JavaScript, web development'da kullanıcı deneyimini artıran önemli bir programlama dilidir. Bu video, JavaScript'in temel kavramlarını ve kullanımını açıklamaktadır.

JavaScript'in Temel Özellikleri

- **DOM Manipulation**: JavaScript, web sayfasının içeriğini ve yapısını değiştirebilir, bu da etkileşimli ve dinamik web siteleri oluşturmayı sağlar.
- **Event Handling**: Kullanıcı etkileşimlerine (örneğin, buton tıklamaları) yanıt vererek web sitelerini daha ilgi çekici hale getirir.

JavaScript'in Kullanım Alanları

- **User Experience**: Form verilerini doğrulama, etkileşimli formlar oluşturma ve kullanıcıya gerçek zamanlı geri bildirim sağlama gibi işlevler sunar.
- **Dynamic Content**: Sayfanın tamamını yenilemeden yeni içerik yükleyebilir, örneğin sosyal medya akışları gibi.
- **Animations and Visual Effects**: Web sitelerinin estetiğini artırmak için animasyonlar ve görsel efektler oluşturabilir.

JavaScript Kodunu HTML'de Kullanma

- **Inline JavaScript**: JavaScript kodunu doğrudan HTML dosyasında <script> etiketi ile ekleyebilirsiniz.
- **External JavaScript File**: Daha büyük projeler için, JavaScript kodunu ayrı bir dosyada tutmak ve HTML dosyasına bağlamak önerilir.

JavaScript ile Bilgi Gösterimi

- **console.log**: Değerleri kontrol etmek ve hata ayıklamak için kullanılır.
- **DOM Alteration**: Web sayfasının içeriğini dinamik olarak değiştirmek için kullanılabilir.
- **Pop-up Dialogues**: Kullanıcıya mesaj göstermek veya girdi almak için alert, confirm, ve prompt gibi yöntemler sunar.

JavaScript, web development'da veri yönetimi ve değişkenlerin kullanımı üzerine odaklanan bir programlama dilidir. Bu video, JavaScript'teki değişkenler ve veri türleri hakkında temel bilgileri sunmaktadır.

Değişkenlerin Tanımı

- **Variables**: JavaScript'te verileri saklamak ve yönetmek için kullanılır. Değişkenler, belirli bir **name** ile tanımlanan veri saklama alanlarıdır.
- **Declaration**: Değişken oluşturmak için var, let veya const anahtar kelimeleri kullanılır. Bu, JavaScript'e bellek alanı ayırdığınızı bildirir.

Değişken Türleri

- **var**: Fonksiyon seviyesinde kapsam sağlar. Yani, var ile tanımlanan bir değişken, tanımlandığı fonksiyon içinde her yerden erişilebilir.

- **let**: ES6 ile tanıtılmıştır ve blok seviyesinde kapsam sağlar. Yani, let ile tanımlanan bir değişken, yalnızca tanımlandığı blok içinde geçerlidir.
- **const**: Değişmez değerler için kullanılır. Bir kez değer atandıktan sonra, const ile tanımlanan bir değişkenin değeri değiştirilemez.

Veri Türleri

- **Primitive Data Types**: JavaScript'te temel veri türleri arasında **String**, **Number**, **Boolean**, **Undefined**, ve **Null** bulunur.
- **Composite Data Types**: Birden fazla değeri tek bir birim olarak saklayabilen veri türleridir. Örneğin, **Array** ve **Object**.

Dinamik Tip Belirleme

- JavaScript, dinamik olarak tip belirleyen bir dildir. Yani, bir değişkenin veri türü, atanan değere göre çalışma zamanında belirlenir.

JavaScript'te **functions** (fonksiyonlar) ve **types of functions** (fonksiyon türleri) üzerine odaklanan bu video, temel kavramları açıklamaktadır.

Fonksiyonların Tanımı

- **Function**: JavaScript'te, belirli bir görevi yerine getiren ve tekrar kullanılabilen bir **block of code** (kod bloğu) olarak tanımlanır.
- **Reusable**: Fonksiyonlar, kodun daha düzenli ve okunabilir olmasını sağlar, bu da karmaşık uygulamaların geliştirilmesinde önemlidir.

Fonksiyon Türleri

- **Non-parameterized functions**: Parametre almayan ve belirli bir girdi gerektirmeyen fonksiyonlardır. Genellikle içsel mantık veya dışsal faktörlere dayanarak çalışır.
- **Parameterized functions**: Bir veya daha fazla **parameter** (parametre) alarak çalışan fonksiyonlardır. Bu parametreler, fonksiyonun tanımında belirtilir ve çağrıldığında belirli değerler ile geçilir.

Fonksiyon Yazım Yöntemleri

- **Function declaration**: Geleneksel bir yöntemdir. function anahtar kelimesi ile başlar, ardından fonksiyon adı ve parametreler gelir.
- **Function expression**: Fonksiyonu bir değişken ile tanımlayarak kullanılır. Bu yöntem, fonksiyonun bir değer olarak saklanmasını sağlar.

Fonksiyon Türleri

- **Named function**: Belirli bir isme sahip olan ve bu isimle çağrılabilen fonksiyonlardır. Hata ayıklama için faydalıdır.
- **Immediately Invoked Function Expression (IIFE)**: Tanımlandıktan hemen sonra çalışan bir fonksiyondur. Genellikle değişkenleri kapsüllemek için kullanılır.

- **Arrow function**: ES6 ile tanıtılan daha kısa bir yazım şeklidir. Kısa ve basit fonksiyonlar için idealdir.
- **Anonymous function**: İsimlendirilmemiş fonksiyonlardır. Genellikle diğer fonksiyonlara argüman olarak geçilir veya değişkenlere atanır.

Fonksiyonların Tanımı

- **Function**: JavaScript'te, belirli bir görevi yerine getiren ve tekrar kullanılabilen bir **block of code** (kod bloğu) olarak tanımlanır. Fonksiyonlar, function anahtar kelimesi ile başlar ve genellikle bir **name** (isim) ve **parameters** (parametreler) içerir.
- **Return statement**: Fonksiyonun sonucunu döndürmek için kullanılır. Bu, fonksiyonun çağrıldığı yere kontrolü geri verir.

Prototiplerin Kullanımı

- **Prototype**: JavaScript'te, bir nesnenin özelliklerini ve yöntemlerini tanımlamak için kullanılır. Tüm JavaScript nesneleri, new anahtar kelimesi ile oluşturulan nesneler için bir prototipe sahiptir.
- **Inheritance**: Prototipler, nesnelerin özelliklerini ve yöntemlerini miras almasını sağlar. Bu, kodun yeniden kullanılabilirliğini artırır.

Fonksiyon Örnekleri

- **Add function**: İki parametre alarak bu parametrelerin toplamını döndüren bir fonksiyon örneğidir. Eğer parametreler string ise, bunları birleştirir.
- **Car function**: Üç parametre alan bir fonksiyon örneğidir. this anahtar kelimesi, mevcut nesne örneğini ifade eder ve nesneye özgü özelliklere erişim sağlar.

Self-Executing Functions

- **Immediately Invoked Function Expression (IIFE)**: Tanımlandıktan hemen sonra çalışan bir fonksiyondur. Genellikle değişkenleri kapsüllemek ve global alanı kirletmemek için kullanılır.

Dizilerin Tanımı

- **Array**: JavaScript'te, birden fazla değeri saklamak için kullanılan bir **data structure** (veri yapısı)dır. Diziler, farklı **data types** (veri türleri) içerebilir; örneğin, **numbers**, **strings**, **objects** veya diğer diziler.
- **Ordered**: Diziler, belirli bir sıraya göre düzenlenir ve her bir eleman, dizinin **index** (indeks) numarası ile erişilebilir. JavaScript'te diziler **zero-indexed** (sıfırdan başlayan indeks) olarak tanımlanır.

Dizi Oluşturma ve Erişim

- **Creating an array**: Diziler, köşeli parantezler [] kullanılarak oluşturulur. Örneğin, let fruits = ['apple', 'banana', 'cherry']; şeklinde tanımlanabilir.
- **Accessing elements**: Dizinin elemanlarına erişmek için indeks numarası kullanılır. Örneğin, fruits[0] ifadesi 'apple' değerini döndürür.

Dizi Uzunluğu ve Değiştirme

- **Length property**: Bir dizinin eleman sayısını bulmak için length özelliği kullanılır. Örneğin, fruits.length ifadesi 3 değerini döndürür.
- **Mutable**: Diziler, oluşturulduktan sonra değiştirilebilir. Örneğin, fruits[2] = 'strawberry'; ifadesi, 'cherry' değerini 'strawberry' ile değiştirir.

Dizi Yöntemleri

- **Built-in methods**: JavaScript, diziler üzerinde işlem yapmak için birçok yerleşik yöntem sunar. Örneğin, push, pop, shift, unshift, splice, slice, concat, map, ve filter gibi yöntemler, dizilerin manipülasyonunu kolaylaştırır.
- **Example methods**: fruits.push('orange'); ifadesi, dizinin sonuna 'orange' eklerken, fruits.pop(); ifadesi dizinin sonundaki elemanı kaldırır.

Dizilerin Kullanım Alanları

- **Multidimensional arrays**: Diziler, diğer dizileri içerebilir ve bu da çok boyutlu diziler oluşturmayı sağlar. Bu, karmaşık veri yapıları için faydalıdır.
- **Common use cases**: Diziler, liste saklama, veri üzerinde döngü yapma, yığın ve kuyruk uygulamaları gibi birçok görevde kullanılır.

Dizi Yöntemlerinin Tanımı

- **Array methods**: JavaScript, diziler üzerinde işlem yapmayı kolaylaştıran yerleşik fonksiyonlar sunar. Bu yöntemler, dizilerin elemanları üzerinde işlem yapmayı ve verileri dönüştürmeyi sağlar.
- **Common methods**: En yaygın kullanılan dizi yöntemleri arasında forEach, map, filter, reduce, ve find bulunmaktadır.

forEach Yöntemi

- **forEach**: Bu yöntem, bir dizi üzerindeki her bir eleman için belirli bir **callback function** (geri çağırma fonksiyonu) çalıştırır. Yeni bir dizi oluşturmaz, sadece yan etkiler için kullanılır.
- **Example**: Kullanıcı nesneleri içeren bir dizide, her bir kullanıcıya hoş geldin e-postası göndermek için forEach kullanılabilir.

map Yöntemi

- **map**: Bu yöntem, her bir dizi elemanına belirli bir fonksiyon uygulayarak yeni bir dizi oluşturur. Orijinal diziyi değiştirmez.
- **Example**: Ürün nesnelerinden fiyatları çıkarmak için map kullanılabilir.

filter Yöntemi

- **filter**: Belirli bir koşulu karşılayan elemanları içeren yeni bir dizi oluşturur. Orijinal diziyi filtreler.
- **Example**: Fiyat aralığına göre ürünleri filtrelemek için filter kullanılabilir.

reduce Yöntemi

- **reduce**: Bir diziyi tek bir değere indirgemek için kullanılır. Her bir eleman üzerinde belirli bir fonksiyon çalıştırarak toplam veya başka bir hesaplama yapar.
- **Example**: Sipariş fiyatlarının toplamını hesaplamak için reduce kullanılabilir.

find Yöntemi

- **find**: Belirli bir koşulu karşılayan ilk elemanı bulur. Eğer koşulu karşılayan bir eleman yoksa undefined döner.
- **Example**: Belirli bir ID'ye sahip çalışanı bulmak için find kullanılabilir.

DOM Nedir?

- **Document Object Model (DOM)**: HTML ve XML belgelerinin programatik olarak temsilini sağlayan bir yapıdır. JavaScript, DOM aracılığıyla web sayfalarının içeriğini ve yapısını değiştirebilir.

Elementlere Erişim

- **getElementById**: Belirli bir **id**'ye sahip HTML elementine erişmek için kullanılır. Örneğin, document.getElementById('myElement') ifadesi, 'myElement' id'sine sahip elementi döndürür.
- **getElementsByTagName**: Belirli bir **tag name** (etiket adı) ile eşleşen tüm elementleri içeren bir **NodeList** döndürür.
  Örneğin, document.getElementsByTagName('p') ifadesi, tüm <p> etiketlerini döndürür.

Elementleri Değiştirme

- **innerHTML**: Bir elementin içeriğini almak veya ayarlamak için kullanılır.
  Örneğin, element.innerHTML = 'New Content'; ifadesi, elementin içeriğini günceller.
- **style**: Elementin **inline CSS** stilini değiştirmek için kullanılır.
  Örneğin, element.style.color = 'red'; ifadesi, elementin metin rengini kırmızı yapar.

Yeni Element Oluşturma

- **createElement**: Yeni bir HTML elementi oluşturmak için kullanılır. Örneğin, let newElement = document.createElement('div'); ifadesi, yeni bir elementi oluşturur.
- **appendChild**: Oluşturulan elementi mevcut bir elementin altına eklemek için kullanılır.
  Örneğin, parentElement.appendChild(newElement); ifadesi, newElement'i parentElem ent'in altına ekler.

Olay Yönetimi

- **Event listeners**: Kullanıcı etkileşimlerini dinlemek için kullanılır. Örneğin, element.addEventListener('click', function() { ... }); ifadesi, elemente tıklandığında belirli bir fonksiyonu çalıştırır.
- **Prevent default**: Olayın varsayılan davranışını engellemek için kullanılır. Örneğin, form gönderimini engellemek için event.preventDefault(); ifadesi kullanılabilir.

Client-Side Script Nedir?

- **Client-side script**: HTML belgesine eşlik eden veya doğrudan HTML belgesine gömülen bir programdır. Bu betikler, kullanıcının cihazında çalışır ve genellikle sayfa yüklendiğinde veya belirli olaylar gerçekleştiğinde tetiklenir.

Betik Kullanım Alanları

- **Form validation**: Kullanıcıdan alınan verilerin doğruluğunu kontrol etmek için betikler kullanılabilir. Örneğin, bir formun doğru şekilde doldurulup doldurulmadığını kontrol etmek için JavaScript kullanılabilir.
- **Dynamic content**: Betikler, HTML sayfasındaki içerikleri dinamik olarak oluşturmak veya değiştirmek için kullanılabilir. Örneğin, kullanıcı etkileşimlerine göre içerik güncellenebilir.

Script Tag Kullanımı

- **Script tag**: HTML belgesine betik eklemek için kullanılır. Betikler, HTML belgesinin <head> veya <body> bölümünde yer alabilir.
- **Inline vs External scripts**: Kısa betikler için doğrudan HTML içinde yazılabilirken, uzun betikler için harici bir dosyaya (src attribute) işaret eden bir yöntem tercih edilir.

noscript Tag Kullanımı

- **noscript tag**: JavaScript'in devre dışı bırakıldığı veya desteklenmediği durumlarda alternatif içerik sağlamak için kullanılır. Bu tag içindeki içerik, betik çalışmadığında görüntülenir.

Event Binding

- **Event binding**: Olayların tetiklenmesiyle belirli fonksiyonların çalıştırılmasıdır. Örneğin, onload olayı, sayfa yüklendiğinde bir betiği çalıştırmak için kullanılabilir.
- **Event handlers**: Belirli bir olay gerçekleştiğinde (örneğin, bir butona tıklama) çalışacak fonksiyonlardır. Örneğin, onclick olayı, bir butona tıklandığında belirli bir işlemi gerçekleştirmek için kullanılabilir

SEO: Arama motorları, **noscript** içeriğini de tarayabilir, bu nedenle önemli bilgileri burada sunmak, SEO açısından faydalı olabilir.

DOM Nedir?

- **DOM**: HTML veya XHTML ile JavaScript arasında bir programlama arayüzüdür. Web sayfalarının içeriğini, yapısını ve stilini dinamik olarak erişip güncellemeye olanak tanır.
- **DOM Hierarchy**: DOM, bir ağaç yapısı şeklinde düzenlenmiştir. En üstteki **window object**, tüm diğer nesneleri kontrol eder.

Temel DOM Nesneleri

- **Window Object**: Tarayıcı ortamını kontrol eder ve global bir nesne olarak işlev görür. JavaScript kodu, bu nesne üzerinden çalışır.
- **Document Object**: HTML sayfasındaki tüm elemanlara erişim sağlar. Her yüklü HTML belgesi, bir **document object** olarak temsil edilir.
- **Location Object**: Sayfanın URL bilgilerini içerir ve sayfanın konumunu yönetir.
- **Navigator Object**: Kullanıcının tarayıcısı hakkında bilgi sağlar, ancak tarayıcılar arasında tutarsızlık gösterebilir.
- **Screen Object**: Kullanıcının ekranı hakkında bilgi verir, örneğin ekran boyutları.

DOM Düzeyleri

- **DOM Level 1**: Temel özellikleri tanımlar ve çoğu tarayıcı tarafından desteklenir.
- **DOM Level 2**: Daha ayrıntılı özellikler sunar ve HTML elemanlarına erişim için API'ler içerir.

Node Türleri

- **Element Nodes**: HTML etiketlerini temsil eder (örneğin, <html>, <head>, <body>).
- **Text Nodes**: HTML etiketleri arasındaki metinleri temsil eder.

Önemli Noktalar

- DOM, web sayfalarının dinamik olarak güncellenmesini sağlar ve JavaScript ile etkileşimde bulunmak için kritik bir yapı sunar.
- Farklı tarayıcılar, DOM standartlarına farklı seviyelerde uyum gösterir, bu nedenle uyumluluk önemlidir.

DOM (Document Object Model) Seviyeleri, web tarayıcılarının HTML ve XML belgeleriyle etkileşimde bulunmasını sağlayan bir yapı sunar. DOM, farklı seviyelerde özellikler ve işlevsellik sunarak geliştiricilerin belgeleri daha etkili bir şekilde yönetmelerine olanak tanır. İşte DOM seviyeleri hakkında detaylı bir açıklama:

1. DOM Level 1

- **Tanım**: DOM Level 1, DOM'un ilk sürümüdür ve temel özellikleri tanımlar. Bu seviye, HTML ve XML belgeleri için temel bir yapı sağlar.
- **Ana Özellikler**:

    - **Core DOM**: Temel nesne türlerini ve bunların nasıl etkileşimde bulunacağını tanımlar. Örneğin, document, element, attribute, text gibi nesneler.
    - **HTML DOM**: HTML belgeleri için özel işlevsellik sunar. HTML etiketlerine erişim ve manipülasyon için gerekli yöntemleri içerir.
- **Kullanım**: DOM Level 1, çoğu modern tarayıcı tarafından desteklenir ve geliştiricilere temel belge yapısını anlamaları için bir temel sağlar.

2. DOM Level 2

- **Tanım**: DOM Level 2, Level 1'e ek olarak daha fazla özellik ve işlevsellik sunar. Bu seviye, daha karmaşık belgelerle çalışmayı kolaylaştırır.
- **Ana Özellikler**:
    - **Events**: DOM Level 2, olay yönetimi için bir API sunar. Bu, kullanıcı etkileşimlerini (örneğin, tıklama, fare hareketi) dinlemek ve yanıt vermek için kullanılır.
    - **Style**: CSS stillerine erişim ve manipülasyon için yöntemler içerir. Bu, sayfanın görünümünü dinamik olarak değiştirmeyi sağlar.
    - **Traversal and Range**: DOM ağaç yapısında gezinmeyi ve belirli bir aralıkta işlem yapmayı kolaylaştıran yöntemler sunar.
- **Kullanım**: DOM Level 2, daha etkileşimli ve dinamik web sayfaları oluşturmak için gereklidir. Modern tarayıcılar bu seviyeyi destekler.

3. DOM Level 3

- **Tanım**: DOM Level 3, önceki seviyelere ek olarak daha fazla özellik ve esneklik sunar. Bu seviye, daha karmaşık uygulamalar için gerekli olan işlevselliği sağlar.
- **Ana Özellikler**:
    - **XPath**: XML belgelerinde belirli düğümleri seçmek için kullanılan bir dil. XPath, DOM ile birlikte kullanılarak belgelerde daha karmaşık sorgular yapmayı sağlar.
    - **Load and Save**: Belgeleri yükleme ve kaydetme işlevselliği sunar. Bu, belgelerin dış kaynaklardan alınmasını veya dosya sistemine kaydedilmesini kolaylaştırır.
    - **Validation**: Belgelerin geçerliliğini kontrol etmek için yöntemler içerir. Bu, XML belgelerinin doğru bir şekilde yapılandırıldığından emin olmayı sağlar.
- **Kullanım**: DOM Level 3, daha karmaşık ve veri odaklı uygulamalar geliştirmek için gereklidir. Bu seviye, modern web uygulamalarında yaygın olarak kullanılır.

Özet

- **DOM Level 1**: Temel yapı ve nesne türleri.
- **DOM Level 2**: Olay yönetimi, stil manipülasyonu ve ağaçta gezinme.

- **DOM Level 3**: XPath desteği, belge yükleme/kaydetme ve geçerlilik kontrolü.

Bu seviyeler, geliştiricilerin web sayfalarını daha etkili bir şekilde yönetmelerine ve etkileşimli uygulamalar oluşturmalarına olanak tanır. Eğer başka bir konuda daha fazla bilgi isterseniz, lütfen belirtin!

Basic DOM Model for Browsers (Tarayıcılar için Temel DOM Modeli), web sayfalarının yapısını ve içeriğini temsil eden bir ağaç yapısıdır. Bu model, JavaScript'in HTML belgeleriyle etkileşimde bulunmasını sağlar. Şimdi bu modeli daha ayrıntılı bir şekilde inceleyelim.

DOM Modelinin Yapısı

- **Ağaç Yapısı**: DOM, bir ağaç yapısı şeklinde düzenlenmiştir. Bu ağaçta her bir eleman bir **node** (düğüm) olarak adlandırılır. Ağaç yapısının en üstünde **root node** (kök düğüm) bulunur ve bu genellikle **document** nesnesidir.
- **Node Türleri**:
    - **Element Nodes**: HTML etiketlerini temsil eder.
      Örneğin, <html>, <head>, <body>, , <p> gibi etiketler element node'lardır.
    - **Text Nodes**: HTML etiketleri arasındaki metinleri temsil eder. Örneğin, bir <p> etiketinin içindeki metin bir text node'dur.

Temel DOM Nesneleri

- **Window Object**: DOM hiyerarşisinin en üstünde yer alır ve tarayıcı ortamını kontrol eder. JavaScript kodu, bu nesne üzerinden çalışır.
- **Document Object**: HTML sayfasındaki tüm elemanlara erişim sağlar. Her yüklü HTML belgesi, bir document object olarak temsil edilir.
- **Diğer Nesneler**:
    - **Location Object**: Sayfanın URL bilgilerini içerir.
    - **Navigator Object**: Kullanıcının tarayıcısı hakkında bilgi verir.
    - **Screen Object**: Kullanıcının ekranı hakkında bilgi sağlar.

Örnek

Aşağıda basit bir HTML belgesinin DOM modelini gösteren bir örnek verilmiştir:

```
<!DOCTYPE html>
<html>
<head>
  <title>My Web Page</title>
</head>
<body>
```

```
    <h1>Welcome to My Web Page</h1>
    <p>This is a simple paragraph.</p>
</body>
</html>
```

Bu HTML belgesinin DOM yapısı şu şekilde görünür:

- **document**
    - **html** (Element Node)
        - **head** (Element Node)
            - **title** (Element Node)
                - "My Web Page" (Text Node)
        - **body** (Element Node)
            - **h1** (Element Node)
                - "Welcome to My Web Page" (Text Node)
            - **p** (Element Node)
                - "This is a simple paragraph." (Text Node)

Önemli Noktalar

- DOM, web sayfalarının dinamik olarak güncellenmesini sağlar. JavaScript, bu model aracılığıyla sayfa içeriğini değiştirebilir, yeni elemanlar ekleyebilir veya mevcut elemanları kaldırabilir.
- DOM, tarayıcılar arasında farklılık gösterebilir, bu nedenle uyumluluk önemlidir.

Bu temel DOM modeli, web geliştirme sürecinde JavaScript ile etkileşimde bulunmak için kritik bir yapı sunar.

Node Types

- W3C DOM Level 2, HTML belgelerinde doğrudan uygulanabilir 7 node type tanımlar.
- Örnekler:
    - ELEMENT_NODE (1)
    - ATTRIBUTE_NODE (2)
    - TEXT_NODE (3)
    - COMMENT_NODE (8)

Accessing Document Elements

- Tarayıcı, belge yüklendiğinde formlar, resimler, bağlantılar gibi nesneler için diziler oluşturur.

- Örnek erişim yöntemleri:
    - document.forms[0].elements[0] ile bir form elemanına erişim.
    - document.getElementById("idValue") ile id değeri ile bir elemana erişim.

Naming Conventions

- id attribute, bir HTML elemanını tanımlamak için kullanılır ve benzersiz olmalıdır.
- id değeri, scriptlerde referans almak için kullanılır ve tırnak içinde belirtilmelidir.