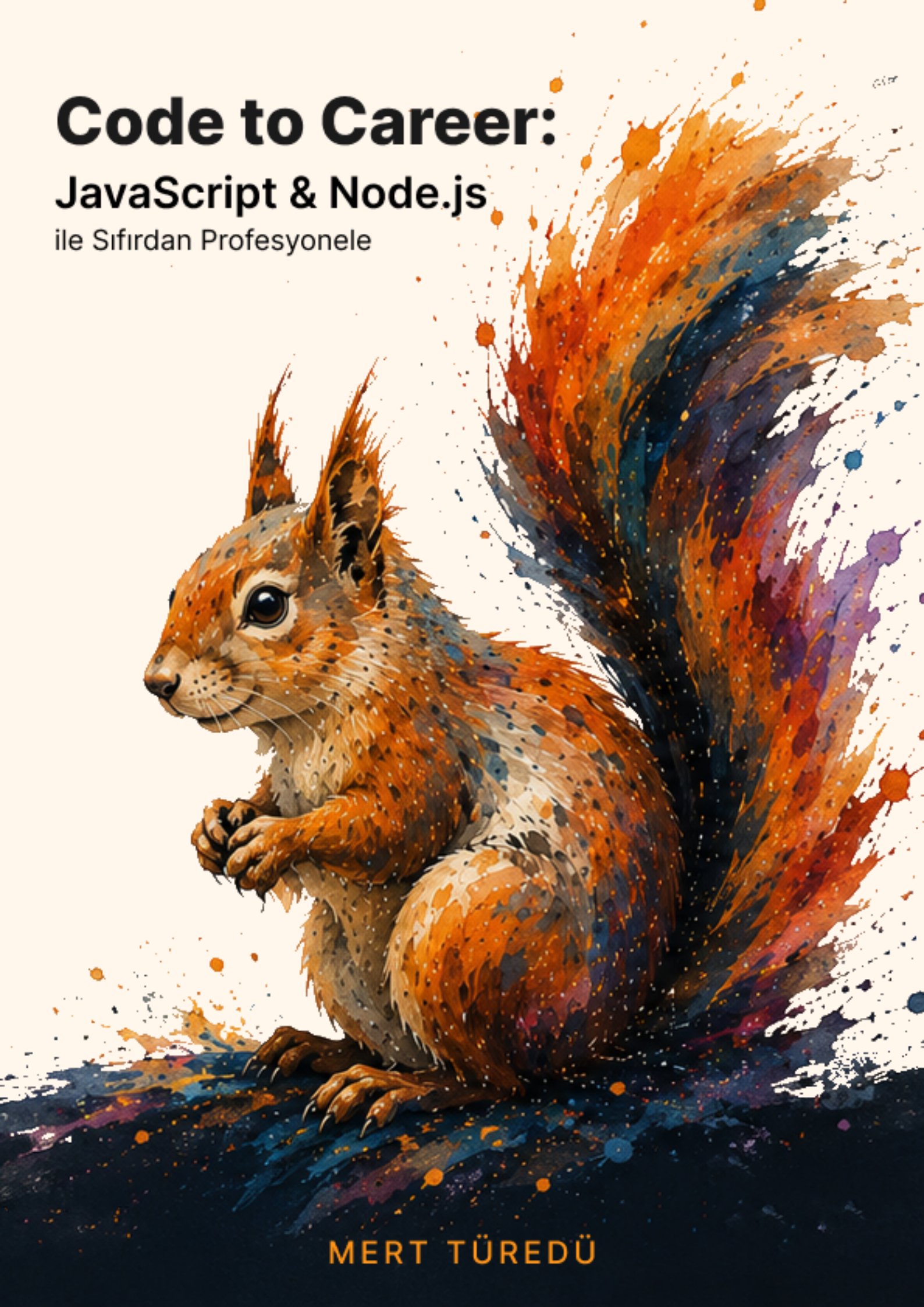
Code to Career:

JavaScript & Node.js

ile Sıfırdan Profesyonele



MERT TÜREDÜ

Koddan Kariyere: JavaScript & Node.js ile Sıfırdan Profesyonele

Birinci Baskı, 2026

Errata & geri bildirim: **mertturedu.thedev@gmail.com** (konu: `[Errata]`)

Kapak Hakkında

Bu kitabın kapağındaki sincap rastgele seçilmedi.

Bir sincap, kışa hazırlanırken binlerce tohumu tek tek toplar, saklar, bazen unutur, bazen yeniden bulur — ama asla durmaz. Onu bu kadar dayanıklı yapan tek bir büyük sıçrayış değil, gün be gün tekrarlanan küçük bir alışkanlıktır: bugün bir tohum daha, yarın bir tane daha.

Yazılımcı olmak da buna çok benziyor. Bir gecede "usta" olmazsın; her gün öğrendiğin küçük bir kavram, çözdüğün küçük bir hata, okuduğun bir satır kod, zamanla birikir. **Merak** seni yeni bir konuya çeker, **çalışkanlık** o konuda derinleşmeni sağlar, **sabır** anlamadığın yerde pes etmemeni sağlar, **hazırlık** ise bugün öğrendiğin şeyin yarın işine yarayacağına güvenmektir — tıpkı sincabın kışın tam olarak ne zaman geleceğini bilmeden tohum biriktirmesi gibi.

Bu kitabı elinize aldığınızda, aslında siz de kendi tohumlarınızı biriktirmeye başlıyorsunuz. Bazı bölümler ilk seferde tam oturmayabilir — sincap da her tohumu ilk seferde bulmuyor. Önemli olan, biriktirmeye devam etmek.

İyi çalışmalar, ve sabırlı ol — kışın geldiğinde hazır olacaksın.

— Mert Türedü

Ön Söz

İlk JavaScript kodum bir hata mesajıydı.

`console.log` yazacağım yerde `console.lo` yazmıştım. Terminal kırmızı bir şey döktü, ben de ne anlama geldiğini bilmeden kapattım. O günden sonra hata mesajlarını okumayı öğrenmem beş yıl daha sürdü, çünkü kimse bana okumayı öğretmemişti, sadece ne yapacağımı söylemişlerdi.

Bu kitap o boşluğu kapatmak için burada.

JavaScript öğrenmek isteyenin önünde bugün yüzlerce kaynak var: YouTube videoları, blog yazıları, interaktif platformlar, resmi dokümanlar. Hepsi iyi niyetli. Ama hiçbiri birbirine bağlı değil. Birinden `var` öğreniyorsunuz, ötekinde `let`'in neden icat edildiğini okuyorsunuz, üçüncüsünde ise `const` kullanın deniyor. Aralarındaki farkı bağlayan bir ses yok.

Ben o bağlantıyı kurmaya çalıştım. Sıfırdan başlayıp üretime hazır kod yazana kadar geçen yolu tek bir seste, tek bir sırada anlatan bir kaynak. "Bilgisayar ne iş yapar?" diye sorarak başlar, async/await'in neden icat edildiğini, TypeScript'in gerçekte ne çözdüğünü, Node.js'in event loop'unun neden bu kadar garip göründüğünü anlatır. Her konunun tarihini de söyler, çünkü bir şeyin neden var olduğunu bilmek, onu nasıl kullanacağınızı öğrenmekten çok daha uzun süre akılda kalır.

Kimin için?

Hiç kod yazmamışsanız: KISIM ZERO'dan başlayın, hiçbir şeyi atlamayın. O bölümler "çok kolay" görünebilir, ama temel oturmazsa ilerisi sendelemeye başlar.

Biraz JavaScript biliyorsanız: İlk bölümleri hızlı tarayın. Bölüm 4 (fonksiyonlar), Bölüm 11–13 (async) ve Bölüm 14 (OOP) asıl duracağınız yerler.

Başka bir dil bilip JS'e geçiyorsanız: Tip sisteminin olmadığına ve `this`'in Java'dakinden çok farklı davrandığına hazır olun. Bölüm 4'ün `this` kısmı (4.7) ve TypeScript bölümü sizin için yazıldı.

Tazeleme yapıyorsanız: Her bölümün sonundaki Cheat Sheet'ler ve Bölüm 29 yeterli.

Mülakat hazırlığı yapıyorsanız: §A.1–§A.10 (Algoritmik Düşünce) bölümlerine doğrudan gidin. Bu bölümler mülakat odaklı algoritmik düşünceyi; Big O, Two Pointers, Binary Search, DP gibi kalıpları sıfırdan öğretir. Bölüme girmeden önce tanı testini çözün, o testi geçebiliyorsanız bu bölümü atlayabilirsiniz.

Nasıl okumalı?

Sırayla. Bu kitap doğrusaldır; her bölüm bir öncekinin üzerine inşa edilir. Atlamak cazip geliyor, biliyorum. Ama 3. bölümü atladığınızda 8. bölümde "hâlâ anlamıyorum" diyeceksiniz ve geri döneceğiniz şey 3. bölüm olacak.

Alıştırmaları yapın. Okumak değil yazmak öğretir. Bir alıştırmayı görünce önce kendiniz deneyin; en az 10 dakika takılırsanız çözüme bakın.

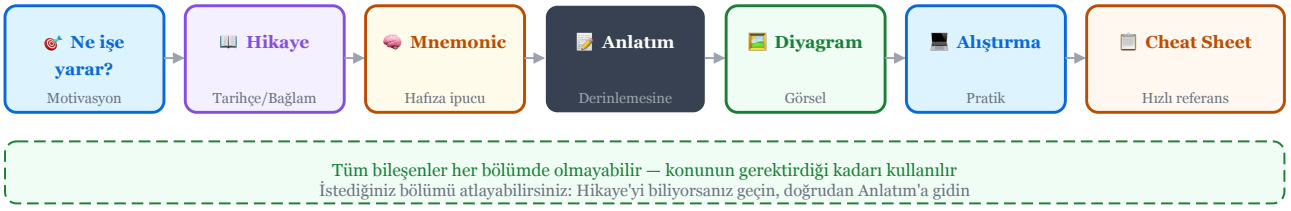
Acele etmeyin. Özellikle Bölüm 11–13 (asenkron programlama) çoğu okuyucunun üç kez okumak zorunda kaldığı bölümler. Bu normal. Ben de üç kez okudum.

Her örnek kodu çalıştırın. Çıktıyı görmek okumaktan beş kat daha fazla şey öğretir.

Tanımadığınız bir terimi aratın. `runtime`, `scope`, `callback` gibi İngilizce terimlerle karşılaşırsanız kitabın sonundaki **Bölüm 28 — Sözlük (Türkçe ↔ İngilizce)** bölümüne bakın. Yeni terimler ilk geçtikleri yerde açıklanır; yine de bir şey tanıdık gelmezse Sözlük her zaman oradadır.

Kitabın yapısı

Her bölüm bir havuzdan beslenir: konunun bir cümlelik özeti, tarihçe (neden icat edildi?), kısa bir akılda kalıcı formül, derinlemesine anlatım, diyagram, alıştırma ve bölüm sonu cheat sheet. Bunların hepsi her bölümde olmaz; konu gerektirmiyorsa veya zaten başka bir yerde ele alındıysa o parça yoktur. Siz de okurken aynı şeyi yapabilirsiniz: tarihçeyi biliyorsanız geçin, cheat sheet'e bakıp yeterliyse ilerleyin.



ŞEKİL 0.0 *Her Bölümün Pedagojik Yapısı*

Bu bileşenlerin tamamı her bölümde olmaz, konu ne gerektiriyorsa o kadarı var. Onları bir **menü** gibi düşünün: işinize yarayanı okuyun, gereksiz gördüğünüzü atlayın. Bir konuyu zaten biliyorsanız Hikaye'yi geçip doğrudan Anlatım'a veya Cheat Sheet'e gidebilirsiniz. Tanıdık yapı, kitabı *gezinmesi kolay* kılmak içindir; sizi tek bir okuma biçimine zorlamak için değil.

Okuma modunuzu seçin

Kitabı herkese aynı hızda okutmak doğru olmaz. Nerede durduğunuza göre üç mod öneririm:

MOD	KİM İÇİN	NASIL
🦕 **Tam okuyucu**	Hiç kod yazmamış; temelini sağlam kurmak istiyor	Her bölümü sırayla oku, hikayeyi ve mnemonik'i atma, her alıştırmayı yap
🚶 **Normal okuyucu**	Biraz programlama biliyor	Hikaye ve mnemonik'i hızlıca tara, anlatıma geç, alıştırmaların yarısını yap
🏃 **Hızlı tarama**	Başka dil biliyor, JS'i keşfediyor	Bölüm girişini, tablolarını ve cheat sheet'i oku; sadece ilgi çekici alıştırmalar

`İLERİ` **ve** `UZMAN` **etiketli bölümler** uzmanlık konularıdır; ilk okumada atlayıp ileride geri dönebilirsiniz. Her bölümün etiketinin tam açıklaması için aşağıdaki **Bölümler Nasıl İşaretlendi?** tablosuna bakın.

Hikaye, mnemonik ve cheat sheet bölümleri o konuda işinize yaramıyorsa geçebilirsiniz. Zaten bildiğiniz bir şeyin tarihçesini okumak zorunda değilsiniz, doğrudan kod örneğine atlayın.

Bir not

Kod yazmayı öğrenmek bazen sinir bozucudur. Hatalar alırsınız, bir şey çalışmaz, saatlerce takılırsınız. Bu, sürecin **doğal** bir parçasıdır, kötü bir geliştirici olduğunuzun değil, gerçekten öğrendiğinizin işaretidir. Deneyimli geliştiriciler de aynı şeyi yaşar; sadece tıkandıklarında ne yapacaklarını bilirler. Bu kitap size hem JavaScript'i hem de o "ne yapacağını bilme" becerisini kazandırmayı hedefler.

İyi okumalar.

Errata & Geri Bildirim

Bu hacimde bir kitapta birkaç hata kalması kaçınılmazdır, yanlış kod çıktısı, yazım hatası, yanıltıcı bir açıklama. Bulduğunuz her hatayı **mertturedu.thedev@gmail.com** adresine bildirirseniz memnun olurum. Konu satırına `[Errata]` yazmanız mesajın kaybolmamasını sağlar; hangi bölüm, hangi sayfa ve kısa bir açıklama eklerseniz düzeltme çok daha hızlı yapılır.

Geri bildirim de aynı adrese: bir konunun daha iyi anlatılmasını, eksik gördüğünüz bir başlığı veya anlamakta zorlandığınız bir örneği yazabilirsiniz.

🗺️ Öğrenme Yolculuğu

Bu kitap doğrusal bir yapıdadır; her bölüm bir öncekinin üzerine inşa edilir. Aşağıdaki tablo kitabın tamamını tek bakışta özetler; her satır için **ne öğreneceğinizi** ve terminalinizde hemen deneyebileceğiniz **küçük bir örneği** görebilirsiniz.

BÖLÜM	KONU	NE ÖĞRENECEKSİN	HEMEN DENE
Z.1	Programlama mantığı	Algoritma nedir; bilgisayar talimatları nasıl işler	`node -e "console.log(1 + 1)"`
Z.2	Editor · Terminal · Tarayıcı	Üç temel geliştirme aracı ve aralarındaki fark	`ls` → `node dosya.js` → F12
Z.3	Ortam kurulumu	Node.js ve VS Code kurulumu; ilk dosyayı çalıştırma	`node --version`
Z.4	İlk program	Değişken, işlem, çıktı; dört satırda çalışan program	`const x = 5; console.log(x * 2)`
Z.5	Zihinsel Model	Değişken = etiketli kutu; fonksiyon = input→makine→output	`let n = 42; console.log(typeof n)`
Z.6	İlk Koşul ve Döngü	if/else ile karar; for ile tekrar; while ile koşullu döngü	`if (yas >= 18) { ... } else { ... }`
Böl. 0	Giriş	Kitabı nasıl okuyacaksın; okuma modları; bölüm yapısı	*(okunur, çalıştırılmaz)*
Böl. 0.5	Debug Mindset	Hata mesajı okuma; SyntaxError / ReferenceError / TypeError; REPL döngüsü	Kasıtlı hatalı kod yazıp mesajı okuma
Böl. 1	JavaScript Tarihi & V8	JS nasıl doğdu; V8 nedir; JIT derleme; Node.js = V8 + libuv	`node -e "console.log(process.version)"`
Böl. 2	Değişkenler & Tipler	`const`, `let`, `var`; 7 primitive tip; `typeof`; `===`; falsy; scope; hoisting; NaN; coercion	`typeof null` → `"object"` (tarihi hata)
Böl. 3	Operatörler	`?.`, `??`, `||`, `&&`, ternary; modern operatörlerin tuzakları	`user?.adres?.sehir ?? "bilinmiyor"`
Böl. 4	Fonksiyonlar	Parametreler, return, arrow fn, closure, first-class	`const çarp = (a, b) => a * b`
Böl. 4.2	Functional Programming	Pure function, immutability, map/filter/reduce, currying	`[1,2,3].filter(x => x > 1).map(x => x * 2)`
Böl. 5	Object	`{ }` sözdizimi, erişim, spread, destructuring; prototype chain temeli	`const { ad } = kullanici`
Böl. 6	Array	`map`, `filter`, `reduce`, spread, destructuring	`[1,2,3].map(x => x * 2)`

BÖLÜM	KONU	NE ÖĞRENECEKSİN	HEMEN DENE
Böl. 7	Map & Set	Sözlük (Map) ve benzersiz küme (Set)	`new Set([1,1,2])` → `{1, 2}`
Böl. 8	String & Regex	`split`, `join`, `trim`, `includes`, regex temeli	`"  ali  ".trim()` → `"ali"`
Böl. 9	Akış Kontrolü	`if/else`, `switch`, ternary, erken return	`x > 0 ? "pozitif" : "negatif"`
Böl. 10	Hata Yönetimi	`try/catch/finally`, hata türleri, `throw`	`throw new Error("Geçersiz giriş")`
A.1	Algoritma & Big O	Zaman/alan karmaşıklığı; O(1), O(n), O(log n) sezgisi	`[1,3,5,7,9].indexOf(7)` → O(n)
A.2	İki Gösterici	Sıralı dizide iki uçtan ortaya: O(n²)'yi O(n)'e indir	palindrome, Two Sum sorted
A.3	Kayan Pencere	Ardışık alt dizi/string'de kaydıran pencere: O(n)	en uzun tekrarsız substring
A.4	Hash Map Kalıpları	O(1) lookup; "daha önce gördüm mü?"; frekans sayacı	`new Map()` ile Two Sum
A.5	Stack & Queue	LIFO/FIFO; parantez eşleştirme; BFS için kuyruk	`()[]{}` geçerli mi?
A.6	Linked List	Pointer zinciri; fast & slow; tersine çevirme	döngü tespiti
A.7	Binary Search	Sıralı uzayda yarıya böl: O(log n)	döndürülmüş dizide arama
A.8	Tree: DFS & BFS	Recursive DFS; queue ile BFS; BST doğrulama	seviye-seviye geziş
A.9	Graph & Union-Find	Adjacency list; ada sayısı; bileşen birleştirme	Number of Islands
A.10	Dinamik Programlama	Memoization; tabulation; alt problem önbelleği	Fibonacci → Coin Change
Böl. 11	Promise	Asenkron zincir, `.then`, `.catch`, `.finally`	`fetch(url).then(r => r.json())`
Böl. 12	async / await	Asenkron kodu senkron gibi yazmak	`const veri = await fetch(url)`
Böl. 13	Event Loop (dikkat)	Neden `setTimeout(fn, 0)` bekler; microtask kuyruğu	`console.log` sıra testi
Böl. 14	OOP & Class	`class`, `constructor`, `extends`, `super`; altta prototype zinciri	`class Hayvan { ses() {} }`

BÖLÜM	KONU	NE ÖĞRENECEKSİN	HEMEN DENE
Böl. 15	CommonJS	`require` , `module.exports` , `__dirname` ; node_modules çözümleme	`const fs = require("fs")`
Böl. 16	ES Modules	`import / export` (ESM); default vs named export; tree-shaking	`export const PI = 3.14`
Böl. 17	Node.js Çekirdeği	`fs` , `path` , `process` , Buffer, stream temeli	`fs.readFileSync("a.txt", "utf8")`
Böl. 18	HTTP & REST API	`http.createServer` , Express, route, middleware	`app.get("/", (req, res) => ...)`
Böl. 19	Crypto & HMAC	Hash, HMAC, JWT token üretimi; şifreleme temeli	`crypto.createHash("sha256")...`
Böl. 20	AsyncLocalStorage	Request context takibi; trace ID; bağlamı otomatik taşıma	Her log satırına trace ID ekle
Böl. 21	Streams & SSE	Büyük veri akışı, server-sent events, backpressure	`res.write("data: merhaba\n\n")`
Böl. 22	Tasarım Örüntüleri	Singleton, Factory, Observer, Strategy	Gerçek Node.js projelerinde tanıma
Böl. 23	Firestore & NoSQL	Document-based DB; collection/query; SQL vs NoSQL farkı	`db.collection("users").add({...})`
Böl. 24	Cloud & Docker	Deployment, konteyner, ortam değişkenleri	`docker build -t app .`
Böl. 24.1–24.2	Güvenlik & Performans	SQL injection, XSS, rate limiting, profiling	`helmet()` middleware
Böl. 24.3–24.14	Gözlemlenebilirlik & DevOps	Logging, tracing, CI/CD, Kubernetes	Yapılandırılmış log formatı
Böl. 25	Proje Yapısı	Klasör organizasyonu, kod temizliği, DRY	Monolith → modüler refactor
Böl. 26.1	TypeScript	Statik tip, `interface` , generic, utility types	`const x: number = 5`
Böl. 26.2–26.4	Test	Unit (Vitest), integration, TDD döngüsü	`expect(topla(1,2)).toBe(3)`
§B0–B11	Browser & DOM	DOM manipülasyonu, Fetch API, Web Worker	`document.querySelector("#btn")`
§F1–F15	Frontend Ekosistemi	React, Vue, Next.js, bundler (Vite)	`<button onClick={...}>Tıkla</button>`

> 📎 **Bu tablo kitabın ana omurgasını gösterir. §A.1–§A.10 Algoritmik Düşünce** bölümleri §10 ile §11 arasında yer alır, LeetCode tarzı problem çözme, Big O, veri yapıları ve klasik algoritmalar. Tabloda yer almayan diğer önemli bölümler: **Bölüm 18.3** (WebSocket), **Bölüm 18.4** (GraphQL & gRPC), **Bölüm 23.11** (PostgreSQL & SQL), **Bölüm 23.1** (Prisma / Drizzle ORM), **Bölüm F9** (Bun & Deno). Tam liste için **İçindekiler**'e bakın.

🛠️ **Mini Projeler** (kitap boyunca): CLI Hesap Makinesi → Async Veri Çekici → HTTP Echo Server → JWT Auth → Realtime Chat → Final: Production-ready full-stack uygulama.

> **Async + Event Loop** (Bölüm 11–13) çoğu okuyucunun tıkandığı yerdir. Acele etmeyin; gerekirse iki kez okuyun.

🏷️ Bölümler Nasıl İşaretlendi?

Her bölüm başlığının hemen altında bir **zorluk etiketi** ve **tahmini okuma süresi** bulunur:

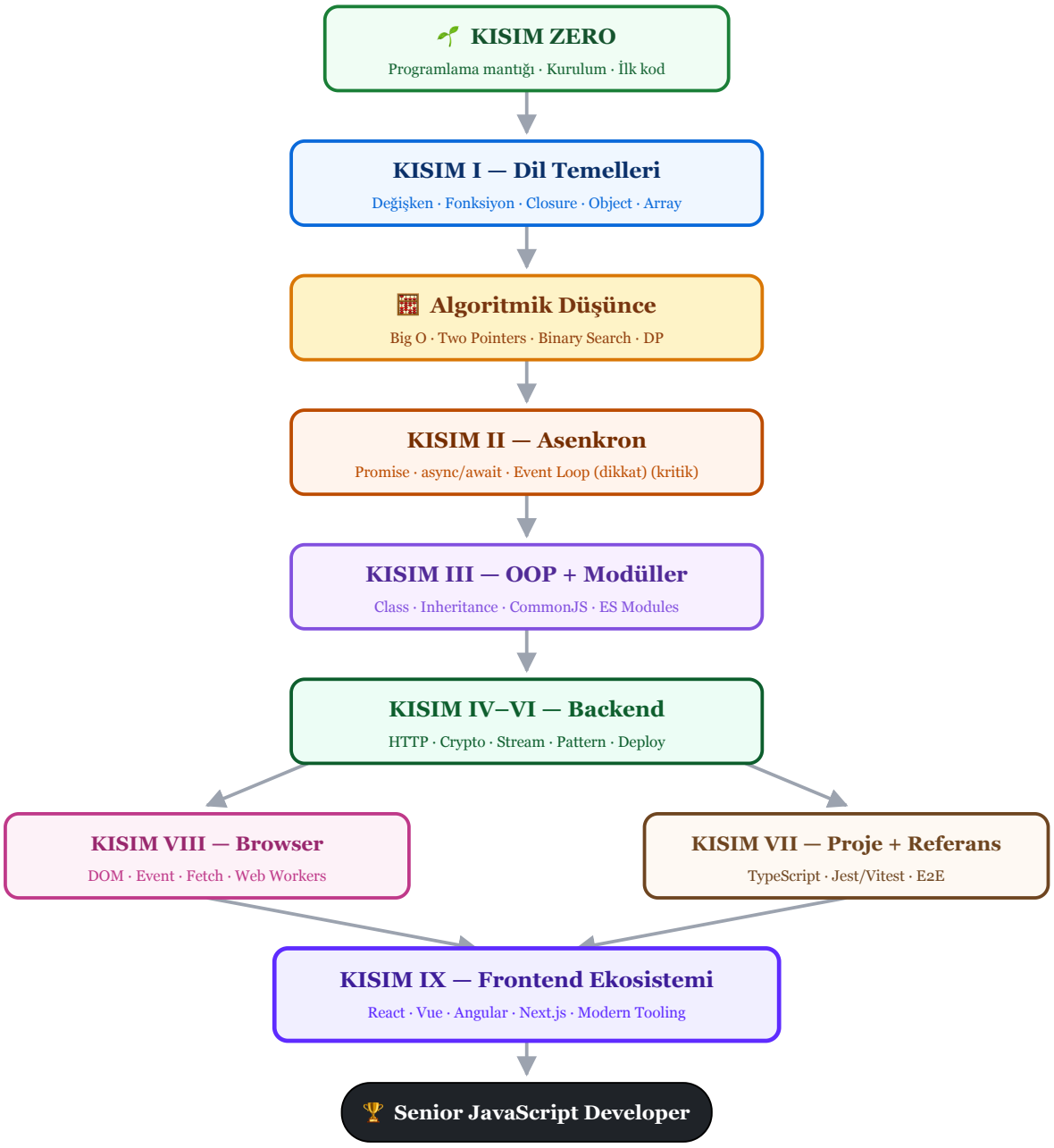
ETİKET	ANLAMI	TAHMİNİ SÜRE
BAŞLANGIÇ	Hiçbir ön bilgi gerektirmez; kimse atlamamalı	15–30 dk
TEMEL	Önceki bölümleri okuduysanız hazırsınız	20–40 dk
ORTA	Dikkat ve biraz pratik ister; acele etmeyin	30–50 dk
İLERİ	Karmaşık konu; ilk turda atlanabilir; YOL HARİTASI kutusuna bakın	30–60 dk
UZMAN	İsteğe bağlı derinleşme; yalnızca o konuya ihtiyacınız varsa	30–60 dk

İlk okuma stratejisi. etiketli bölümleri sırayla okuyun. veya bir bölüme geldiğinizde panik yapmayın: bu bölümlerin başında bir 🧭 **YOL HARİTASI** kutusu vardır. O kutu size üç şeyi söyler, (1) bu konunun ön koşulu nedir, (2) şimdi mi okumalısınız yoksa sonraya mı bırakmalısınız, (3) atlarsanız nereden devam edeceksiniz. Bir konuyu ertelemek başarısızlık değildir; doğru sırada öğrenmektir.

"**X**" **ekli bölümler** (Bölüm 2.1, 4X, 6X, 11X…) ana anlatıya birer **derinleşme eki**'dir, bir konunun ileri ayrıntılarını ana akıştan ayrı tutar. İlk turda atlanabilir; o konuyu gerçekten kullanmaya başladığınızda geri dönülür. Başlıktaki "X" harfi "bu bir derinleşmedir, zorunlu değildir" anlamına gelir.

🌳 Beceri Ağacı (Skill Tree)

Aşağıdaki harita, öğreneceğiniz becerilerin birbirine nasıl bağlandığını gösterir. Her kutu bir önceki seviyenin üzerine inşa edilir, atlamadan ilerleyin.



ŞEKİL 0.1B *JavaScript & Node.js Beceri Ağacı: Sıfırdan Senior'a*

İçindekiler

KODDAN KARİYERE · JAVASCRİPT & NODE.JS İLE SIFIRDAN PROFESYONELE

152 BÖLÜM

🌱 KISIM ZERO: İlk Adım (Programlama'ya yeni başlayan için) — 6 BÖLÜM

Bölüm Z.1 — Bilgisayar Nasıl Düşünür? · Bölüm Z.2 — Editor, Terminal, Browser: Üç Arkadaş · Bölüm Z.3 — Setup: Aletlerini Hazırla · Bölüm Z.4 — İlk Programını Yazıyorsun · Bölüm Z.5 — Mental Model: Kutular ve Makineler · Bölüm Z.6 — İlk Koşulun ve İlk Döngün: if, for, while

📚 KISIM I: Temeller — 18 BÖLÜM

Bölüm 0 — Giriş: Bu kitap nedir, nasıl okuyacaksın · Bölüm 0.5 — Debug Mindset: Kod Yazmadan Önce Düşünmeyi Öğren · Bölüm 1 — JavaScript: Tarih, V8, ve Mental Model · Bölüm 2 — Değişkenler ve Tipler · Bölüm 2.1 — Generator + Iterator: Tembel Veri Üretimi · Bölüm 2.2 — Symbol + BigInt + Number Derin · Bölüm 3 — Operatörler ve Karşılaştırma · Bölüm 4 — Fonksiyonlar: JavaScript'in Kalbi · Bölüm 4.1 — Proxy + Reflect: Metaprogramming · Bölüm 4.2 — Functional Programming Derin · Bölüm 5 — Object'ler · Bölüm 5.1 — Object Derin: Descriptor, Getter/Setter, Inheritance · Bölüm 6 — Array'ler ve İşlemleri · Bölüm 6.1 — Array Modern API: ES2022+ · Bölüm 7 — Map, Set ve Collection'lar · Bölüm 8 — String'ler ve Regex Temel · Bölüm 8.1 — Regex Derin: Lookbehind, Named Groups, Unicode · Bölüm 8.2 — Mini Proje 1: CLI Hesap Makinesi

🧮 Algoritmik Düşünce (§A.1 – §A.10) — 11 BÖLÜM

🎯 Tanı Testi — Bu Bölümü Atlayabilir Misin? · Bölüm A.1 — Algoritma Düşüncesi & Big O Notasyonu · Bölüm A.2 — İki Gösterici (Two Pointers) · Bölüm A.3 — Kayan Pencere (Sliding Window) · Bölüm A.4 — Hash Map & Hash Set Kalıpları · Bölüm A.5 — Stack & Queue · Bölüm A.6 — Linked List · Bölüm A.7 — Binary Search · Bölüm A.8 — Tree: DFS & BFS · Bölüm A.9 — Graph: DFS, BFS & Union-Find · Bölüm A.10 — Dinamik Programlama Giriş

🔄 KISIM II: Kontrol Akışı + Asenkron — 8 BÖLÜM

Bölüm 9 — Control Flow (if, for, switch) · Bölüm 10 — Hata Yönetimi · Bölüm 11 — Asenkron JavaScript: Promises · Bölüm 11.1 — Modern Async: for-await-of, AbortSignal, Promise.withResolvers · Bölüm 12 — async/await Derin Bakış · Bölüm 12.1 — AbortController + Cancellation · Bölüm 13 — Event Loop'un Anatomisi · Bölüm 13.5 — Mini Proje 2: Async Veri Çekici

🧬 KISIM III: OOP ve Modüller — 4 BÖLÜM

Bölüm 14 — Class'lar ve OOP · Bölüm 14.1 — Modern Class: Private Methods, Static Blocks, Decorators · Bölüm 15 — CommonJS Modül Sistemi · Bölüm 16 — ES Modules (Modern)

🌐 KISIM IV: Node.js Spesifik — 11 BÖLÜM

Bölüm 17 — Node.js Temelleri · Bölüm 17.1 — Worker Threads + Cluster + child_process · Bölüm 17.2 — File Handling Derin · Bölüm 17.3 — Date/Time + Timezone + Intl · Bölüm 18 — Framework'süz HTTP Server'lar · Bölüm 18.1 — API Design + REST Best Practices · Bölüm 18.2 — HTTP Caching Headers · Bölüm 18.3 — WebSocket · Bölüm 18.4 — GraphQL ve gRPC · Bölüm 18.5 — Dosya Yükleme (File Upload) · Bölüm 18.6 — Mini Proje 3: HTTP Echo Server

🔐 KISIM V: Güvenlik, Context, Stream — 11 BÖLÜM

Bölüm 19 — Crypto ve HMAC · Bölüm 19.1 — OAuth 2.0 + Session Management · Bölüm 19.2 — Validation Derin (Zod/Joi/Yup) · Bölüm 19.5 — Mini Proje 4: JWT Auth Sistemi · Bölüm 20 — AsyncLocalStorage: Request Context · Bölüm 21 — Server-Sent Events · Bölüm 21.1 — Streams ve EventEmitter · Bölüm 21.2 — Background Jobs ve Queue'lar · Bölüm 21.3 — State Machines · Bölüm 21.4 — Reactive Programming (RxJS) · Bölüm 21.5 — Mini Proje 5: Real-time Chat (SSE)

🏗️ KISIM VI: Pattern'ler ve Altyapı — 29 BÖLÜM

Bölüm 22 — Pattern'ler (Ring Buffer, LRU, Pub/Sub) · Bölüm 23 — Firestore Entegrasyonu · Bölüm 23.1 — ORM/Query Builder: Prisma, Drizzle, Kysely · Bölüm 23.2 — Database Migration Patterns: Zero-Downtime · Bölüm 23.3 — Redis Cache Patterns · Bölüm 23.4 — Message Queues: RabbitMQ, Kafka, AWS SQS · Bölüm 23.5 — npm Ekosistemi · Bölüm 23.6 — Database Patterns · Bölüm 23.7 — CLI Tool Building · Bölüm 23.8 — API Documentation · Bölüm 23.9 — Source Maps · Bölüm 23.10 — i18n + Localization · Bölüm 23.11 — SQL ve PostgreSQL: İlişkisel Veritabanı · Bölüm 24 — Cloud Deployment ve Serverless · Bölüm 24.1 — Güvenlik Temelleri · Bölüm 24.1.1 — Saldırı Senaryoları · Bölüm 24.2 — Performance ve Profiling · Bölüm 24.3 — Debugging Derin · Bölüm 24.4 — Logging ve Observability · Bölüm 24.5 — Process Management (PM2, systemd) · Bölüm 24.6 — Reverse Proxy (Nginx, Caddy) · Bölüm 24.7 — GitHub Actions CI/CD · Bölüm 24.8 — Edge Computing · Bölüm 24.9 — Email Sending · Bölüm 24.10 — Webhook Pattern · Bölüm 24.11 — Feature Flags · Bölüm 24.12 — Microservices Patterns · Bölüm 24.13 — Production Operations Pipeline · Bölüm 24.14 — Kubernetes Temelleri

🎓 KISIM VII: PROJE YAPISI + REFERANS

22 BÖLÜM

Bölüm 25 — Gerçek Bir Node.js Projesinin Yapısı · Bölüm 25.1 — Mimari Kararlar · Bölüm 26 — Sıkça Yapılan Hatalar ·
Bölüm 26.1 — TypeScript Giriş · Bölüm 26.1.1 — TypeScript: Generics Derin · Bölüm 26.1.2 — TypeScript: Type Manipulation (İleri) ·
Bölüm 26.1.3 — TypeScript: Utility Types Tam Liste · Bölüm 26.1.4 — TypeScript: Type Guards + Narrowing ·
Bölüm 26.1.5 — TypeScript: tsconfig + Strict Mode · Bölüm 26.2 — Testing (Jest + Vitest) · Bölüm 26.3 — E2E Testing: Playwright + Cypress ·
Bölüm 26.4 — Storybook + Visual Testing · Bölüm 27 — Sonraki Adımlar · Bölüm 28 — Sözlük: Türkçe ↔ İngilizce ·
Bölüm 29 — Quick Reference Cheatsheet · Bölüm 29.1 — Tüm Cheat Sheet'ler: Konsolide Referans ·
Bölüm 30 — Cevap Anahtarı (Quiz Cevapları) · Bölüm 30A — Mini Proje Çözümleri · Bölüm 31 — Kodlama Alıştırmaları (Yazılabilir Kod Alanı)
· Bölüm 32 — Capstone Projesi: NoteFlow API · 🎓 Genel Değerlendirme Testi · 📖 Alıştırma Çözümleri

🌐 KISIM VIII: Browser + DOM

12 BÖLÜM

Bölüm B0 — HTML ve CSS Temelleri · Bölüm B1 — DOM Temelleri · Bölüm B2 — Event Handling: Click, Bubble, Delegate ·
Bölüm B3 — Fetch API + AJAX · Bölüm B4 — Web Storage: localStorage, sessionStorage, IndexedDB, Cookies ·
Bölüm B5 — Web Workers (Browser) · Bölüm B6 — Service Workers + PWA · Bölüm B7 — DOM Observers: Intersection, Mutation, Resize ·
Bölüm B8 — Canvas + WebGL Giriş · Bölüm B9 — WebAssembly Giriş · Bölüm B10 — Browser History + Routing ·
Bölüm B11 — WebRTC: Gerçek Zamanlı İletişim

🎨 KISIM IX: Frontend Framework'ler + Ekosistem

15 BÖLÜM

Bölüm F1 — React Temelleri · Bölüm F2 — React Hooks Derin · Bölüm F3 — Next.js: React + SSR/SSG/RSC ·
Bölüm F4 — Vue 3: Composition API + Reactivity · Bölüm F5 — Angular: TS + DI + RxJS · Bölüm F6 — Svelte / SolidJS — Yeni Nesil ·
Bölüm F7 — State Management: Redux Toolkit, Zustand, TanStack Query · Bölüm F8 — Backend Frameworks: Express, Fastify, Hono, NestJS ·
Bölüm F9 — Bun + Deno: Yeni Runtime'lar · Bölüm F10 — React Native + Expo: Mobile JS · Bölüm F11 — Electron + Tauri: Desktop JS ·
Bölüm F12 — Build Tools: Vite, Webpack, esbuild, Rollup · Bölüm F13 — Modern Tooling Ekosistemi 2026 ·
Bölüm F14 — AI / LLM API Entegrasyonu · Bölüm F15 — Web Erişilebilirliği (a11y)

📎 Ekler

7 BÖLÜM

Ek A — package.json Anatomy · Ek B — Dockerfile Anatomy · Ek C — gcloud / git Cheatsheet · Ek D — Faydalı Linkler ve Kaynaklar ·
Ek E — Karar Rehberi: Hangisini Ne Zaman Kullanmalı? · Ek F — Resmi Kaynaklar ve Referanslar ·
Ek G — Buradan Sonra Nereye? Kariyer Yol Haritası

KISIM ZERO – İLK ADIM

Bölüm Z.1 – Bilgisayar Nasıl Düşünür?

● ○ ○ ○ ○ BAŞLANGIÇ

> ◎ NE İŞE YARAR?
> Programlamanın temel mantığını anlamak: bilgisayar bir tarif okur ve uygular. Kod yazmak, bu tarifi yazmaktan ibarettir.

Bilgisayarla ilk kez iş yapan çoğu kişi onu "akıllı bir yardımcı" sanır. Gerçek bunun tam tersi — ve çok daha ilginç: bilgisayar, dünyanın en hızlı ama en *harfiyen* çırağıdır. Ne derseniz tam onu yapar; eksiğini de, fazlasını da. Bu bölümde o çırağa iş tarif etmeyi öğreneceksiniz.

Bilgisayar tam olarak yazdığınızı yapar

ŞEKİL Z.1.1 *Bilgisayar: hızlı ama itaatkâr; her talimatı açıkça söylemeniz gerekir*

Bilgisayarlar son derece hızlıdır, saniyede milyarlarca işlem yaparlar. Ama tek başlarına hiçbir karar veremezler. İki temel kural geçerlidir:

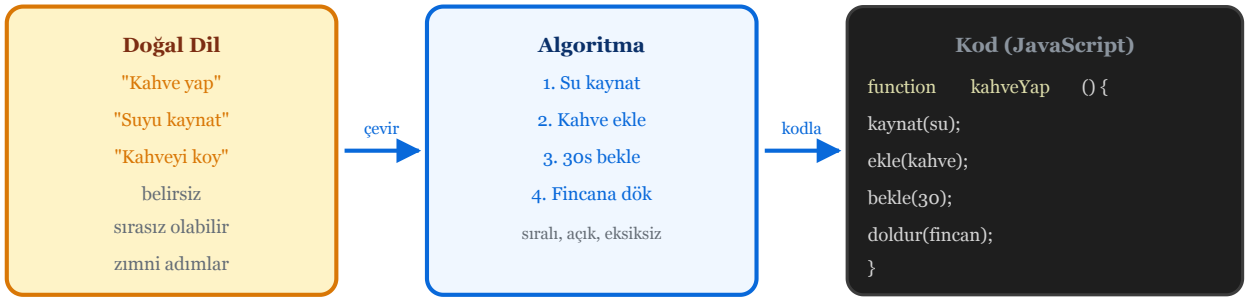
1. **Söylediğiniz her şeyi yaparlar.**
2. **Söylemediğiniz hiçbir şeyi yapmazlar.**

Bu kurallar neden bu kadar katı? Çünkü bilgisayarlar bağlam bilmez; "ne demek istediğinizi" çıkarsayamazlar. Bir insan asistana "kahve yap" dersiniz, o bağlamdan ne kastettiğinizi çıkarır. Bilgisayar bağlam yorumlayamaz; her adım açık yazılmak zorundadır.

Programcılığın özü budur: niyetinizi **adım adım** ve **kesin** ifade etmek. Doğal dilde "kahve yap" yeterli olur; bilgisayara şunları söylemeniz gerekir:

ÇIKTI

```
1. Cezveye su koy
2. Ocağı yak
3. Suyun kaynamasını bekle
4. Kahveyi suya ekle
5. 30 saniye karıştır
6. Fincana boşalt
```

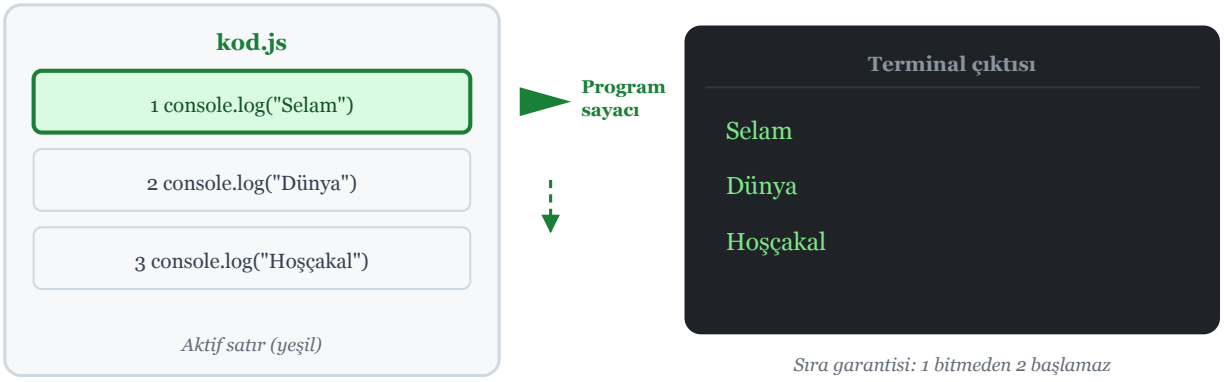


ŞEKİL Z.1.2 *Algoritma: günlük tarif → numaralandırılmış adımlar → kod*

Bu liste, bir **algoritma**. Kod yazmak, bunu bilgisayarın anlayacağı bir dile çevirmektir.

Tarifi yazdık; peki bilgisayar bu adımları hangi *sırayla* işler?

Komutlar yukarıdan aşağı çalışır



ŞEKİL Z.1.3 *Sequential execution: program sayacı (pointer) her satırı sırayla işler*

JavaScript'te her satır sırayla yorumlanır:

```js
console.log("Selam");
console.log("Dünya");
console.log("Hoşçakal");
```

Çalıştırdığınızda:

```
Selam
Dünya
Hoşçakal
```

İlk satır biter, sonra ikinci satıra geçilir. Bu davranışa **sequential execution** denir ve programlamanın en temel modelidir. İleride bunu **concurrent execution** (eş zamanlı çalıştırma) ile karşılaştıracağız (Bölüm 11–13).

Bir incelik daha var: yazdığınız dosya ile çalışan şey aynı değildir.

Kod, çalışan programla aynı şey değildir

Tarif (kod) kalır — kek (program) yenir, masada bir şey kalmaz

ŞEKİL Z.1.4 *Kod (disk) → node komutu → Program (RAM): tarif ile pişen yemek farkı*

Bu fark öğrenirken gözden kaçar:

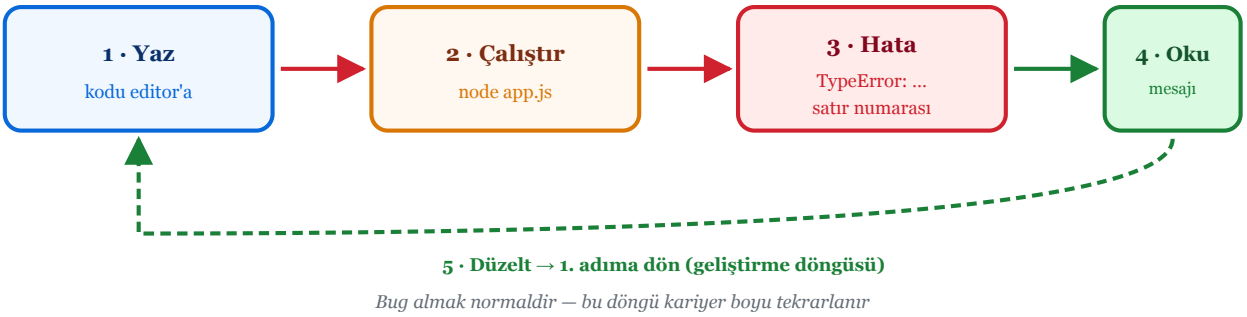
- **Kod**: bir metin dosyasıdır, statiktir. Diskte durur.
- **Program**: o kodun çalıştırıldığı an. Bellekte yaşar, bittiğinde kaybolur.

ŞEMA

```
[ kod.js ]  ——— node ile çalıştırılır ———→  [ aktif process ]
   metin                                       bellek + CPU
   diskte                                      geçici
```

Bir tarif kâğıdı ile pişen kek arasındaki fark gibi. Kâğıt kalır, kek yenir, masada bir şey kalmaz.

Bug = tarifteki hata



ŞEKİL Z.1.5 *Bug döngüsü: hata al → oku → düzelt → tekrar çalıştır*

Bilgisayar yazdıklarınızı **tam olarak** yapar. Bir şey beklediğiniz gibi çalışmıyorsa, bu kötü haber değil, iyi haber. Çünkü hatayı bilgisayar değil siz yapmışsınızdır ve sizin düzeltme yetkiniz var.

JS

```
console.lo("Selam");
```

JavaScript hatası:

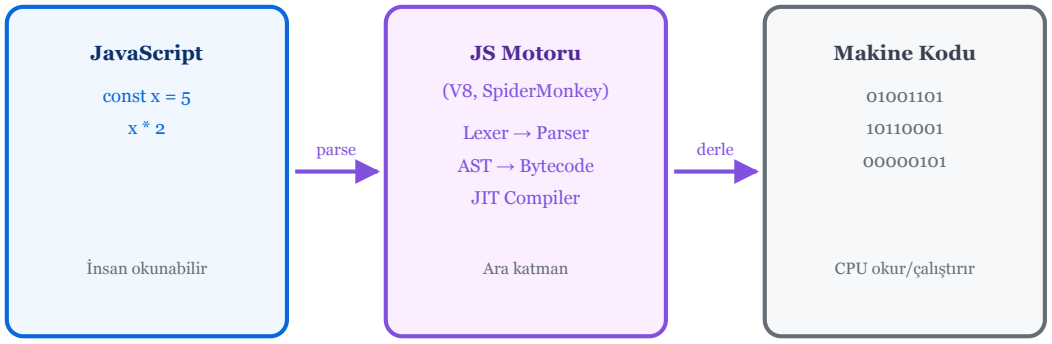
```
TypeError: console.lo is not a function
```
HATA

Bilgisayar size şunu söylüyor: "console nesnesinin `lo` adında bir fonksiyonu yok. `log` mu demek istemiştin?" Düzeltirsiniz, devam edersiniz. Bu döngü, yazıp test etmek, hata almak, düzeltmek, geliştiriciliğin günlük rutinidir.

Bug sözcüğünün ilginç bir kökeni var: 1947'de Harvard Mark II bilgisayarında çalışan Grace Hopper ve ekibi, hata kaynağını ararken makinenin içinde gerçek bir güveye rastladı. Defterlerine "first actual case of bug being found" yazdılar; bu kayıt bugün Smithsonian Ulusal Amerikan Tarihi Müzesi'nde sergilenmektedir. O günden beri yazılım hatalarına "bug", hata avlamaya "debugging" denir.

Peki bu tarifleri bilgisayara hangi dille anlatıyoruz?

Programlama dili nedir?



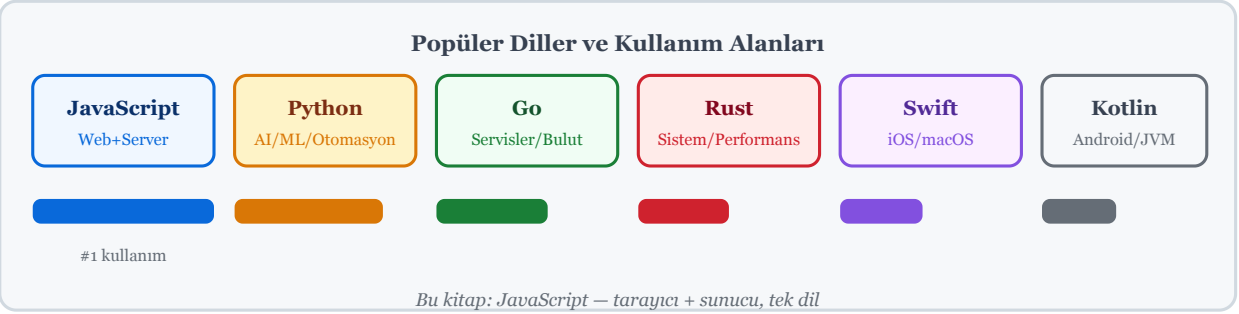
Her dil aynı temel mantık: insan-okunabilir → çalıştırılabilir

ŞEKİL Z.1.6 *Soyutlama katmanları: insan kodu → JS motoru → makine kodu → CPU*

Bilgisayarın asıl konuştuğu dil **makine kodu**dur (0 ve 1'ler). Bu dilde program yazmak insan için neredeyse imkânsızdır. Bu yüzden ara diller geliştirildi:

DİL	YAYGIN KULLANIM ALANI
JavaScript	Web tarayıcı, sunucu, mobil, masaüstü
Python	Veri bilimi, makine öğrenmesi, otomasyon
Go	Sistem servisleri, yüksek eş zamanlılık
Rust	Performans-kritik sistemler, güvenli C alternatifi
Swift	Apple platformları (iOS, macOS)
Kotlin	Android, JVM tabanlı uygulamalar

Her dilin kendi sözdizimi (syntax) ve felsefesi vardır, ama temel mantık aynıdır: insan-okunabilir kodu makine-çalıştırabilir hale çevirmek.



ŞEKİL Z.1.7 *Popüler dillerin kullanım alanları*

Bu kitabın konusu **JavaScript**. Dünyada en yaygın kullanılan dildir; başlangıçta sadece tarayıcılarda çalışıyordu, 2009'dan itibaren (Node.js ile) sunucularda da koşmaya başladı. Bugün bir dilde hem frontend hem backend yazabilmek başlı başına bir avantajdır.

– HİKÂYE

HİKAYE — Alan Turing ve hesaplanabilirlik *Peki bir bilgisayarın çözemeyeceği problem var mıdır? 1936'da matematikçi Alan Turing tam bu soruyu sordu — ve yanıtladı. Geliştirdiği teorik model olan* ***Turing makinesi****, tüm modern bilgisayarların kavramsal temelini oluşturdu. Turing aynı zamanda "hangi problemler algoritmayla çözülebilir, hangileri çözülemez?" sorusunu matematiksel olarak kanıtladı. Bu çalışma, programlama dillerinin varlık nedenini belirler: insan düşüncesini — en azından belirli bir kategorisini — makineye aktarmak. JavaScript yazdığınızda bu teorik temelin üstünde duruyorsunuz.*

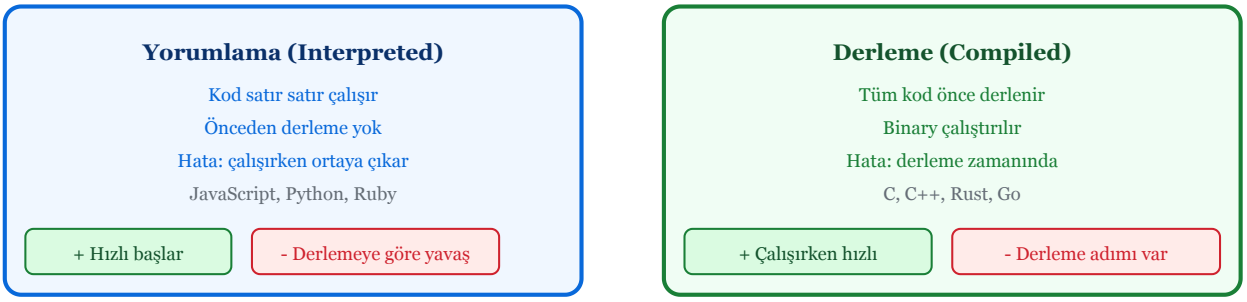
Hata kültürü

Yeni başlayanlar genelde hatadan korkar. Profesyonel geliştiriciler için durum farklıdır: bir hatayı görmek, eldeki bir bilgi parçası kazanmaktır. Hatayı okumayı öğrenmek, sözdizimini öğrenmekten daha önemlidir.

İlerleyen bölümlerde hata yönetimini (Bölüm 10) ve "debug zihniyeti"ni (Bölüm 0.5) ayrı ayrı işleyeceğiz. Şimdilik şu sezgi yeterli:

> Hata, programlama dilinin size verdiği geri bildirimdir. Susmak (sessiz başarısızlık) daha tehlikelidir.

Bu bölümden çıkarımlar



ŞEKİL Z.1.8 *Yorumlama vs Derleme: JS yorumlanır (JIT ile hızlandırılır)*

KAVRAM	ÖZET
Algoritma	Bir problemi çözmek için adım adım yöntem
Sequential execution	Komutlar sırayla, yukarıdan aşağı çalışır
Kod vs Program	Kod = metin, Program = çalışan örnek
Bug	Yazılan koddaki hata, normaldir, kaçınılmazdır
Programlama dili	İnsan ile bilgisayar arasındaki arayüz

CHEAT SHEET

BÖLÜM Z.1

KAVRAM	TANIM
Bilgisayar	Hızlı ama itaatkâr ve körü körüne uyan; her talimatı açıkça tarif ister
Program	Bilgisayara verilen talimatlar dizisi
Bug	Yazdığın tarifteki hata; bilgisayarın suçu değil
Algoritma	Bir problemi çözmek için adım adım yöntem

KONTROL SORULARI: BÖLÜM Z.1

1. Aşağıdaki kahve tarifini Türkçeden "programlama mantığına" çevir (numaralandırılmış adımlar olarak): "*Önce su kaynat. Su kaynayınca kahveyi bardağa koy. Kaynar suyu bardağa dök. Karıştır.*"
2. Bilgisayar sana neden "ne yapacağını sor" sorusu sormaz?
3. Aşağıdaki kod ne yazdırır?

```js
console.log("a");
console.log("b");
console.log("a");
```

4. Bug = ?
5. Tarif vs kek, JavaScript'te kod vs program ne karşılığı?

Artık bilgisayarın talimat beklediğini biliyorsunuz; bir sonraki bölümde bu talimatları yazmak için kullanacağınız üç araca geçiyoruz: editor, terminal ve tarayıcı.

▼ YANITLARI GÖR

1. Örnek: `1. Su kaynat. 2. Kahveyi koyacağa koy. 3. Suyu üstüne dök. 4. 3 dakika bekle. 5. Fincan doldur.` — Her adım açık, sıralı ve tek iş yapıyor.
2. Bilgisayar verilen talimatları tam olarak uygular; hangi adımın eksik ya da hatalı olduğunu anlamaz. Sana ne yapacağını soran kişi değil, ne söylersen yapan bir araçtır.
3. `merhaba` — `console.log` parantez içindeki değeri ekrana yazar.
4. Bug: programda istenen davranışı bozan hata. Hata bulma/düzeltme işlemine debugging denir.
5. Tarif = kod (talimatlar), pişen kek = program çıktısı (çalışan sonuç). JavaScript kodu çalışınca bir program oluşur.

Bölüm Z.1: Bilgisayar Nasıl Düşünür? – Test Soruları

1. Bilgisayarın temel çalışma kuralı hangisidir?

A) Kendi başına karar alabilir
B) Tam olarak söyleneni yapar, söylenmeyeni yapmaz
C) Hatalı talimatları otomatik düzeltir
D) Doğal dili anlayabilir
E) En mantıklı adımı seçer

2. Algoritma en doğru şekilde nasıl tanımlanır?

A) Belirli bir problemi çözmek için adım adım yazılmış talimat dizisi
B) Bilgisayarın içindeki işlemci
C) Bir programlama dilinin adı
D) Kodun çalışmasını sağlayan yazılım
E) Hatalı kodların toplandığı liste

3. Bir algoritmada hangi özellik kesinlikle bulunmalıdır?

A) En az 10 adımdan oluşmalı
B) Yalnızca matematik problemleri için geçerli
C) Her adım kesin ve belirsizlik içermeyen talimatlar içermeli
D) Adımlar olabildiğince kısa tutulmalı
E) Tüm adımlar aynı anda çalışabilmeli

4. Bir öğrenci `sonuç = 10 / 0` kodunu çalıştırıyor. JavaScript bu durumda ne yapar?

A) Matematiksel kural olarak işlemi reddeder ve durur
B) Otomatik olarak 0 döndürür
C) `Infinity` döndürür; hata vermez
D) İşlemi kullanıcıya sorar
E) Sadece uyarı gösterir, sonuç üretmez

5. "Kod" ile "program" arasındaki fark nedir?

A) İkisi aynı şeydir, sadece farklı isimlerdir
B) Kod statik metindir (tarif), program çalışan süreçtir (pişen kek)
C) Kod makine dilidir, program insan diliyle yazılır
D) Program koddan daha kısadır
E) Kod yalnızca JavaScript için kullanılır

6. Bilgisayara `kahve yap` talimatı versek ne olur?

A) Kahve tarifi internetten arar
B) En mantıklı adımları tahmin eder
C) Adımları sorar
D) Sisteme bağlı kahve makinesini çalıştırır
E) Komutu tanımadığı için hata verir veya yanlış çalışır

7. Alan Turing'in programlamaya temel katkısı nedir?

A) İlk programlama dilini icat etti
B) İlk bilgisayarı inşa etti
C) Hangi problemlerin bilgisayarla çözülebileceğini matematiksel olarak tanımladı
D) Bug terimini yarattı
E) JavaScript'in temelini attı

8. Aşağıdaki adımlardan hangisi `su kaynat` algoritmasında yanlış sıradadır?

`1. Su koy → 2. Ocağı yak → 3. Suyun kaynamasını bekle → 4. Fincana boşalt → 5. Cezveye kahve koy`

A) Adım 1 ve 2 yer değiştirmeli
B) Adım 2 son olmalı
C) Adım 3 atlanmalı
D) Adım 4 ve 5 yer değiştirmeli; kahve önce konur
E) Sıra doğrudur

9. Programlamanın özünü en iyi hangi ifade özetler?

A) Bilgisayara ne yapmasını istediğinizi düşünmek
B) Kodun güzel görünmesini sağlamak
C) Niyetinizi adım adım ve kesin ifade etmek
D) Hata mesajlarını anlamak
E) Programlama dilini ezberleme

10. Bir programcı `total = 5 + 3` yazdığında bilgisayar ne yapar?

A) 5 ve 3'ü karşılaştırır
B) Hatayı kullanıcıdan düzeltmesini ister
C) Ekrana 5 + 3 yazdırır
D) `total` adlı bellek hücresine 8 değerini yazar
E) İşlemi gelecekte yapar

▼ CEVAPLAR VE AÇIKLAMALAR

1 → B

`Bilgisayar tam söyleneni yapar, söylenmeyeni yapmaz`, kitabın en temel kuralı. Bilgisayar ne sezgi ne empati sahibidir; sadece açık ve sıralı talimatları çalıştırır. Diğer seçenekler bilgisayarı insanlaştırmaktadır.

2 → A

Algoritma; bir problemi çözmek için belirli sırada yazılmış, her biri açık ve sonlu adımlardan oluşan talimat kümesidir. Programlama dilinden bağımsızdır, kâğıda bile yazılabilir.

3 → C

Bir algoritmanın en temel özelliği her adımın **kesin ve belirsizlik içermemesi**dir. "Su kaynat" değil "suyu 100°C'ye ulaşana kadar ısıt" gibi. Belirsiz talimat → belirsiz sonuç. Adım sayısı, konu alanı veya eş zamanlılık algoritmanın zorunlu özellikleri değildir.

4 → C

JavaScript'te `10 / 0` işlemi `Infinity` döndürür; hata vermez. Bilgisayar "bu anlamsız" demez; tam olarak söyleneni yapar. Python gibi diller `ZeroDivisionError` fırlatır. Bu fark, her dilin sıfıra bölme davranışını ayrı tanımladığını gösterir. `console.log(10 / 0)` → `Infinity`, `console.log(-10 / 0)` → `-Infinity`, `console.log(0 / 0)` → `NaN`.

5 → B

Kod, yazılı talimatlardır, bir tarif gibi. Program ise o kodun çalıştırılmasıyla hayata geçen süreçtir, pişen kek gibi. Aynı kod birden fazla kez çalıştırılınca her biri ayrı bir program örneği (process) oluşturur.

6 → E

Bilgisayar doğal dili anlamaz. `kahve yap` belirsiz bir talimat; her adımı ayrı ayrı, açıkça yazmanız gerekir. Belirsiz talimat → ya hata ya yanlış sonuç.

7 → C

Alan Turing, hesaplanabilirlik teorisiyle hangi problemlerin algoritmik olarak çözülebileceğini, hangilerinin çözülemeyeceğini matematiksel olarak ortaya koydu. Bu, bilgisayar biliminin teorik temeli sayılır.

8 → D

Adım 4'te `fincana boşalt` yapılıyor, ama kahve henüz eklenmemiş. Doğrusu: `kahveyi ekle → karıştır → fincana boşalt`. Algoritmada sıra kritiktir, yanlış sıra yanlış sonuç üretir.

9 → C

Programlama; niyeti bilgisayarın anlayabileceği, adım adım ve kesin talimatlara dönüştürme disiplinidir. Güzel kod veya hata ayıklama bunun sonucudur, özü değil.

10 → D

Bilgisayar `5 + 3` işlemini yaparak 8 üretir ve bu değeri `total` adlı bellek hücresine (değişkene) atar. Ekrana yazdırmak için ayrıca `console.log(total)` gerekirdi.

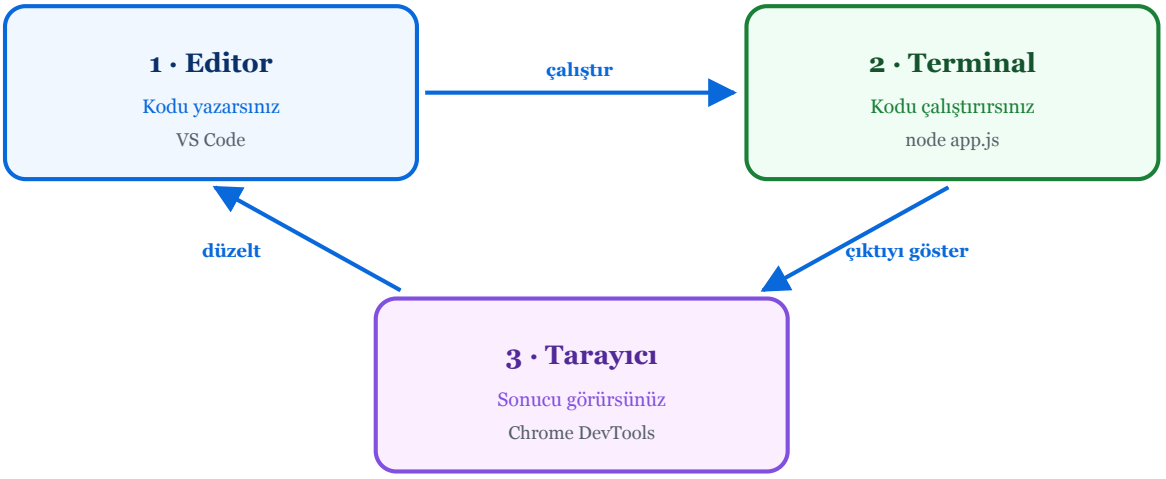
KISIM ZERO – İLK ADIM

Bölüm Z.2 – Editor, Terminal, Browser: Üç Arkadaş

● ○ ○ ○ ○ BAŞLANGIÇ

> ◎ NE İŞE YARAR?
>
> Bir geliştiricinin gün boyu kullandığı üç temel aracı tanımak: kodu **yazdığınız** editor, kodu **çalıştırdığınız** terminal, sonucu **gördüğünüz** tarayıcı.

Bu üçlüyü bir ekip gibi düşünün: biri *yazar* (editor), biri *çalıştırır* (terminal), biri *gösterir* (tarayıcı). Gün boyu aralarında dönüp duracağınız üç arkadaş.



ŞEKİL Z.2.1 *Geliştirme döngüsü: Yaz → Çalıştır → Gör*

1. Editor: Kodun yazıldığı yer

ŞEKİL Z.2.2 *VS Code arayüzü: kenar çubuğu, editör, terminal ve renklendirme*

Editor (kod yazma uygulaması), bir kelime işlemcisinden farklıdır. Microsoft Word'de kod yazmak şu nedenlerle pratik değildir:

- Sözdizimi renklendirmesi yoktur (her şey siyah).
- Akıllı tamamlama (autocomplete) sunmaz.
- Yazım denetimi kodu yanlış vurgular.
- Görünmeyen biçimlendirme karakterleri ekler.

Onun yerine **kod editörü** kullanılır. En yaygın seçenek **Visual Studio Code** (kısaca VS Code), Microsoft tarafından geliştirilen, ücretsiz, açık kaynaklı.

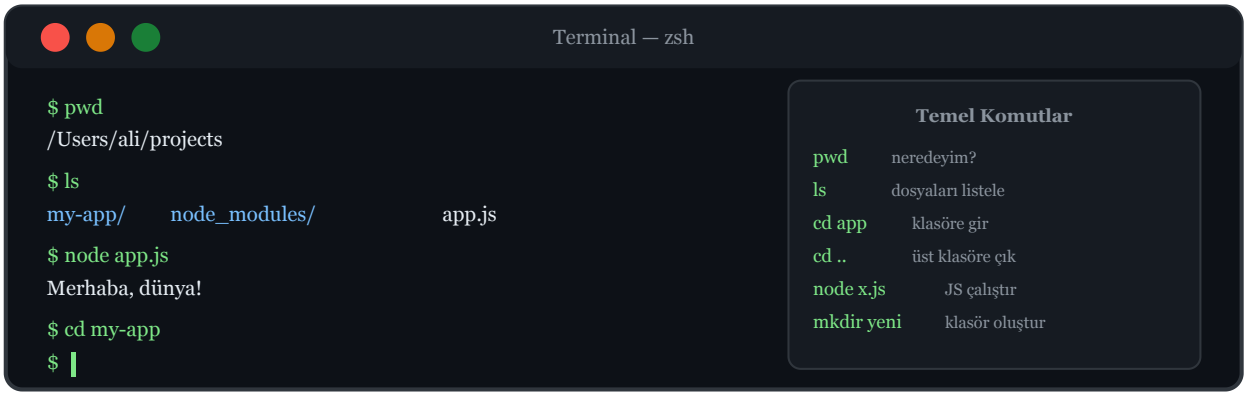
VS Code'un sağladıkları:

ÖZELLİK	FAYDASI
Sözdizimi renklendirme	`function` mavi, string yeşil; kod daha okunabilir
Akıllı tamamlama	`arr.` yazınca `map, filter, ...` listesi açılır
Hata altçizgisi	Yanlış sözdizimini kırmızı dalgalı çizgi ile gösterir
Entegre terminal	Editor içinde shell açabilirsiniz
Git entegrasyonu	Dosya değişikliklerini görsel olarak gösterir
Eklenti ekosistemi	Prettier, ESLint, GitLens gibi binlerce eklenti

Diğer popüler editörler: WebStorm (JetBrains, ücretli ama güçlü), Sublime Text (hafif), Neovim (terminal tabanlı, ileri kullanıcılar için).

Kodu yazan arkadaş tamam — ama bir metin dosyası tek başına kımıldamaz. Sahneye *çalıştıran* arkadaş çıkıyor.

2. Terminal: Komutların çalıştırıldığı yer



ŞEKİL Z.2.3 *Terminal temel komutları: ls, cd, pwd, node*

Terminali, klavyeyle gezilen bir Finder/Explorer gibi düşünebilirsiniz: orada tıkladığınız her şeyi burada yazarak yaparsınız.

Terminal (ya da "command line") yazılı komutlarla bilgisayarla etkileşim kurduğunuz **penceredir**. Shell ise o pencerede çalışan **yorumlayıcıdır**: komutları anlayan ve çalıştıran program. macOS/Linux'ta en yaygın shell `bash` veya `zsh`'tir; terminal açtığınızda arka planda shell çalışır. Grafiksel arayüzden (GUI) önce bilgisayarlar sadece terminal üzerinden kullanılırdı. Bugün hâlâ kullanılır çünkü:

- Otomasyon kolaydır (script'leyebilirsiniz).
- Uzaktan sunuculara bağlanmak için tek seçenektir (SSH).
- GUI'nin yapamadığı bazı şeyleri yapabilir (özel git komutları, ağ analizi, vb.).

Temel komutlar:

› TERMİNAL

```
ls              # Bulunduğunuz dizindeki dosyaları listele
cd folder       # folder dizinine geç
cd ..           # Bir üst dizine çık
pwd             # Şu an hangi dizindeyim?
mkdir new       # Yeni dizin oluştur
rm file.txt     # Dosya sil
mv old.js new.js   # Yeniden adlandır veya taşı
cp a.txt b.txt  # Dosya kopyala
cat file.txt    # Dosya içeriğini göster
```

JavaScript için en kritik komut:

› TERMİNAL

```
node script.js
```

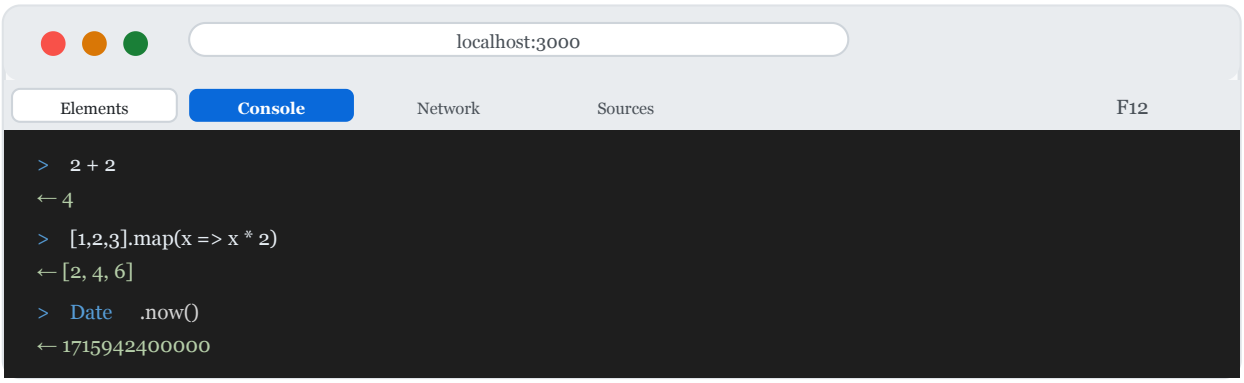
Bu satır, `script.js` dosyanızdaki JavaScript kodunu çalıştırır. Çıktı (örneğin `console.log(...)`) terminale yazılır.

Terminal nereden açılır?

- **macOS**: `Cmd+Space` → "Terminal" yazıp Enter
- **Windows**: Başlat → "PowerShell" veya "Windows Terminal"
- **Linux**: `Ctrl+Alt+T` çoğu dağıtımda
- **VS Code içinde**: `Ctrl+`` (backtick) ile entegre terminal açılır

Yazan ve çalıştıran hazır; sıra sonucu *gösteren* arkadaşta.

3. Tarayıcı: Sonucun görüntülendiği yer



ŞEKİL Z.2.4 *Browser DevTools Console: F12 ile açılır, anında JS çalıştırabilirsiniz*

Tarayıcı (Chrome, Firefox, Safari, Edge) sadece web sayfası göstermez, bir **JavaScript çalıştırma ortamıdır**. Açtığınız her web sayfasının arkasında JavaScript kodu çalışır.

İlk tarayıcıyı 1990'da CERN'de Tim Berners-Lee yazdı; bugün cebinizdeki telefon, o ilk hayalin çok ötesinde bir çalışma ortamı taşıyor.

Tarayıcının iki kullanımı vardır:

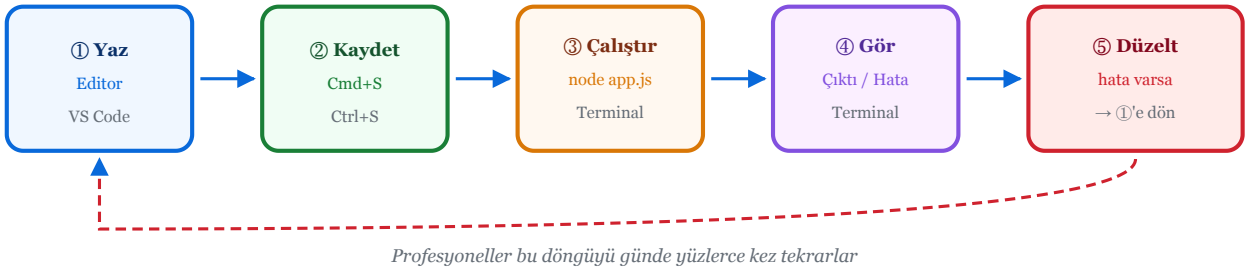
1. Web uygulamanızı görmek: Eğer bir web sitesi geliştiriyorsanız, sonuç tarayıcıda görünür.

2. JavaScript test etmek (DevTools): `F12` (Windows/Linux) veya `Cmd+Option+I` (Mac) tuşuyla **Developer Tools** açılır. Console sekmesinde anında JavaScript çalıştırabilirsiniz:

```js
> 2 + 2
4
> "merhaba".length
7
> Date.now()
1715942400000
```

Bu, hızlı deneyler için son derece kullanışlıdır. Tarayıcı konsolu, "JavaScript için hesap makinesi" gibi düşünülebilir.

Geliştirme döngüsü: 3 araç birlikte



ŞEKİL Z.2.5 *Tam geliştirme döngüsü: yaz → kaydet → çalıştır → gör → düzelt*

Tipik bir geliştirme oturumu şu adımlardan oluşur:

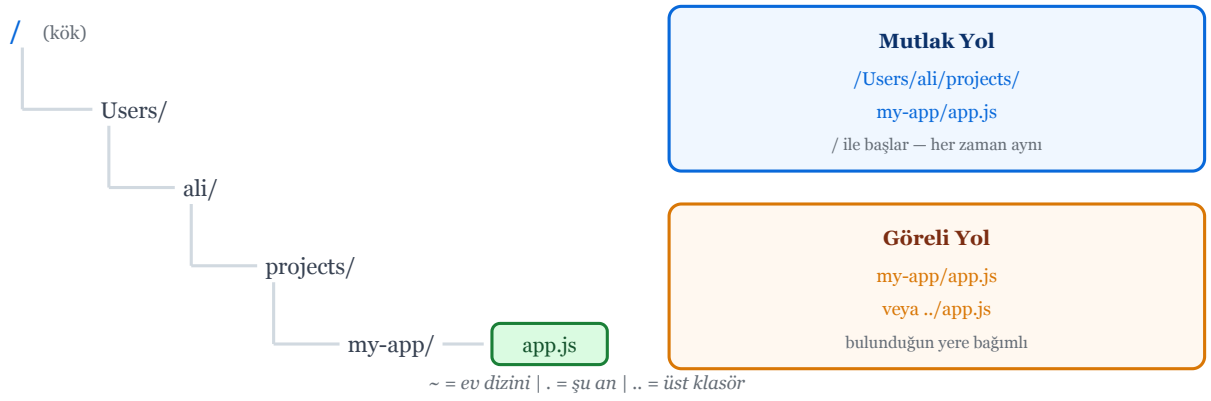
ADIM	ARAÇ	EYLEM
1	Editor	Kodu yazın veya düzenleyin
2	Editor	Dosyayı kaydedin (`Cmd+S` / `Ctrl+S`)
3	Terminal	`node app.js` ile çalıştırın
4	Terminal	Çıktıyı veya hatayı görün
5	(gerekirse) Tarayıcı	Web sonucunu görüntüleyin
6	Editor	Düzeltin, 2. adıma dönün

Profesyonel geliştiriciler bu döngüyü günde yüzlerce kez tekrarlar. Bu nedenle üç pencereyi de aynı anda açık tutmak alışkanlık hâline gelir.

Dene: VS Code'da `merhaba.js` adında yeni bir dosya oluştur. İçine `console.log("İlk programım çalışıyor!")` yaz, kaydet (`Cmd+S` / `Ctrl+S`), terminali aç ve `node merhaba.js` komutunu çalıştır. Çıktıyı gör. Döngüyü tamamladın.

Üç arkadaşla çalışırken hepsinin ortak bir dile ihtiyacı var: dosyaların adresi.

Dosya yolu (path) kavramı



ŞEKİL Z.2.6 *Dosya sistemi ağacı: mutlak yol vs göreli yol*

Bilgisayarınızdaki her dosyanın bir **yolu** vardır. Yol, dosyanın klasör yapısındaki konumunu belirtir:

DİZİ

```
/Users/ali/projects/my-app/src/app.js
└──┘ └─┘ └──────┘ └──────┘ └─┘ └──┘
  │    │     │        │     │    │
 kök  ev  projeler   app   src  dosya
```

İki tür yol vardır:

- **Mutlak yol** (absolute): `/Users/ali/projects/my-app/src/app.js` : kökten başlar.
- **Göreli yol** (relative): `src/app.js` : bulunduğunuz dizine göredir.

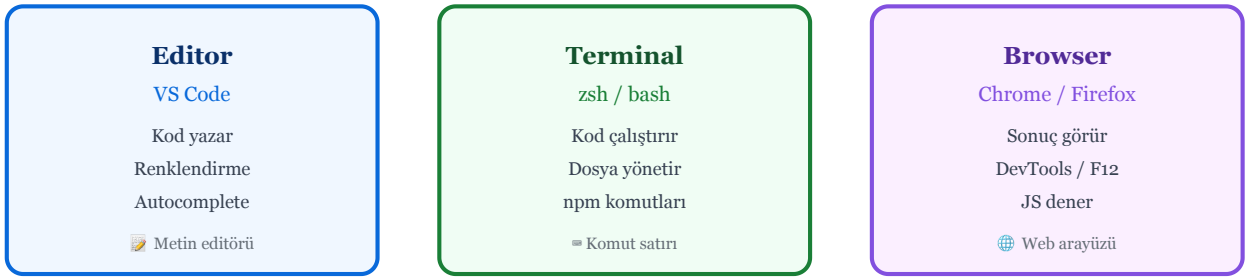
Terminal komutları çoğunlukla göreli yol kullanır:

```
cd ~/projects/my-app    # ~ kullanıcı dizini için kısayol
node src/app.js         # bulunduğum dizinden göreli
```
› TERMİNAL

SEMBOL	ANLAMI
`~`	Kullanıcı ev dizini (`/Users/ali` veya `/home/ali`)
`.`	Bulunduğum dizin
`..`	Bir üst dizin
`/`	Kök dizin

CHEAT SHEET

BÖLÜM Z.2



ŞEKİL Z.2.7 *Editor vs Terminal vs Browser: her aracın rolü*

ARAÇ	İŞİ
Editor (VS Code)	Kodu **yazdığın** yer
Terminal	Komutları **çalıştırdığın** yer (`node app.js`)
Browser	Sonucu **gördüğün** yer (web için)
`cd`	Klasör değiştir
`ls` (Mac) / `dir` (Win)	Klasör içeriğini listele
`pwd`	Şu an neredeyim

KONTROL SORULARI: BÖLÜM Z.2

1. Editor'ın asıl işi nedir? Kod çalıştırır mı?
2. Terminal'de `node server.js` yazınca ne olur?
3. `cd /Users/zeynep/Desktop` komutu ne yapar?
4. Browser DevTools'u nasıl açılır (Mac'te kısayol)?
5. Geliştirme döngüsünü kendi cümlenle yaz.

Araçları tanıdınız; sıradaki adım onları kendi bilgisayarınıza kurmak ve ilk programı çalıştırmak.

▼ YANITLARI GÖR

1. Editor'ın asıl işi kod yazmaktır; metni düzenler, sözdizimini renklendirir. Kodu çalıştırmaz — kodu çalıştıran Node.js ya da tarayıcıdır.
2. `node server.js` komutu Node.js yorumlayıcısını başlatır ve `server.js` dosyasını okuyup çalıştırır. Dosyada hata varsa hata mesajı basar, yoksa kodun yaptığı işi gerçekleştirir.
3. `cd /Users/zeynep/Desktop` komutu terminaldeki çalışma dizinini (current working directory) değiştirir — Zeynep'in masaüstü klasörüne geçiş yapar.
4. Mac'te: `Cmd + Option + I` veya `Cmd + Option + J` (Console için). Alternatif: sağ tık → "İncele" (Inspect) → DevTools açılır.
5. Geliştirme döngüsü: Kod yaz → Kaydet → Çalıştır (`node dosya.js`) → Çıktıyı gör → Hatayı düzelt → Tekrar çalıştır. Bu döngü her özellik için tekrarlanır.

Bölüm Z.2 — Editor, Terminal, Browser: Üç Arkadaş — Test Soruları

1. Bir code editor'ın (VS Code) temel görevi nedir?

A) Kodu çalıştırmak
B) Kodu yazmak ve düzenlemek
C) Web sayfalarını görüntülemek
D) JavaScript motorunu barındırmak
E) Dosyaları sunucuya yüklemek

2. Terminalde `node server.js` yazınca ne olur?

A) server.js dosyası tarayıcıda açılır
B) Dosya VS Code'da düzenlemeye açılır
C) Dosya sunucuya yüklenir
D) Node.js çalışma ortamı server.js dosyasını çalıştırır
E) Dosyanın içeriği terminalde yazdırılır

3. Mac'te Chrome DevTools'u açmanın kısayolu nedir?

A) Cmd+Option+I
B) Cmd+D
C) F5
D) Ctrl+Shift+I
E) Ctrl+Alt+C

4. Doğru geliştirme döngüsü hangi sırayladır?

A) Çalıştır → Yaz → Gör → Düzelt
B) Gör → Yaz → Çalıştır → Düzelt
C) Düzelt → Yaz → Çalıştır → Gör
D) Yaz → Gör → Çalıştır → Kaydet
E) Yaz → Çalıştır → Gör → Hata varsa Düzelt

5. Backend (sunucu) Node.js kodu çalıştırıldığında çıktı nerede görülür?

A) Terminalde
B) Tarayıcı ekranında
C) VS Code'un editör alanında
D) Chrome DevTools Console'da
E) Dosyanın sonuna yazılır

6. VS Code'da entegre terminali açmak için kullanılan kısayol nedir?

A) Ctrl+T
B) Cmd+Enter
C) Ctrl+` (backtick)
D) Alt+T
E) Cmd+Shift+P

7. İlk web tarayıcısını kim ve hangi kurumda geliştirdi?

A) Bill Gates, Microsoft
B) Brendan Eich, Netscape
C) Ryan Dahl, Google
D) Linus Torvalds, Helsinki Üniversitesi
E) Tim Berners-Lee, CERN

8. Terminalde `cd /Users/zeynep/Desktop` komutu ne yapar?

A) Desktop klasörünü siler
B) Terminaldeki aktif dizini Desktop klasörüne değiştirir
C) Desktop klasörünü kopyalar
D) Desktop'taki dosyaları listeler
E) Desktop klasörü oluşturur

9. Editor, Terminal ve Tarayıcı üçlüsünde her birinin rolü sırasıyla nedir?

A) Çalıştır, Yaz, Gör
B) Kaydet, Derle, Yayınla
C) Gör, Yaz, Çalıştır
D) Yaz, Çalıştır, Gör
E) Düzenle, Test et, Yayınla

10. Bir frontend JavaScript kodu nerede çalışır?

A) Terminalde
B) VS Code içinde
C) Tarayıcının JavaScript motorunda
D) Node.js'te
E) Dosya sisteminde

▼ CEVAPLAR VE AÇIKLAMALAR

1 → B
Editor yalnızca kodu yazma ve düzenleme aracıdır. Kodu **çalıştırmaz.** Çalıştırma işi terminal'e (Node.js), görüntüleme ise tarayıcıya aittir.

2 → D
Node.js runtime'ı server.js dosyasını okur ve JavaScript kodunu satır satır çalıştırır. Çıktılar terminalde görünür. Tarayıcı bu süreçte dahil değildir.

3 → A
Mac'te `Cmd+Option+I` Chrome DevTools'u açar. Windows/Linux için `F12` veya `Ctrl+Shift+I`. DevTools, tarayıcıda çalışan JavaScript'i inceleme, hata ayıklama ve network isteklerini görme imkânı sunar.

4 → E
Doğru döngü: **Yaz (editor) → Çalıştır (terminal) → Gör (terminal/tarayıcı) → Hata varsa Düzelt → Tekrar Çalıştır**. Bu döngü kariyeriniz boyunca tekrarlanır.

5 → A
Node.js programı terminalde (`node dosya.js` komutuyla) çalıştırılır ve `console.log` çıktıları terminalde görünür. Tarayıcı yalnızca HTTP üzerinden gelen frontend kodunu çalıştırır.

6 → C
`Ctrl+`` (backtick) VS Code'un entegre terminalini açar. Bu kısayol Mac ve Windows'ta aynıdır. Entegre terminal, ayrı bir terminal uygulaması açmadan doğrudan editörde çalışmayı sağlar.

7 → E
Tim Berners-Lee, 1990'da CERN'de ilk grafik web tarayıcısını yazdı (adı: WorldWideWeb). Aynı zamanda HTTP protokolünü ve HTML dilini de geliştirdi. Web'in mucidi olarak kabul edilir.

8 → B
`cd` (change directory) komutu terminaldeki aktif çalışma dizinini değiştirir. Sonrasında `node dosya.js` gibi komutlar bu dizinde çalışır.

9 → D
Editor (VS Code) → kodu **yazarsınız**. Terminal → kodu **çalıştırırsınız**. Tarayıcı → sonucu **görürsünüz**. Bu üçlü web geliştirme döngüsünün temelidir.

10 → C
Frontend JavaScript, kullanıcının tarayıcısındaki JavaScript motorunda (Chrome'da V8, Firefox'ta SpiderMonkey) çalışır. Backend JavaScript ise sunucudaki Node.js ortamında çalışır.

Tarihsel Arka Plan: Araçların Evrimi

Bu kitap boyunca bu üçlüyü kullanacağız

ŞEKİL Z.2.0 *Geliştirici araçlarının tarihi: vi/emacs → shell → tarayıcı → VS Code + Chrome*

KISIM ZERO – İLK ADIM

Bölüm Z.3 – Geliştirme Ortamının Kurulumu

● ○ ○ ○ ○ BAŞLANGIÇ

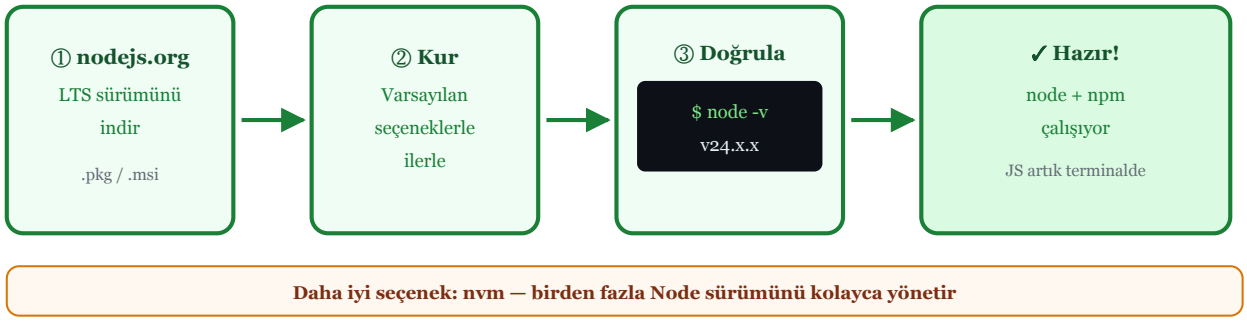
> ◎ NE İŞE YARAR?
> Bilgisayarınıza JavaScript çalıştırmak için gereken iki temel yazılımı kurmak: Node.js (çalışma ortamı) ve VS Code (editor). Bu bölümün sonunda terminalde "node --version" çıktısı alabilecek ve ilk kodunuzu çalıştırabileceksiniz.

– HİKAYE

*2009'da Ryan Dahl, Google'ın Chrome için geliştirdiği V8 JavaScript motorunu tarayıcıdan ayırarak doğrudan işletim sistemi üzerinde çalıştırmanın yolunu buldu. Sonuç: **Node.js**. O güne kadar JavaScript yalnızca tarayıcıda yaşıyordu; Node.js sayesinde sunucularda, komut satırı araçlarında, masaüstü uygulamalarında; her yerde çalıştırılabilir hale geldi. Bugün Node.js, dünyanın en yaygın çalışma ortamlarından biridir.*

Dürüst olalım: kurulum, programlamanın en eğlencesiz kısmıdır — ve yeni başlayanların en çok pes ettiği yer de burasıdır. İyi haber: bu bir kez yapılan, yaklaşık 20 dakikalık bir yatırımdır; sonrası ömür boyu kod. Bir şey ters giderse panik yok — aşağıdaki sorun-giderme tablosu tam bunun için var.

Adım 1: Node.js kurulumu

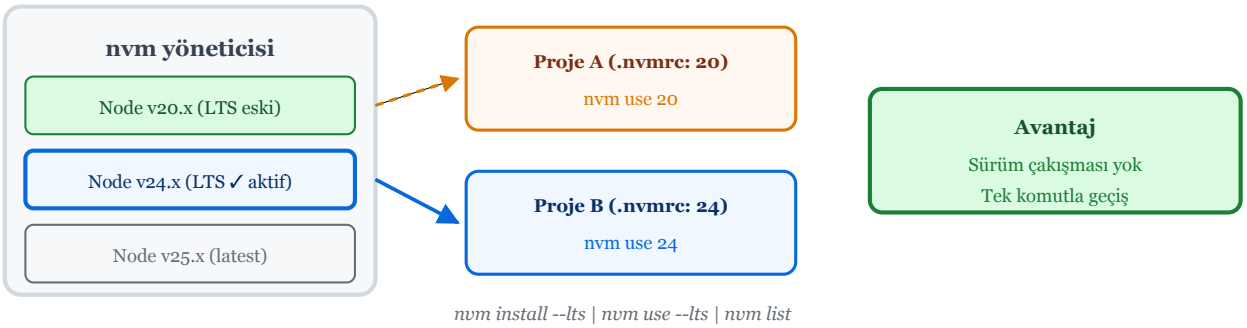


ŞEKİL Z.3.1 *Node.js kurulum akışı: download → install → doğrula*

nodejs.org↗ adresine gidin ve **LTS** (Long-Term Support) sürümünü indirin. LTS sürümü production'da kullanılmaya uygun, stabil sürümdür. Bu kitap yazıldığında güncel LTS Node 24.

Yükleme:

- **macOS / Windows**: İndirdiğiniz `.pkg` veya `.msi` dosyasını açın ve varsayılan seçeneklerle ilerleyin.
- **Linux**: Dağıtımınıza özgü paket yöneticisini kullanın (`apt` , `dnf` , `pacman`), ya da nvm↗ ile sürüm yönetimi yapın.



ŞEKİL Z.3.2 *nvm ile sürüm yönetimi: proje A v20, proje B v24*

Daha iyi alternatif: `nvm` (Node Version Manager): Birden fazla Node sürümünü kolayca yönetmenize olanak tanır. İleride farklı projeler farklı Node sürümleri isteyebilir.

```
# nvm ile Node kurulumu
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
nvm install --lts
nvm use --lts
```

› TERMİNAL

Doğrulama:

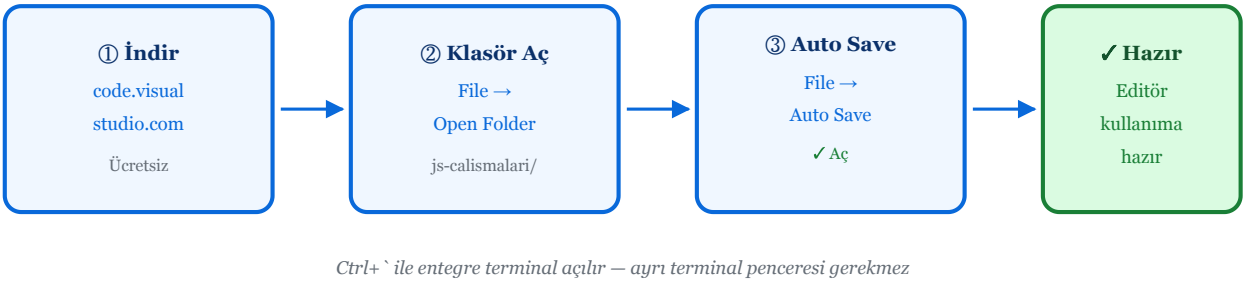
```
node --version    # v24.x.x
npm --version     # 10.x.x
```

› TERMİNAL

`node` ve `npm` (Node Package Manager) komutları çalışıyorsa kurulum başarılı.

Node kuruldu — motor hazır. Şimdi kod yazacağınız tezgâhı kuralım.

Adım 2: VS Code kurulumu



ŞEKİL Z.3.3 *VS Code ilk açılış adımları: klasör aç → auto save → dil ayarı*

code.visualstudio.com ↗ adresinden işletim sisteminize uygun sürümü indirin ve kurun.

İlk açılışta yapılacaklar:

1. **Bir klasör açın**: `File → Open Folder` ile boş bir klasör seçin (örn. `Desktop/js-calismalari`).
2. **Türkçe arayüzü seçebilirsiniz**: `Cmd+Shift+P` → "Configure Display Language" → "Türkçe". Ama hata mesajları İngilizce kalır, çoğu profesyonel İngilizce arayüzde çalışır.
3. **Auto save'i açın**: `File → Auto Save`. Değişiklikler kendiliğinden kaydedilir.

Adım 3: İlk JavaScript dosyanız

VS Code'da:

1. `Cmd+N` (macOS) veya `Ctrl+N` (Windows) ile yeni dosya açın.
2. Şu kodu yazın:

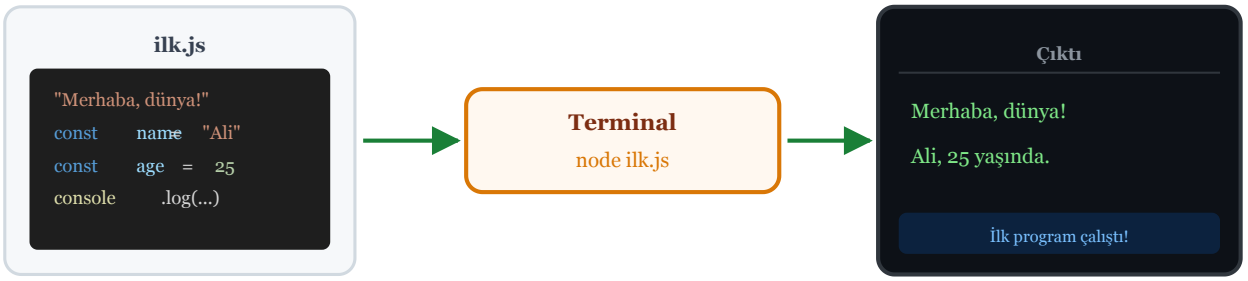
```js
console.log("Merhaba, dünya!");

const name = "Ali";
const age = 25;
console.log(`${name}, ${age} yaşında.`);
```

3. Kaydedin: `Cmd+S` / `Ctrl+S`, dosya adı `ilk.js`.

Önemli: Dosya uzantısı **mutlaka** `.js` olmalı. VS Code uzantıdan dosya tipini anlar ve buna göre renklendirme yapar.

Adım 4: Kodu çalıştırma



Ctrl+` backtick → VS Code entegre terminal

ŞEKİL Z.3.4 *İlk programı çalıştırmak: dosya → terminal → çıktı*

VS Code'un entegre terminalini açın: `Ctrl+`` (backtick, klavyenin sol üst köşesinde, 1'in solundaki tuş).

Terminale şunu yazın:

```
node ilk.js
```

› TERMİNAL

Çıktı:

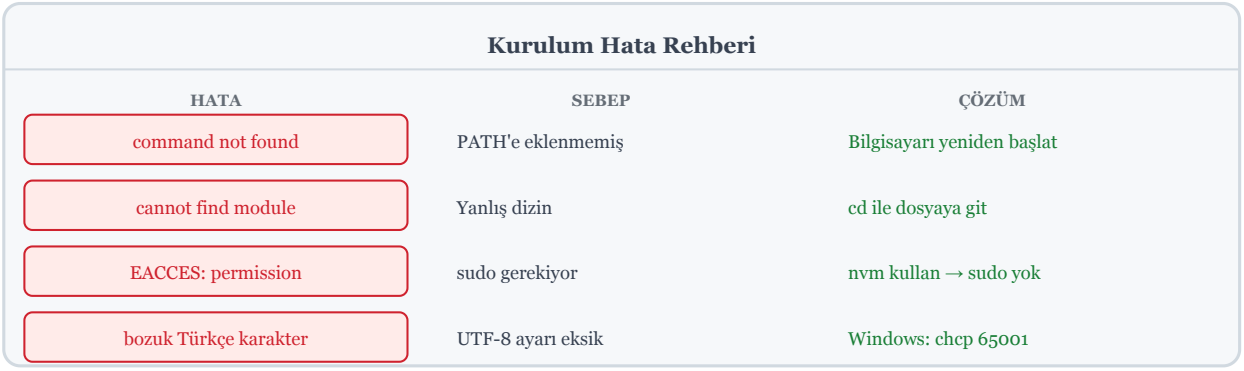
```
Merhaba, dünya!
Ali, 25 yaşında.
```

ÇIKTI

🎉 İlk JavaScript programınız çalıştı. Bu küçük başarı sayılır, geliştirme döngüsünün tamamını (yaz → kaydet → çalıştır → çıktıyı gör) başarıyla tamamladınız.

Bir şey ters mi gitti? Herkeste gider — panik yok. En sık takılınan noktalar ve çözümleri:

Sıkça karşılaşılan kurulum sorunları



ŞEKİL Z.3.5 *Yaygın kurulum hataları ve çözümleri*

HATA	SEBEP	ÇÖZÜM
`node: command not found`	PATH'e eklenmemiş veya kurulum yarım	Bilgisayarı yeniden başlatın; çözmezse Node'u tekrar kurun
`cannot find module './ilk.js'`	Yanlış dizindeyseniz	`cd` komutuyla dosyanın olduğu klasöre geçin
Terminal'de Türkçe karakterler bozuk	UTF-8 ayarı eksik	macOS/Linux'ta sorun yoktur; Windows'ta `chcp 65001` deneyin
VS Code'da Türkçe karakterler bozuk	Dosya encoding'i yanlış	Sağ alt köşede "UTF-8" yazıyor olmalı; değilse tıklayıp değiştirin
`EACCES: permission denied` (npm install)	Sudo gerekiyor (kötü yaklaşım)	Bu uyarı yerine nvm kullanın, sudo'ya gerek kalmasın

Önerilen VS Code eklentileri

ŞEKİL Z.3.6 *Temel VS Code eklentileri ve işlevleri*

VS Code'un sağ kenarındaki **Extensions** ikonuna (dört kare) tıklayın ve şunları aratıp yükleyin:

EKLENTİ	İŞİ
ESLint	Kodda olası hata ve kötü pratikleri kırmızı altçizgi ile gösterir
Prettier	Kodu otomatik formatlar (tutarlı girinti, satır uzunluğu)
GitLens	Git tarihçesini editor içinde görmenize olanak verir
Error Lens	Hata mesajlarını satır sonunda doğrudan gösterir
Pretty TypeScript Errors	TS hatalarını okunabilir hale getirir

Bonus: pnpm ve bun

> ▪ NOT
>
> Node.js alternatifi olan **Bun** ve **Deno** ile ayrıntılı karşılaştırma için Bölüm F9 ve F13'e bakın. Şimdilik standart `npm` + Node.js yeterli.

İleride **pnpm** (npm'in disk dostu alternatifi) veya **bun** (çok hızlı yeni bir *runtime*, yani kodu çalıştıran ortam, Node.js'in alternatifi) ile tanışacaksınız. Şimdilik standart `npm` yeterli, ama not olarak bilin:

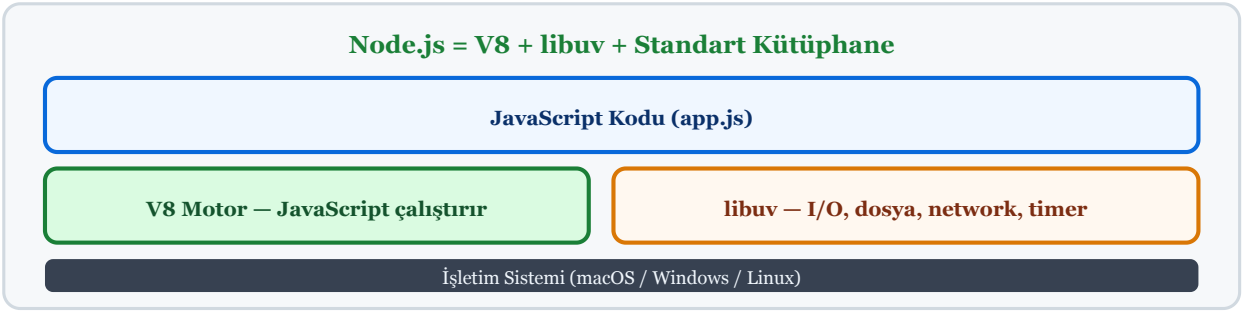
› TERMİNAL

```
# pnpm
npm install -g pnpm

# bun
curl -fsSL https://bun.sh/install | bash
```

Modern tooling konusunu Bölüm F13'te detaylı işliyoruz.

CHEAT SHEET BÖLÜM Z.3



ŞEKİL Z.3.7 *Node.js mimarisi: V8 motoru + libuv + standart kütüphane*

ADIM	KOMUT
Node sürümü	`node --version`
İlk dosya	`echo "console.log('hi')" > app.js`
Çalıştır	`node app.js`
Paket kur	`npm install <name>`
Proje başlat	`npm init -y`

KONTROL SORULARI: BÖLÜM Z.3

1. `node --version` ne işe yarar?
2. VS Code'da terminal nasıl açılır?
3. Bir `.js` dosyası nasıl çalıştırılır?
4. `console.log("Selam")` ile `console.log('Selam')` farklı mı? (Cevap: hayır, JS'te `'` ve `"` aynı.)
5. Sen ilk programını çalıştırdın mı? (Bunu kendine sor, devam et.)

Ortam hazır; bir sonraki bölümde gerçek bir program yazıp değişken ve konsol komutlarını elinizle deneyeceksiniz.

▼ YANITLARI GÖR

1. `node --version` kurulu Node.js'in sürüm numarasını gösterir (örn. `v24.1.0`). Kurulu olmadığında hata döner.
2. VS Code'da: `Ctrl + ``` (backtick) veya üst menüden Terminal → New Terminal. Mac'te `Cmd + ``` .
3. `node dosyaAdi.js` komutuyla. Terminalde önce dosyanın bulunduğu klasöre `cd` ile gidilir, ardından bu komut çalıştırılır.
4. Hayır, farklı değil. JavaScript'te tek tırnak `'` ve çift tırnak `"` string tanımlamak için birbirinin yerine kullanılabilir. Önemli olan açılış ve kapanış tırnağının eşleşmesidir.
5. Kişisel deneyim sorusu — ilk `console.log("Merhaba Dünya")` çalıştırdıktan sonra evet diyebilirsin.

Bölüm Z.3: Geliştirme Ortamının Kurulumu — Test Soruları

1. Node.js indirirken `LTS` seçeneği ne anlama gelir?

A) Latest TypeScript Support; TypeScript desteği olan sürüm
B) Lite Terminal Setup; hafif kurulum sürümü
C) Long-Term Support; uzun vadeli destek alan kararlı sürüm
D) Linux Terminal Standard; Linux için özel sürüm
E) Last Tested Software; son test edilmiş sürüm

2. Node.js kurulumunun başarılı olduğunu doğrulamak için ne yazılır?

A) `npm start`
B) `node install`
C) `javascript -v`
D) `node -v` veya `node --version`
E) `check node`

3. Node.js kurulduğunda yanında otomatik gelen paket yöneticisi hangisidir?

A) yarn
B) pnpm
C) bun
D) npm
E) pip

4. nvm (Node Version Manager) ne işe yarar?

A) Node.js için grafik arayüz sağlar
B) npm paketlerini görselleştirir
C) Aynı bilgisayarda birden fazla Node.js sürümünü yönetir
D) VS Code eklentisi olarak çalışır
E) Node.js güncellemelerini bildirir

5. JavaScript kaynak dosyalarının uzantısı nedir?

A) `.java`
B) `.js.txt`
C) `.javascript`
D) `.jscript`
E) `.js`

6. Bir `.js` dosyasını Node.js ile çalıştırmak için terminal komutu nedir?

A) `run merhaba.js`
B) `execute merhaba.js`
C) `start merhaba.js`
D) `npm merhaba.js`
E) `node merhaba.js`

7. Node.js'i kim geliştirdi ve ne zaman?

A) Brendan Eich, 1995
B) Ryan Dahl, 2009
C) Tim Berners-Lee, 1990
D) Linus Torvalds, 2003
E) Douglas Crockford, 2008

8. VS Code'da yeni bir dosya oluşturmak için kısayol nedir?

A) Cmd+N (Mac) / Ctrl+N (Windows)
B) Ctrl+S
C) Ctrl+O (Open)
D) Alt+F
E) Ctrl+P

9. Node.js çalışma ortamı ile tarayıcı JavaScript ortamı arasındaki temel fark nedir?

A) Node.js daha yavaştır
B) Node.js'te `window`, `document` gibi tarayıcı API'leri yoktur; bunun yerine `fs`, `http` gibi sunucu API'leri vardır
C) Tarayıcıda `const` / `let` kullanılamaz
D) Node.js yalnızca TypeScript çalıştırır
E) Tarayıcı daha fazla bellek kullanır

10. Kurulum sonrasında ilk `node -v` çıktısı `v24.1.0` görünüyor. Bu ne demektir?

A) Node.js 24. versiyon, 1. güncelleme paketi, 0. hata düzeltmesi kurulu
B) 24 GB bellek kullanılıyor
C) Kurulum 24 saniye sürdü
D) v24 deneysel sürümdür, kullanılmamalı
E) Sadece sürüm 24 LTS değildir

▼ CEVAPLAR VE AÇIKLAMALAR

1 → C
LTS (Long-Term Support), uzun vadeli güvenlik ve hata düzeltme desteği sağlayan kararlı sürümdür. Production ve öğrenme ortamı için önerilir. Current sürümü ise yeni özellikleri erken dener ama daha az kararlı olabilir.

2 → D
`node -v` veya `node --version` komutunu terminalde çalıştırdığınızda `v24.x.x` gibi bir sürüm numarası görüyorsanız Node.js başarıyla kurulmuştur. Hata alırsanız kurulum tamamlanmamış demektir.

3 → D
npm (Node Package Manager), Node.js ile birlikte otomatik yüklenir. `npm -v` ile sürümünü kontrol edebilirsiniz. yarn, pnpm, bun ise alternatif paket yöneticileridir, ayrıca kurulmaları gerekir.

4 → C
nvm, aynı bilgisayarda birden fazla Node.js sürümü kurmanıza ve aralarında kolayca geçiş yapmanıza olanak tanır. Farklı projeler farklı Node sürümleri gerektirebilir; nvm bu sorunu çözer.

5 → E

JavaScript dosyaları `.js` uzantısını kullanır. TypeScript `.ts`, React bileşenleri `.jsx` veya `.tsx` kullanabilir. Dosya adını kaydederken uzantıyı unutmak kodu çalıştıramamaya yol açar.

6 → E

`node dosyaadı.js` komutu Node.js runtime'ını belirtilen JavaScript dosyasını çalıştırmak için başlatır. `npm start` ise `package.json`'daki `start` scriptini çalıştırır, doğrudan dosya değil.

7 → B

Ryan Dahl, 2009'da Google'ın V8 JavaScript motorunu tarayıcıdan ayırarak doğrudan işletim sistemi üzerinde çalıştırdı. Bu sayede JavaScript sunucularda da çalışabilir hale geldi.

8 → A

`Cmd+N` (Mac) veya `Ctrl+N` (Windows) VS Code'da yeni boş bir dosya açar. Ardından `Cmd+S` / `Ctrl+S` ile kaydedip uzantı verebilirsiniz.

9 → B

Node.js'te `window`, `document`, `localStorage` gibi tarayıcı API'leri bulunmaz. Bunun yerine dosya sistemi (`fs`), ağ (`http`), süreç kontrolü (`process`) gibi sunucu API'leri vardır. JavaScript dili aynı, ortam farklıdır.

10 → A

Semantic versioning (SemVer) formatında `MAJOR.MINOR.PATCH` yapısı kullanılır. `v24.11.0` → major sürüm 24, minor 11, patch 0. Çift major numaraları (20, 22, 24) LTS sürümlerini belirtir.

KISIM ZERO – İLK ADIM

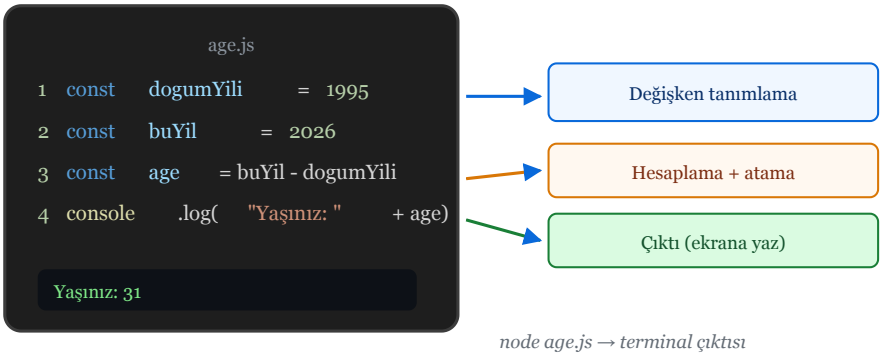
Bölüm Z.4 – İlk Programınızı Yazıyorsunuz

● ○ ○ ○ ○ BAŞLANGIÇ

> ◎ NE İŞE YARAR?
>
> Şimdiye kadar kavramları okudunuz. Bu bölüm sıra **sizde**: yazacak, çalıştıracak ve sonucu kendi gözlerinizle göreceksiniz. Birkaç satırlık küçük programlar üzerinden değişken, ifade, çıktı kavramlarını somut hale getireceğiz.

1972'de Bell Labs'ta Brian Kernighan, B programlama dilinin tanıtım dokümanı için en küçük çalışan programı yazdı: `printf("hello, world\n")`. Bu basit satır, sonraki on yıllarda her programlama kitabının ve her geliştiricinin ilk adımı haline geldi. "Hello World" yazıp çalıştığını görmek bir geleneğin parçasıdır.

Adım adım: Yaş hesaplayıcı



ŞEKİL Z.4.1 *Yaş hesaplayıcı: 4 satır, 3 kavram (değişken, işlem, çıktı)*

VS Code'da yeni bir dosya açın (`Cmd+N` / `Ctrl+N`). Adını `age.js` koyup `js-calismalari` klasörünüze kaydedin (`Cmd+S` / `Ctrl+S`).

İçine şu kodu yazın (kopyalamak yerine elle yazmanızı öneririm; kas hafızası kuruluyor):

```js
const dogumYili = 1995;
const buYil = 2026;
const age = buYil - dogumYili;
console.log("Yaşınız: " + age);
```

Terminalden çalıştırın:

```
node age.js
```

› TERMİNAL

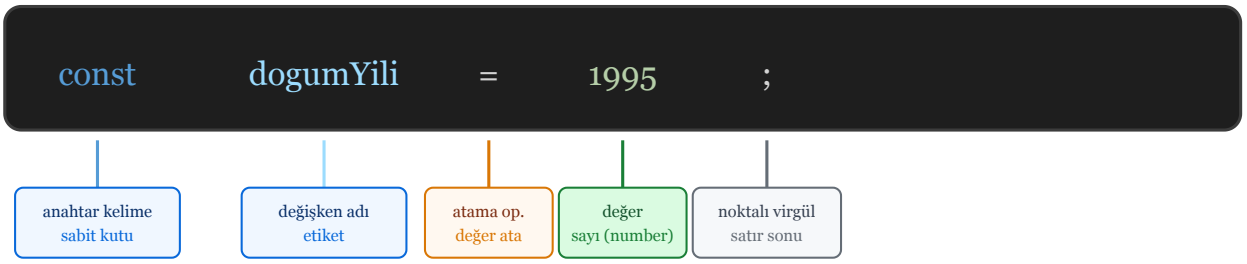
Çıktı:

```
Yaşınız: 31
```

ÇIKTI

Dene: Doğum yılını kendininkiyle değiştir ve kaydet. Terminalden tekrar çalıştır; sonuç değişti mi? Sonra `age` yerine kendi adını tutan bir değişken ekle ve `"Merhaba, Yaşınız: "` çıktısı üret.

Her satır ne yapıyor?



Sağ taraf önce hesaplanır, sonra sol taraftaki isme atanır

ŞEKİL Z.4.2 *Değişken anatomisi: const · isim · = · değer · ;*

Bu küçük programda 4 ayrı kavram bir araya geldi. Tek tek inceleyelim:

Satır 1-2: Değişken tanımlama

```js
const dogumYili = 1995;
const buYil = 2026;
```

Anatomi:

PARÇA	ANLAMI
`const`	"Sabit değişken oluştur"; değeri sonradan değiştirilemez
`dogumYili`	Değişkenin adı (geliştiricinin verdiği isim)
`=`	Atama operatörü ("şu değeri ata")
`1995`	Atanacak değer (bir sayı)
`;`	Satırın sonu (zorunlu değil ama önerilir)

Türkçeye çevirirsek: "`dogumYili` *adında değiştirilemez bir değişken oluştur ve içine 1995 sayısını koy.*"

Satır 3: Hesaplama ve atama

```js
const age = buYil - dogumYili;
```

Sağ taraf önce hesaplanır: `2026 - 1995 = 31`. Sonra sonuç `age` adıyla saklanır. Bu sıralama önemli, JavaScript her zaman önce ifadeyi (expression) çözer, sonra atamayı yapar.

Satır 4: Çıktı

```js
console.log("Yaşınız: " + age);
```

PARÇA	ANLAMI
`console.log(...)`	Parantez içindekini ekrana yaz
`"Yaşınız: "`	Metin (string); çift tırnak veya tek tırnak fark etmez
`+`	İki şeyi birleştir (concatenation)
`age`	`31` değerinin bulunduğu değişken

Sonuç: `"Yaşınız: " + 31` → `"Yaşınız: 31"` (sayı otomatik metne çevrilir; buna **tip dönüşümü / type coercion** denir; ileride detaylı inceleyeceğiz).

Küçük bir test: sence `"1" + "1"` kaç eder — `2` mi, `"11"` mi? Cevap `"11"`; `+` operatörü string görünce *birleştirir*. Ama `"5" - 3` → `2`; çıkarma her zaman *sayısallaştırır*. Bu çelişki JavaScript'in en ünlü sürprizidir — Bölüm 2'de tuzaklarıyla birlikte inceleyeceğiz.

Yorum satırları

Kod yazarken "bu satır ne yapıyor?" sorusunu not bırakmak istersiniz. Bu notlar **yorum** (comment) olarak yazılır. Bilgisayar yorumları tamamen **yok sayar**; sadece sizin veya ekibiniz için var.

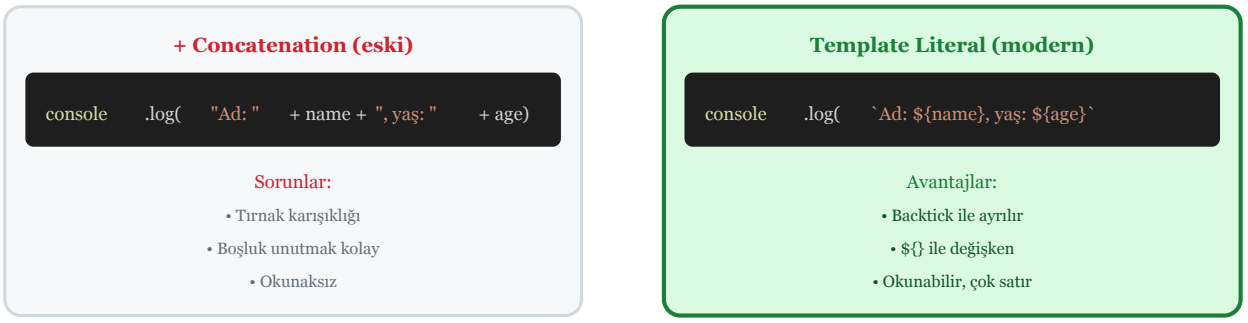
```js
// Bu bir tek satır yorum — # ile değil, // ile başlar
const age = 31; // değişkenin sağına da yorum eklenebilir

/*
  Bu çok satırlı yorum.
  Uzun açıklamalar, geçici olarak devre dışı bırakılan kod için kullanılır.
*/
const name = "Ali";
```

> • İPUCU
>
> "Bu kodu yaz" yerine "neden bu kodu yazdım" sorunu yanıtlayan yorumlar değerlidir. `// sayı artır` yerine `// loop her iterasyonda bir mesaj gruplar` gibi.

`+` ile birleştirme çalışır, ama gözü yorar ve hata davet eder. Profesyoneller aynı işi daha temiz yapar:

Modern alternatif: Template literal



ŞEKİL Z.4.3 *String birleştirme: + concatenation vs template literal*

`+` ile birleştirme yerine, günümüz JavaScript'te **template literal** (şablon dizgisi) kullanılır. Tırnak yerine backtick (`` ` ``) ve `${ifade}` sözdizimi:

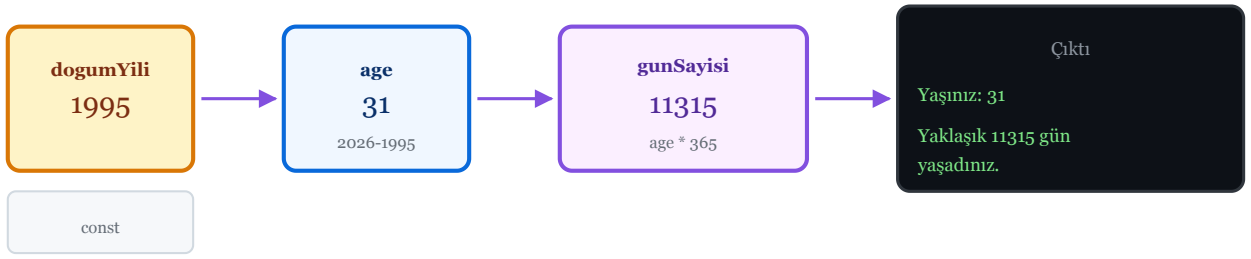
```JS
console.log(`Yaşınız: ${age}`);
```

Avantajları:

- Birden fazla değişken eklemek kolay: `` `${ad} ${soyad}, ${yas} yaşında` ``
- Satır sonları korunur (multi-line string)
- Okunabilirliği artırır

Bu kitap boyunca template literal'i tercih edeceğiz.

Pratik 1: Yaşam günü hesabı



ŞEKİL Z.4.6 *Değişken zinciri: birinden diğerine hesap yapma*

Aşağıdakini kendiniz deneyin:

1. `age.js` dosyasındaki doğum yılını **kendi doğum yılınızla** değiştirin.
2. Yeni bir değişken ekleyin: `const gunSayisi = age * 365;`
3. Yeni bir `console.log` ekleyin: `console.log(`Yaklaşık ${gunSayisi} gün yaşadınız.`);`
4. Çalıştırın.

Beklenen çıktı (1995 doğumlu için):

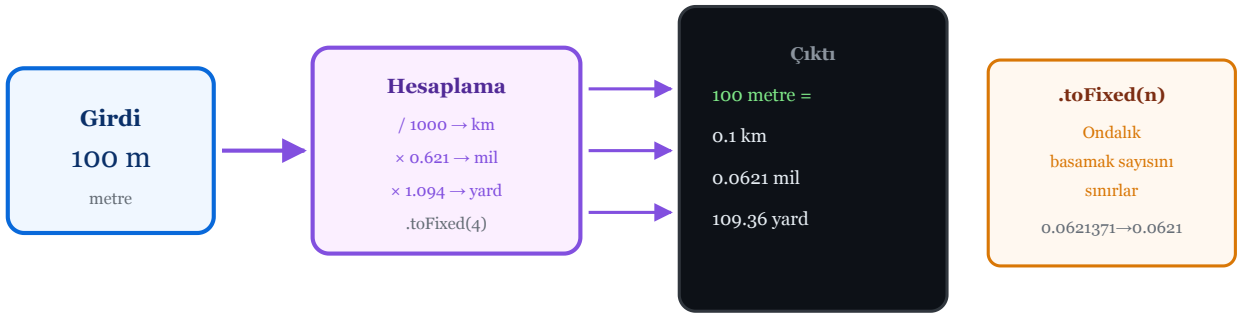
```ÇIKTI
Yaşınız: 31
Yaklaşık 11315 gün yaşadınız.
```

Sorun mu çıktı? Hata mesajını dikkatle okuyun. Genellikle hatanın satır numarası ve sebebi yazar:

```HATA
SyntaxError: missing ) after argument list
    at /path/to/age.js:4
```

Bu mesaj "4. satırda kapanış parantezi eksik" diyor. Bu, geliştirmenin normal bir parçası.

Pratik 2: Birim çevirici



ŞEKİL Z.4.4 *Birim çevirici veri akışı: metre → kilometre, mil, yard*

İkinci bir dosya oluşturun: `cevirici.js`. Aşağıdaki birim dönüşümlerini hesaplayan bir program yazın:

```js
const metre = 100;
const kilometre = metre / 1000;
const mil = kilometre * 0.621371;
const yard = metre * 1.09361;

console.log(`${metre} metre =`);
console.log(`  ${kilometre} kilometre`);
console.log(`  ${mil.toFixed(4)} mil`);
console.log(`  ${yard.toFixed(2)} yard`);
```

`.toFixed(n)` ondalık basamak sayısını sınırlar (`0.0621371` → `0.0621`).

Pratik 3: Saat-saniye çevirici

```js
const saat = 3;
const saniye = saat * 60 * 60;
console.log(
  `${saat} saat = ${saniye.toLocaleString("tr-TR")} saniye`,
);
```

`toLocaleString("tr-TR")` sayıyı Türk biçiminde gösterir (10.800).

ALIŞTIRMA Z.4 — Toplam fiyat

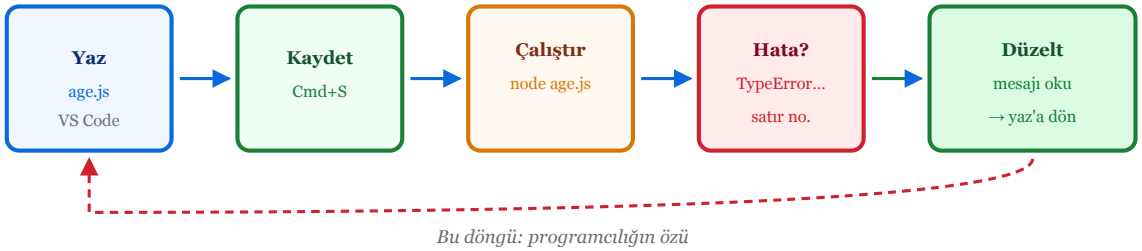
Üç ürünün fiyatını topla. `kalem = 5`, `defter = 12`, `silgi = 3`. Toplamı bir `toplam` değişkenine koy, yazdır.

```
// Kodunu buraya yaz:
```

Çözümü gör → (Çözümler bölümü)

CHEAT SHEET

BÖLÜM Z.4



ŞEKİL Z.4.5 *Geliştirme döngüsü tamamlandı: yaz → kaydet → çalıştır → hata → düzelt*

KONU	ÖRNEK
Değişken	`const name = "Ali";`
Yazdır	`console.log(name);`
Matematik	`const age = 2026 - 1995;`
Birleştir	`` `Merhaba ${isim}` ``

KONTROL SORULARI: BÖLÜM Z.4

1. `//` ne işe yarar?
2. `const age = 31;` cümlesini Türkçe'ye çevir.

> • STRİNG Mİ, NUMBER MI?
>
> JavaScript'te tırnak içindeki her şey **string**'dir: `"3"`, `"merhaba"`, `"42"`. Tırnaksız sayılar ise **number**'dır: `3`, `42`, `3.14`. `"3 + 5"` bir string ifadesidir; `3 + 5` ise sayısal toplama işlemidir.

3. `"3 + 5"` ile `3 + 5` farklı mı? Çıktısı?
4. `console.log(1 + 1)` ne yazdırır?
5. `console.log("1" + "1")` ne yazdırır? (UYARI: tuzak)

▼ YANITLARI GÖR

Cevaplar: 1) Yorum, bilgisayar yok sayar. 2) "yas adında kalıcı kutu yarat, içine 31 koy". 3) "3 + 5" string, çıktı `"3 + 5"`; `3 + 5` matematik, çıktı `8`. 4) `2`. 5) `"11"` (string concat).

İlk programınızı yazdınız; bir sonraki bölümde bu değişkenlerin ve fonksiyonların zihinsel modelini netleştireceğiz: değişken = kutu, fonksiyon = makine.

Bölüm Z.4: İlk Programınızı Yazıyorsunuz — Test Soruları

1. Aşağıdaki kod çalıştırıldığında konsola ne yazdırılır?

```js
const dogumYili = 1995;
const buYil = 2026;
console.log("Yaşınız: " + (buYil - dogumYili));
```

A) `Yaşınız: 19952026`
B) `Yaşınız: buYil - dogumYili`
C) `Yaşınız: 31`
D) Hata verir
E) `31`

2. JavaScript'te tek satır yorum nasıl yazılır?

A) `/* yorum */`
B) `// yorum`
C) `<!-- yorum -->`
D) `# yorum`
E) `-- yorum`

3. Aşağıdaki ifadenin sonucu nedir?

```js
console.log("3" + 5);
```

A) `"35"`
B) `35`
C) `8`
D) Hata
E) `3+5`

4. Aşağıdaki ifadenin sonucu nedir?

```js
console.log("3" - 5);
```

A) `-2`
B) Hata
C) `"3-5"`
D) `35`
E) `undefined`

5. `console.log()` fonksiyonu ne işe yarar?

A) Değişkeni siler
B) Değere değer atar
C) Programı sonlandırır
D) Değeri terminale (konsola) yazdırır
E) Bir dosyaya yazar

6. "Hello World" programını yazan ilk kişi kimdir ve hangi dilin dokümanında yer aldı?

A) Dennis Ritchie, C dili
B) Guido van Rossum, Python
C) Brendan Eich, JavaScript
D) Bjarne Stroustrup, C++
E) Brian Kernighan, B dili

7. VS Code'da dosya kaydetme kısayolu nedir?

A) Ctrl+Enter
B) Alt+S
C) Cmd+S (Mac) / Ctrl+S (Windows)
D) Cmd+W
E) Ctrl+X

8. Aşağıdaki kodda kaç değişken tanımlanmıştır?

```js
const name = "Ali";
let age = 25;
const city = "İstanbul";
```

A) 1
B) 2
C) 0
D) 4
E) 3

9. Geliştirme döngüsü `Yaz → Kaydet → Çalıştır → Hata var sa Oku → Düzelt → Tekrar Çalıştır` şeklinde tarif edilir. Bu döngü ne zaman biter?

A) İlk hata giderildiğinde
B) Kaydet adımı yapıldığında
C) 100 kez çalıştırıldığında
D) Program beklenen sonucu verdiğinde ve hata kalmadığında
E) Hiçbir zaman bitmez

10. Aşağıdakilerden hangisi `const` ile tanımlanmış bir değişken için geçerlidir?

A) Değeri sonradan değiştirilebilir
B) Tanımlandıktan sonra değeri değiştirilemez
C) Aynı isimle ikinci kez tanımlanabilir
D) Fonksiyon kapsamlıdır
E) Hoist edilir ve `undefined` değer alır

▼ CEVAPLAR VE AÇIKLAMALAR

1 → C
Parantez içinde `2026 - 1995 = 31` hesaplanır ve sonuç sayı 31'dir. `+` operatörü bir taraf string (`"Yaşınız: "`), diğer taraf sayı (31) olunca string birleştirme yapar: `"Yaşınız: 31"` .

2 → B
`//` ile satır sonuna kadar olan her şey yorum sayılır ve JavaScript motoru tarafından görmezden gelinir. Çok satırlı yorum için `/* ... */` kullanılır. `#` Python'da, `<!--` HTML'de, `--` SQL'de yorum belirtir.

3 → A

`+` operatöründe bir operand string ise diğeri de stringe dönüştürülür. `"3"` string, `5` sayı → `"3" + "5"` → `"35"`. Bu JavaScript'in en yaygın tuzaklarından biridir.

4 → A

`-` operatörü her zaman sayısal işlem yapar. `"3"` string → `3` sayısına dönüştürülür → `3 - 5 = -2`. `-`, `*`, `/`, `%` operatörleri `+`'ın aksine string birleştirme yapmaz.

5 → D

`console.log()` JavaScript'in en temel çıktı fonksiyonudur. Verilen değeri (string, sayı, nesne vb.) konsola yazdırır. Node.js'te terminale, tarayıcıda DevTools Console'a yazar.

6 → E

1972'de Brian Kernighan, Bell Labs'ta B programlama dilinin tanıtım dokümanı için `printf("hello, world\n")` yazdı. Bu basit gelenek daha sonra C dili kitabında da yer aldı ve tüm programlama dünyasına yayıldı.

7 → C

`Cmd+S` (Mac) / `Ctrl+S` (Windows) aktif dosyayı kaydeder. Kaydedilmemiş dosyalar VS Code'da sekme başlığında nokta (•) ile belirtilir. Kodu çalıştırmadan önce daima kaydetmeyi alışkanlık edinin.

8 → E

`name`, `age` ve `city` olmak üzere 3 değişken tanımlanmıştır. `const` ve `let` anahtar kelimeleri değişken türünü belirtir; her tanımlama ayrı bir değişken oluşturur.

9 → D

Döngü; program doğru çalıştığında, beklenen çıktıyı verdiğinde ve bilinen hata kalmadığında tamamlanmış sayılır. Ancak bu döngü kariyeriniz boyunca tekrar eder, her yeni özellik veya düzeltme için.

10 → B

`const` ile tanımlanan değişkene sonradan yeniden değer atanamaz (`const x = 5; x = 10;` → TypeError). Ancak `const` ile tanımlanan bir nesne veya dizinin **içeriği** değiştirilebilir, bu, referansın değişmemesi anlamına gelir.

Örneğin Sonu

Bu bir **önizleme örneğidir**.

Elinizdeki bu örnek, *Koddan Kariyere: JavaScript & Node.js ile Sıfırdan Profesyonele* kitabının yalnızca ilk bölümlerini içerir. Kitabın tamamı 157+ bölüm, alıştırma ve çözümleriyle birlikte çok daha kapsamlıdır.

Kitabın tamamına ulaşmak için Amazon KDP veya Apple Books üzerinden *Koddan Kariyere*'i edinebilirsiniz.

– Mert Türedü

Sıfırdan Profesyonele

Tek Yol. Eksiksiz.

Bu kitap, ilk kod satırından gerçek dünyada kullanılabilecek, üretime hazır uygulamalar geliştirmeye kadar JavaScript ve Node.js'de ustalaşman için eksiksiz bir yol haritasıdır.

Sadece nasıl yapıldığını öğrenmeyecek, neden öyle yapıldığını da anlayacaksın. Her konu; tarihçesi, detaylı açıklamaları, görselleri, alıştırmaları ve bilgiyi kalıcı hâle getiren özet notlarıyla birlikte sunulur.

İster tamamen yeni başlayan biri ol, ister JavaScript'e geçen deneyimli bir geliştirici, bu kitap sana uyum sağlar. İstersen baştan sona sırayla oku, istersen kendi öğrenme yolunu seç.

JavaScript Temellerinden İleri Düzey Konulara

Değişkenler, fonksiyonlar, closure'lar, nesne yönelimli programlama (OOP), asenkron programlama, modüller, ES özellikleri ve çok daha fazlası.

Node.js Temellerinden Gerçek Projelere

Event Loop, Stream'ler, HTTP, REST API'leri, kimlik doğrulama, veritabanları, test süreçleri ve canlı ortama dağıtım (deployment).

Veritabanları ve SQL

Gerçek örneklerle PostgreSQL, sorgular, optimizasyon teknikleri ve en iyi uygulamalar.

Modern Araçlar ve En İyi Uygulamalar

ES Modülleri, TypeScript, test süreçleri, tasarım desenleri, güvenlik, performans ve daha fazlası.

Uygulamalı Alıştırmalar ve Hızlı Referans Notları

Öğrenirken pratik yap. Her bölüm, alıştırmalar ve hızlı başvuru notlarıyla sona erer.

Mülakatlar İçin Algoritmik Düşünme

Problem çözme becerini algoritmik kalıplar, Big O analizi, veri yapıları ve klasik mülakat sorularıyla geliştir.

MERT TÜREDÜ

Resources, updates and corrections:
mertturedu.thedev@gmail.com