WEB & YAZILIM GELİŞTİRME SERİSİ · MODÜL 5

JavaScript

Web'e etkileşim: değişkenler, veri türleri ve operatörler; koşullar, döngüler ve fonksiyonlar; DOM ile sayfayı canlandırma, olaylar ve modern JavaScript. Gerçek kod ve özgün diyagramlarla, sıfırdan.

Değişkenler · Mantık · DOM · Modern JS · Eğitim amaçlıdır

Bu Kitap Hakkında

Bu modül, web'e davranış ve etkileşim katan JavaScript'i sıfırdan öğretir. Dört seviye ve yirmi altı bölüm boyunca değişkenlerden veri türlerine, koşul ve döngülerden fonksiyonlara, DOM ile sayfa manipülasyonundan olaylara ve modern JavaScript'ten (map/filter/reduce, async/await, fetch) güvenli koda kadar her şeyi gerçek kod örnekleriyle ele alır.

Her bölümde çalışan kod blokları, konuyu görselleştiren özgün bir diyagram (değişken anatomisi, if/else kararı, döngü çevrimi, DOM ağacı, olay akışı), 'konsolda ne görürsün?' kartı ve bir alıştırma yer alır. Bu, on altı modüllük 'Web & Yazılım Geliştirme' serisinin beşinci modülüdür; HTML iskeletine ve CSS görünümüne burada davranış katarsın. Kod örneklerini kendi editöründe ve tarayıcı konsolunda deneyerek ilerlemen önerilir. Bu seri eğitim amaçlıdır.

Web & Yazılım Geliştirme Serisi · Modül 5

İçindekiler

★ JavaScript Sözlüğü 61

SEVİYE 1

İlk Adımlar

JavaScript'in temelleri: JS nedir, sayfaya eklemek, değişkenler, veri türleri, operatörler, diziler, nesneler ve konsolla hata ayıklama.

BÖLÜM 01

JavaScript Nedir?

HTML sayfanın iskeletini, CSS görünümünü kurar. JavaScript ise sayfaya davranış ve etkileşim katar: tıklamalara tepki vermek, içeriği değiştirmek, veri hesaplamak, sunucuyla konuşmak. Web'i "canlı" yapan dildir.

Üçlünün son parçası

- **HTML:** yapı (ne var).
- **CSS:** görünüm (nasıl görünür).
- **JavaScript:** davranış (ne yapar, nasıl tepki verir).

Şema 1.1 — JavaScript, web'in üçlüsünü tamamlayan "davranış" katmanıdır.

JavaScript nerede çalışır?

- Tarayıcıda (sayfayı canlandırır) — bu modülün odağı.
- Sunucuda (Node.js ile) — ileri modüllerde.
- Gerçek bir programlama dilidir: değişken, mantık, döngü, fonksiyon.

İlk JavaScript satırın

```
console.log("Merhaba, JavaScript!");
alert("Sayfa yüklendi!");
```

İPUCU

JavaScript, Java ile **aynı şey değildir** (isim benzerliği tarihsel bir pazarlama tercihidir). Tarayıcıdaki tek programlama dili odur; bu yüzden web geliştirmenin vazgeçilmezidir. Bu modülde gerçek, çalışan kod yazacaksın.

Konsolda ne görürsün? ÇIKTI

`console.log("Merhaba, JavaScript!")` çalıştığında, tarayıcının geliştirici konsoluna **Merhaba, JavaScript!** yazısı düşer. `console`, kodun ne yaptığını görmenin en temel yoludur.

Alıştırma 8 dk

JS'i tanı:

1. HTML, CSS ve JS'in rollerini kendi cümlelerinle yaz.
2. JavaScript'in nerelerde çalıştığını say.
3. Bir `console.log` satırı yaz.

BÖLÜM 02

JS'i Sayfaya Eklemek

JavaScript'i HTML sayfasına eklemenin birkaç yolu vardır. Profesyoneller genelde ayrı bir dosya kullanır; tıpkı CSS'te olduğu gibi.

Ekleme yolları

- **Satır içi:** `onclick="..."` — küçük, dağınık, kaçın.
- **İç:** `<script>` bloğu doğrudan HTML içinde.
- **Dış (önerilen):** ayrı `.js` dosyası + `<script src>`.

Dış JS dosyası (önerilen)

```
<!-- index.html, body sonunda -->
<script src="app.js"></script>

// app.js
console.log("app.js çalıştı");
```

Şema 2.1 — JavaScript'i sayfaya ekleme yolları.

İPUCU

`<script>` etiketini genelde `<body>`'nin **sonuna** koy; böylece sayfa içeriği önce yüklenir, JS sonra çalışır ve öğeleri bulabilir. Hızlı denemeler için tarayıcının geliştirici araçlarındaki **Konsol**'a doğrudan kod yazabilirsin.

Konsolda ne görürsün? ÇIKTI

`app.js` bağlanıp çalışınca, içindeki `console.log("app.js çalıştı")` konsola **app.js çalıştı** yazar — bu, dosyanın doğru bağlandığını doğrulamanın basit yoludur.

◎ Alıştırma

8 dk

JS'i bağla:

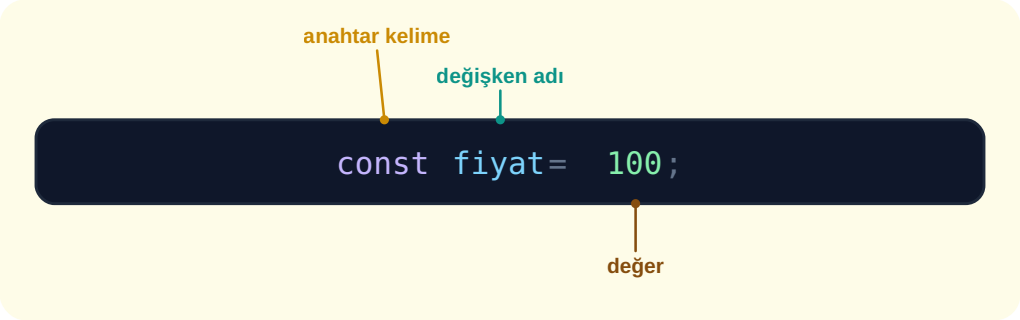
1. Bir `app.js` dosyası oluştur.
2. Onu `<script src>` ile sayfaya bağla.
3. Tarayıcı konsolunu açıp çıktıyı gör.

BÖLÜM 03

Değişkenler

Değişkenler, veriyi saklayan ve isimle eriştiğin kutulardır: bir sayı, bir metin, bir liste. JavaScript'te neredeyse her şey değişkenlerle başlar.

Bir değişkenin anatomisi



Şema 3.1 — Bir değişken tanımı: anahtar kelime, ad ve değer.

let, const ve var

- `const` — değeri değişmeyecek (sabit). Varsayılan tercihin olsun.
- `let` — değeri sonradan değişebilen.
- `var` — eski yöntem; artık önerilmez.

Değişken tanımlama

```
const ad = "Ayşe";
let puan = 0;
puan = puan + 10;   // let değiştirilebilir
// ad = "Veli";     // const değiştirilemez — hata
```

İPUCU

Kuralın basit olsun: **önce** `const` kullan; bir değeri gerçekten değiştirmen gerekiyorsa `let`'e geç. `var`'ı hiç kullanma. Bu, beklenmedik hataları azaltır ve kodun niyetini netleştirir (bu değer sabit mi, değişken mi?).

Konsolda ne görürsün? ÇIKTI

Yukarıdaki kodda `puan` önce 0'dır, sonra 10 olur. `console.log(puan)` dersen konsola **10** yazar. `const ad`'ı değiştirmeye çalışırsan tarayıcı hata verir.

◎ Alıştırma

10 dk

Değişken kullan:

1. Bir `const` ve bir `let` tanımla.
2. `let`'in değerini değiştir.
3. `const`'u değiştirmeye çalışıp hatayı gör.

BÖLÜM 04

Veri Türleri

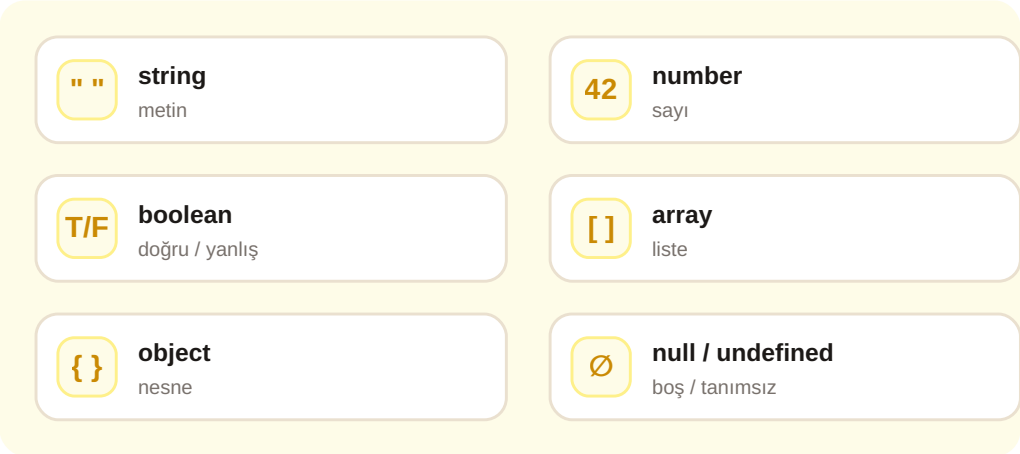
JavaScript'te veriler farklı türlerde olur: metin, sayı, doğru/yanlış... Her türün kendi davranışı vardır. Türleri tanımak, hataları önler.

Temel veri türleri

- **String** (metin): `"Merhaba"` — tırnak içinde.
- **Number** (sayı): `42`, `3.14`.
- **Boolean** (mantıksal): `true` / `false`.
- **null / undefined**: değer yok / henüz tanımsız.

Farklı türler

```
const ad = "Ayşe";        // string
const yas = 30;           // number
const aktif = true;       // boolean
const adres = null;       // değer yok
console.log(typeof yas);  // "number"
```



Şema 4.1 — JavaScript'in temel veri türleri.

İPUCU

`typeof` ile bir değerin türünü öğrenebilirsin: `typeof "abc"` → `"string"`. Tür karışıklığı yaygın bir hata kaynağıdır: `"5" + 5` metin birleştirip `"55"` verir, `5 + 5` ise `10`. Sayı mı metin mi olduğuna dikkat et.

Konsolda ne görürsün? ÇIKTI

`typeof yas` konsola **"number"** yazar. Türleri bilmek, `"5" + 5` gibi sürprizleri (sonuç: "55") anlamanı sağlar.

Alıştırma 8 dk

Türleri keşfet:

1. Her türden (string, number, boolean) birer değişken yaz.
2. `typeof` ile türlerini konsola yazdır.
3. `"5" + 5` ve `5 + 5` sonuçlarını karşılaştır.

BÖLÜM 05

Operatörler

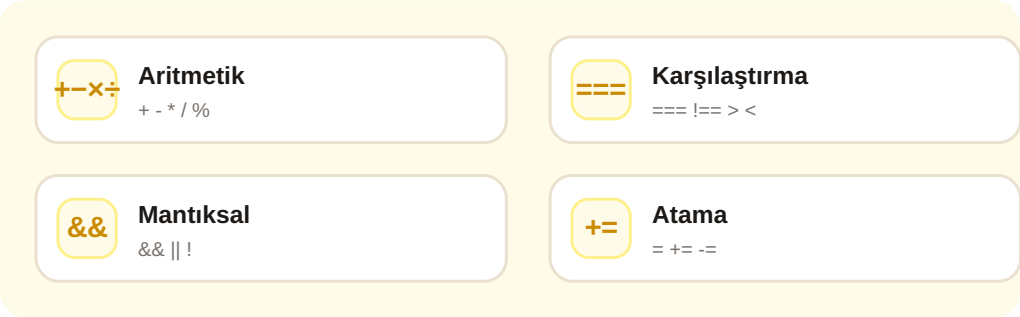
Operatörler, değerlerle işlem yapmanı sağlar: toplama, karşılaştırma, mantıksal birleştirme. Hesaplamaların ve kararların temelidir.

Operatör türleri

- **Aritmetik:** `+ - * / %` (mod: kalan).
- **Karşılaştırma:** `=== !== > < >= <=`.
- **Mantıksal:** `&&` (ve), `||` (veya), `!` (değil).
- **Atama:** `=`, `+=`, `-=`.

Operatörler iş başında

```
const toplam = 5 + 3;         // 8
const esit = (5 === 5);       // true
const yetiskin = yas >= 18 && aktif;
let sayac = 0; sayac += 1;    // 1
```



Şema 5.1 — Operatör türleri.

İPUCU

Karşılaştırmada **her zaman** `===` (üç eşittir) kullan, `==` değil. `===` hem değeri hem türü kontrol eder; `==` ise tür dönüşümü yaparak şaşırtıcı sonuçlar verebilir (örn. `0 == ""` bazı durumlarda true olur). `===` güvenlidir.

Konsolda ne görürsün? ÇIKTI

`5 + 3` konsola **8**, `5 === 5` ise **true** yazar. `%` (mod) operatörü kalanı verir: `10 % 3` → **1**. Bunlar mantık ve hesaplamanın yapı taşlarıdır.

◎ Alıştırma

8 dk

Operatörleri dene:

1. Birkaç aritmetik işlem yapıp sonucu yazdır.
2. `===` ile iki değeri karşılaştır.
3. `&&` ve `||` ile mantıksal bir ifade kur.

BÖLÜM 06

Diziler (Arrays)

Bir dizi (array), sıralı birden çok değeri tek bir değişkende tutar: bir alışveriş listesi, kullanıcı adları, puanlar. Her öğeye bir indeks (sıra numarası) ile erişirsin.

Dizi nedir?

Şema 6.1 — Bir dizi: indeksli kutular (0'dan başlar).

- Köşeli parantezle oluşturulur: `[...]`.
- İndeks 0'dan başlar: ilk öğe `dizi[0]`.
- `dizi.length` — öğe sayısı.

Dizilerle çalışmak

```
const meyveler = ["elma", "armut", "kiraz"];
console.log(meyveler[0]);     // "elma"
console.log(meyveler.length); // 3
meyveler.push("muz");         // sona ekle
```

İPUCU

İndeksin **0'dan başladığını** unutma: 3 öğeli bir dizide indeksler 0, 1, 2'dir (3 değil). `dizi.push()` sona ekler, `dizi.pop()` sondan çıkarır. Diziler, çok sayıda veriyi düzenli tutmanın temel yoludur — döngülerle birlikte çok güçlüdür (Seviye 2).

Konsolda ne görürsün? ÇIKTI

`meyveler[0]` konsola **"elma"**, `meyveler.length` ise **3** yazar. `push("muz")` 'dan sonra dizi dört öğeli olur.

◎ Alıştırma

10 dk

Dizi kullan:

1. Sevdiğin 3 şeyden bir dizi oluştur.
2. İlk ve son öğeyi indeksle yazdır.
3. `push` ile bir öğe ekle, `length`'i kontrol et.

BÖLÜM 07

Nesneler (Objects)

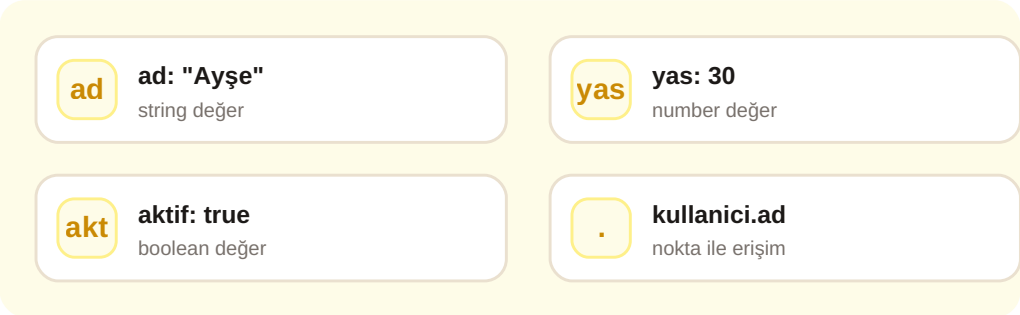
Bir nesne (object), ilişkili verileri anahtar–değer çiftleriyle bir arada tutar: bir kullanıcının adı, yaşı ve durumu gibi. Gerçek dünyadaki "şeyleri" modellemenin yoludur.

Anahtar–değer çiftleri

- Süslü parantezle oluşturulur: `{...}`.
- Her özellik bir **anahtar: değer** çiftidir.
- Değere nokta ile erişilir: `kullanici.ad`.

Bir nesne tanımlamak ve kullanmak

```
const kullanici = {
  ad: "Ayşe",
  yas: 30,
  aktif: true
};
console.log(kullanici.ad);   // "Ayşe"
kullanici.yas = 31;          // değeri güncelle
```



Şema 7.1 — Bir nesne: anahtar–değer çiftleri.

İPUCU

Diziler "sıralı liste" için, nesneler "isimli özellikler" için idealdir. Çoğu gerçek veri ikisinin birleşimidir: örneğin bir **kullanıcı dizisi**, her biri nesne olan kullanıcıları tutar. Nokta gösterimi (`kullanici.ad`) en sık kullandığın erişim yoludur.

Konsolda ne görürsün? ÇIKTI

`kullanici.ad` konsola **"Ayşe"** yazar. `kullanici.yas = 31` dedikten sonra `kullanici.yas` artık **31**'dir. Nesneler, ilişkili bilgileri tek bir yapıda toplar.

◎ Alıştırma

10 dk

Nesne oluştur:

1. Kendini tanımlayan bir nesne yaz (ad, yas, sehir).
2. Bir özelliğe nokta ile eriş ve yazdır.
3. Bir özelliğin değerini güncelle.

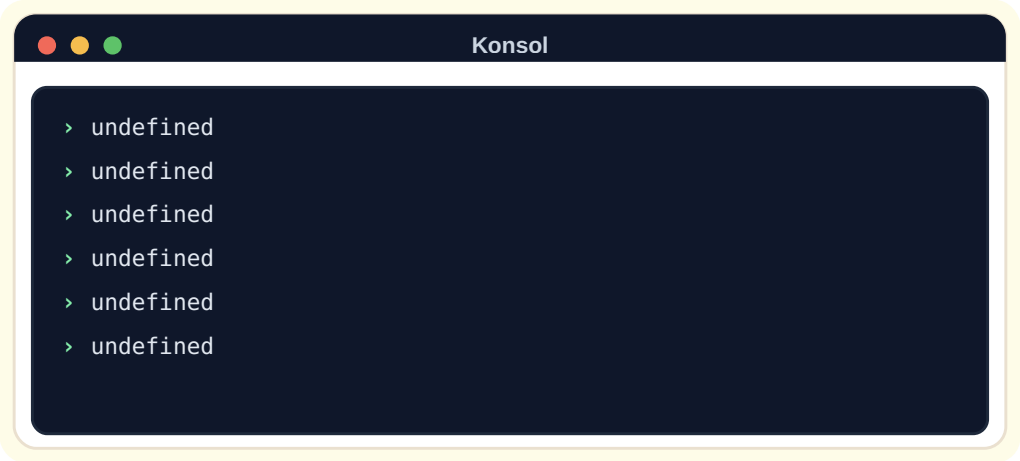
BÖLÜM 08

Konsol ve Hata Ayıklama

Her geliştirici hata yapar; önemli olan onları bulup düzeltebilmektir. Tarayıcının konsolu, kodunun ne yaptığını görmenin ve hataları çözmenin en güçlü aracıdır.

Konsolun gücü

- `console.log(...)` — değerleri görmek için.
- Hata mesajları (kırmızı) — neyin, nerede bozulduğunu söyler.
- Geliştirici araçları (F12) — Konsol sekmesi.



Şema 8.1 — Tarayıcı konsolu: çıktıları ve hataları gösterir.

Hata ayıklama dostların

```
 console.log("buraya kadar çalıştı");
console.log("değer:", puan);
console.table(meyveler);   // diziyi tablo gibi göster
```

İPUCU

Bir şey çalışmadığında **panik yapma**: konsoldaki kırmızı hata mesajını oku — genelde sorunun türünü ve satırını söyler. Kodun arasına `console.log` serpiştirerek "nereye kadar çalışıyor?" diye izlemek, en sık kullanılan hata ayıklama tekniğidir.

Konsolda ne görürsün? ÇIKTI

Konsol, `console.log` çıktılarını sırayla gösterir; bir hata olursa kırmızı bir mesaj (örn. `ReferenceError: ... is not defined`) belirir ve hangi satırda olduğunu belirtir. Bu mesajlar düşmanın değil, en iyi rehberindir.

◎ Alıştırma

10 dk

Hata ayıkla:

1. Tanımlanmamış bir değişkeni yazdırıp hatayı oku.
2. Kodun arasına `console.log` ekleyip akışı izle.
3. Bir `console.table` ile bir diziyi görüntüle.

SEVİYE 2

Mantık ve Akış

Programa karar ve tekrar yeteneği: koşullar (if/else), karşılaştırma ve mantık, döngüler, dizi döngüleri, fonksiyonlar ve kapsam ile ok fonksiyonları.

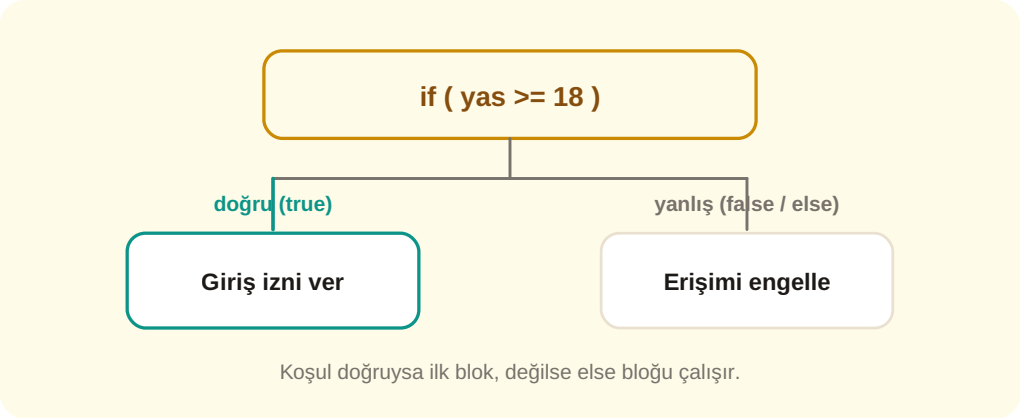
BÖLÜM 09

Koşullar (if / else)

Programlar karar verebilmelidir: "kullanıcı giriş yaptıysa şunu göster, yapmadıysa bunu". `if / else` yapısı, koşula göre farklı kod çalıştırmanı sağlar.

Koşul mantığı

- `if (koşul) { ... }` — koşul doğruysa çalışır.
- `else { ... }` — koşul yanlışsa çalışır.
- `else if` — birden çok durumu sırayla kontrol eder.



Şema 9.1 — if/else: koşul doğruysa bir yol, yanlışsa diğer yol.

Koşullu karar

```
const yas = 20;
if (yas >= 18) {
  console.log("Giriş izni verildi");
} else {
  console.log("Erişim engellendi");
}
```

İPUCU

Koşulu okunur tut: karmaşık iç içe `if`'ler yerine, erken çıkış (önce hatalı durumları ele alıp dönmek) genelde daha temizdir. Çok sayıda sabit durum varsa `else if` zinciri veya `switch` kullanılır. Koşulun her zaman `true` / `false` ürettiğinden emin ol.

Konsolda ne görürsün? ÇIKTI

`yas = 20` için konsola **Giriş izni verildi** yazılır (çünkü 20 >= 18 doğrudur). `yas`'ı 15 yaparsan **Erişim engellendi** yazılır. Program, koşula göre farklı davranır.

◎ Alıştırma

10 dk

Karar verdir:

1. Bir sayıya göre "çift mi tek mi" yazan bir `if/else` yaz (ipucu: `% 2`).
2. Bir nota göre "geçti/kaldı" kararı ver.
3. `else if` ile üç durumlu bir kontrol kur.

BÖLÜM 10

Karşılaştırma ve Mantık

Koşulların kalbinde karşılaştırma ve mantıksal operatörler vardır. Bunları ve "truthy/falsy" kavramını anlamak, doğru kararlar kurmanın anahtarıdır.

Karşılaştırma ve mantık operatörleri

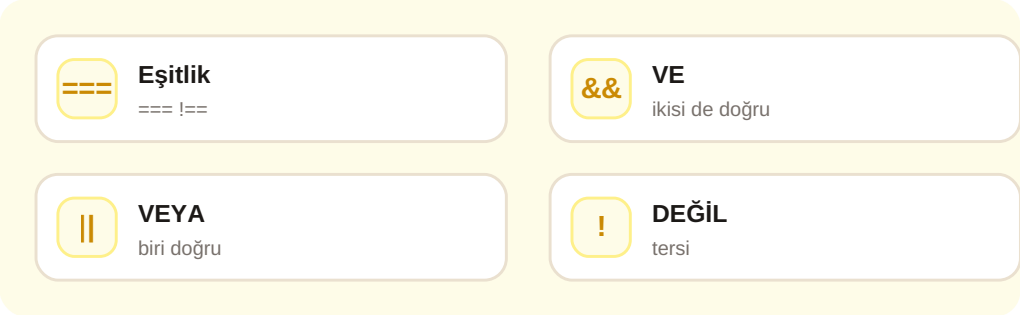
- `===` / `!==` — eşit / eşit değil (tür dahil).
- `> < >= <=` — büyüklük karşılaştırması.
- `&&` (ve), `||` (veya), `!` (değil).

Truthy ve falsy

- "Falsy" değerler koşulda `false` gibi davranır: `0`, `""`, `null`, `undefined`.
- Diğer her şey "truthy"dir (koşulda doğru sayılır).

Mantıksal ifadeler

```
const giris = true, yas = 25;
if (giris && yas >= 18) { console.log("Erişim var"); }
if (!giris) { console.log("Lütfen giriş yap"); }
const isim = "" || "Misafir";   // "" falsy -> "Misafir"
```



Şema 10.1 — Karşılaştırma ve mantık operatörleri.

İPUCU

`||` operatörü varsayılan değer vermek için sık kullanılır: `const isim = girilen || "Misafir"` — girilen boşsa "Misafir" atanır. Truthy/falsy'yi bilmek bu tür kısa yolları güvenle kullanmanı sağlar. Yine de eşitlik için her zaman `===` kullan.

Konsolda ne görürsün? ÇIKTI

İlk koşul (`giris && yas >= 18`) doğru olduğu için konsola **Erişim var** yazılır. `isim` değişkeni ise boş metin falsy olduğundan **"Misafir"** olur.

Alıştırma 8 dk

Mantığı dene:

1. `&&` ile iki koşulu birleştir.
2. `!` ile bir koşulu tersine çevir.
3. `||` ile bir varsayılan değer ver.

BÖLÜM 11

Döngüler (Loops)

Aynı işi defalarca yapmak (bir listedeki her öğeyi işlemek, 1'den 100'e saymak) için döngüler kullanılır. Döngü, kod tekrarını ortadan kaldırır.

for ve while

- `for` — belli sayıda tekrar (sayaçlı).
- `while` — koşul doğru olduğu sürece tekrar.
- Her ikisi de "koşul doğruysa gövdeyi çalıştır, sonra tekrar kontrol et" mantığını izler.

Şema 11.1 — Döngü mantığı: koşul doğru oldukça gövde tekrarlanır.

for ve while döngüleri

```
for (let i = 1; i <= 3; i++) {
  console.log("Tekrar " + i);
}
let sayac = 0;
while (sayac < 3) { sayac++; }
```

İPUCU

Döngülerde en sık hata **sonsuz döngüdür**: koşul hiç yanlış olmazsa program kilitlenir. `while` kullanırken döngü gövdesinin koşulu eninde sonunda yanlışa çevirdiğinden emin ol (örn. sayacı artır). `for` döngüsü sayacı kendi yönettiği için bu hataya daha az açıktır.

Konsolda ne görürsün? ÇIKTI

`for` döngüsü konsola sırayla **Tekrar 1**, **Tekrar 2**, **Tekrar 3** yazar. `i` 1'den başlar, her turda 1 artar, 3'ü geçince durur.

◎ Alıştırma

10 dk

Döngü kur:

1. `for` ile 1'den 5'e kadar say.
2. Bir `while` döngüsü yaz (koşulu mutlaka değişsin).
3. Çift sayıları yazdıran bir döngü kur.

BÖLÜM 12

Dizi Döngüleri

Diziler ve döngüler birlikte çok güçlüdür: bir dizinin her öğesini tek tek işlemek, JavaScript'te en sık yaptığın işlerden biridir. Bunun için özel, okunur yöntemler vardır.

Diziyi dolaşmanın yolları

- `for...of` — her öğeyi sırayla verir (en okunur).
- `forEach` — her öğe için bir fonksiyon çalıştırır.
- Klasik `for` (indeksle) — indekse ihtiyaç varsa.

Diziyi dolaşmak

```
const meyveler = ["elma", "armut", "kiraz"];
for (const meyve of meyveler) {
  console.log(meyve);
}
meyveler.forEach(m => console.log(m));
```

Şema 12.1 — Dizi döngüsü: her öğe sırayla işlenir.

İPUCU

Öğelerin değerine ihtiyacın varsa `for...of` en temiz seçimdir; indekse (kaçıncı öğe?) ihtiyacın varsa klasik `for` veya `forEach((oge, i) => ...)` kullan. Seviye 4'te `map` / `filter` ile bu işleri daha da güçlü yapacaksın.

Konsolda ne görürsün? ÇIKTI

Her iki döngü de konsola sırayla **elma**, **armut**, **kiraz** yazar. `for...of` sana doğrudan öğeyi verir; indeksle uğraşmana gerek kalmaz.

Alıştırma 10 dk

Diziyi dolaş:

1. Bir diziyi `for...of` ile yazdır.
2. Aynısını `forEach` ile yap.
3. Bir sayı dizisinin toplamını bir döngüyle hesapla.

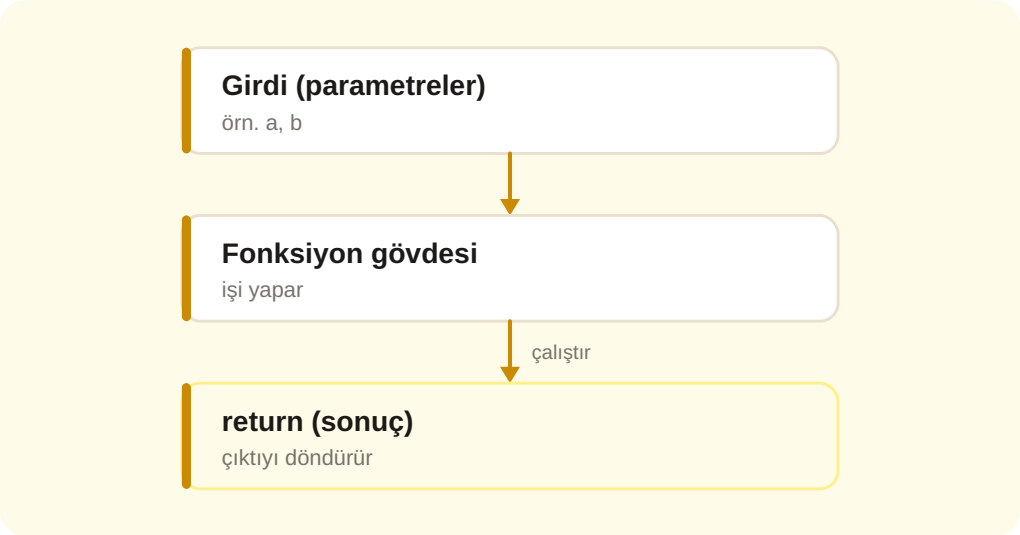
BÖLÜM 13

Fonksiyonlar

Fonksiyon, bir işi yapan ve yeniden kullanabileceğin bir kod paketidir: girdi alır, iş yapar, sonuç döndürür. Kodu düzenli, tekrarsız ve test edilebilir kılar.

Bir fonksiyonun parçaları

- **Parametre:** fonksiyona giren değer(ler).
- **Gövde:** yapılan iş.
- **return:** geri döndürülen sonuç.



Şema 13.1 — Fonksiyon bir makine gibidir: girdi → işlem → çıktı.

Fonksiyon tanımlama ve çağırma

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

İPUCU

İyi bir fonksiyon **tek bir işi** iyi yapar ve adı ne yaptığını söyler (`topla` , `kullaniciGetir`). Aynı kodu iki kez yazdığını fark edersen, onu bir fonksiyona taşı. `return` unutulursa fonksiyon `undefined` döndürür — sonuç lazımsa `return` yazmayı unutma.

Konsolda ne görürsün? ÇIKTI

`topla(5, 3)` çağrısı **8** döndürür ve `sonuc` 'a atanır; konsola **8** yazılır. Aynı fonksiyonu farklı sayılarla tekrar tekrar çağırabilirsin.

Alıştırma 12 dk

Fonksiyon yaz:

1. İki sayıyı çarpan bir fonksiyon yaz.
2. Bir isim alıp "Merhaba, [isim]" döndüren fonksiyon yaz.
3. Fonksiyonunu farklı değerlerle birkaç kez çağır.

BÖLÜM 14

Kapsam ve Ok Fonksiyonları

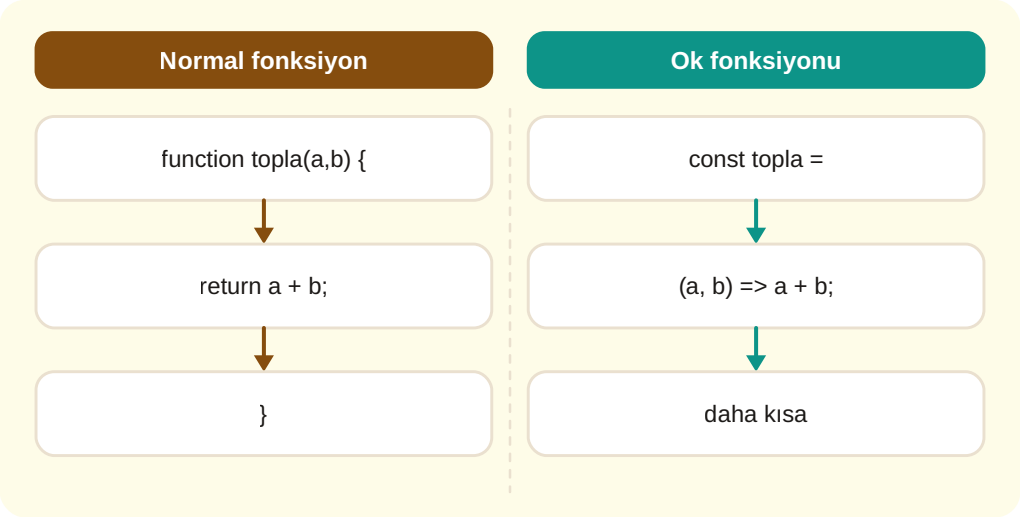
Değişkenlerin nerede görünür olduğu (kapsam) ve fonksiyon yazmanın modern, kısa yolu (ok fonksiyonları), temiz JavaScript için bilmen gereken iki kavramdır.

Kapsam (scope)

- Bir fonksiyon/blok içinde tanımlanan değişken, sadece orada görünür.
- Dışarıdan o değişkene erişilemez (bilgi gizleme).
- Bu, isim çakışmalarını ve hataları önler.

Ok fonksiyonları (arrow functions)

- Daha kısa söz dizimi: `(a, b) => a + b`.
- Özellikle `map`, `filter` gibi yerlerde çok kullanılır.



Şema 14.1 — Aynı iş, iki yazım: normal ve ok fonksiyonu.

Ok fonksiyonu

```
const kare = x => x * x;
console.log(kare(4));        // 16
const selam = isim => "Merhaba, " + isim;
```

İPUCU

Tek satırlık, kısa işlemler için ok fonksiyonları idealdir ve çok okunur: `sayilar.map(n => n * 2)`. Kapsam ise "değişkenimi neden dışarıdan göremiyorum?" sorusunun cevabıdır — değişkenler tanımlandıkları blokta yaşar. İkisi de modern JS'in günlük parçasıdır.

Konsolda ne görürsün? ÇIKTI

`kare(4)` konsola **16** yazar. Ok fonksiyonu, `function` ve `return` yazmadan aynı sonucu daha kısa verir — özellikle dizi metotlarında bunu çok kullanacaksın.

Alıştırma 10 dk

Modern yaz:

1. Bir normal fonksiyonu ok fonksiyonuna çevir.
2. Bir ok fonksiyonuyla bir sayının karesini al.
3. Bir fonksiyon içindeki değişkene dışarıdan erişmeyi dene (kapsamı gözlemle).

SEVİYE 3

Sayfayla Etkileşim (DOM)

JavaScript'i sayfaya bağlamak: DOM nedir, element seçme, içerik ve stil değiştirme, olaylar, form ve girdi ile eleman oluşturma.

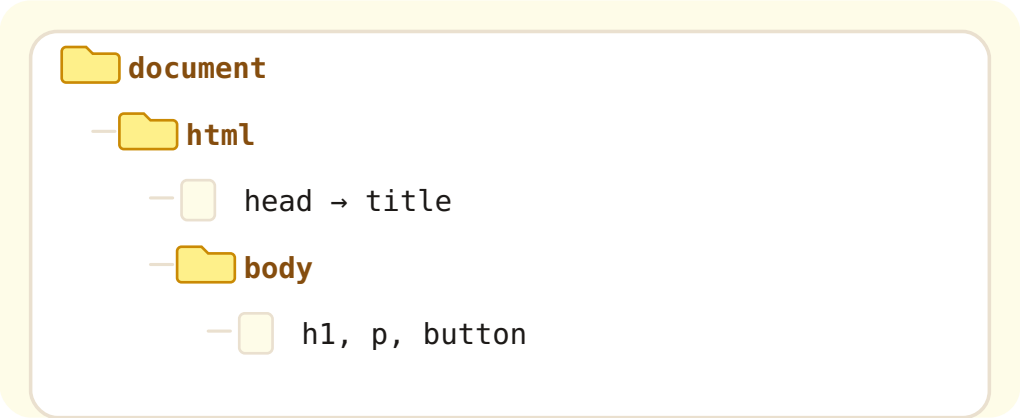
BÖLÜM 15

DOM Nedir?

Tarayıcı, HTML sayfanı bir ağaç yapısına dönüştürür: DOM (Document Object Model). JavaScript bu ağaca erişip öğeleri okuyabilir ve değiştirebilir — işte sayfayı "canlı" yapan budur.

Sayfa bir ağaçtır

- `document` — tüm sayfayı temsil eden nesne.
- Her HTML etiketi bir "düğüm" (node) olur.
- JS, bu düğümleri seçip değiştirebilir.



Şema 15.1 — DOM: HTML sayfasının ağaç biçiminde temsili.

DOM'a ilk dokunuş

```
console.log(document.title);   // sayfa başlığı
console.log(document.body);    // body öğesi
```

İPUCU

DOM, HTML'in **canlı bir kopyası** gibidir: JavaScript ile DOM'u değiştirdiğinde, sayfa anında güncellenir (dosyayı kaydetmeden, yenilemeden). Bu yüzden DOM'u anlamak, etkileşimli sayfalar yapmanın temelidir. `document` her şeyin başladığı yerdir.

Konsolda ne görürsün? ÇIKTI

`document.title` konsola sayfanın `<title>`'ındaki metni yazar. `document` üzerinden sayfanın her öğesine ulaşabilirsin — sonraki bölümde nasıl seçeceğini göreceksin.

◎ Alıştırma

8 dk

DOM'u tanı:

1. Konsola `document.title`'ı yazdır.
2. `document.body`'yi inceleyip ağaç yapısını gör.
3. DOM'un ne olduğunu kendi cümlelerinle açıkla.

BÖLÜM 16

Element Seçme

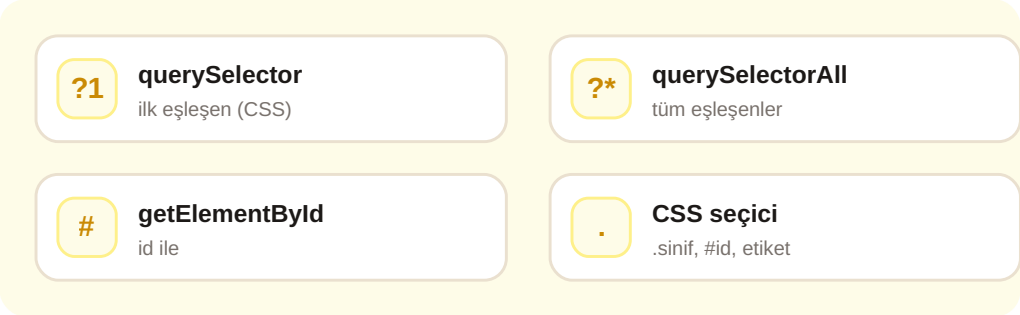
Bir öğeyi değiştirmeden önce onu "seçmen" gerekir. JavaScript, CSS seçicilerine benzer yöntemlerle sayfadaki öğeleri bulmanı sağlar.

Seçim yöntemleri

- `querySelector(".sinif")` — CSS seçiciyle ilk eşleşeni bulur (en yaygın).
- `querySelectorAll("p")` — eşleşen tüm öğeleri bulur (liste).
- `getElementById("id")` — id ile tek öğe.

Öğe seçme

```
const baslik = document.querySelector("h1");
const dugme = document.querySelector("#gonder");
const tumKartlar = document.querySelectorAll(".kart");
console.log(baslik.textContent);
```



Şema 16.1 — Element seçme yöntemleri.

İPUCU

`querySelector` ve `querySelectorAll`'ı öğrenmen çoğu zaman yeterli; CSS seçicilerinin aynısını kullandıkları için (`.sinif`, `#id`, `etiket`) öğrenmesi kolaydır. Seçtiğin öğeyi bir değişkende tut; böylece ona tekrar tekrar erişebilirsin.

Konsolda ne görürsün? ÇIKTI

`querySelector("h1")` sayfadaki ilk `<h1>`'i seçer; `baslik.textContent` ise o başlığın metnini konsola yazar. Seçmek, değiştirmenin ilk adımıdır.

◎ Alıştırma

10 dk

Öğe seç:

1. Bir başlığı `querySelector` ile seç ve metnini yazdır.
2. Bir id'li öğeyi seç.
3. `querySelectorAll` ile birden çok öğe seç.

BÖLÜM 17

İçerik ve Stil Değiştirme

Öğeyi seçtikten sonra onu değiştirebilirsin: metnini, içeriğini, sınıfını veya stilini. İşte sayfayı dinamik olarak güncellemek budur.

Değiştirme yolları

- `textContent` — öğenin metnini değiştirir (güvenli).
- `innerHTML` — HTML içeriğini değiştirir (dikkatli kullan).
- `classList.add/remove/toggle` — CSS sınıfı ekler/çıkarır.
- `style.renk` — doğrudan stil (genelde sınıf tercih edilir).

İçerik ve stili güncelleme

```
const baslik = document.querySelector("h1");
baslik.textContent = "Yeni Başlık";
baslik.classList.add("vurgulu");
baslik.style.color = "#CA8A04";
```

Şema 17.1 — Bir öğenin içeriğini ve görünümünü değiştirme yolları.

İPUCU

Metin değiştirmek için `textContent`'i tercih et; `innerHTML` güçlüdür ama dışarıdan gelen (kullanıcı) veriyi doğrudan `innerHTML`'e koymak **güvenlik açığıdır** (XSS). Görünüm değişikliklerini genelde `classList` ile yap (CSS'te tanımlı sınıfları aç/kapat) — bu, stili ve mantığı ayrı tutar.

Konsolda ne görürsün? ÇIKTI

Bu kod çalışınca sayfadaki başlığın metni anında **"Yeni Başlık"** olur, "vurgulu" sınıfı eklenir ve rengi altın olur — hepsi sayfa yenilenmeden. JavaScript ile DOM'u değiştirmek, sayfayı canlandırır.

Alıştırma

10 dk

Sayfayı değiştir:

1. Bir öğenin `textContent`'ini değiştir.
2. `classList.toggle` ile bir sınıfı aç/kapat.
3. Bir öğeye `style` ile renk ver.

BÖLÜM 18

Olaylar (Events)

Etkileşimin kalbi olaylardır: kullanıcı bir düğmeye tıkladığında, bir tuşa bastığında veya formu gönderdiğinde, JavaScript buna tepki verebilir. `addEventListener` bunun için kullanılır.

Olay dinleme

- Bir öğe seç, ona bir "dinleyici" ekle.
- `addEventListener("click", fonksiyon)` — tıklamayı dinler.
- Olay olunca, verdiğin fonksiyon çalışır.

Şema 18.1 — Olay akışı: kullanıcı eylemi → olay → fonksiyon.

Tıklamaya tepki vermek

```
 const dugme = document.querySelector("#gonder");
dugme.addEventListener("click", () => {
  console.log("Düğmeye tıklandı!");
});
```

İPUCU

`addEventListener` en çok kullanacağın yapıdır. Olay türleri çoktur: `"click"`, `"input"` (yazarken), `"submit"` (form), `"keydown"` (tuş). Dinleyici olarak genelde ok fonksiyonu verirsin. Bu yapı, statik bir sayfayı kullanıcıya tepki veren bir uygulamaya dönüştürür.

Konsolda ne görürsün? ÇIKTI

Bu kod eklendikten sonra, kullanıcı her "Gönder" düğmesine tıkladığında konsola **Düğmeye tıklandı!** yazılır. Dinleyici, sen kaldırana kadar her tıklamada tekrar çalışır.

Alıştırma

12 dk

Olay dinle:

1. Bir düğmeye `click` dinleyicisi ekle.
2. Tıklayınca bir metni değiştir (DOM + olay).
3. Bir sayacı tıkladıkça artıran bir düğme yap.

BÖLÜM 19

Form ve Girdi

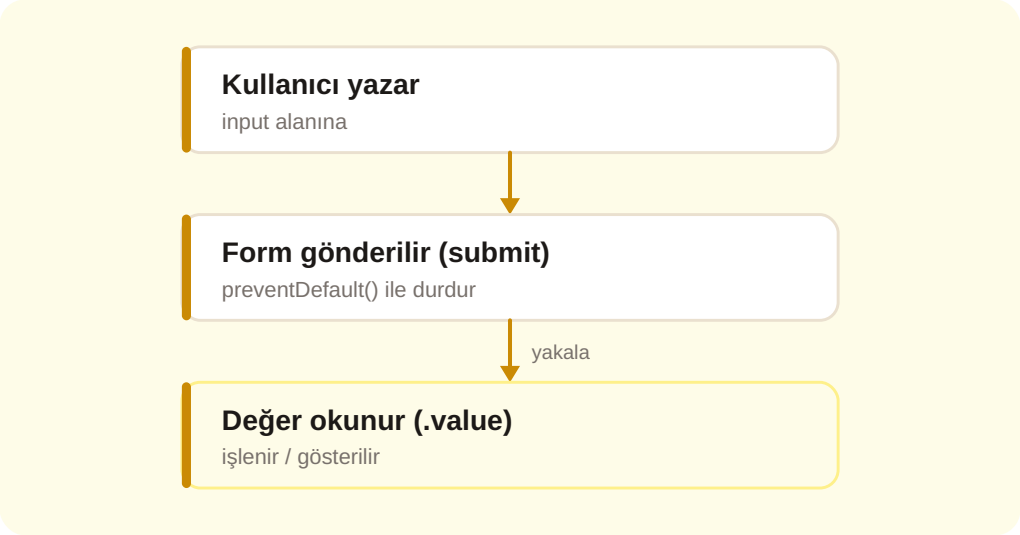
Kullanıcıdan veri almak (giriş, arama, yorum) formlarla olur. JavaScript ile girdiyi okuyabilir, form gönderimini kontrol edebilir ve veriyi işleyebilirsin.

Girdiyi okumak

- Bir `<input>` seç, değerini `.value` ile oku.
- `submit` olayını dinle.
- `event.preventDefault()` — sayfanın yenilenmesini engelle.

Form girdisini işlemek

```
 const form = document.querySelector("#form");
const girdi = document.querySelector("#ad");
form.addEventListener("submit", (e) => {
  e.preventDefault();
  console.log("Merhaba, " + girdi.value);
});
```



Şema 19.1 — Form işleme akışı: yaz → gönder → değeri oku.

İPUCU

Form gönderiminde `e.preventDefault()` kritiktir: yoksa tarayıcı sayfayı yeniler ve JavaScript'in işi yarıda kalır. Girdiyi **her zaman doğrula** (boş mu, geçerli mi?) ve dışarıdan gelen veriye güvenme — bu, güvenli kodun temelidir (Seviye 4).

Konsolda ne görürsün? ÇIKTI

Kullanıcı adını yazıp formu gönderince sayfa yenilenmez (`preventDefault` sayesinde) ve konsola **Merhaba, [yazılan ad]** yazılır. Böylece girdiyi alıp dilediğin gibi işleyebilirsin.

Alıştırma 12 dk

Formu işle:

1. Bir input'un değerini `.value` ile oku.
2. `submit`'i dinleyip `preventDefault` kullan.
3. Girilen adı sayfada bir yere yazdır.

BÖLÜM 20

Eleman Oluşturma

JavaScript yalnızca var olan öğeleri değiştirmekle kalmaz; yenilerini oluşturup sayfaya ekleyebilir. Bir yapılacaklar listesine madde eklemek gibi dinamik içerikler böyle yapılır.

Yeni öğe ekleme adımları

- `document.createElement("li")` — yeni öğe oluştur.
- İçeriğini ayarla (`textContent`).
- `append()` ile bir ataya ekle. `remove()` ile sil.

Şema 20.1 — Yeni eleman oluşturma akışı: oluştur → doldur → ekle.

Listeye madde eklemek

```
 const liste = document.querySelector("#liste");
const madde = document.createElement("li");
madde.textContent = "Yeni görev";
liste.append(madde);   // listede görünür
```

İPUCU

Yeni oluşturulan öğe, `append` ile bir ataya eklenene kadar **sayfada görünmez** (sadece bellekte durur). Bu üç adım — oluştur, doldur, ekle — dinamik arayüzlerin temelidir: yorumlar, liste maddeleri, kartlar hep böyle eklenir. `remove()` ile de öğeyi kaldırırsın.

Konsolda ne görürsün? ÇIKTI

Bu kod çalışınca `#liste` öğesinin sonuna **"Yeni görev"** yazan yeni bir madde eklenir ve anında ekranda görünür. Bunu bir olay (örn. düğme tıklaması) ile birleştirince, kullanıcı kendi maddelerini ekleyebilir.

Alıştırma 12 dk

Eleman ekle:

1. `createElement` ile bir `<li>` oluştur.
2. İçeriğini ayarlayıp bir listeye `append` et.
3. Bir düğmeye basınca listeye madde ekleyen kod yaz.

SEVİYE 4

Modern JavaScript

Profesyonel JS: modern söz dizimi, dizi metotları (map/filter/reduce), asenkron JavaScript, fetch ile veri çekme, hatalar ile güvenli kod ve etkileşimli bir uygulama.

BÖLÜM 21

Modern Söz Dizimi

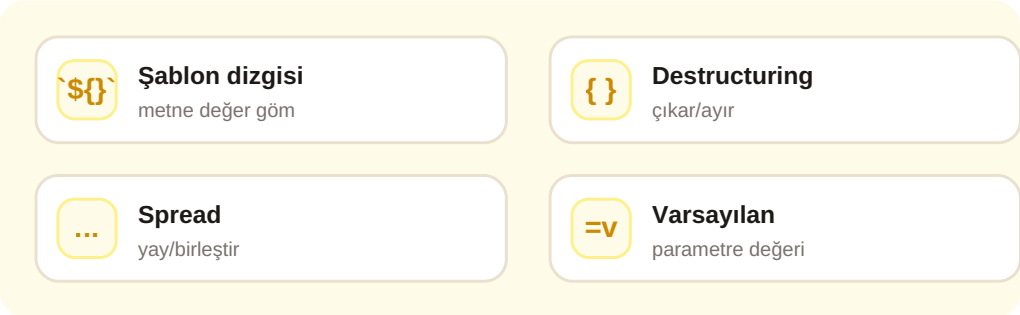
Modern JavaScript (ES6+), kodu daha kısa ve okunur yazmanı sağlayan güçlü araçlar getirdi. Bunlar her gün kullanılan, hayat kolaylaştıran özelliklerdir.

Sık kullanılan modern özellikler

- **Şablon dizgileri:** `` `Merhaba ${ad}` `` — değişkeni metne gömer.
- **Destructuring:** `const { ad, yas } = kullanici;`.
- **Spread:** `[...dizi1, ...dizi2]` — birleştir/kopyala.
- **Varsayılan parametre:** `function selam(ad = "Misafir")`.

Modern söz dizimi

```
const ad = "Ayşe", yas = 30;
console.log(`${ad}, ${yas} yaşında`);
const kullanici = { ad, yas };          // kısa nesne
const { sehir = "Bilinmiyor" } = kullanici;
const hepsi = [...[1,2], ...[3,4]];      // [1,2,3,4]
```



Şema 21.1 — Hayat kolaylaştıran modern JavaScript özellikleri.

İPUCU

Şablon dizgileri (`` ` ` `` ters tırnak) metin birleştirmeyi çok kolaylaştırır: `"Merhaba " + ad + "!"` yerine `` `Merhaba ${ad}!` ``. Destructuring ve spread ise özellikle dizilerle/nesnelerle çalışırken kodu kısaltır. Bunlar modern JavaScript'in günlük dilidir.

Konsolda ne görürsün? ÇIKTI

Şablon dizgisi konsola **Ayşe, 30 yaşında** yazar. Destructuring'de `sehir` nesnede olmadığından varsayılan **"Bilinmiyor"** kullanılır. Spread ise iki diziyi **[1,2,3,4]** olarak birleştirir.

◎ Alıştırma

10 dk

Modern yaz:

1. Bir şablon dizgisiyle bir cümle kur.
2. Bir nesneden destructuring ile değer çıkar.
3. Spread ile iki diziyi birleştir.

BÖLÜM 22

Dizi Metotları (map, filter, reduce)

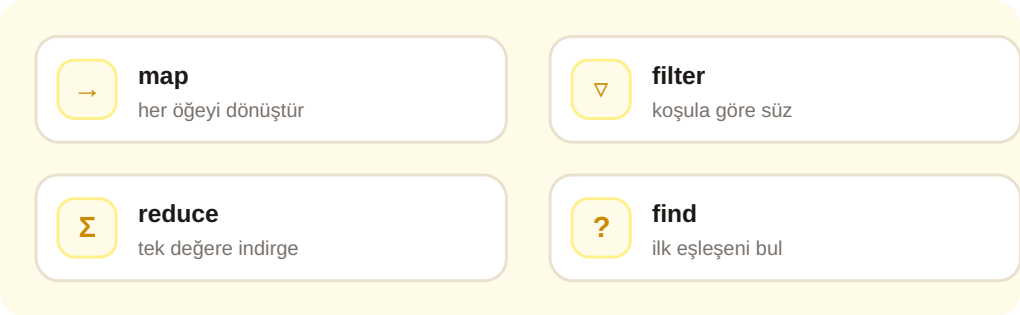
Modern JavaScript'in en güçlü araçlarından biri dizi metotlarıdır. map, filter ve reduce ile dizileri döngü yazmadan, okunur biçimde dönüştürür, süzer ve özetlersin.

Üç temel metot

- `map` — her öğeyi dönüştür, yeni dizi üret.
- `filter` — koşulu sağlayan öğeleri seç.
- `reduce` — diziyi tek bir değere indirge (örn. toplam).

map, filter, reduce

```
const sayilar = [1, 2, 3, 4];
const kareler = sayilar.map(n => n * n);      // [1,4,9,16]
const ciftler = sayilar.filter(n => n % 2 === 0); // [2,4]
const toplam = sayilar.reduce((a, n) => a + n, 0); // 10
```



Şema 22.1 — Dizileri dönüştüren, süzen ve özetleyen metotlar.

İPUCU

Bu metotlar **orijinal diziyi değiştirmez**, yeni bir sonuç üretir — bu, hataları azaltan güvenli bir yaklaşımdır. `map` "dönüştür", `filter` "ayıkla", `reduce` "topla/özetle" diye düşün. Bir kez alışınca, çoğu döngünün yerini bunlar alır ve kodun çok daha okunur olur.

Konsolda ne görürsün? ÇIKTI

`map` konsola **[1, 4, 9, 16]** (kareler), `filter` **[2, 4]** (çiftler) ve `reduce` **10** (toplam) verir. Hepsi tek satırda, döngü yazmadan.

◎ Alıştırma

12 dk

Dizileri dönüştür:

1. Bir sayı dizisini `map` ile iki katına çıkar.
2. `filter` ile 10'dan büyükleri seç.
3. `reduce` ile toplamı hesapla.

BÖLÜM 23

Asenkron JavaScript

Bazı işler zaman alır: bir sunucudan veri çekmek, bir dosya okumak. JavaScript bunları beklerken donmaz; işi "arka planda" yürütür ve bittiğinde devam eder. Buna asenkron çalışma denir.

Asenkron kavramı

- Uzun işler (örn. ağ isteği) beklenirken kod akmaya devam eder.
- **Promise:** "ileride gelecek bir sonuç" sözü.
- **async / await:** asenkron kodu okunur biçimde yazma yolu.

Şema 23.1 — Asenkron akış: beklerken bloklamaz, sonuç gelince devam eder.

async / await

```
async function veriGetir() {
  const yanit = await fetch("/veri.json");
  const veri = await yanit.json();
  console.log(veri);
}
```

İPUCU

`await` "bu iş bitene kadar bekle, ama programın geri kalanını dondurmadan" demektir ve yalnızca `async` fonksiyon içinde kullanılır. Asenkron kavramı başta soyut gelir; ama "veriyi iste, gelince kullan" kalıbını bir kez kurunca netleşir. Sonraki bölümde gerçek bir API'den veri çekeceksin.

Konsolda ne görürsün? ÇIKTI

`veriGetir()` çağrıldığında, `fetch` verisi gelene kadar fonksiyon "bekler" ama tarayıcı donmaz; veri gelince konsola yazılır. Bu, modern web uygulamalarının sunucuyla konuşma biçimidir.

Alıştırma 10 dk

Asenkronu kavra:

1. Asenkron çalışmanın neden gerektiğini kendi cümlelerinle yaz.
2. Bir `async` fonksiyon iskeleti yaz.
3. `await`'in ne işe yaradığını açıkla.

BÖLÜM 24

Veri Çekme (fetch & API)

Modern uygulamalar verilerini genelde bir sunucudan/API'den çeker: hava durumu, ürün listesi, kullanıcı bilgisi. `fetch` ile bu veriyi alır, JSON olarak okur ve sayfanda kullanırsın.

fetch akışı

- `fetch(url)` — bir adrese istek gönderir.
- Yanıt gelir; `.json()` ile JSON verisine çevrilir.
- **API:** verinin alındığı uç nokta (genelde JSON döner).

Şema 24.1 — fetch akışı: istek → yanıt → JSON → kullan.

API'den veri çekmek

```
async function urunleriGetir() {
  const yanit = await fetch("https://api.ornek.com/urunler");
  const urunler = await yanit.json();
  urunler.forEach(u => console.log(u.ad));
}
```

İPUCU

JSON (JavaScript Object Notation), web'de verinin taşındığı standart biçimdir ve JavaScript nesnelerine çok benzer. `fetch` her zaman başarılı olmaz (ağ hatası, 404...); bu yüzden gerçek kodda `try/catch` (sonraki bölüm) ile hataları ele almak gerekir. Çoğu ücretsiz API ile pratik yapabilirsin.

Konsolda ne görürsün? ÇIKTI

`urunleriGetir()` çağrılınca API'den ürün listesi gelir ve her ürünün adı konsola yazılır. Gerçek bir uygulamada bu veriyi DOM'a (Seviye 3) ekleyerek sayfada gösterirsin.

Alıştırma 12 dk

Veri çek:

1. Bir `fetch` + `await` + `.json()` akışı yaz.
2. Gelen veriyi `console.log` ile incele.
3. Gelen bir listeyi `forEach` ile dolaş.

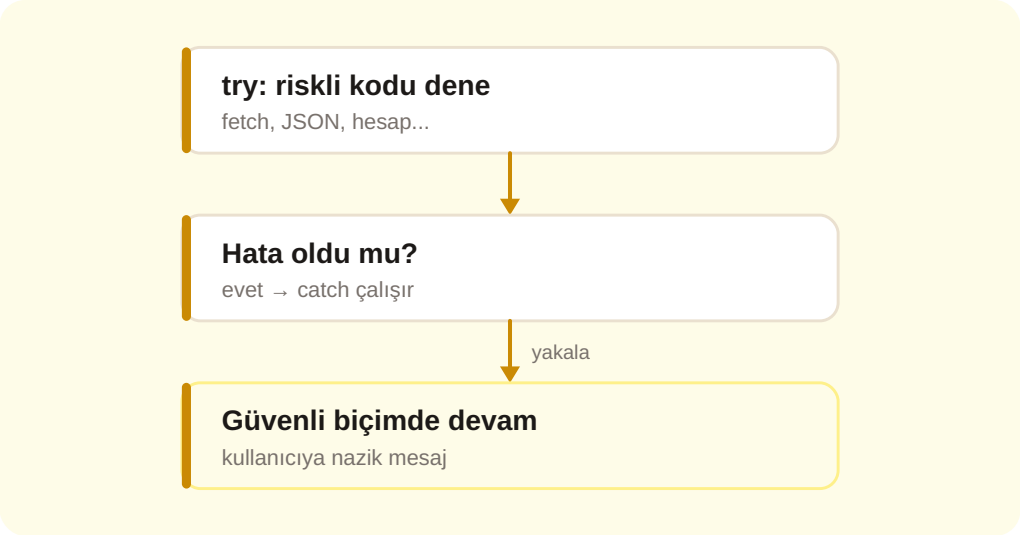
BÖLÜM 25

Hatalar ve Güvenli Kod

Gerçek dünyada işler ters gider: ağ kopar, kullanıcı hatalı veri girer, beklenmeyen değerler gelir. Sağlam kod, bu durumları öngörür ve nazikçe ele alır.

Hataları ele almak

- `try { ... } catch (e) { ... }` — riskli kodu güvenle çalıştır.
- Hata olursa `catch` bloğu devreye girer; program çökmez.
- Kullanıcı girdisini **her zaman doğrula** (boş, tür, aralık).



Şema 25.1 — try/catch: hata olsa bile program çökmeden devam eder.

try / catch ile güvenli kod

```
async function veriGetir() {
  try {
    const yanit = await fetch("/veri.json");
    if (!yanit.ok) throw new Error("Sunucu hatası");
    return await yanit.json();
  } catch (e) {
    console.log("Veri alınamadı:", e.message);
  }
}
```

İPUCU

İki ilke: **(1)** hata olabilecek her yeri (ağ, JSON çevirme, dış veri) `try/catch` ile sar; **(2)** kullanıcıdan veya dışarıdan gelen hiçbir veriye güvenme — doğrula. Güvenlik (örn. `innerHTML`'e ham veri koymama) ve sağlamlık, amatör ile profesyonel kodu ayıran en önemli farklardandır.

Konsolda ne görürsün? ÇIKTI

Veri başarıyla gelirse fonksiyon onu döndürür; bir hata olursa (ağ koptu, sunucu hatası) program çökmez — `catch` bloğu çalışır ve konsola **Veri alınamadı: [sebep]** yazılır. Kullanıcı, çöken bir sayfa yerine nazik bir mesaj görür.

Alıştırma 10 dk

Güvenli yaz:

1. Riskli bir işi `try/catch` içine al.
2. Bir kullanıcı girdisini (boş mu?) doğrula.
3. Hata durumunda kullanıcıya nazik bir mesaj göster.

BÖLÜM 26

Bitirme: Etkileşimli Bir Uygulama

Tüm JavaScript bilgini birleştirip küçük, gerçek bir uygulama kuruyorsun: örneğin bir "yapılacaklar" listesi. Seçme, olay, DOM güncelleme ve veri yönetimi bir arada.

Bir uygulamanın iskeleti

- **Veri:** bir dizide görevleri tut.
- **Olay:** "Ekle" düğmesini ve girdiyi dinle.
- **DOM:** her görevi listeye ekle; silmeyi bağla.

Şema 26.1 — Etkileşimli uygulama döngüsü: seç → dinle → veri → DOM.

Mini "yapılacaklar" mantığı

```
 const girdi = document.querySelector("#yeni");
const liste = document.querySelector("#liste");
document.querySelector("#ekle").addEventListener("click", () => {
  if (!girdi.value) return;          // doğrula
  const li = document.createElement("li");
  li.textContent = girdi.value;
  liste.append(li);
  girdi.value = "";                  // alanı temizle
});
```

İPUCU

Bu küçük uygulama, bu modülün tamamını bir araya getirir: değişkenler, koşul (doğrulama), olaylar, DOM ve eleman oluşturma. Gerçek projeler de aynı döngüyü izler — sadece daha büyük ölçekte. Bunu yapabiliyorsan, JavaScript'in temelini sağlam attın demektir. Sıradaki adım: algoritma düşüncesini güçlendirmek (Modül 6).

Konsolda ne görürsün? ÇIKTI

Kullanıcı bir görev yazıp "Ekle"ye basınca: girdi boş değilse yeni bir madde oluşturulur, listeye eklenir ve giriş alanı temizlenir — hepsi anında, sayfa yenilenmeden. İşte etkileşimli bir web uygulaması.

Alıştırma 20 dk

Uygulamanı kur:

1. Bir input, bir "Ekle" düğmesi ve bir liste içeren HTML yap.
2. Düğmeye tıklayınca girdiyi listeye madde olarak ekle.
3. Boş girdiyi engelle (doğrulama) ve ekledikten sonra alanı temizle.
4. İstersen her maddeye bir "sil" düğmesi ekle.

EK

JavaScript Sözlüğü

En sık kullanılan JavaScript yapıları ve işlevleri. Bir başvuru kaynağı olarak saklayabilirsin.

let / const	`Değişken tanımlama`	typeof	`Bir değerin türü`
=== / !==	`Eşitlik karşılaştırma`	&& / || / !	`Mantıksal operatörler`
if / else	`Koşullar`	for / while	`Döngüler`
function / =>	`Fonksiyon / ok fonksiyonu`	dizi.push / .map	`Dizi metotları`
querySelector	`Element seçme (DOM)`	addEventListener	`Olay dinleme`
fetch()	`Veri çekme (API)`	try / catch	`Hata yakalama`

JavaScript'in özeti ÇIKTI

JavaScript, **değişkenlerle** veriyi tutar, **operatör ve koşullarla** karar verir, **döngü ve fonksiyonlarla** işi düzenler ve **DOM ile** sayfayı canlı biçimde değiştirir. HTML iskeletine ve CSS görünümüne bu davranış katmanı eklenince, statik bir sayfa gerçek bir uygulamaya dönüşür.