelerdir.

Örneğin, **toplama** (+), **çıkarma** (-), **çarpma** (*), **bölme** (/) gibi aritmetik operatörler kullanılabilir.

Eşitliği kontrol etmek için **eşittir** (=) veya **esit değil** (!) gibi karşılaştırma operatörleri kullanılır.

Aşağıda JavaScript operatörlerinin listesi verilmiştir:

Aritmetik operatörler: +, -, *, /, % (mod)

Atama operatörleri: =, +=, -=, *=, /=, %=

Karşılaştırma operatörleri: ==, ===, !=, !==, >, <, >=, <=

Mantıksal operatörler: &&, ||, !

typeof operatörü: girdığınız verinin türünü verir Örn: `typeof(5) //number`

JavaScript ayrıca birkaç özel amaçlı operatörler içerir, örneğin ternary operatör (`? :`) ve spread operatörü (`...`).

Aritmetik Operatörler

JavaScript **aritmetik** operatörleri(**+**, **-**, *****, **/**, **%**), matematiksel işlemleri gerçekleştirmek için kullanılır. Örneğin;

```
let x = 5;
let y = 7;
let toplam = x + y;
console.log(toplam); // 12
```

```
let r = 11;
let s = 3;
let kalan = r % s;
console.log(kalan); // 2
```

Atama Operatörleri

JavaScript atama operatörleri (`=`, `+=`, `-=`, `*=`, `/=`, `%=`), değişkenlere değerler atamak için kullanılır. Örneğin;

= (Atama) operatörü, bir değişkene bir değer atar. Örneğin, `x = 20` komutu `x` değişkenine 20 değerini atar.

```
let x = 5;  
let y = 2;  
x = y;  
console.log(x); // 2
```

$+=$ operatörü, bir değişkenin mevcut değerine bir değer ekler. Örneğin, $x += 5$ komutu x değişkeninin mevcut değerine 5 ekler.

```
let a = 5;  
a += 2;  
console.log(a); // 7
```

$-=$ operatörü, bir değişkenin mevcut değerinden bir değer çıkarır. Örneğin, $x -= 3$ komutu x değişkeninin mevcut değerinden 3 çıkarır.

```
let b = 10;  
b -= 5;  
console.log(b); // 5
```

$*$ = operatörü, bir değişkenin mevcut değeriyle bir değer çarpar. Örneğin, $x * =$

2 komutu x değişkeninin mevcut değerini 2 ile çarpar.

```
let c = 5;  
c *= 2;  
console.log(c); //10
```

$/=$ operatörü, bir değişkenin mevcut değerini bir değerle böler. Örneğin, $x /=$ 2 komutu x değişkeninin mevcut değerini 2 ile böler.

```
let d = 10;  
d /= 2;  
console.log(d); //5
```

Karşılaştırma Operatörleri

JavaScript karşılaştırma operatörleri(`==`, `==`, `!=`, `!==`, `>`, `<`, `>=`, `<=`), veriler arasındaki ilişkiyi kontrol etmek için kullanılır.Bazı Örnekler;

`== (Eşit) operatörü`,iki değerin eşit olup olmadığını kontrol eder. Örneğin,

`x == 5` komutu `x` değişkeninin değerin `5` ile eşit olup olmadığını kontrol eder.

```
let x = 5;  
console.log(x == 5);  
console.log(x == '5');
```

== (Tam Eşit) operatörü, iki değerinin eşit olup olmadığını ve veri tipinin eşit olup olmadığını kontrol eder. Örneğin, `x == 5` komutu `x` değişkeninin değerinin 5 ile eşit olduğunu ve veri tipinin aynı olup olmadığını kontrol eder.

```
let x = 7;  
console.log(x === "7"); // false  
console.log(x === 7); // true
```

!= (Eşit Değil) ve != (Tam Eşit Değil) operatörü, iki değerin eşit olmadığını kontrol eder. Tam eşit değil, ek olarak veri tipine de bakar.

```
let x = "mehmet";  
console.log(x != "ahmet"); // true  
console.log(x != "mehmet"); // false
```

< (**Küçük**) operatörü, sol tarafındaki değerin sağ tarafındaki değerden büyük olup olmadığını kontrol eder.

< (**Küçük**) operatörü, sol tarafındaki değerin sağ tarafındaki değerden küçük olup olmadığını kontrol eder.

>= (**Büyük veya Eşit**) operatörü, sol tarafındaki değerin sağ tarafındaki değerden büyük veya eşit olup olmadığını kontrol eder

<= (**Küçük veya Eşit**) operatörü, sol tarafındaki değerin sağ tarafındaki değerden küçük veya eşit olup olmadığını kontrol eder

Mantıksal Operatörler

JavaScript mantıksal operatörleri(**&&**, **||**, **!**), gerçek veya yanlış değerleri kontrol etmek için kullanılır.

&& (VE) operatörü, iki koşulun da gerçek olması gerektiğini kontrol eder.

Örneğin, $x > 5 \&\& y < 10$ komutu x değişkeninin 5'ten büyük olması ve y değişkeninin 10'dan küçük olması gerektiğini kontrol eder.

```
let x = 6;  
let y = 9;  
console.log(x > 5 && y < 10); // true
```

|| (VEYA) operatörü, iki koşuldan en az biri gerçek olması gerektiğini kontrol eder. Örneğin, $x > 5 \text{ || } y < 10$ komutu x değişkeninin 5'ten büyük olması veya y değişkeninin 10'dan küçük olması gerektiğini kontrol eder.

```
let x = 8;  
let y = 15;  
console.log(x > 5 || y < 10); // true
```

!(DEĞİL) operatörü, bir koşulun gerçek olmadığını kontrol eder. Örneğin, $!(x > 2)$ komutu x değişkeninin 2'den büyük olmadığını kontrol eder.

```
let x = 4;  
console.log(!(x > 5)); // true
```

JavaScript Fonksiyonlar

Fonksiyon , belirli sayıda verileri kullanarak bunları işleyen ve bir sonuç üreten komut grubudur.

Fonksiyonların genel özellikleri:

Bir amaca hizmet eden program parçacıklarıdır.

Fonksiyon çalıştırıldığında, bir işlemi yerine getirmesi yada bir değer döndürmesi istenecektir.

Çağrılmış olan bir fonksiyon ya geriye bir sonuç değeri döndürür ve/veya fonksiyon içerisindeki operasyonlarını işlemelerini tamamlayıp çağrılan yere geri döner.

Fonksiyonların Yapısı

```
function fonksiyonAdı(parametre1,parametre2){  
    // Yapmasını istediğiniz işlemler  
}
```

Not: Bütün fonksiyonlar parametre almak zorunda değil.

Parametresiz basit bir fonksiyon örnek verelim:

```
function mesaj() {  
    alert("merhaba javascript dünyası");  
}  
mesaj(); //fonksiyon çağrılmıyor
```

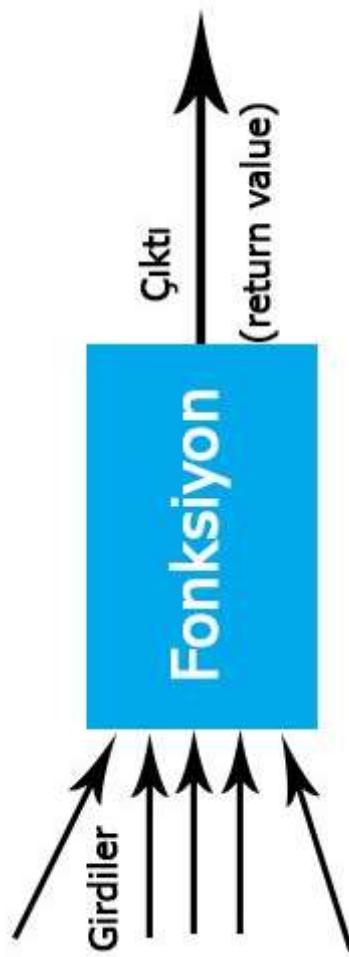
Parametre Kavramı

JavaScript fonksiyonları, **girdi** olarak değerler alabilirler. Bu değerlere **fonksiyon parametrelleri** denir. Fonksiyon tanımlaması sırasında, parametreler fonksiyonun parantez içinde tanımlanır. Fonksiyon çağrılrıken, **gerçek değerler** parametreler geçirilir. Örneğin:

```
function topla(a, b) {  
    return a + b;  
}  
  
let sonuc = topla(5, 3); // değerler sırasıyla parametrelerle atandı  
console.log(sonuc); // 8
```

Bu örnekte, topla fonksiyonu iki parametresi olan **a** ve **b** alır. Bu fonksiyon çağrırlarken, değerler **5** ve **3** olarak **parametrelere geçirilir**. Fonksiyon çalıştırıldığında, **a** ve **b** değerleri toplanır ve sonuç **8** olarak geri döndürülür.

Fonksiyonlar, birden fazla parametre alabilir veya hiçbir parametre almayıabilir. Ayrıca, fonksiyonlar geriye değer döndürebilir veya döndürmeyebilir. Önemli olan, fonksiyonun yaptığı işi anlamak ve okunaklı hale getirmek.



Fonksiyonlarda geri dönüş (`return`)

Fonksiyonlarda `return` sözcüğünün iki önemli işlevi vardır:

- Fonksiyon işlemleri sonucu oluşan değeri döndürür
- Fonksiyonu sonlandırır

Sonuç döndüren `return`:

`return` fonksiyon içerisinde her yerde kullanılabilir.

Kod `return` satırına eriştiğinde fonksiyon **`durur`** ve değer, fonksiyonun **çağırıldığı yere** geri **gönderilir**. En basit örnek iki değeri toplayan bir fonksiyon olabilir:

```
function topla(a, b) {  
    return a + b;  
}  
  
let sonuc = topla(5, 3);  
  
console.log(sonuc); // 8
```

Kod return satırına eriştiğinde fonksiyon **durdur** ve **a+b** değeri fonksiyonun çağırıldığı yere geri gönderdi.

Bir fonksiyon içerisinde birden fazla return fonksiyonu da olabilir:

```
let yas = 17
function yasKontrolu(yas) {
  if (yas > 18) {
    return true;
  } else {
    return confirm('Ebevenylerinin izni var mı?');
  }
}
```

return her zaman değer döndürmek zorunda değildir. Aşağıdaki örnekte fonksiyondan anında çıkışmayı sağlar:

```
function filmGoster( age ) {  
    if ( !yasKontrolu(yas) ) {  
        return; // yaşı 18den kucuk olduğundan fonksiyon durduruldu  
    }  
  
    alert( "Filmleri izleyebilirsin" );  
}
```

Yukarıdaki kodda eğer yasKontrolu(yas) **false** döndürür ise return devreye girer filmGoster fonksiyonu alerte erişemeyecektir.

Fonksiyonu İsimlendirme

Fonksiyonlar **eylemidir**. Bundan dolayı isimleri **yüklem** olmalıdır. **Net** olmalı ve fonksiyonun ne işe yaradığını ifade edebilmeliidir. Böylece kim kodu okursa, ne yazıldığına dair bir fikri olur.

Genel itibarı ile **eylemi tanımlayan** Ön ek kullanmak iyi bir yöntemdir. Bu ön ekler ile ilgili birlikte kod yazdığınız kişilere ile uyum içerisinde olmalısınız.

Örneğin "**show**" fonksiyonu her zaman bir şeyleri gösterir.

Fonksiyonlar örneğin şöyle başlayabilir:

- "get..." – değer döndürür,
- "calc..." – bir şeyler hesaplar,
- "create..." – bir şeyler yaratır,
- "check..." – bir şeyler kontrol eder ve boolean döndürür.

Not: İngilizce'de bu daha kolay önce eylemi yazıyorlar. Türkçe'de fiil genelde sonda olduğundan dolayı sıkıntı yaşanmaktadır. Fonksiyonlarınızı adlandırırken İngilizce adlandırırsanız okunması daha kolay olacaktır.

- sendMessage(..) // mesaj gönderir
- getAge(..) // yaşı döndürür
- calcSum(..) // toplamı hesaplar ve geri döndürür.
- createForm(..) // form oluşturur ve genelde geri döndürür.
- checkPermission(..) // izni kontor eder. true/false

JavaScript Object (Nesne)

JavaScript nesneleri, verileri ve işlevleri gruplandırmak için kullanılır. Bir JavaScript nesnesi, bir dizi anahtar-değer çifti içerebilir. Her anahtar bir dize olur ve değer bir nesne, dizi, sayı, metin veya diğer değişken tipi olabilir. JavaScript nesneleri, JSON formatını kullanır. Not: Yandaki örnekte “This” car objesini referans almak için kullanılmıştır

```
const car = {  
    marka: "Toyota",  
    model: "Camry",  
    year: 2020,  
    color: "white",  
    start: function() {  
        console.log(this.model + "Engine started.")  
    },  
    console.log(car.marka); // Toyota  
    console.log(car.model); // Camry  
    car.start(); // Camry Engine started
```

Nesne Özelliklerine erişim

JavaScript Özelliğlerine iki şekilde erişebiliriz.

`nesneAdı.ozellikAdı` yani önceki örnek için `car.marka` şeklinde erişilebilir.

Veya

`nesneAdı["ozellikAdı"]` yani önceki örnek için `car["marka"]` şeklinde erişilebilir.

Nesne İşlevlerine erişim

`nesneAdı.İşlevAdı();` yani önceki örnek için `car.start()` şeklinde erişilebilir.

JavaScript Events (Olaylar)

JavaScript olayları, kullanıcının bir web sayfasındaki belirli bir eylemi gerçekleştiği anda gerçekleşen olayları ifade eder. Örneğin, bir kullanıcı bir düğmeye tıkladığında, bir metin kutusuna veri girdiğinde veya bir sayfanın yüklenliğinde olaylar gerçekleşebilir. JavaScript ile bu olayları algılamak ve tepki vermek mümkündür.

JavaScript olayları, HTML etiketlerine tanımlanabilir veya JavaScript kodunda tanımlanabilir. Örneğin, bir HTML düğmesine onclick özelliği ekleyerek bir JavaScript fonksiyonunu çağırabilirsiniz.

```
<button onclick="myFunction()"> Fonksiyon Çalıştır</button>
```

Yukarıdaki örnekte butona tıklandığında **onclick** niteliği ile belirttiğiniz fonksiyon çalışır.

JavaScript kodunda ise, **addEventListener** metodу kullanarak olayları **algılamak** ve **tepki vermek** mümkünür.

Bu metod, bir nesnenin belirli bir olaya nasıl tepki vereceğini tanımlar.

```
let button = document.querySelector('button');

button.addEventListener('click', function() {
    console.log('Butona tıklandı');
});
```

Bu örnekte, "button" nesnesi seçilir, daha sonra addEventListener metodu ile click olayına tepki vermek için fonksiyon tanımlanır.

Bu sayede kullanıcı butona tıkladığı anda konsola Butona tıklandı yazar.

Javascript eylemlerinin bir listesi;

onload: Sayfa yüklenliğinde gerçekleşir.

onclick: Bir nesnenin üzerine tıklandığında gerçekleşir.

onmouseover: Fare nesnenin üzerine geldiğinde gerçekleşir.

onchange: Bir form elemanındaki değer değiştiğinde gerçekleşir.

onsubmit: Form gönderildiğinde gerçekleşir.

onkeydown: Bir tuşa basıldığında gerçekleşir.

onkeyup: Bir tuş bırakıldığında gerçekleşir.

onresize: Bir pencerenin boyutları değiştiğinde gerçekleşir.

onscroll: Bir sayfanın kaydırıldığından gerçekleşir.

Bu liste, JavaScript olaylarının sadece birkaçı değildir. JavaScript, birçok farklı olayı destekler ve bu olayları kullanarak web sayfalarınızda dinamik içerik oluşturabilirsiniz.

JavaScript String (Metin)

JavaScript'de, bir dizi karakterleri ifade eden veri tipi "String" olarak adlandırılır. Stringler, tek veya çift tırnak içinde tanımlanabilir. Örneğin:

```
let tanim1 = 'Hello, world!';
let tanim2 = "Hello, world!";
```

String tanımlarının içerisinde tırnak işaretlerini onları çevreleyen tırnak işaretini aynı olmadığı müddetçe kullanabilirsiniz. Örneğin:

```
let tanim3 = "He is called 'Johnny'";
let tanim4 = 'He is called "Johnny"';
```

JavaScript **string metotları**, string veri türünün işlemini yapmak için kullanılan metotları içerir. String metotlarının örnekleri:

concat(): Bir stringin sonuna başka bir stringi ekler.

```
"Hello".concat(" ", "World!") -> "Hello World!"
```

endsWith(): Bir stringin belirli bir karakter ile bitmediğini kontrol eder.

```
"Hello World!".endsWith("!") -> true
```

includes(): Bir stringin içinde belirli bir karakterin varlığını kontrol eder.

```
"Hello World!".includes("o") -> true
```

repeat(): Bir stringi belirli bir sayıda tekrarlamak için kullanılır.

```
"Hello!".repeat(3) -> "Hello!Hello!Hello!"
```

length: JavaScript Array ve String objelerinin **length** özelliği, objenin elemanlarının sayısını temsil eder. Array'erde elemanların sayısını, String'erde karakterlerin sayısını verir. Örnek: "Hello World!".length -> 12

replace(): Bir stringin içinde belirli bir karakter veya karakter dizisi yerine başka bir karakter veya karakter dizisi ile değiştirir.

```
"Hello World!".replace("World", "Dünya") -> "Hello Dünya!"
```

search(): Bir stringin içinde belirli bir karakter veya karakter dizisi arar ve ilk görüldüğü indeksi döndürür. `"Hello World!".search("o") -> 4`

slice(): Bir stringin belirli bir kısmını almak için kullanılır.

```
"Hello World!".slice(0,5) -> "Hello"
```

split(): Bir stringi belirli bir karakter veya karakter dizisi kullanarak parçalara ayırmak için kullanılır.

```
"Hello World!".split(' ', 1) -> ["Hello"]
```

toLowerCase() / toUpperCase(): Bir stringdeki tüm harfleri küçük veya büyük harfe dönüştürür.

```
"Hello World!".toLowerCase() -> "hello world!"
```

Template Literals(back ticks)

JavaScript Template Literals, ES6 (ECMAScript 6) ile birlikte gelen bir Özelliğidir. Template literals, daha okunaklı ve kolay bir şekilde string işlemleri yapmak için kullanılan bir yazım şeklidir. Bunu kullanmak için string oluşturken tırnak işaretini yerine **back-ticks** (` `) kullanılır.

Template literals, bir string içinde değişkenleri veya hesaplamaları direkt olarak kullanabilmemizi sağlar. Bu yazım şekli, herhangi bir string içerisinde değişkenleri veya hesaplamaları **\$\$** içinde yazabiliriz.

Ayrıca, template literals içerisinde birçok satır yazabiliriz.

```
let name = "İnci";
let age = 10;
console.log(`Merhaba ismim ${name} ve ${age} yaşındayım.);
```

Bu örnekte, name ve age değişkenleri back-tick `` arasında \${} içerisinde değişken direkt olarak kullanılmıştır.

Sonuç olarak konsola "Merhaba ismim İnci ve 10 yaşındayım." yazdırılacaktır.

Backtick oluşturmak için klavyenizde "alt" tuşuna basılı tutarken İki Kere `` (virgül) e basın

Number Methods

JavaScript, sayılarla çalışmak için birçok yerleşik metoda sahiptir. Aşağıda bazı metodların kısa açıklamalarını içeren bir liste bulabilirsiniz:

toFixed() - Bir sayıyı belirli bir ondalık basamağa yuvarlar ve sonucu bir dizi olarak döndürür.

2.3456789 için toFixed(2) şöyle dönecektir "2.35"

toPrecision() - Bir sayıyı belirli bir anamlı rakam sayısına yuvarlar ve sonucu bir dizi olarak döndürür.

2.3456789 için toPrecision(2) şöyle dönecektir "2.3"

toString() - Bir sayıyı metin olarak döndürür.

```
var x = 123; x.toString() söyle绝对不会 "123"
```

~~ - Bir sayı metodunu değiştir ama metni sayı olarak döndürür.

```
var x = "123" let z = ~~x // 123
```

toFixed . dan sonraki hane sayısını belirler ve son haneyi yuvarlar. Örneğin:

```
let x = 9.656;
```

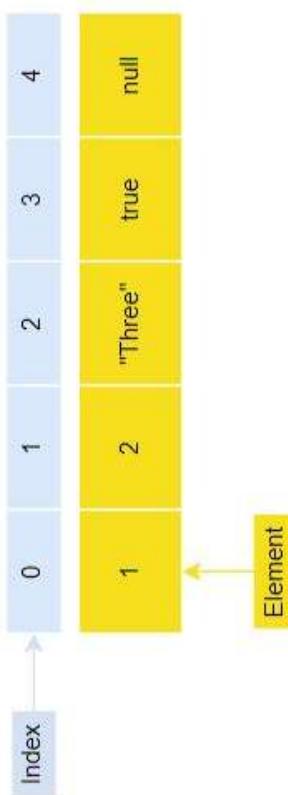
```
x.toFixed(0) // 10  
x.toFixed(2) // 9.66  
x.toFixed(3) // 9.656  
x.toFixed(6) // 9.656000
```

Javascript Array (Dizi)

JavaScript array, birden fazla değerin saklandığı bir veri yapısıdır. Array'ler, dizi (liste) şeklinde düzenlenmiş elemanları içerebilir. Elemanlar, dizinin başlangıç indeksi olan 0'dan başlayarak sıralanır.

JavaScript array'leri oluşturmak için, `new keyword`'ü kullanabilir ve `Array()` constructor'ı ile oluşturulur. Ayrıca, bir değişken ile tanımlandıktan sonra `[] (köşeli parantez)` işaretini de oluşturabilir.

```
let fruits = ["Apple", "Banana", "Orange"];
let numbers = new Array(1, 2, 3, 4, 5);
```



Array Methodları

JavaScript Array'leri, dizi içerisinde elemanların eklenmesi, silinmesi, değiştirilmesi gibi işlemler yapmamıza sağlanır. Bu metodlar ve dizilerin diğer özellikleri ile birlikte JavaScript Array'leri, veri yapılarının kolay yönetimi için kullanılabilir.

Bu metodlardan bilmeniz gerekenleri deneyebilmek için aşağıdaki gibi bir array oluşturalım.

```
const dizi = [1, 2, 3, 4, 5, 6];
```

forEach()

Bu metod, dizi içerisindeki her eleman için fonksiyon çalıştırılmaya yarar

```
dizi.forEach(  
  function (sayı) {console.log(sayı);}  
 ) // 1 2 3 4 5 6
```

Dizi de tanımladığımız her bir elementi sayı olarak adlandırdı
forEach ile her bir sayı için

```
dizi.forEach(  
  function (sayı , index) {  
    console.log(index+ "surasındaki rakam:  
      + sayı );})  
 // 0 sırasındaki rakam 1  
 // 1 sırasındaki rakam 2
```

fonksiyonu çalıştırıldı.
Eğer dönen elemanların index numarasına da istiyorsak yandaki gibi kullanmalıyız;

includes()

Bu metod bir dizinin belirli bir elemanı içerip içermediğini belirler, içeriyorsa **true** içermiyorsa **false** değeri döndürür.

```
var hayvanlar = ['kedi', 'köpek', 'yaraşa'];

console.log(pets.includes('kedi'));
// ekran çıktısı: true

console.log(pets.includes('at'));
// ekran çıktısı: false
```

filter()

Bu metod belirlediğimiz bir kurala göre tüm dizi elemanlarını kontrol eder ve kurallı geçen dizi elemanlarıyla yeni bir dizi oluşturur.

```
// array dizisinde bulunan 3'ten büyük sayıları filtrele  
const filtrelen = dizi.filter(sayı => sayı > 3);  
  
console.log(dizi); // ekran Çıktısı: [1, 2, 3, 4, 5, 6]  
  
console.log(filtrelen); // ekran Çıktısı: [4, 5, 6]
```

map()

Bu metod bir dizide değişiklik yaparak yeni bir dizi oluşturmamıza yardımcı olur. Aşağıda ki Örnekte var olan dizimin içinde ki sayıları bir artırarak yeni bir dizi oluşturacağız. Ardından var olan dizimiz ile yeni oluşturduğumuz diziyi ekranaya yazdıracağız burada ki önemli nokta eski dizimizde hiçbir değişikliğin olmadığıdır.

```
// dizideki sayıların hepsini dön ve 10 ekle
const onEklendi = dizi.map(
    function(sayı){ return sayı + 10}
);
console.log(onEklendi); // ekran çıktısı: [11, 12, 13, 14, 15, 16]
console.log(dizi); // ekran çıktısı: [1, 2, 3, 4, 5, 6]
```

Bu metodun `forEach` ile bir farkı olmadığını düşünebilirsiniz ancak `forEach` metodu ile tüm dizi elemanlarını döndürdüğümüzde yeni bir değişkene atamayız. Aynı örneği birde `forEach` ile yaparak ekran çıktılarını inceleyelim.

```
const onEklendi = dizi.forEach(sayı => sayı + 10);
console.log(oneAdded); // ekran çıktısı: undefined
```

İki metod arasındaki temel fark, map döngüsünün sonucu `return` ile geri **döndürken**, `foreach` döngüsü sonucu **geri döndürmez**. `Map` metodu, her bir dizi elemanını tek tek işleyerek **yeni bir dizi** oluşturur. `ForEach` metodu ise, her bir dizi elemanını tek tek işler ve geriye herhangi bir değer geri **döndürme** zorunluluğu yoktur.

reduce()

JavaScript'te "reduce" metodu, dizideki tüm elemanları tek bir değere indirmek için kullanılır. Reduce metodu, bir dizi içindeki tüm elemanları tek tek işler ve her işlem sonunda dizideki bir sonraki elemana geçer. Sonunda, dizideki tüm elemanlar işlendiğinden sonra, reduce metodunu geriye tek bir değer döndürür. Örnek olarak, bir dizideki tüm sayıların toplamını hesaplamak için kullanabiliriz

```
let diziToplam = dizi.reduce(function(sayıToplamı, anlıkDeger) {  
    return sayıToplamı + anlıkDeger;  
});  
console.log(diziToplam); // 21
```

push()

JavaScript Array `push()` methodu, Array'in sonuna eleman eklemek için kullanılır.

Bu method, Array'in `length` özelliğini artırrarak, yeni elemani sona ekler.

Örnek olarak daha önce tanımladığımız sayı dizisine yeni rakamlar ekleyelim.

```
dizi.push(10, 11, 12);  
console.log(dizi); // 1 2 3 4 5 6 10 11 12
```

Veya;

```
let fruits = ["Apple", "Banana"];  
fruits.push("Orange");  
console.log(fruits); // ["Apple", "Banana", "Orange"]
```

pop()

JavaScript Array pop() methodu, Array'in sonundaki son elemanı silmek için kullanılır. Bu method, Array'in length özelliğini azaltarak, son elemanı diziden çıkarır. Örnek olarak:

```
let meyveler = ["Elma", "Armut", "Muz"];
let sonMeyve = fruits.pop();
console.log(meyveler); // ["Elma", "Armut"]
console.log(sonMeyve); // "Muz"
```

Bu örnekte, fruits dizisinin sonunda yer alan "Orange" elemanı pop() methodu ile silinmiştir. fruits dizisi artık ["Apple", "Banana"] elemanlarını içermektedir. Ayrıca pop() methodu son elemanı geri döndürür. Örnekte lastFruit değişkenine atamıştır.

Diğer Array Methodları

concat(): Bir veya daha fazla diziyi birleştirir.

join(): Array elemanlarını bir string olarak birleştirir.

reverse(): Array elemanlarının sıralamasını ters çevirir.

sort(): Array elemanlarını sıralar.

slice(): Array elemanları arasında belirli bir kısmı almak için kullanılır.

splice(): Array'den belirli elemanları silmek veya eklemek için kullanılır.

shift(): Array'in başındaki ilk elemanı siler.

unshift(): Array'in başına eleman ekler.

JavaScript Date (Tarih)

JavaScript **Date**, JavaScript programlama dili için **tarih** ve saat verilerini işlemek için kullanılan bir nesnedir. Tarih ve saat verilerini **almak**, **ayarlamak** ve **birimlendirmek** için kullanılır.

JavaScript'te Date nesnesi oluşturmak için üç yöntem kullanabilirsiniz:

new Date() : Bu yöntem, mevcut tarih ve saatı almak için kullanılır. Örnek:

```
var tarih = new Date();
Console.log(tarih)
```

new Date(dateString): Bu yöntem, verilen tarih ve saat bilgilerini içeren bir string'i kullanarak Date nesnesi oluşturur. Örnek:

```
let dogumGunum = new Date('January 10, 2004');
console.log(dogumGunum); // Sat Jan 10 2004 GMT+0200 (GMT+03:00)
```

new Date(year, month, day, hours, minutes, seconds, milliseconds):

Bu yöntem, yıl, ay, gün, saat, dakika, saniye ve milisaniye bilgilerini kullanarak Date nesnesi oluşturur. Örnek:

```
let birTarih = new Date(2022, 11, 25, 10, 45, 32);
document.write(birTarih); // Sun Dec 25 2022 10:45:32 GMT+0300 (GMT+03:00)
```

Date Get Methodları:

JavaScript Date nesnesi, tarih ve saat verilerini almak, ayarlamak ve biçimlendirmek için birçok metot içerir. Aşağıda bazı önemli metodlar ve örnekleri verilmiştir:

getDate(): Bu metot, günü (1 ile 31 arası) döndürür. Örnek:

```
let mevcutTarih = new Date();
console.log(mevcutTarih.getDate()); // şu anki ayın günü
```

getMonth(): Bu metot, ayı (0 ile 11 arası) döndürür. Örnek:

```
mevcutTarih.getMonth() // şu anki ay ( Ocak için 0, Şubat için 1 , vb.)
```

getFullYear(): Bu metod, yılı (4 haneli) döndürür. Örnek:

```
mevcutTarih.getFullYear(); // 4 haneli yılı döndürür 2022
```

setDate(), setMonth(), setFullYear() : Tarihleri ayarlamak için kullanılır.

```
mevcutTarih.setDate(25); // tarih, şu anki ayın 25. gününe ayarlandı  
mevcutTarih.setMonth(11); // ay, Aralık olarak şu anki yıla ayarlandı  
mevcutTarih.setFullYear(2022); // yıl, 2022 olarak ayarlandı
```

toLocaleDateString(): Kullanıcının sistemi tarafından belirlenen tarih ve saat formatına göre biçimlendirilir.döndürür.

```
mevcutTarih.toLocaleDateString() // 12/25/2022
```

JavaScript Math (Matematik) Object

JavaScript **Math object**, matematiksel işlemler için kullanılan bir **global nesnedir**. Bu nesne, birçok matematik **fonksiyonlarını** içerir, Örneğin **trigonometrik fonksiyonlar**, **logaritma fonksiyonları** ve **rasgele sayılar üretebilmek** için kullanılabilirsiniz metodlar. Aşağıda bazı **önemli Math metodları** ve örnekleri verilmiştir :

Math.abs(x) : Bu metot, x'in mutlak değerini döndürür. Örnek:

```
Math.abs(-5); // 5
```

Math.round(x) : Bu metot, x'in yuvarlanmış değerini döndürür.

```
Math.round(4.6) // 5  
Math.round(4.4) // 4
```

Math.floor(x) : Bu metot, x'den küçük olan en büyük tam sayıyı döndürür.

```
Math.floor(7.9) // 7
```

Math.ceil(x) : Bu metot, x'den büyük olan en en küçük tam sayıyı döndürür.

```
Math.ceil(2.1) // 3
```

Math.random() : Bu metod, 0 ile 1 arasında rasgele bir sayı döndürür.

```
Math.random() // 0.743209342
```

Math.min(a,b,c,...) : Bu metod, verilen sayılardan en küçük olanı döndürür.

```
Math.min(4,8,2,6) // 2
```

Math.max(a,b,c,...) : Bu metod, verilen sayılardan en büyük olanı döndürür.

```
Math.ceil(2.1) // 3
```

JavaScript Boolean(True/False)

- JavaScript Boolean, true veya false değerlerini içerebilen bir veri türüdür.
- Boolean değerleri, koşulların veya ifadelerin doğru olup olmadığını belirlemek için kullanılır. Örneğin, bir değişkenin boş olup olmadığını belirlemek için == null, undefined veya "" kullanılabilir ve bu durum true veya false olarak döndürür. Aynı şekilde bir sayının pozitif olup olmadığını belirlemek için > 0 kullanılır.

```
let doğru = true;  
let yanlış = false;
```

JavaScript Boolean(True/False)

- JavaScript Boolean, true veya false değerlerini içerebilen bir veri türüdür.
- Boolean değerleri, koşulların veya ifadelerin doğru olup olmadığını belirlemek için kullanılır. Örneğin, bir değişkenin boş olup olmadığını belirlemek için == null, undefined veya "" kullanılabilir ve bu durum true veya false olarak döndürür. Aynı şekilde bir sayının pozitif olup olmadığını belirlemek için > 0 kullanılır.

```
let dogru = true;  
let yanlis = false;
```

JavaScript'te, **birçok** işlem veya fonksiyon Boolean değerler döndürebilir.

Örneğin, bir ifadelin doğruluğunu test etmek için kullanılan `"=="` ve `"!="` operatörleri Boolean değer döndürür.

Ayrıca, bir değişkenin tanımlı olup olmadığını kontrol etmek için kullanılan `typeof` operatorü Boolean değer döndürür. Örnek:

```
console.log(5 > 3); // true  
console.log(typeof variable !== 'undefined'); // true
```

JavaScript If Else (Koşullar)

JavaScript **if else**, bir koşuluun **gerçekleşip gerçekleşmediğine** göre kod bloğunuun **çalışmasını denetleyen** bir yapıdır. Eğer koşul **true** ise, "**if**" bloğu çalışır, aksi halde "**else**" bloğu çalışır. Örneğin, bir sayının pozitif veya negatif olduğunu belirlemek için **if else** kullanılabılır:

```
let sayı = -5;
if (sayı >= 0) {
    console.log("sayı pozitiftir");
} else {
    console.log("sayı negatiftir");
}
```

Ve birden çok **else if** bloğu eklenerek daha kompleks karar mekanizmaları oluşturulabilir. Örnek:

```
let sayı = 5;  
if (sayı > 0) {  
    console.log("sayı pozitiftir");  
} else if (sayı < 0) {  
    console.log("sayı negatiftir");  
} else {  
    console.log("sayı sıfırdır");  
}
```

JavaScript Switch

JavaScript switch, bir değişkenin **birden fazla** değer arasında eşleştirilmesini **yapmak** için kullanılan bir yapıdır. Switch yapısı, **if else** yapısına **benzer** şekilde çalışır ancak daha okunaklı ve kolay anlaşılır olması açısından tercih edilir.

Switch yapısı, bir değişkenin **değeri** ile **eşleşen case** bloğunu **çalıştırır** ve eşleşme olmazsa **default** bloğu çalıştırır. Örneğin, bir kullanıcının üyelik durumunu belirtmek için switch case yapısı kullanılabilir:

Üyelik değişkeni "temel" veya "premium" olduğunda ilgili `case` bloğu çalışır. Eğer üyelik değişkeni başka bir değer içeriyorsa `default` bloğu çalışır.

Switch yapısı, daha fazla sayıda eşleşme yaparken daha etkilidir. Ayrıca, switch yapısı `if else` yapısına göre daha hızlı çalışır.

```
let uyelik = "premium";
switch (uyelik) {
  case "temel":
    console.log("Temel üyelik");
    break;
  case "premium":
    console.log("Premium üyelik");
    break;
  default:
    console.log("Geçersiz üyelik");
}
```

Break ve Default Ne işe yarar?

JavaScript'de switch yapısında kullanılan **break** anahtar kelimesi, eşleşen case bloğunda yer alan tüm kodların çalıştırılmasını **durdurur**. **break** anahtar kelimesi kullanılmadan yazılırsa, switch yapısı eşleşen case bloğundan sonraki **tüm case** bloklarının kodlarını da **çalıştırır**. Bu yüzden, switch yapısının kullanılması sırasında her case bloğunda **mutlaka** **break** anahtar kelimesi kullanılmalıdır.

JavaScript'de switch yapısında kullanılan **default** bloğu, eşleşen case bloğu **bulunamazsa** çalıştırılan kod bloğudur. default bloğu, switch yapısının **en sonunda** yer almalıdır ve **sadece bir** tane olmalıdır.

JavaScript For Loop (Döngü)

JavaScript for **loop** (döngüsü) , belirli bir koşula göre belirli bir kod bloğunu tekrar edilmesini sağlar. for loop yapısı, döngü sayacının başlangıç, bitiş ve artım değerlerini tanımlayan bir yapıdır.

Bu örnekte, döngü sayacı **i** başlangıç değeri **1** ile başlar, **10'a** kadar devam eder ve **her** döngüde **1 artar**. Bu sayede **1'den 10'a** kadar olan tüm sayılar yazdırılır.

```
for (let i = 1; i <= 10 ; i++) {  
    console.log(i);  
}
```

Örnek olarak, bir dizinin tüm elemanlarını yazdırmak için kullanılır:

```
let isimler = ['Ahmet', 'Mehmet', 'Osman'];

for (let i = 0; i < isimler.length; i++) {
    console.log(isimler[i]);
}
```

Bu örnekte, döngü sayacı `i` başlangıç değeri `0` ile başlar, isimler dizinin `uzunluğu` kadar devam eder ve `her` döngüde `1` artar. Bu sayede isimler dizinin tüm elemanları yazdırılır.

For In Loop Nedir ?

JavaScript for in loop (döngüsü) , bir nesnenin(object) tüm özelliklerini dolaşmak için kullanılır. Örnek olarak, bir objenin tüm özelliklerini ve değerlerini yazdırmak için kullanılabilir.

```
let kullanici = {  
    isim: "Furkan",  
    yas: 20,  
    email: "furkan@ornek.com"  
};  
  
for (let anahtar in kullanici) {  
    console.log(anahtar + ":" + kullanici[anahtar]);  
}
```

Yandaki örneğin çıktısı:

```
isim: Furkan  
yas: 20  
email: furkan@ornek.com
```

For Of Loop Nedir ?

JavaScript for of loop, dizi veya iterable objeler üzerinde dolaşmak için kullanılır. Bu loop, dizi elemanlarının değerlerini tek tek alır ve işlem yapar. Örnek olarak, bir dizinin tüm elemanlarını yazdırırmak için kullanılabilir:

```
let meyveler = ["elma", "armut", "muz"];
for (let meyve of meyveler) {
  console.log(meyve);
}
```

Bu örnekte, for of loop kullanılarak meyveler dizisi içindeki tüm elemanlar dolaşılır ve eleman değerleri yazdırılır.

JavaScript While Loop

JavaScript while loop (döngüsü), belirli bir koşula göre belirli bir kod bloğunun tekrar edilmesini sağlar. while loop yapısı, koşulun doğru olduğu sürece kod bloğunu tekrar eder. Ancak, dikkatli kullanılması gerekir çünkü koşul asla false olamazsa, döngü sonsuz olarak çalışmaya devam edebilir.

```
let i = 1;  
while (i <= 10) {  
    console.log(i);  
    i++;  
}
```

Bu örnekte, `while loop` kullanılarak döngü sayacı `i` **başlangıç** değeri 1 ile başlar ve `10'a kadar` devam eder.
Her döngüde sayacın değeri **1 artırılır** ve sayacın değeri `10'a` ulaştığında döngü **sona erer**.

Do While Loop Nedir ?

JavaScript **do while loop** (döngüsü), while loop yapısına benzer şekilde belirli bir koşula göre belirli bir kod bloğunun tekrar edilmesini sağlar. Ancak, do while loop yapısında ilk önce kod bloğu çalıştırılır ve daha sonra koşul kontrol edilir. Yani, do while loop yapısında koşul doğru olduğu sürece kod bloğu en az bir kere çalıştırılır. Örnek olarak, kullanıcı girişini almak için kullanılabilir:

```
let giriş;
do {
    giriş = prompt("10'dan büyük bir sayı girin:");
} while (giriş <= 10);
```

Bu örnekte, do while loop kullanılarak kullanıcıdan bir sayı girişi alınır. Eğer girilen sayı 10'dan küçükse, kullanıcı tekrar bir sayı girene kadar döngü devam eder.

JavaScript Try Catch

JavaScript try and catch yapısı, kodunuzda oluşabilecek potansiyel hata veya exceptionları (istisnalar) yakalamak için kullanılan bir yapıdır. `try` bloğu içerisinde olası bir hata oluşabilecek kod yer alır. Eğer hata oluşursa, `catch` bloğu içerisindeki kod çalışır. Bu sayede, hata oluşması durumunda programın çalışmasının **durdurulmaması** ve hata ile nasıl başa çıkacağımızın yollarının belirlenmesi sağlanmış olur. Örnek olarak:

```
try {  
    let x = y + 1; // y değerini tanımlamadık  
} catch (error) {  
    console.log('Hata: ' + error);  
}
```

Bu örnekte, try bloğu içerisinde **y** değişkeni tanımlanmamış olduğundan **hata** oluşur ve **catch** bloğu içerisindeki kod çalışır. Bu sayede, hata ile karşılaşlığımızda ne yapacağımız belirlenmiş olur.

try and catch yapısı, hata oluşması durumunda programın çalışmasını **durdurulmasını**, hata ile nasıl başa çıkacağımızın yollarının **belirlenmesini** ve **hata mesajlarının loglanması** sağlar. Ayrıca, catch bloğu içerisinde hata ile ilgili **detaylı bilgileri** elde etmek için **error nesnesi** kullanılabilir. Örneğin, hata mesajını, hata tipini veya hata oluştugu satırı gibi bilgileri içerebilir.

Değişkenlerde Scope

JavaScript scope, bir değişkenin erişilebilirliği olarak tanımlanabilir.

Scope, bir değişkenin hangi kod bloğunda tanımlı olduğuna ve hangi kod bloğunda erişilebildiğine bağlıdır. JavaScript'de, iki tür scope vardır: global scope ve local scope.

Değişken bir fonksiyonun içinde tanımlanırsa Local Scope olur eğer fonksiyonun dışında tanımlanırsa Global Scope olur.

Global scope:

```
let x = 10; // Global scope içinde tanımlanmış bir değişken  
  
function myFunction() {  
    console.log(x); // Erişilebilir  
  
    console.log(x); // Erişilebilir  
}
```

Bu örnekte, x global scope içinde tanımlanmış bir değişken olduğu için, fonksiyon içinde veya dışında erişilebilir.

Local scope:

```
function myFunction() {  
    let y = 20; // Local scope içinde tanımlanmış bir değişken  
    console.log(y); // Erişilebilir  
}  
  
console.log(y); // ReferenceError: y is not defined
```

Bu örnekte, `y` local scope içinde tanımlanmış bir değişken olduğu için, `sadece myFunction` fonksiyonu içinde erişilebilir. `Dışarıda` erişmeye çalışırsak `hata` alırız.

let const var arasındaki scope farkları:

var ile tanımlanan değişkenler global veya fonksiyon çevresi içinde tanımlanır ve bu çevreler dışında erişilemez. Örneğin:

```
var x = 10; // Global scope içinde tanımlanmış bir değişken
if (true) {
  var x = 20; // x değişkeni değiştirilir
}
console.log(x); // 20
```

Bu örnekte, x değişkeni global scope içinde tanımlanmıştır ve if bloğu içinde değiştirilir.

`let` ve `const` ile tanımlanan değişkenler ise sadece fonksiyonlar veya **global scope** içinde değer alırlar, ancak **block scope**(`if` , `for` , `while` bloğu)'da değer almazlar. Örneğin:

```
let y = 10; // Global scope içinde tanımlanmış bir değişken
if (true) {
  let y = 20; // y değişkeni if bloğu içinde tanımlanmıştır
}
console.log(y); // 10
```

Bu örnekte, `y` değişkeni **global scope** içinde tanımlanmıştır ancak `if` bloğu içinde değiştirilir ve `if` bloğu dışında değişmez.

`const` ile tanımlanan değişkenler ise `sadece tanımlanırken` değer alırlar ve daha sonra `değiştirilemezler`. Örneğin:

```
const z = 10; // Global scope içinde tanımlanmış bir değişken  
z = 20; // TypeError: Assignment to constant variable.
```

Bu örnekte, `z` değişkeni `global scope` içinde `tanımlanmıştır` ancak daha sonra `değiştirilemez`.

Açıkça görüldüğü gibi, `var`, `let` ve `const` ile tanımlanan değişkenler arasında `scope` farkları vardır ve kullanım amacına göre kullanılmalı

JavaScript “**use strict**”

JavaScript Use Strict katı mod anlamına gelmektedir. JS'nin katı modda yürütülmesini sağlar, bu da kodlama uygulamalarında olası **hatalara** yol açabilecek yaygın hataların belirlenmesine yardımcı olur. Use Strict kullanıldığında hatalı işlemlere **anında müdahale** edilir ve hata yapma olasılığınızı **en aza** indirir. temiz bir kod yazmak için geliştiriciler Use Strict kullanır.

“**use strict**” sayfanın en başında yer alması gerekir üstüne herhangi bir kod yazılmaz.

Strict mode içinde, tanımsız değişkenlerin kullanılmasına izin verilmez.

```
"use strict";  
x = 10; // ReferenceError: x is not defined
```

Strict mode içinde, yinelelenen parametrelerde izin verilmez

```
"use strict";  
function Sum(val, val) { // normal şartlarda hata vermezken  
    return val + val; // katı modda hata verir  
}
```

Bunlar strict modun zorunluluqlarından sadece bir kaçındı daha fazlası için araştırılabilirsiniz.

Arrow Function

JavaScript'te Arrow Function (Ok Fonksiyonları), fonksiyon tanımlama işlemini daha kısa ve okunaklı hale getirmek için kullanılan bir yöntemdir.

Bu fonksiyonlar, function anahtar kelimesi yerine => işaretini kullanılarak tanımlanır. Örneğin:

```
let myFunction = function (param) {  
    return param;  
};
```

Yukarıdaki fonksiyon
Arrow function ile yandaki
Şekilde yazılabilir.

```
let myFunction = (param) => {  
    return param;  
};
```

Ayrıca, tek bir döndürme ifadesi olan fonksiyonlar için bu kullanım daha da
kısaltılabilir. Aşağıdaki şekilde kullanırsak yazılan satırı direkt return eder

```
let myFunction = () => { "Hello World!" }
```

Arrow function kullanımı ile kodun okunabilirliği ve anlaşılabilirliği arttırlar,
yazımı kolaylaştırırlar ve kodun boyutu azaltılır.

JSON (Javascript Object Notation)

JavaScript Object Notation (JSON) bir veri aktarma biçimidir. JavaScript içinde, JSON, nesneler, diziler ve diğer veri türlerini serileştirmek (yazmak) veya okumak için kullanılır. JSON, veri alışverişi için popüler bir seçenekdir çünkü veriyi **basit** ve anlaşılır bir şekilde ifade eder ve çok sayıda programlama dili tarafından desteklenir.

JSON, veriyi **iki** temel yapıda ifade eder: nesneler ve diziler.

Bir nesne, **anahtar-değer** çiftlerinden oluşur. Örneğin:

JSON, JavaScript içinde `JSON.stringify()` fonksiyonu ile string ifade `object` çevrilir. `JSON.parse()` fonksiyonu ile string ifade `object` çevrilir.

```
let ÖrnekObje = { "name": "John", "age": 30 };
let myJSON = JSON.stringify(ÖrnekObje);
console.log(myJSON); // {"name":"John","age":30}

let myParsedJSON = JSON.parse(myJSON);
console.log(myParsedJSON); // { name: "John", age: 30 }
```

JSON, web uygulamaları, mobil uygulamalar ve API'ler arasında veri alışverişinde sıkılıkla kullanılır. Çünkü hem sunucu hem de istemci tarafında kolayca işlenebilir ve anlaşılabılır bir şekilde veri gönderip alınabilir.

ASYNC(Asenkron)

JavaScript içinde **Async** (Asenkron), **birden fazla işlemi aynı anda yürütmek** için kullanılan bir yöntemdir. Asenkron işlemler, işlemlerin gerçekleştirilmesi sırasında diğer işlemlere engel **olmamasını** sağlar. Örneğin, bir web sayfasının yüklenmesi sırasında arka planda bir veri sunucusundan **veri alınması** gibi.

JavaScript içinde asenkron işlemler, **callback fonksiyonları**, **promise** ve **async/await** gibi yollarla gerçekleştirilebilir.

Callback fonksiyonları: Bir işlem tamamlandığında çağrılabilecek olan bir fonksiyonu tanımlar. Örneğin:

```
function getData(callback) {  
    setTimeout(() => {  
        let data = 'Hello World!';  
        callback(data);  
    }, 2000);  
}  
  
getData(function (data) {  
    console.log(data);  
});
```

Bu örnekte, `getData` fonksiyonu `setTimeout` sayesinde 2 saniye sonra bir veri döndürür ve dönen veri `console.log` ile yazdırılır.

`setTimeout` yapılması istenen işlemlerin belirlenildiğiimiz süre sonrasında yapmaya başlanması sağlanır.

Promise: Promise, JavaScript'te asenkron işlemler için bir yol sunar.

Promise, asenkron işlemleri daha okunaklı ve anlaşılır hale getirmek için kullanılır.

Bir Promise objesi, bir işlemin gelecekte gerçekleşeceği veya gerçekleşmeyebileceği sonucunu temsil eder.

Bu objenin içinde, `resolve` ve `reject` fonksiyonları yer alır. `resolve` fonksiyonu, işlemin **başarılı** olması durumunda çağrırlır ve işlem sonucunu döndürür. `reject` fonksiyonu ise, işlemin **başarisız** olması durumunda çağrırlır ve hata mesajını döndürür.

Promise objesini kullanmak için `.then` ve `.catch` metodlarını kullanabilirsiniz.

Örneğin:

```
let promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve('Veri sunucudan çekildi');
  }, 2000);
});
```

```
promise.then((data) => console.log(data))
  .catch((error) => console.log('Hata: ' + error));
```

Bu örnekte, bir `promise` objesi oluşturuldu ve 2 saniye sonra `resolve` fonksiyonu çağırılarak "Veri sunucudan çekildi" mesajı döndürdü. `.then` metodu ile bu mesaj ekranaya yazdırılır.

Promise, asenkron işlemlerin yürütülmesinde JavaScript'te **önemli** bir kavramdır ve JavaScript programlamasında sıkça kullanılır. Bu sayede programın devam etmesini beklemeden **arka planda** işlemler yapabilir ve işlemlerinin sonucunu alırken daha okunaklı ve anlaşılır bir kod yazabiliyorsınız.

Async/Await: JavaScript içinde asenkron işlemleri daha okunaklı ve anlaşılır hale getirmek için kullanılan bir yöntemdir. Async/await, promise'lerin yerine kullanılabilir ve işlemlerin gerçekleştirilmesi sırasında programın ara vermemesini sağlar. Örneğin:

```
async function getData() {  
  let data = await fetch('https://example.com/data.json');  
  return data;  
}  
  
getData()  
.then((data) => console.log(data))  
.catch((error) => console.log(error));
```

Bu örnekte, `getData` fonksiyonu bir web adresinden veri alır ve `console.log` ile yazdırır.

`Async/await` yapısı, JavaScript içinde asenkron işlemlerin yazımını daha okunaklı ve anlaşılır hale getirmekte, ayrıca kodun okunabilirliğini artırmaktadır.

Not: `Async/await` kullanırken, işlemleri gerçekleştirmek için bir `promise` nesnesi oluşturmanız gereklidir.

DOM (Document Object Model)

JavaScript içinde Document Object Model (DOM), bir web sayfasının HTML veya XML dokümanının yapısını ve içeriğini ifade etmek için kullanılan bir **modeldir**.

DOM, JavaScript tarafından kullanılarak web sayfasının **dinamik** olarak **değiştirilmesi** ve **yönetilmesi** sağlar.

DOM, bir web sayfasının HTML dokümanını, bir ağaç yapısı olarak ifade eder. Bu ağaç yapısı, web sayfasının HTML **etiketlerini**, **özelliklerini** ve **İçeriğini** temsil eder. Örneğin:

Bu HTML dokümanı, aşağıdaki DOM ağacını oluşturur:

- Document
- html
- head
- title (Blogum)
- body
- h1 (Sayfama Hoşgeldiniz)
- p (Paragraf alanı)

JavaScript içinde DOM, document nesnesi aracılığıyla erişilir.

```
let baslikAlani = document.querySelector("h1");
```

DOM Methodları:

JavaScript içinde DOM, birçok metod içermektedir. Bu metodlar, web sayfasının HTML dokümanının yapısını ve içeriğini değiştirmek için kullanılır. Aşağıda bazı önemli DOM metodları ve örnekleri verilmiştir:

getElementById(id): Belirli bir ID'ye sahip bir elementi döndürür. Örneğin:

```
<div id="myDiv">Burası bir div alanı</div>
<script>
let myDiv = document.getElementById('myDiv');
console.log(myDiv.innerText); // Burası bir div alanı
</script>
```

querySelector(selector): Elementi cssteki seçicisine göre seçer ve döndürür.

```
<h1 class="baslik-alani">Sayfama Hoşgeldiniz</h1>  
- javascript -  
let baslik = document.querySelector('.baslik-alani');  
console.log(baslik.innerText); // Sayfama Hoşgeldiniz
```

querySelectorAll(selector): Belirli bir seçiciye sahip tüm elementleri döndürür.

```
<p>Paragraf alanı </p> <p>Paragraf alanı 2</p>  
- javascript -  
let paragraflar = document.querySelectorAll('p');  
paragraflar.forEach((p) => { console.log(p.textContent); });  
// Paragraf alanı Paragraf alanı 2
```

innerText: Bu metod, elementin içeriğini döndürür veya değiştirir. İçerik, HTML etiketleri olmadan döndürülür yanı sadece seçilen elementin içindedi yazılar döner.

```
<div id="kutucuk">
  <p>Bu bir deneme yazısıdır</p>
</div>
- javascript -
let kutucuk = document.getElementById('kutucuk');

console.log(kutucuk.innerText); // Bu bir deneme yazısıdır

kutucuk.innerText = "Artık kutunun içinde bu yazıcak"; // içeriğini değiştirdik

console.log(kutucuk.innerText); // Artık kutunun içinde bu yazıcak
```

innerHTML: Elementin içeriğini döndürür veya değiştirir. İçerik, HTML etiketleri birlikte döndürülür ve içerik değiştirilirken yine html etiketleri kullanılır. Örneğin:

```
<div id="kutucuk">  
  <p>Yeni bir <span>PARAGRAF</span> alanı</p>  
</div>  
  
let yeniKutu = document.getElementById('kutu');  
  
console.log(yeniKutu.innerHTML); // <p>Yeni bir <span>PARAGRAF</span> alanı</p>  
  
yeniKutu.innerHTML = '<p>Bundan sonra <span> içerik </span> böyle olacak </p>';  
  
console.log(yeniKutu.innerHTML); //<p>Bundan sonra <span> içerik</span> böyle olacak</p>
```

createElement(Etiket İsmi): Belirttiğimiz HTML etiketini oluşturur.

```
<div id="kutucuk"></div>
-
javascript -
let kutucuk = document.getElementById('kutucuk');
let yeniP = document.createElement('p'); // Yeni bir p elementi oluşturuldu
yeniP.innerText = 'Paragraf etiketi eklendi'; // P elementinin içeriği güncellendi
kutucuk.appendChild(yeniP); // P elementi divin içine eklendi
```

appendChild(element): Seçtiğiniz elementin içine yeni bir element ekler

```
let liste = document.getElementById("liste");
let yeniSatir = document.createElement("li"); // Yeni li elementi oluşturuldu
yeniSatir.innerText = "Yeni liste elemani"; // li nin içeriği güncellendi
liste.appendChild(yeniSatir); // yeni liste elemansı listenin içine eklendi
```

classList: Bu metod, elementin **class** özelliğine ait tüm değerleri **DOMTokenList** adında bir **obje** olarak döndürür. Bu objede **add**, **remove**, **toggle** gibi metodlar mevcut ve classların ekleme, çıkarılması veya toggling işlemleri yapabilirsiniz.

- javascript -

```
<div id="kutucuk" class="bilgi-kutusu kirmizi"></div>
```

```
let kutucuk = document.getElementById('kutucuk');

console.log(kutucuk.classList); // " kirmizi bilgi-kutusu"

kutucuk.classList.add('aktif'); // " aktif kirmizi bilgi-kutusu"

kutucuk.classList.remove('kirmizi'); // " aktif bilgi-kutusu "
```

style: Bu metod, elementin **style** özelliklerine ait tüm değerleri bir

CSSStyleDeclaration objesi olarak döndürür. Bu objede **backgroundColor**, **color**, **fontSize** ve diğer css özelliklerine erişip atama yapabilirsiniz. Örneğin:

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

- javascript -

```
let kutu = document.getElementById('kutu'); //kutuyu seçtik
```

```
kutu.style.color = 'black'; // yazılar siyah olarak ayarlandı
```

```
kutu.style.backgroundColor = 'red'; //arkaplan kırmızı oldu
```

DOM Methodlarını kullanırken dikkat edilmesi gereken birkaç şey vardır:

- * DOM elementleri yalnızca sayfa yüklenliğinde erişilebilir. Ve DOM manipülasyonlarını en **az** oranda yapmalısınız. Çok fazla DOM manipülasyonu **performansı düşürebilir**. Özellikle döngüler içinde yapılan manipülasyonlar sayfa yüklenme süresini uzatabilir.
- * Belirli bir elementi aratırken seçicileri **doğru** olarak kullanmalısınız. Örneğin, **getElementById** metodu yalnızca **ID'ye** sahip elementleri arar, **querySelector** metodu ise **CSS seçicileri** kullanarak elementleri arar.
- * Metodlar arasındaki farkları iyi bilmelisiniz. Örneğin, **getElementById** metodu **yalnızca ilk** bulunan elementi döndürürken, **querySelectorAll** metodu **tüm** elementleri döndürür.

BOM (Browser Object Model)

JavaScript BOM (Browser Object Model) tarayıcının öğelerini temsil eden JavaScript objelerini içerir. BOM, JavaScript ile tarayıcının penceresi, ekranı, uyarıları ve diğer özellikleri kontrol etmenizi sağlar. BOM, JavaScript ile tarayıcının davranışını değiştirebilmenizi ve kullanıcının etkileşimi yönetebilmenizi sağlar. Örnek olarak, BOM, kullanıcının pencereyi kapatmasını engelleyebilir veya pencere boyutunu değiştirebilir. BOM, Window objesini temsil eder ve tüm global değişkenlerin altında yer alır. Örnek olarak, navigator, history, location, screen, document, gibi objeler BOM içinde yer alır.

window : objesi, tarayıcı penceresini temsil eder ve BOM (Browser Object Model) içinde yer alır. JavaScript ile tarayıcının penceresi, ekranı, uyarıları ve diğer özelliklerini kontrol etmenizi sağlar.

Window objesi, birçok metod ve özellik içerir, bunlar arasında en yaygın olanlar:

open(): Yeni bir pencere veya sekme açar.

close(): Açıktı olan pencereyi kapatır.

alert(): Kullanıcıya uyarı mesajı gösterir.

prompt(): Kullanıcıdan veri almak için bir mesaj gösterir.

confirm(): Kullanıcıdan "evet" veya "hayır" seçenekleri için bir mesaj gösterir.

scrollTo(): Sayfa içerisinde belirli bir konuma kaydırır.

resizeTo(): Pencere boyutunu değiştirir.

setTimeout(): Belirli bir zaman sonra bir fonksiyonu çalıştırırmak için kullanılır.

clearTimeout(): setTimeout() ile belirlenen bir zamanı iptal etmek için kullanılır.

Ayrıca window objesi **document** objesine erişebilir, **location** objesine erişebilir, **history** objesine erişebilir, **screen** objesine erişebilir vb. Örnek olarak, **window.location** objesi, kullanıcının şu anda ziyaret ettiği web sayfasının **URL'sini** içerir ve **window.history** objesi, kullanıcının tarayıcı geçmişini içerir.

pop up mesajlarının kullanımı: alert(), confirm() ve prompt()

alert(): Bu method kullanıcıya bir pop-up mesaj gösterir ve kullanıcı OK tuşuna basana kadar bekler.

```
alert("Merhaba, bu bir alert mesajıdır.");
```

prompt(): Kullanıcıya bir pop-up mesaj gösterir ve kullanıcı bir değer girip "OK" veya "Cancel" tuşuna basana kadar bekler. Kullanıcı "OK" tuşuna bastığında girilen değer, "Cancel" tuşuna bastığında ise null döner. Örnek:

```
let ad = prompt("Adınız nedir?");  
console.log(`Merhaba ${ad}`);
```

confirm(): Kullanıcıya bir pop-up mesaj gösterir ve kullanıcı "OK" veya "Cancel" tuşuna basana kadar bekler. Kullanıcı "OK" tuşuna bastığında true, "Cancel" tuşuna bastığında false döner. Örnek:

```
let kabulEt = confirm("Bu mesajı kabul ediyor musunuz?");  
if(kabulEt) {  
    console.log("Kabul edildi"); //kullanıcı ok tuşuna basarsa burası çalışır  
} else {  
    console.log("Reddedildi"); //cancel tuşuna basarsa burası çalışır  
}
```

Bu methodlar ile kullanıcıya bilgi, onay veya girdi almak için kullanılabilir. Ancak, bu methodların kullanımını eskimiş olup ve kullanıcı deneyimini **olumsuz** etkileyebileceğinden, mümkün olduğunda **kullanmamak** daha iyidir. Bunun yerine, web sayfasının içeriği ile **interaktif** olmak için JavaScript veya diğer teknolojileri kullanabilirsiniz.

window.screen: objesi, kullanıcının ekranının özelliklerini temsil eder ve BOM (Browser Object Model) içinde yer alır. screen objesi, kullanıcının ekranının boyutlarını, çözünürlüğünü, renk derinliğini ve diğer özelliklerini içerir.

Window.screen objesi, birçok metod ve özellik içerir, bunlar arasında en yaygın olanlar:

screen.width: Ekranın genişliğini piksel cinsinden döndürür.

screen.height: Ekranın yüksekliğini piksel cinsinden döndürür.

Öğrenilen bu bilgilerle yazılım, ekran özelliklerine göre daha uygun görüntüler sağlamak için kullanılabilir.

window.location : Kullanıcının gezinmekte olduğu web sayfasının URL bilgilerini temsil eder. location objesi, web sayfasının URL'sini, protokolünü, hostname'ını, pathname'ını, search parametrelerini ve diğer bilgileri içerir.

window.location objesi, birçok metod ve özellik içerir, bunlar arasında en yaygın olanlar:

location.href: Web sayfasının tam URL'sini döndürür.

location.reload(): Web sayfasını yeniden yükler.

location.assign(url): Web sayfasını belirtilen URL'ye yönlendirir.

location.replace(url): Web sayfasını belirtilen URL'ye yönlendirir ve geçmişteki sayfayı siler.

window.history : objesi, kullanıcının tarayıcı geçmişini temsil eder.

history objesi, kullanıcının ziyaret ettiği web sayfalarının adreslerini, geçmiş(Kullanıcı hakları gereği bazı sınırlar vardır) ve ileriyi yönetmenize olanak tanır.

window.history objesi, birçok metod ve özellik içerir, bunlar arasında en yaygın olanlar:

history.back(): Tarayıcı bir önceki sayfaya geri yönlendirir.

history.forward(): Tarayıcı bir sonraki sayfaya ilerletir.

history.go(n): Tarayıcının n adım sonraki/önceki sayfasına yönlendirir.

history.pushState(state, title, url): Tarayıcı geçmişine yeni bir kayıt ekler.

window.navigator : Kullanıcının tarayıcı ve cihaz bilgilerini temsil eder.
navigator objesi, tarayıcının adını, sürümünü, cihaz türünü, işletim sistemini, dili ve diğer bilgileri içerir.
window.navigator objesi, birçok metod ve özellik içerir, bunlar arasında en yaygın olanlar:

navigator appName(): Tarayıcının adını döndürür.

navigator appVersion(): Tarayıcının sürümünü döndürür.

navigator language(): Kullanıcının tarayıcı dili döndürür.

navigator onLine(): Tarayıcının internet bağlantısının açık olup olmadığını döndürür.

Javascript API

JavaScript API (Application Programming Interface), bir uygulamanın veya sistemin diğer uygulamalar veya sistemlerle nasıl **etkileşime** girebileceğini tanımlayan bir **arayüzüdür**. JavaScript API'leri, web tarayıcılarının veya JavaScript Kütüphanelerinin içinde yer alır ve birçok fonksiyon ve **metodlar** içerirler.

JavaScript API'leri genellikle aşağıdaki işlevleri yerine getirir:

Veri erişimi: Örneğin, bir web sayfasındaki verileri okuma, ekleme veya silme.

Geolocation: Kullanıcının konum bilgisi alma ve kullanma.

Geolocation API

JavaScript Geolocation API, web tarayıcılarının kullanıcının konum bilgisini almasını ve kullanmasını sağlar. Bu API, navigator.geolocation objesini kullanarak çalışır. Bu objenin içinde, kullanıcının konum bilgisini almak için getCurrentPosition() ve watchPosition() gibi metodlar yer alır.

getCurrentPosition() metodu, kullanıcının anlık konum bilgisini almak için kullanılır. Bu metod, **iki** parametre alır: **success** callback ve **error** callback. Success callback, kullanıcının konum bilgisi **alındığında** çalışacak fonksiyonu ve error callback ise kullanıcının konum bilgisi **alinamadığında** çalışacak fonksiyonu temsil eder.

watchPosition() metodu ise, kullanıcının konum bilgisinde meydana gelen değişiklikleri **izlemek** için kullanılır. Bu metod da iki parametre alır: success callback ve error callback.

```
navigator.geolocation.getCurrentPosition(konumuGoster);  
  
function konumuGoster(konum) {  
    console.log('Enlem: ' + konum.coords.latitude);  
    console.log('Boylam: ' + konum.coords.longitude);  
}
```

navigator.geolocation objesi kullanılarak kullanıcının **anlık** konum bilgisi alınmıştır. **getCurrentPosition** metodу ile alınan bilgi **konumuGoster** fonksiyonuna gönderilmiştir. Bu fonksiyon içinde kullanıcının **Enlem** ve **Boylam** bilgisi ekranaya yazdırılmıştır.

Webstorage API

JavaScript Web Storage API, web tarayıcılarının kullanıcının tarayıcıda saklamasını sağlar. Bu API, iki temel saklama seçeneği sunar:

sessionStorage: Bu API, verileri ancak tarayıcı penceresi açıkken saklar. Tarayıcı kapatıldığında veya sekme kapatıldığında veriler silinir.

localStorage: Bu API, verileri tarayıcı kapatılana kadar saklar.

Web storage API, `setItem()` ve `getItem()` metodlarını kullanarak verileri saklar ve okur. `setItem()` metodu, veriyi saklamak için kullanılır ve **iki** parametre alır: **anahtar** ve **değer**. `getItem()` metodu ise, saklanmış veriyi okumak için kullanılır ve sadece anahtar parametresi alır.

```
localStorage.setItem('isim', 'Serra');
let kalıcıisim = localStorage.getItem('isim'); // Serra
localStorage.removeItem('isim') // isim kaldırıldı
//sessionstorage
sessionStorage.setItem('isim', 'Bilal');
let geçiciisim = sessionStorage.getItem("isim"); // Bilal
```

Bu örnekte, localStorage ve sessionStorage kullanılarak veri saklama ve okuma işlemleri gerçekleştirilmektedir. setItem metodу ile "isim" anahtarına "John Doe" değeri atanmıştır ve getItem metodу ile aynı anahtara sahip veri okunmuştur.

Web Storage API, kullanıcının bilgisayarında veri saklamak istediğiniz durumlarda kullanabileceğiniz önemli bir araçtır.

Fetch API

JavaScript Fetch API, web tarayıcılarının veya JavaScript kodlarının, bir web sunucusundan veri almasını sağlar. Bu veriler, genellikle **JSON** formatında olur ve JavaScript kodları tarafından kolayca kullanılabilir hale getirilir. Fetch API, **XMLHttpRequest** veya **jQuery ajax** gibi eski araçların yerine kullanılabilir ve kodunuza daha okunaklı ve anlaşılır hale getirir.

Fetch API, **fetch()** fonksiyonunu kullanarak çalışır. Bu fonksiyon, bir URL'yi parametre olarak alır ve bu URL'den **veri** almaya çalışır. Fetch işlemi, asenkron bir işlemidir ve **fetch()** fonksiyonu, bir **promise** objesi döndürür. Bu objenin içinde, verilerin okunabilir bir şekilde alınması için **json()** veya **text()** gibi metodlar yer alır.

```
fetch('https://jsonplaceholder.typicode.com/posts')
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.log('Hata: ' + error));
```

Bu örnekte, `fetch()` fonksiyonu ile `jsonplaceholder.typicode.com` adresinden veri alınmaya çalışılıyor.

Dönen veri alma işlemi başarılı olursa promise objesinde `then()` metodу ile response objesinin içindeki `json` verisi `okunuyor` ve konsola yazdırılıyor. Eğer işlem başarısız olursa konsola hata yazdırılıyor.

JavaScript, modern web uygulamalarının vazgeçilmez bir parçasıdır. Bu dökümanda, JavaScript'in temel kavramlarından, kontrol yapılarına, fonksiyonlar ve nesnelerin kullanımına, web tarayıcılarının JavaScript API'lerine ve modern JavaScript yazılım geliştirme teknolojilerine kadar birçok konu ele alınmıştır.

JavaScript, dinamik ve esnek bir dildir ve web uygulamalarınızda kullanabileceğiniz birçok fonksiyon ve özellik sunar. Dökümanda öğrendiğiniz bilgileri практике uygulayarak, güçlü ve etkili web uygulamaları oluşturabilirsiniz.

Ancak JavaScript öğrenmenin sadece bu dökümanda yer alan bilgilerle sınırlı olmadığını unutmayın. JavaScript dünyası hızla gelişmekte ve sürekli olarak yeni özellikler ve teknolojiler eklenmektedir. Bu nedenle, JavaScript ve web geliştirme ile ilgili sürekli olarak öğrenmeye ve araştırmaya devam etmeniz gerekmektedir.