

Fonksiyonlar, Kapsam, Diziler ve Nesneler

Fonksiyon tanımlama ■ Ok fonksiyonları ■ Kapsam ve closure ■ , , ,
Nesne yapıları

Prof. Dr. Özal YILDIRIM

Fırat Üniversitesi — Teknoloji Fakültesi

Yazılım Mühendisliği Bölümü

✓ Hafta 10 Özeti

- ▶ `const/let`, veri tipleri
- ▶ Operatörler, `===` kuralı
- ▶ `if/else`, `switch`
- ▶ `for`, `while`, `for...of`

Artık **akış kontrolünü** biliyoruz.

→ Bu Hafta

Kodu **tekrar kullanılabilir parçalara** bölmeyi ve **veriyi düzenli tutmayı** öğreniyoruz. **Fonksiyonlar + Diziler** + **Nesneler** — gelecek haftanın DOM çalışmasının **temeli**.

*“Aynı kodu ikinci kez yazıyorsanız, **onu bir fonksiyona dönüştürme zamanı** gelmiştir.”*

- ▶ Fonksiyon tanımlamanın üç yöntemini karşılaştırabilecek,
- ▶ Parametre, argüman ve `return` kavramlarını doğru kullanabilecek,
- ▶ Ok fonksiyonlarını (`=>`) kısa yazımlarıyla yazabilecek,
- ▶ Global, fonksiyon ve blok kapsamını ayırt edebilecek,
- ▶ Dizi metotlarını (`push`, `find`, `map`, `filter`, `reduce`, `sort`) uygulayabilecek,
- ▶ Nesne oluşturup özelliklerine erişebilecek ve güncelleyebilecek,
- ▶ Yıkım ataması (destructuring) ve yayılma (`...`) operatörünü kullanabilecek,
- ▶ Nesne dizileri üzerinde zincirleme işlemler yapabileceksiniz.

Bu Haftanın İçeriği

- 1 Fonksiyonlar
- 2 Kapsam (Scope)
- 3 Diziler
- 4 Nesneler
- 5 Yapay Zeka ile Fonksiyonlar
- 6 Özet ve Değerlendirme



Fonksiyonlar

BÖLÜM 1

Neden Fonksiyon?

✗ Tekrar eden kod

```
let kdv1 = 89.90 * 0.20;
let top1 = 89.90 + kdv1;
console.log("Toplam: " + top1.toFixed(2));

let kdv2 = 45.00 * 0.20;
let top2 = 45.00 + kdv2;
console.log("Toplam: " + top2.toFixed(2));

let kdv3 = 120.00 * 0.20;
let top3 = 120.00 + kdv3;
console.log("Toplam: " + top3.toFixed(2));
```

✓ Fonksiyon ile

```
function kdvliFiyat(fiyat, oran = 0.20) {
    return fiyat + fiyat * oran;
}

console.log(kdvliFiyat(89.90).toFixed(2));
console.log(kdvliFiyat(45.00).toFixed(2));
console.log(kdvliFiyat(120.00).toFixed(2));

// KDV oranı değişirse TEK yerde değişir
```

💡 İpucu

DRY ilkesi: Don't Repeat Yourself — Kendini Tekrar Etme. Fonksiyonların üç kazancı: **tekrar kullanılabilirlik**, **tek noktadan değişiklik**, **okunabilirlik**.

Fonksiyon Tanımlamanın Üç Yolu

```
1 // 1) Fonksiyon BILDIRIMI (declaration) - "hoisted" edilir
2 function selamla(ad) {
3   return `Merhaba ${ad}!`;
4 }
5
6 // 2) Fonksiyon IFADESI (expression) - degiskene atanir
7 const selamla2 = function(ad) {
8   return `Merhaba ${ad}!`;
9 };
10
11 // 3) OK FONKSIYONU (arrow function) - modern, kisa
12 const selamla3 = (ad) => {
13   return `Merhaba ${ad}!`;
14 };
15
16 // Ok fonksiyonu kisaltmalari
17 const selamla4 = ad => `Merhaba ${ad}!`; // tek parametre + tek satir
18 const zar      = () => Math.ceil(Math.random() * 6); // parametresiz
19 const topla    = (a, b) => a + b;
```

Tek satırlık ok fonksiyonlarında süslü parantez ve `return` yazılmaz — **örtülü return** vardır.

Parametreler ve return

```
1 // Varsayılan parametre değeri
2 function indirimliFiyat(fiyat, oran = 0.10) {
3   return fiyat * (1 - oran);
4 }
5 indirimliFiyat(100);           // 90   (oran varsayılan 0.10)
6 indirimliFiyat(100, 0.25);    // 75
7
8 // Birden fazla değer dondurmek: NESNE veya DIZI kullanın
9 function kitapAnaliz(kitaplar) {
10   const toplam    = kitaplar.reduce((t, k) => t + k.fiyat, 0);
11   const ortalama  = toplam / kitaplar.length;
12   return { toplam, ortalama, adet: kitaplar.length };
13 }
14
15 const sonuc = kitapAnaliz(kitaplar);
16 console.log(sonuc.ortalama);
17
18 // Belirsiz sayıda parametre: REST operatörü
19 function enBuyuk(...sayilar) {
20   return Math.max(...sayilar);
21 }
22 enBuyuk(3, 9, 1, 7);    // 9
```

Ok Fonksiyonları: Ne Zaman, Ne Zaman Değil?

Durum	Tercih
Dizi metotları (<code>map</code> , <code>filter</code>)	Ok fonksiyonu — kısa ve okunaklı
Olay dinleyicileri (Hafta 12)	Ok fonksiyonu — <code>this</code> sorunu yaşatmaz
Nesne metotları	Klasik function — <code>this</code> doğru çalışsın
Uzun, karmaşık mantık	Klasik function — adı hata mesajında görünür

```
1 // Ok fonksiyonlarının KENDI `this` degeri YOKTUR
2 const kitap = {
3   ad: "Simyaci",
4   yazdirKlasik: function() { console.log(this.ad); }, // "Simyaci"
5   yazdirOk:    () => { console.log(this.ad); }        // undefined (!)
6 };
7
8 // Kisa yazim (nesne metodu icin en iyisi)
9 const kitap2 = {
10   ad: "Simyaci",
11   yazdir() { console.log(this.ad); } // "Simyaci"
12 };
```



Kapsam (Scope)

BÖLÜM 2

Üç Kapsam Türü

```
1  const globalDegisken = "Her yerden erisilir";    // GLOBAL kapsam
2
3  function ornekFonksiyon() {
4      const fonksiyonIci = "Sadece bu fonksiyon icinde";    // FONKSIYON kapsamı
5      console.log(globalDegisken);    // OK - disaridan icerisi gorulur
6
7      if (true) {
8          let blokIci = "Sadece bu blok icinde";            // BLOK kapsamı
9          const dahaDerin = 42;
10         console.log(fonksiyonIci);    // OK
11     }
12
13     // console.log(blokIci);            // HATA: blokIci is not defined
14 }
15
16 // console.log(fonksiyonIci);    // HATA: fonksiyonIci is not defined
```

Kural: İçeriden dışarıya bakılabilir, dışarıdan içeriye bakılamaz.



Dikkat

Kapsam Zinciri ve Gölgeleme

```
1  const ad = "Global";
2
3  function dis() {
4      const ad = "Dis fonksiyon";
5
6      function ic() {
7          const ad = "Ic fonksiyon";
8          console.log(ad);          // "Ic fonksiyon"
9      }
10
11      ic();
12      console.log(ad);              // "Dis fonksiyon"
13  }
14
15  dis();
16  console.log(ad);                  // "Global"
```

Q Kapsam Zinciri

JavaScript bir değişkeni ararken:

- 1 **Kendi kapsamına** bakar
- 2 Bulamazsa **bir üst** kapsama
- 3 Global'e kadar devam eder
- 4 Yine bulamazsa **ReferenceError**

⚠ Dikkat

Gölgeleme (shadowing): İç kapsamdaki aynı isimli değişken, dıştakini **gizler**. Karışıklığa yol açar — **farklı isimler** kullanın.

Global kirliliğinden kaçının: Global kapsamda çok fazla değişken tanımlamak, **isim çakışmalarına** ve **takip edilemez hatalara** yol açar.



Diziler

BÖLÜM 3

Dizi Temelleri

```
1  const kitaplar = ["Simyaci", "Kurk Mantolu", "Tutunamayanlar"];
2
3  kitaplar[0];           // "Simyaci"      (0'dan baslar!)
4  kitaplar[kitaplar.length - 1]; // son eleman
5  kitaplar.length;       // 3
6  kitaplar.at(-1);       // "Tutunamayanlar" (modern: sondan eris)
7
8  // EKLEME / CIKARMA (diziyi DEGISTIRIR)
9  kitaplar.push("Beyaz Kale"); // SONA ekle
10 kitaplar.pop();             // SONDAN cikar
11 kitaplar.unshift("Yeni");   // BASA ekle
12 kitaplar.shift();           // BASTAN cikar
13 kitaplar.splice(1, 1);      // 1. indeksten 1 eleman sil
14
15 // ARAMA
16 kitaplar.indexOf("Simyaci"); // 0 (yoksa -1)
17 kitaplar.includes("Simyaci"); // true
18 kitaplar.join(", ");        // "Simyaci, Kurk Mantolu, ..."
19
20 // KOPYALAMA (orijinali degistirmez)
21 const kopya = [...kitaplar]; // yayilma (spread)
22 const dilim = kitaplar.slice(0, 2);
```

Modern Dizi Metotları: forEach ve map

```
1  const fiyatlar = [89.90, 45.00, 120.00, 65.50];
2
3  // forEach: her eleman için bir şey YAP - değer DONDURMEZ
4  fiyatlar.forEach((fiyat, indeks) => {
5    console.log(`${indeks + 1}. urun: ${fiyat} TL`);
6  });
7
8  // map: her elemanı DONUSTUR - YENİ dizi dondurur
9  const kdvliFiyatlar = fiyatlar.map(fiyat => fiyat * 1.20);
10 // [107.88, 54, 144, 78.6]
11
12 const etiketler = fiyatlar.map(f => `${f.toFixed(2)} TL`);
13 // ["89.90 TL", "45.00 TL", "120.00 TL", "65.50 TL"]
14
15 // Nesne dizisinde
16 const kitapAdlari = kitaplar.map(kitap => kitap.ad);
```

⚠ Dikkat

En sık karıştırılan ikili: `forEach` bir işlem yapar ve `undefined` döndürür; `map` bir dönüşüm yapar ve **yeni bir dizi** döndürür. `const x = fiyatlar.forEach(...)` yazarsanız `x undefined`

filter, find ve reduce

```
1  const kitaplar = [  
2    { ad: "Simyaci",      fiyat: 89.90,  puan: 8.4,  tur: "roman" },  
3    { ad: "Kurk Mantolu", fiyat: 45.00,  puan: 9.1,  tur: "roman" },  
4    { ad: "Tutunamayanlar", fiyat: 120.00, puan: 9.5,  tur: "roman" },  
5    { ad: "Kozmos",      fiyat: 75.00,  puan: 9.3,  tur: "bilim" }  
6  ];  
7  
8  // filter: kosula uyanlari SEC - YENI dizi  
9  const begenilenler = kitaplar.filter(k => k.puan >= 9);           // 3 kitap  
10 const ucuzlar       = kitaplar.filter(k => k.fiyat < 80);          // 2 kitap  
11  
12 // find: kosula uyan ILK elemani bul - TEK nesne (yoksa undefined)  
13 const aranan = kitaplar.find(k => k.ad === "Kozmos");  
14  
15 // some / every: mantiksal kontrol - true/false  
16 kitaplar.some(k => k.fiyat > 100);    // true  - en az biri  
17 kitaplar.every(k => k.puan > 8);      // true  - hepsi  
18  
19 // reduce: diziyi TEK bir degere indirge  
20 const toplam = kitaplar.reduce((biriken, k) => biriken + k.fiyat, 0);  
21 //                                     ~birikimli      ~mevcut          ~baslangic  
22 const enPahali = kitaplar.reduce((a, b) => a.fiyat > b.fiyat ? a : b);
```

Metot Zincirleme

```
1 // "Puani 9'un üzerindeki romanların adlarını alfabetik sırala"
2 const sonuc = kitaplar
3   .filter(k => k.tur === "roman")           // önce romanları seç
4   .filter(k => k.puan >= 9)                 // sonra puanı yüksek olanları
5   .map(k => k.ad)                           // sadece adlarını al
6   .sort();                                 // alfabetik sırala
7
8 console.log(sonuc); // ["Kürk Mantolu", "Tutunamayanlar"]
9
10 // "Bilim kitaplarının toplam fiyatı"
11 const bilimToplam = kitaplar
12   .filter(k => k.tur === "bilim")
13   .reduce((t, k) => t + k.fiyat, 0);
```

İpucu

Her metot yeni bir dizi döndürdüğü için zincirleme yapabilirsiniz. Kod, **ne yapmak istediğinizi** neredeyse Türkçe gibi anlatır.

⚠ Dikkat

Performans notu: Her adım yeni dizi oluşturur. **Binlerce eleman** varsa **filter**'ları tek koşulda birleştirin: `.filter(k => k.tur === "roman" && k.puan >= 9)`

Sıralama: sort()

```
1 const sayilar = [10, 5, 100, 2];
2
3 sayilar.sort();           // [10, 100, 2, 5] <- YANLIS! Metin olarak siralar
4 sayilar.sort((a, b) => a - b); // [2, 5, 10, 100] artan
5 sayilar.sort((a, b) => b - a); // [100, 10, 5, 2] azalan
6
7 // Nesne dizisi siralama
8 kitaplar.sort((a, b) => b.puan - a.puan); // puana gore azalan
9 kitaplar.sort((a, b) => a.fiyat - b.fiyat); // fiyata gore artan
10
11 // Metin siralama - TURKCE karakterler icin localeCompare
12 kitaplar.sort((a, b) => a.ad.localeCompare(b.ad, "tr"));
13
14 // sort() ORIJINAL diziyi degistirir!
15 const siraliKopya = [...kitaplar].sort((a, b) => a.fiyat - b.fiyat);
```

⚠ Dikkat

İki kritik tuzak: (1) `sort()` varsayılan olarak elemanları *metne çevirip* sıralar — sayılarda mutlaka karşılaştırma fonksiyonu verin. (2) `sort()` **orijinal diziyi değiştirir** — kopyasını almak isterseniz `[...dizi].sort()` yazın



Nesneler

BÖLÜM 4

Nesne Temelleri

```
1  const kitap = {
2    ad: "Simyaci",
3    yazar: "Paulo Coelho",
4    yil: 1988,
5    fiyat: 89.90,
6    turler: ["roman", "felsefe"],           // dizi olabilir
7    yayinevi: { ad: "Can", sehir: "Istanbul" }, // ic ice nesne
8    ozet() {                               // metot (kisa yazim)
9      return `${this.ad} - ${this.yazar}`;
10   }
11 };
12
13 // ERISIM
14 kitap.ad;                               // "Simyaci"        nokta gosterimi
15 kitap["yazar"];                         // "Paulo Coelho"   koseli parantez
16 kitap.yayinevi.sehir;                   // "Istanbul"
17 kitap.turler[0];                        // "roman"
18 kitap.ozet();                           // "Simyaci - Paulo Coelho"
19
20 // GUNCELLEME / EKLEME / SILME
21 kitap.fiyat = 79.90;
22 kitap.stok = 12;                        // yeni ozellik ekler
23 delete kitap.yil;
```

Nesne Yardımcıları

```
1 const kitap = { ad: "Simyaci", yazar: "Coelho", fiyat: 89.90 };
2
3 Object.keys(kitap);    // ["ad", "yazar", "fiyat"]
4 Object.values(kitap);  // ["Simyaci", "Coelho", 89.90]
5 Object.entries(kitap); // [["ad", "Simyaci"], ["yazar", "Coelho"], ...]
6
7 // Nesnede gezinme
8 for (const [anahtar, deger] of Object.entries(kitap)) {
9   console.log(`${anahtar}: ${deger}`);
10 }
11
12 // Özellik var mı?
13 "fiyat" in kitap;           // true
14 kitap.hasOwnProperty("stok"); // false
15 kitap.stok ?? "Bilinmiyor";  // güvenli varsayılan
16
17 // Birleştirme / kopyalama
18 const guncel = { ...kitap, fiyat: 79.90, stok: 5 }; // yayılma
19 const birlesik = Object.assign({}, kitap, { indirim: true });
```



Dikkat

Yıkım Ataması (Destructuring)

```
1  const kitap = { ad: "Simyaci", yazar: "Coelho", fiyat: 89.90, yıl: 1988 };
2
3  // Nesneden degisken cikarma
4  const { ad, yazar } = kitap;
5  console.log(ad);           // "Simyaci"
6
7  // Yeniden adlandirma + varsayilan deger
8  const { ad: kitapAdi, stok = 0 } = kitap;
9
10 // Ic ice
11 const { yayinevi: { sehir } = {} } = kitap;
12
13 // DIZI yikimi
14 const [birinci, ikinci, ...digerleri] = ["a", "b", "c", "d"];
15
16 // Fonksiyon parametresinde - COK KULLANISLI
17 function kartYaz({ ad, yazar, fiyat = 0 }) {
18   return `${ad} / ${yazar} - ${fiyat} TL`;
19 }
20 kartYaz(kitap);
21
22 // Dizi metotlariyle birlikte
23 kitaplar.map(({ ad, puan }) => `${ad}: ${puan}`);
```

Yayılma (Spread) ve Rest Operatörleri

```
1 // YAYILMA (spread) - acar
2 const a = [1, 2, 3], b = [4, 5];
3 const birlesik = [...a, ...b];           // [1,2,3,4,5]
4 const kopya     = [...a];               // bagimsiz kopya
5
6 const temel = { ad: "Simyaci", fiyat: 89.90 };
7 const detay = { ...temel, stok: 5 };     // yeni nesne
8
9 Math.max(...[3, 9, 1]);                 // 9
10
11 // REST - toplar
12 function toplamHesapla(...sayilar) {
13     return sayilar.reduce((t, s) => t + s, 0);
14 }
15 toplamHesapla(1, 2, 3, 4);              // 10
16
17 const { ad, ...kalanlar } = detay;      // ad ayri, kalanlari nesne
```

İpucu

Aynı semboller (...), zıt işler: Sol tarafta (parametre/yıkım) → **toplar** (rest). Sağ tarafta

Uygulama: BookVerse Katalog Fonksiyonları

```
1 const kitaplar = [
2   { id: 1, ad: "Simyaci", yazar: "Paulo Coelho", fiyat: 89.90, puan: 8.4, tur: "roman" },
3   { id: 2, ad: "Kürk Mantolu", yazar: "Sabahattin Ali", fiyat: 45.00, puan: 9.1, tur: "roman" },
4   { id: 3, ad: "Tutunamayanlar", yazar: "Oğuz Atay", fiyat: 120.00, puan: 9.5, tur: "roman" },
5   { id: 4, ad: "Kozmos", yazar: "Carl Sagan", fiyat: 75.00, puan: 9.3, tur: "bilim" }
6 ];
7
8 // 1) Türe göre filtrele
9 const tureGoreGetir = tur => kitaplar.filter(k => k.tur === tur);
10
11 // 2) Fiyat aralığına göre
12 const fiyatAraligi = (min, max) => kitaplar.filter(k => k.fiyat >= min && k.fiyat <= max);
13
14 // 3) Id ile tek kitap bul
15 const kitapBul = id => kitaplar.find(k => k.id === id);
16
17 // 4) İstatistik nesnesi dondur
18 function istatistik(liste = kitaplar) {
19   const toplam = liste.reduce((t, k) => t + k.fiyat, 0);
20   return {
21     adet: liste.length,
22     toplam: Number(toplam.toFixed(2)),
23     ortalamaFiyat: Number((toplam / liste.length).toFixed(2)),
24     ortalamaPuan: Number((liste.reduce((t, k) => t + k.puan, 0) / liste.length).toFixed(2)),
25     enPahalı: liste.reduce((a, b) => (a.fiyat > b.fiyat ? a : b)).ad
26   };
27 }
28
29 // 5) Arama - büyük/küçük harf duyarlı
30 const ara = terim =>
```

Sık Yapılan Hatalar

Hata	Sebeup ve Çözüm
<code>map</code> sonucunu kullanmamak	<code>map</code> yeni dizi döndürür; sadece işlem yapacaksanız <code>forEach</code> kullanın.
<code>forEach</code> sonucunu değişkene atamak	<code>undefined</code> döner. Dönüşüm için <code>map</code> kullanın.
<code>sort()</code> ile sayıların yanlış sıralanması	Karşılaştırma fonksiyonu verin: <code>(a,b) => a-b</code> .
Orijinal diziyi bozmak	<code>sort</code> , <code>splice</code> , <code>push</code> diziyi değiştirir. <code>[...dizi]</code> ile kopyalayın.
<code>return</code> unutmak	Ok fonksiyonunda süslü parantez açtıysanız <code>return şarttır</code> .
Nesne karşılaştırması	<code>{a:1} === {a:1} → false</code> . Nesneler referansla karşılaştırılır.
<code>find</code> yerine <code>filter</code> kullanmak	Tek sonuç için <code>find</code> — dizi değil, nesne döner.

reduce Adım Adım

reduce, başlangıçta korkutucu görünür. **Adım tablosuyla** bakınca çok basittir.

```
const fiyatlar = [89.90, 45.00, 120.00];  
const toplam = fiyatlar.reduce((biriken, mevcut) => biriken + mevcut, 0);
```

Adım	biriken	mevcut	Yeni biriken
Başlangıç	—	—	0
1	0	89.90	89.90
2	89.90	45.00	134.90
3	134.90	120.00	254.90

⚠ Dikkat

Başlangıç değerini (0) unutmayın. Yazmazsanız ilk element başlangıç kabul edilir, baş diğide bato

💡 İpucu

Sadece toplama değil, Başlangıç `{}` veya `[]`

reduce ile İleri Kullanımlar

```
1 // 1) TURE GORE GRUPLAMA
2 const gruplar = kitaplar.reduce((grup, kitap) => {
3   (grup[kitap.tur] ?? = []).push(kitap.ad);
4   return grup;
5 }, {});
6 // { roman: ["Simyaci", "Kürk Mantolu", ...], bilim: ["Kozmos"] }
7
8 // 2) SAYMA
9 const sayim = kitaplar.reduce((s, k) => {
10   s[k.tur] = (s[k.tur] || 0) + 1;
11   return s;
12 }, {});
13 // { roman: 3, bilim: 1 }
14
15 // 3) EN BUYUGU BULMA
16 const enPahalı = kitaplar.reduce((a, b) => (a.fiyat > b.fiyat ? a : b));
17
18 // 4) ID -> NESNE HARİTASI (hızlı erişim için)
19 const harita = kitaplar.reduce((h, k) => ({ ...h, [k.id]: k }), {});
20 harita[3].ad; // "Tutunamayanlar"
```

?? = **operatörü**: "Boşsa ata". grup[tur] ?? = [] → yoksa boş dizi oluştur.

✓ Saf (pure) fonksiyon

```
function zamli(fiyat, oran) {  
  return fiyat * (1 + oran);  
}  
  
zamli(100, 0.2); // 120 - HER ZAMAN  
zamli(100, 0.2); // 120 - aynı sonuç
```

- ▶ Aynı girdi → aynı çıktı
- ▶ Dışarıyı **değiştirmez**
- ▶ Test edilmesi **kolay**

✗ Yan etkili fonksiyon

```
let toplam = 0;  
  
function ekle(fiyat) {  
  toplam += fiyat; // DISARIYI degistiriyor  
  return toplam;  
}  
  
ekle(100); // 100  
ekle(100); // 200 - FARKLI sonuç!
```

- ▶ Sonuç **öngörülemez**
- ▶ Hata ayıklaması **zor**
- ▶ Sırası önemli hale gelir

💡 İpucu

Pratik kural: Fonksiyonlarınızın çoğu **saf** olsun; yan etkileri (DOM değiştirme, konsola yazma, sunucuya istek) **az sayıda ve belirgin** yerde toplayın. Gelecek hafta DOM'a dokunan fonksiyonlar yazacağız — onlar zorunlu olarak yan etkilidir. **Hesaplamaları** ise saf tutun.

Fonksiyonları Düzenlemek

```
1 // ===== VERI =====
2 const kitaplar = [ /* ... */ ];
3
4 // ===== YARDIMCI FONKSIYONLAR =====
5 const paraFormat = t1 => `${t1.toFixed(2)} TL`;
6 const kucult      = m  => m.toLocaleLowerCase("tr");
7
8 // ===== VERI ISLEMLERI (saf) =====
9 const tureGoreGetir = tur => kitaplar.filter(k => k.tur === tur);
10 const kitapBul      = id  => kitaplar.find(k => k.id === id);
11
12 function istatistik(liste) {
13     const toplam = liste.reduce((t, k) => t + k.fiyat, 0);
14     return { adet: liste.length, toplam, ortalama: toplam / liste.length };
15 }
16
17 // ===== CIKTI (yan etkili) =====
18 function raporYaz(liste) {
19     liste.forEach(k => console.log(`${k.ad} - ${paraFormat(k.fiyat)}`));
20 }
```

Bu düzen, dosya büyüdüğçe sizi kurtarır: **veri** → **yardımcılar** → **iş mantığı** → **çıktı**.



Yapay Zeka ile Fonksiyonlar

BÖLÜM 5

Dizi metotları ve fonksiyonlar, yapay zekanın **kusursuz çalıştığı** alanlardan biridir. Tam da bu yüzden **dikkatli olmalısınız**.

Risk

“Bu kitap dizisini filtrele ve sırala” deyip **çalışan kod** aldığınızda, **reduce**’un nasıl çalıştığını **asla öğrenemezsiniz**. Sınavda ve iş görüşmesinde bu soru **mutlaka** sorulur.

İyi Kullanım

- ▶ Yazdığınız döngüyü **dizi metoduna çevirtmek** (ve farkı görmek)
- ▶ **reduce**’u **adım adım izletmek**
- ▶ Kendi çözümünüze **alternatif** istemek
- ▶ Alıştırma üretirmek

Prompt 1 — Döngüyü Dizi Metoduna Çevirtme

Yapay Zeka İstemi (Prompt) — Refactor ve Karşılaştırma

Asagidaki kodu klasik for dongusu ile yazdim:

[KENDI FOR DONGUNUZU YAPISTIRIN]

GOREV: 1) Ayni isi dizi metotlariyle (map/filter/reduce/find) yapan versiyonu yaz. 2) Iki versiyonu su tabloda karsilastir: | Olcut | for dongusu | dizi metodu | Olcutler: satir sayisi, okunabilirlik, orijinal diziyi degistirir mi, hangi durumda hangisi tercih edilmeli
3) Dizi metodu versiyonunu SATIR SATIR acikla. Ozellikle reduce kullandiysan, her adimda "biriken" degerin ne oldugunu bir tablo halinde goster.
4) "Bu iki kod hangi girdide FARKLI sonuc verir?" sorusunu yanitla.
Turkce yaz.

3. madde öğretici kısım: reduce tablosu, bu metodu anlamamanın en hızlı yoludur.

Yapay Zeka İstemi (Prompt) — Fonksiyon İncelemesi

Asagidaki fonksiyonlari yazdim:

[FONKSIYONLARINIZI YAPISTIRIN]

Her fonksiyon icin su olcutlere gore [IYI] / [GELISTIRILEBILIR] / [SORUNLU] etiketi ver:

1) TEK SORUMLULUK - fonksiyon tek bir is mi yapiyor? 2) ISIMLENDIRME - adi ne yaptigini anlatiyor mu? 3) PARAMETRE SAYISI - 3'ten fazla mi? (nesne almali mi?) 4) YAN ETKI - disaridaki bir degiskeni degistiriyor mu? 5) DONUS DEGERI - tutarli bir tip donduruyor mu? 6) HATA DURUMU - bos dizi / null girdide ne olur?

KURAL: Duzeltilmis kod VERME. Her sorun icin "ne yapilmali" seklinde YON tarif et.

En sonda: hangi fonksiyonu ONCE duzeltmeliyim? Neden?

Prompt 3 — Alıştırma Üretimi

Yapay Zeka İstemi (Prompt) — Dizi Metodu Alıştırmaları

Bana dizi metotlari uzerine 10 alistirma hazirla. Asagidaki veriyi kullan:

```
const kitaplar = [ id:1, ad:"Simyaci", yazar:"Coelho", fiyat:89.90, puan:8.4,
tur:"roman", stok:12 , id:2, ad:"Kurk Mantolu", yazar:"Ali", fiyat:45.00, puan:9.1,
tur:"roman", stok:0 , id:3, ad:"Tutunamayanlar", yazar:"Atay", fiyat:120.00, puan:9.5,
tur:"roman", stok:3 , id:4, ad:"Kozmos", yazar:"Sagan", fiyat:75.00, puan:9.3,
tur:"bilim", stok:7 ];
```

Alistirma dagilimi: - 2 x filter, 2 x map, 2 x reduce, 1 x find, 1 x sort, - 2 x zincirleme (birden fazla metot)

HER alistirma icin: - Gorev tanimi (1 cumle) - Beklenen ciktinin TAM hali (dizi/deger olarak yazili) - KOD VERME

En sonda ayri baslik altinda cozumler olsun. Zorluk sirasina gore dizilsin. Turkce.

Yapay Zeka Çıktısını Doğrulama

Beklenen çıktıları doğrulayın: Yapay zeka bazen kendi verdiği veriye göre **yanlış sonuç** yazar. Önce siz çözün, sonra karşılaştırın.

Özet ve Değerlendirme

BÖLÜM 6

⚙️ Fonksiyonlar

`function`, ifade, ok fonksiyonu. Varsayılan parametre, `return`, rest. **DRY**: Aynı kodu iki kez yazmayın.

📦 Kapsam

Global → fonksiyon → blok. **İçeriden dışarıya bakılır**, tersi olmaz. `var` blok kapsamını tanımaz.

☰ Diziler & Nesneler

`map` dönüştürür, `filter` seçer, `find` bulur, `reduce` indirger. `sort` **orijinali değiştirir**. Yıkım ve yayılma (...).

🤖 Yapay Zeka

Bu alanda **kusursuz** çalışır — tam da bu yüzden **kod ürettirmeyin**. Döngünüzü çevirtin, `reduce`'u **adım adım izletin**.

Altın Kural: `map/filter/reduce` **yeni dizi** döndürür; `push/sort/splice` **orijinali değiştirir**. Bu ayrımı bilmek, en zor hataları önler.

Terim	Anlam	Terim	Anlam
Parametre	Tanımdaki değişken	map	Dönüştürür, yeni dizi
Argüman	Çağrıda verilen değer	filter	Seçer, yeni dizi
return	Fonksiyonun döndürdüğü değer	find	İlk eşleşeni bulur
Ok fonksiyonu	$(a) \Rightarrow a * 2$	reduce	Tek değere indirger
Kapsam (scope)	Değişkenin geçerli olduğu alan	sort	Sıralar (orijinali bozar)
Gölgeleme	İç kapsamın dışı gizlemesi	Destructuring	Yıkım ataması
DRY	Kendini tekrar etme	Spread (...)	Yayar / açar
Yan etki	Dışarıdaki değeri değiştirme	Rest (...)	Toplar

Değerlendirme Soruları

1 `map` ile `forEach` arasındaki fark nedir? Hangi durumda hangisini kullanırsınız?

2 Aşağıdaki kodun çıktısı nedir? Neden?

```
const s = [10, 5, 100, 2];  
console.log(s.sort());  
console.log(s.sort((a, b) => a - b));
```

3 `reduce` kullanarak bir sayı dizisinin **ortalamasını** hesaplayan tek satırlık ifade yazınız.

4 Ok fonksiyonları nesne metodu olarak neden önerilmez?

5 `const dizi = [1,2,3]; dizi.push(4);` neden hata vermez? `const` tam olarak neyi korur?

6 “Puanı 9’un üzerinde ve stokta olan kitapların adlarını alfabetik sırala” işlemini **tek zincirle** yazınız.

7 `{...kitap}` ile `structuredClone(kitap)` arasındaki fark nedir?

Laboratuvar Föyü 11

- ▶ Fonksiyon yazımının üç biçimi
- ▶ Kapsam deneyleri
- ▶ 10 dizi metodu alıştırmaları
- ▶ `reduce` adım tablosu
- ▶ BookVerse katalog fonksiyonları modülü
- ▶ Yıkım ve yayılma uygulamaları
- ▶ Yapay zekaya döngü çevirtme deneyi

Hazırlık

Okuma: Kitap Ünite 2.3 — “Fonksiyonlar ve Kapsam” **Kaynaklar:**

- ▶ developer.mozilla.org/docs/Web/JavaScript
- ▶ javascript.info/array-methods

Gelecek hafta: **DOM manipülasyonu** — JavaScript nihayet **sayfayı değiştirecek**.

Önemli: Bu haftaki katalog fonksiyonlarınızı saklayın — Hafta 12’de bunlarla **ekrana kitap kartları** basacağız.

Haftaya Görüşmek Üzere!

Gelecek Hafta:

DOM Manipülasyonu ve Olay Yönetimi

- / ▪
- Mobil menünün hayata geçirilmesi