Bölüm 4.3: Programlamaya Giriş

Bilgisayar ve Programlamaya Giriş

Diller | Çeviri | Problem çözme | Yazılım süreci | Temel yapılar

Sercan KÜLCÜ · Giresun Üniversitesi

© Sercan Külcü. Tüm hakları saklıdır. · Sunumun etkileşimli HTML sürümü: https://sercankulcu.github.io/files/computer/slides/Bolum_04_03_Programlamaya_Giris.html

GİRİŞ

Öğrenme Çıktıları

Kavramlar

Kaynak kod, makine dili ve programlama dilini ayırt etmek.

Çalıştırma

Derleme, yorumlama ve hibrit çalışma biçimlerini açıklamak.

Problem çözme

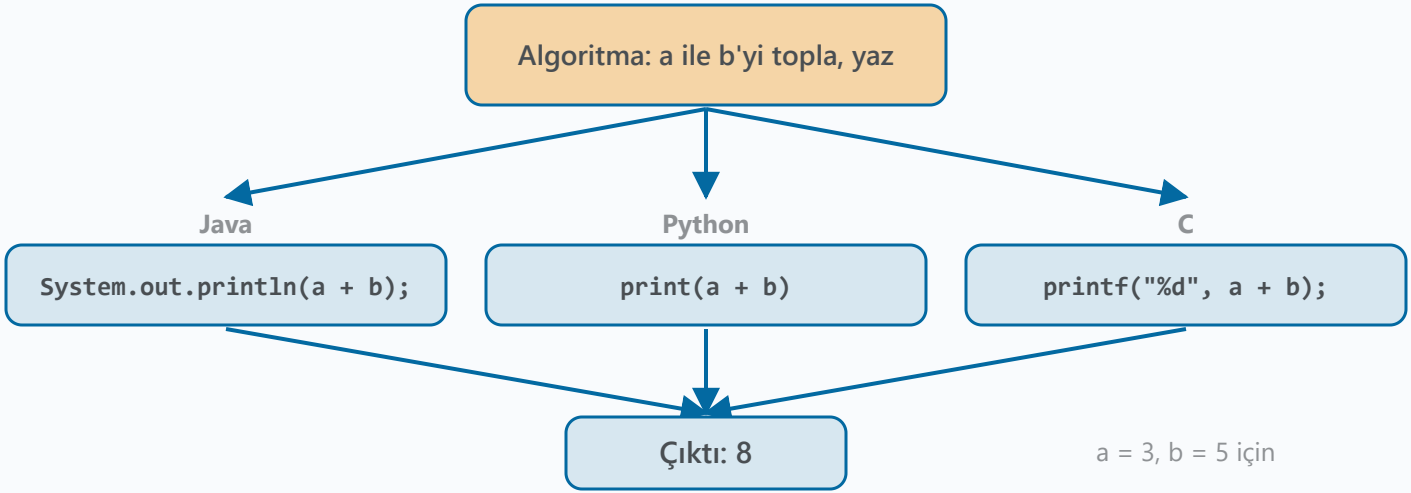
Algoritmadan programa giden süreci kurmak.

Yapılar

Değişken, karar, döngü, fonksiyon ve diziyi tanımak.

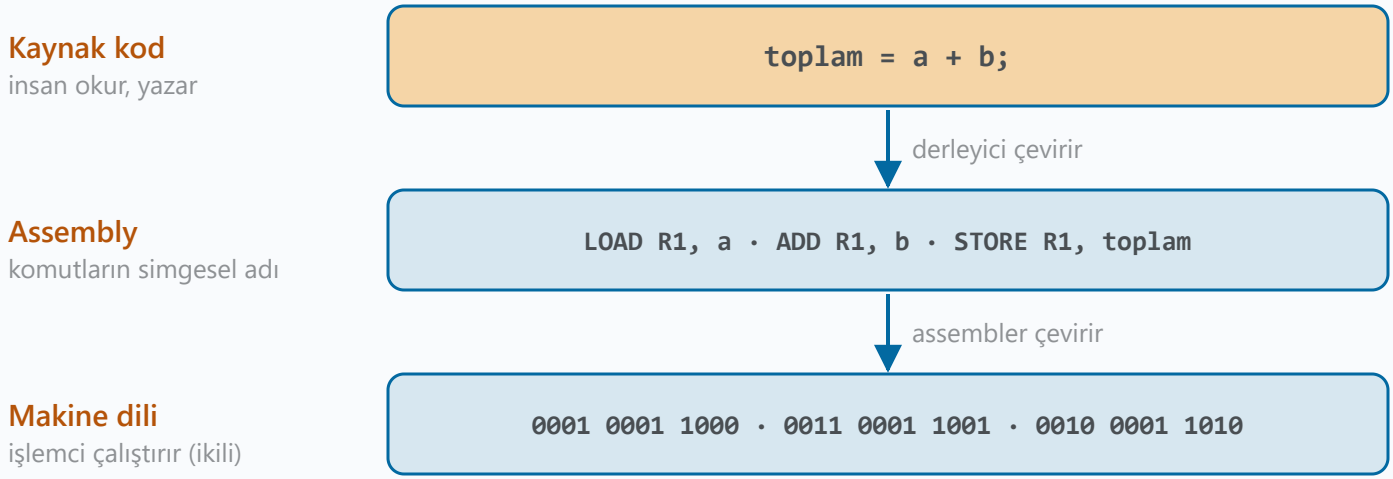
Programlama Dili Nedir?

- İnsanların bilgisayara işlem adımlarını ifade etmesini sağlayan biçimsel bir dildir.
- **Söz dizimi** (syntax) hangi yapıların yazılabileceğini, **anlam** (semantics) bu yapıların ne yaptığını belirler.
- Aynı problem farklı dillerle çözülebilir; seçim gereksinim ve ortama bağlıdır.



TEMEL KAVRAMLAR

Kaynak Kod ve Makine Dili



Makine kodu örnektir (gerçek kodlama işlemciye göre değişir).

- Kaynak kod geliştiricinin okuyup değiştirebildiği program metnidir; makine dili işlemcinin çalıştırdığı ikili komutlardır.
- Derleyici veya yorumlayıcı aradaki köprüyü kurar: programlama dili ile işlemci dili aynı şey değildir.

ÇALIŞTIRMA · ADIM ADIM

Program Nasıl Çalışır?



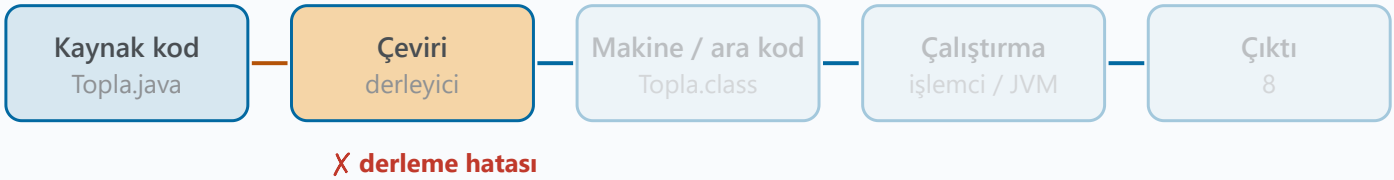
Programcı **kaynak kodu** bir metin dosyasına yazar. Bilgisayar bu metni doğrudan çalıştıramaz.

Adım 1 / 5

Hata türleri: **derleme** (söz dizimi kuralı), **çalışma zamanı** (yürütme sırasında), **mantık** (yanlış sonuç).

ÇALIŞTIRMA · ADIM ADIM

Program Nasıl Çalışır?



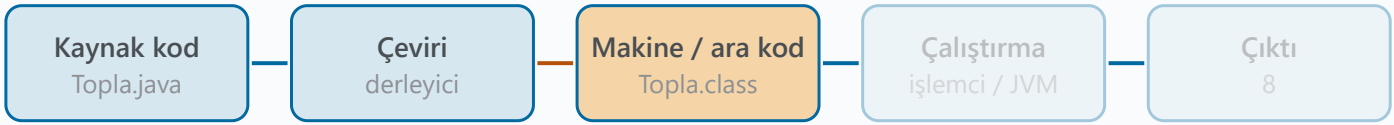
Çeviri: derleyici söz dizimini denetler. Noktalı virgül unutulursa **derleme hatası** burada çıkar, program hiç çalışmaz.

Adım 2 / 5

Hata türleri: **derleme** (söz dizimi kuralı), **çalışma zamanı** (yürütme sırasında), **mantık** (yanlış sonuç).

ÇALIŞTIRMA · ADIM ADIM

Program Nasıl Çalışır?



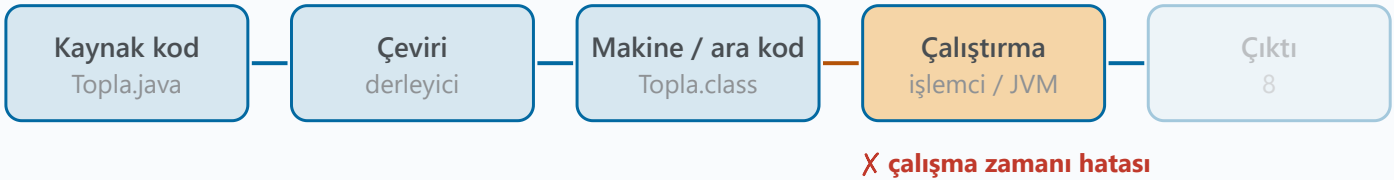
Hata yoksa **makine kodu** ya da Java'da olduğu gibi **ara kod** (bytecode) üretilir.

Adım 3 / 5

Hata türleri: **derleme** (söz dizimi kuralı), **çalışma zamanı** (yürütme sırasında), **mantık** (yanlış sonuç).

ÇALIŞTIRMA · ADIM ADIM

Program Nasıl Çalışır?

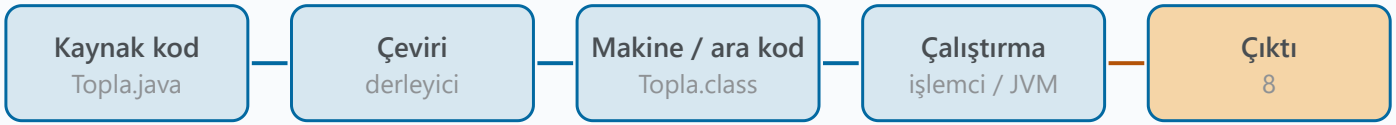


Çalıştırma: komutlar işlemcide (Java'da JVM üzerinden) yürütülür. 0'a bölme gibi **çalışma zamanı hataları** burada ortaya çıkar.

Hata türleri: **derleme** (söz dizimi kuralı), **çalışma zamanı** (yürütme sırasında), **mantık** (yanlış sonuç).

ÇALIŞTIRMA · ADIM ADIM

Program Nasıl Çalışır?



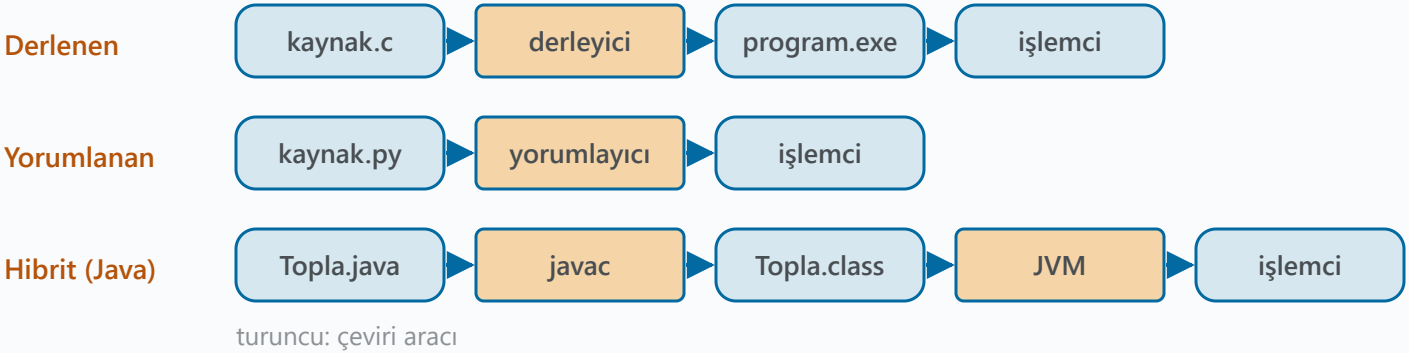
Çıktı, verilen girdiye (a = 3, b = 5) ve program mantığına bağlıdır. Mantık hatası varsa program çalışır ama yanlış sonuç verir.

Adım 5 / 5

Hata türleri: **derleme** (söz dizimi kuralı), **çalışma zamanı** (yürütme sırasında), **mantık** (yanlış sonuç).

Çalıştırma Yaklaşımları

Yaklaşım	İşleyiş	Örnek
Derlenen	Program önceden hedef makine koduna çevrilir	C, C++, Rust
Yorumlanan	Komutlar çalışma sırasında değerlendirilir	Python, JavaScript
Hibrit	Ara kod sanal makinede yürütülür	Java, C#



turuncu: çeviri aracı

ÇALIŞTIRMA · ADIM ADIM

Derleyici ve Yorumlayıcı Adım Adım

Derleyici: önce tümü çevrilir, sonra çalışır

1	`yaz "Merhaba"`	çevriliyor
2	`x ← 10`	çevriliyor
3	`yaz x / 0`	çevriliyor

konsol



Derleyici programın **tamamını** baştan sona okur; henüz hiçbir satır çalışmadı.

Adım 1 / 7

Derleyici söz dizimi hatalarını program çalışmadan **önce** yakalar; yorumlayıcı hemen başlar; hatayı (basit yorumlayıcılarda) o satıra gelince görür. Python ve JavaScript gibi modern yorumlayıcılar önce tüm dosyayı ayrıştırır ve söz dizimi hatasını en başta bildirir; Python ayrıca bytecode'a derler.

ÇALIŞTIRMA · ADIM ADIM

Derleyici ve Yorumlayıcı Adım Adım

Derleyici: önce tümü çevrilir, sonra çalışır

```
1  yaz "Merhaba"      ✓
2  x ← 10             ✓
3  yaz x / 0          ✓
```

konsol

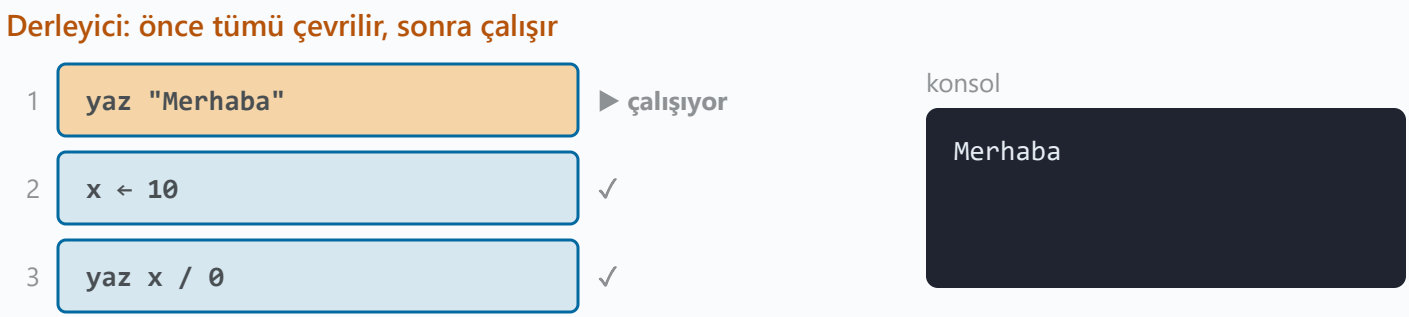


Satır 3'teki 0'a bölme söz dizimi hatası değildir; çeviri başarılı olur ve çalıştırılabilir dosya üretilir.

Derleyici söz dizimi hatalarını program çalışmadan **önce** yakalar; yorumlayıcı hemen başlar; hatayı (basit yorumlayıcılarda) o satıra gelince görür. Python ve JavaScript gibi modern yorumlayıcılar önce tüm dosyayı ayrıştırır ve söz dizimi hatasını en başta bildirir; Python ayrıca bytecode'a derler.

ÇALIŞTIRMA · ADIM ADIM

Derleyici ve Yorumlayıcı Adım Adım



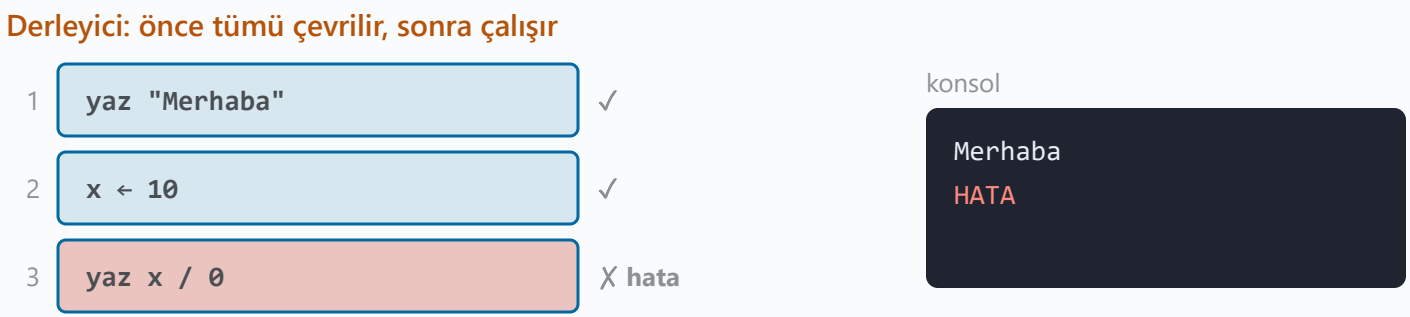
Şimdi çalıştırma başlar: satır 1 → çıktı **Merhaba**.

Adım 3 / 7

Derleyici söz dizimi hatalarını program çalışmadan **önce** yakalar; yorumlayıcı hemen başlar; hatayı (basit yorumlayıcılarda) o satıra gelince görür. Python ve JavaScript gibi modern yorumlayıcılar önce tüm dosyayı ayrıştırır ve söz dizimi hatasını en başta bildirir; Python ayrıca bytecode'a derler.

ÇALIŞTIRMA · ADIM ADIM

Derleyici ve Yorumlayıcı Adım Adım



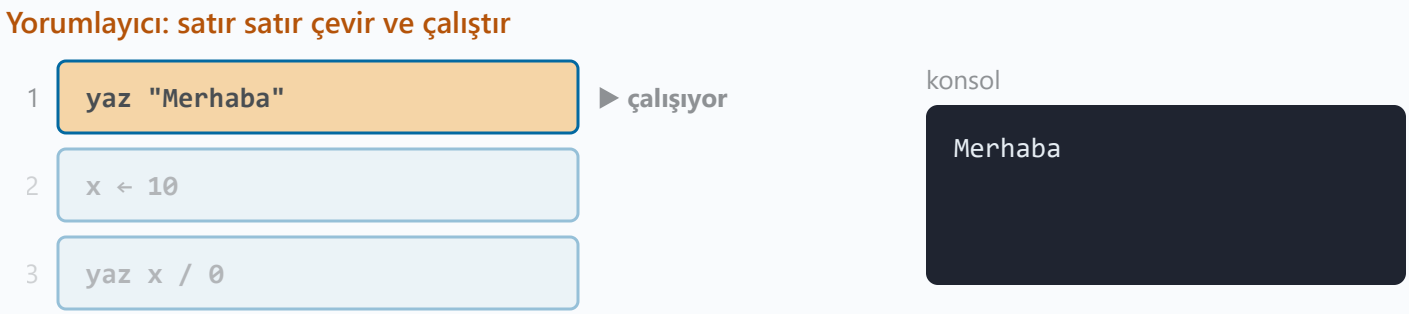
Satır 3 çalışırken **çalışma zamanı hatası** (0'a bölme). Derleyicinin ayıklayamadığı hata türü.

Adım 4 / 7

Derleyici söz dizimi hatalarını program çalışmadan **önce** yakalar; yorumlayıcı hemen başlar; hatayı (basit yorumlayıcılarda) o satıra gelince görür. Python ve JavaScript gibi modern yorumlayıcılar önce tüm dosyayı ayrıştırır ve söz dizimi hatasını en başta bildirir; Python ayrıca bytecode'a derler.

ÇALIŞTIRMA · ADIM ADIM

Derleyici ve Yorumlayıcı Adım Adım



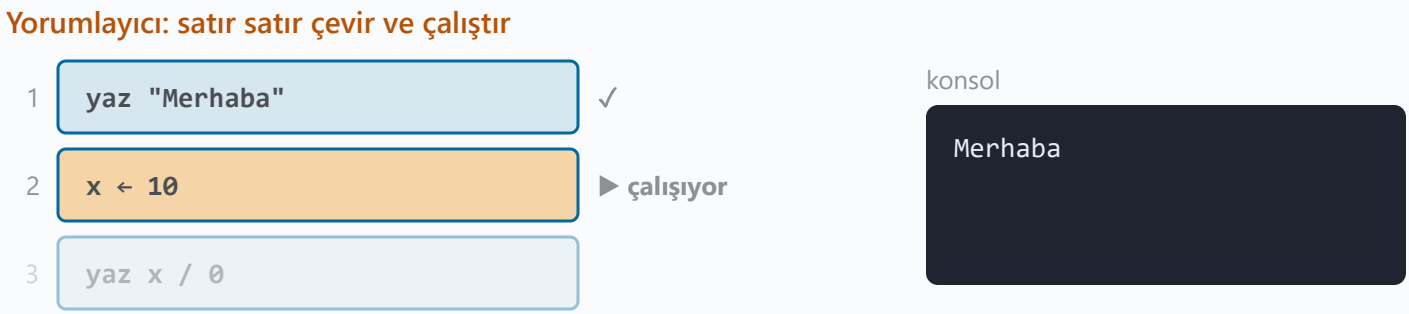
Yorumlayıcı ise **çevir-çalıştır** döngüsüyle ilerler: satır 1 çevrilir ve hemen çalışır → Merhaba.

Adım 5 / 7

Derleyici söz dizimi hatalarını program çalışmadan **önce** yakalar; yorumlayıcı hemen başlar; hatayı (basit yorumlayıcılarda) o satıra gelince görür. Python ve JavaScript gibi modern yorumlayıcılar önce tüm dosyayı ayrıştırır ve söz dizimi hatasını en başta bildirir; Python ayrıca bytecode'a derler.

ÇALIŞTIRMA · ADIM ADIM

Derleyici ve Yorumlayıcı Adım Adım



Satır 2 çevrilip çalışır: **x = 10**. Satır 3 henüz hiç okunmadı.

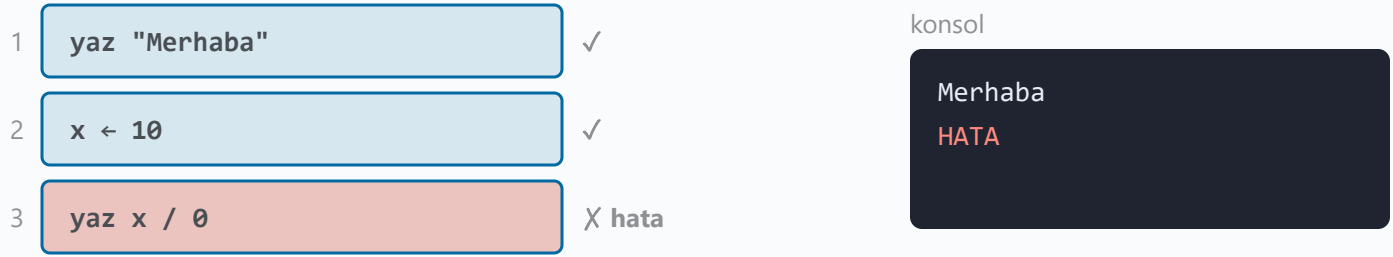
Adım 6 / 7

Derleyici söz dizimi hatalarını program çalışmadan **önce** yakalar; yorumlayıcı hemen başlar; hatayı (basit yorumlayıcılarda) o satıra gelince görür. Python ve JavaScript gibi modern yorumlayıcılar önce tüm dosyayı ayrıştırır ve söz dizimi hatasını en başta bildirir; Python ayrıca bytecode'a derler.

ÇALIŞTIRMA · ADIM ADIM

Derleyici ve Yorumlayıcı Adım Adım

Yorumlayıcı: satır satır çevir ve çalıştır



Satır 3'e gelince hata. Önceki satırlar **çoktan çalışmıştı**. Basit, satır satır çalışan bir yorumlayıcıda bu hata da ancak o satıra gelince fark edilirdi.

Adım 7 / 7

Derleyici söz dizimi hatalarını program çalışmadan **önce** yakalar; yorumlayıcı hemen başlar; hatayı (basit yorumlayıcılarda) o satıra gelince görür. Python ve JavaScript gibi modern yorumlayıcılar önce tüm dosyayı ayrıştırır ve söz dizimi hatasını en başta bildirir; Python ayrıca bytecode'a derler.

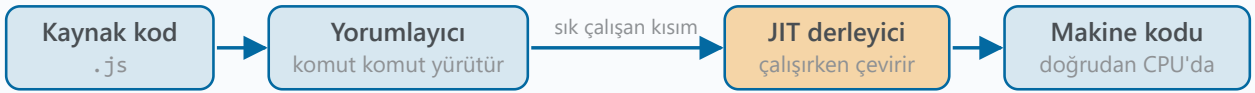
ÇALIŞTIRMA

Python Neden “Yorumlanan” Denir?

Kaynak kod (.py) → **Bytecode** **(derleme)** → **Sanal makine** **(yürütme)**

- **CPython** önce kaynak kodu **bytecode**’a derler, sonra bunu bir sanal makinede komut komut yürütür; makine kodu dosyası üretilmez.
- “Yorumlanan” etiketi bu yürütme biçiminden gelir; derleme adımı programcıdan gizlidir.
- Modern **JavaScript** motorları (ör. V8) **JIT** kullanır: sık çalışan kodu çalışma sırasında makine koduna çevirir.

JavaScript motoru (ör. V8): önce yorumlar, sık çalışan kısmı JIT ile makine koduna çevirir

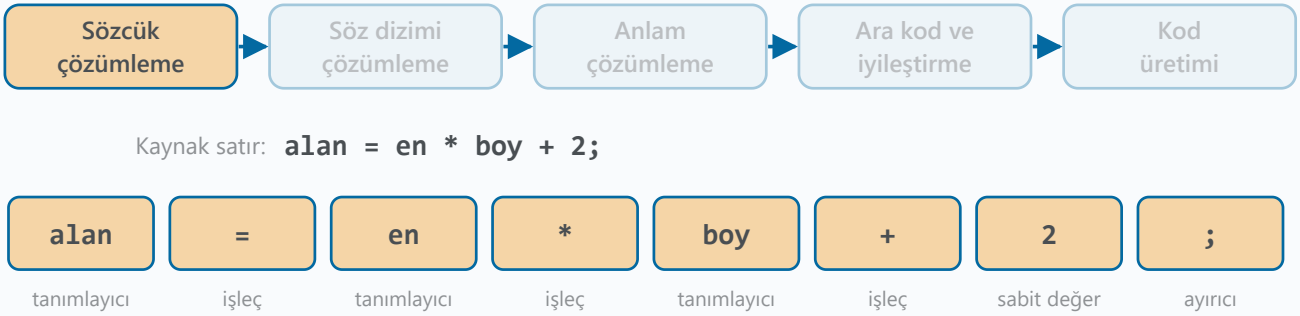


Sonraki çağrılarda yorumlama atlanır; program çalıştıkça hızlanır.

Derlenen / yorumlanan ayrımı dilin değil, **uygulamanın** özelliğidir; pratikte çoğu sistem iki yöntemi birleştirir.

ÇALIŞTIRMA · ADIM ADIM

Derleyicinin Aşamaları



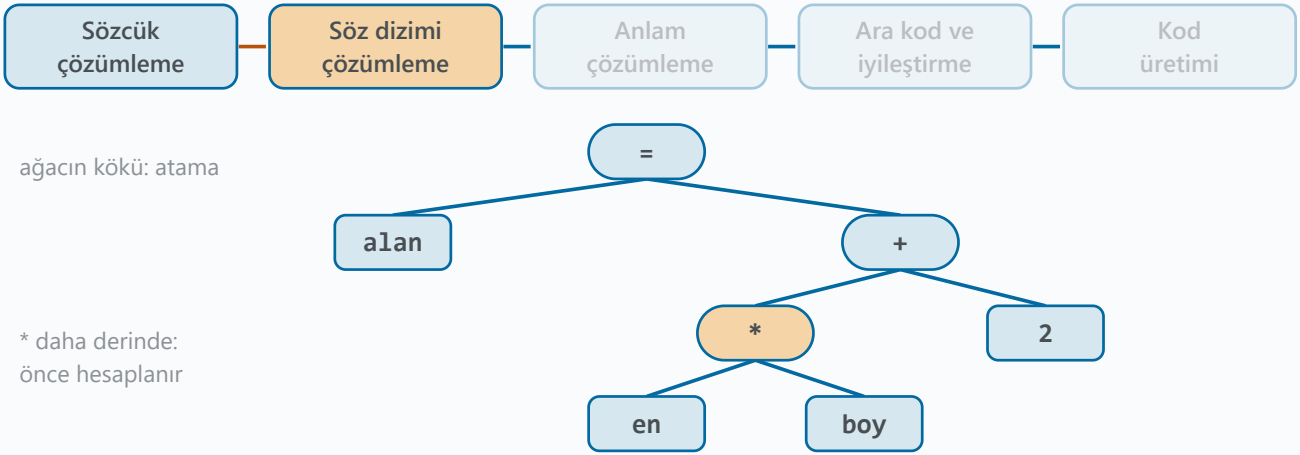
Sözcük çözümleme: karakter dizisi anlamlı parçalara, **belirteçlere** ayrılır; boşluklar ve açıklamalar atılır. `#` gibi dilde olmayan bir karakter ya da kapanmamış bir dizgi burada fark edilir.

Adım 1 / 5

İlk üç aşama hata arar ve hatayı satır numarasıyla bildirir; son iki aşama çalışacak kodu üretir. 0'a bölme gibi değere bağlı hatalar hiçbir aşamada görünmez.

ÇALIŞTIRMA · ADIM ADIM

Derleyicinin Aşamaları



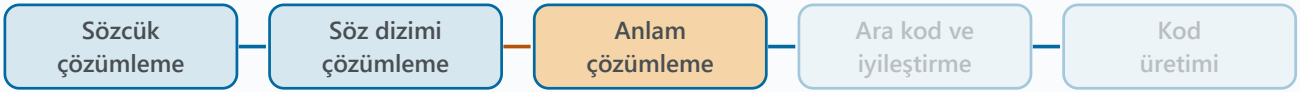
Söz dizimi çözümleme: belirteçler dilin kurallarına göre bir **söz dizimi ağacına** dizilir. Eksik `;` veya kapanmamış parantez bu aşamada yakalanır.

Adım 2 / 5

İlk üç aşama hata arar ve hatayı satır numarasıyla bildirir; son iki aşama çalışacak kodu üretir. 0'a bölme gibi değere bağlı hatalar hiçbir aşamada görünmez.

ÇALIŞTIRMA · ADIM ADIM

Derleyicinin Aşamaları



`en` → tanımlı · tip int ✓

`boy` → tanımlı · tip int ✓

`en * boy + 2` → int × int + int → int ✓

`alan ← int` → alan int: tipler uyumlu ✓

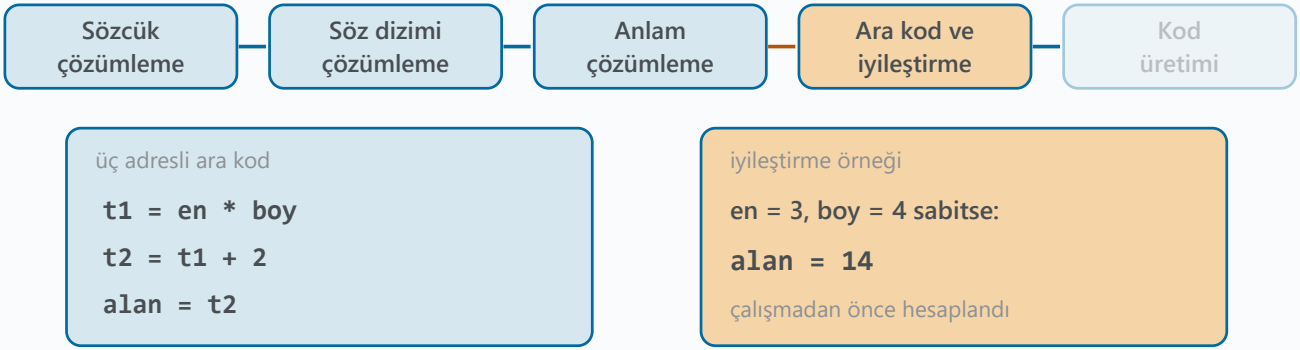
boy hiç bildirilmeseydi → error: cannot find symbol

Anlam çözümleme: adların bildirilip bildirilmediği ve tiplerin uyumu denetlenir. Tanımsız değişken ve tip uyuşmazlığı burada bulunur.

İlk üç aşama hata arar ve hatayı satır numarasıyla bildirir; son iki aşama çalışacak kodu üretir. 0'a bölme gibi değere bağlı hatalar hiçbir aşamada görünmez.

ÇALIŞTIRMA · ADIM ADIM

Derleyicinin Aşamaları



sabit katlama: sonucu değişmeyen işlemler derleme sırasında yapılır

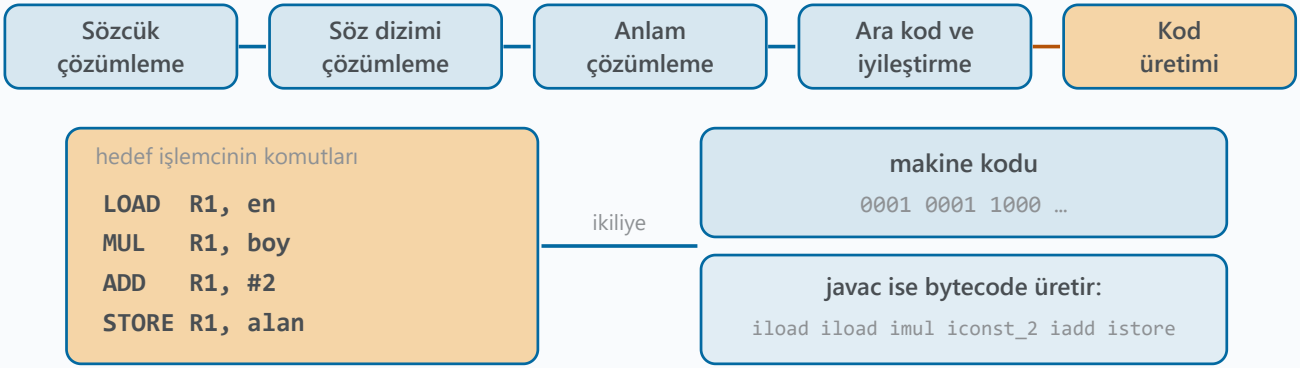
Ara kod ve iyileştirme: ifade, her biri tek işlem yapan basit komutlara bölünür; gereksiz işlemler atılır, sabit ifadeler önceden hesaplanır.

Adım 4 / 5

İlk üç aşama hata arar ve hatayı satır numarasıyla bildirir; son iki aşama çalışacak kodu üretir. 0'a bölme gibi değere bağlı hatalar hiçbir aşamada görünmez.

ÇALIŞTIRMA · ADIM ADIM

Derleyicinin Aşamaları



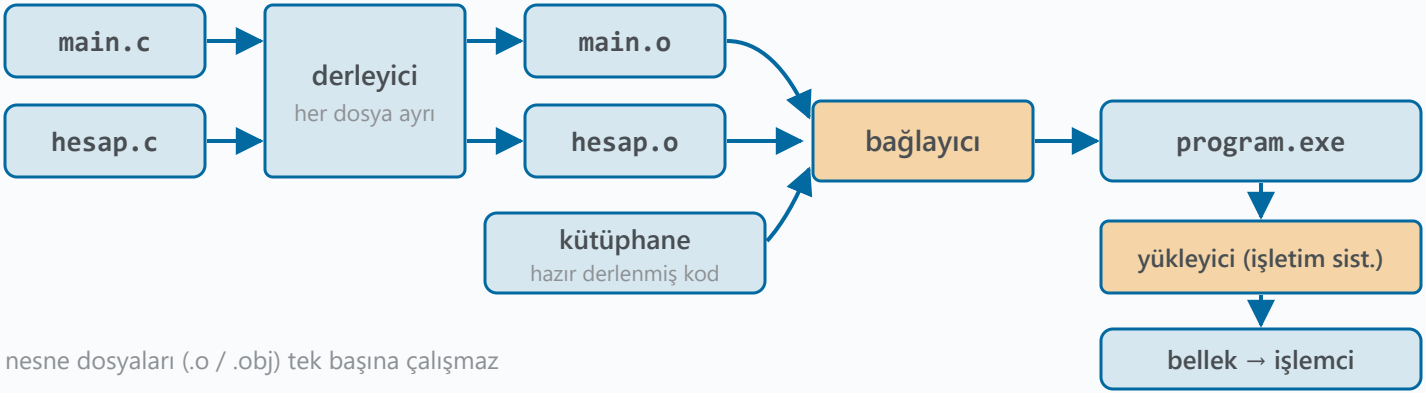
Kod üretimi: hedef işlemcinin komutları seçilir ve ikiliye çevrilir. Java derleyicisinin hedefi gerçek bir işlemci değil, JVM'dir.

Adım 5 / 5

İlk üç aşama hata arar ve hatayı satır numarasıyla bildirir; son iki aşama çalışacak kodu üretir. 0'a bölme gibi değere bağlı hatalar hiçbir aşamada görünmez.

ÇALIŞTIRMA

Bağlayıcı ve Yükleyici



nesne dosyaları (.o / .obj) tek başına çalışmaz

- Büyük programlar birçok kaynak dosyadan oluşur; derleyici her birini ayrı bir **nesne dosyasına** çevirir.
- **Bağlayıcı** (linker) nesne dosyalarını ve kullanılan kütüphane kodunu tek bir çalıştırılabilir dosyada birleştirir. Çağrılan bir fonksiyon hiçbir parçada yoksa **bağlama hatası** verir.
- **Yükleyici** (loader) işletim sisteminin parçasıdır: dosyayı belleğe alır, paylaşılan kütüphaneleri (.dll, .so) bağlar ve ilk komuttan çalıştırmayı başlatır.
- Java'da ayrı bir bağlama adımı yoktur: JVM'nin **sınıf yükleyicisi** .class dosyalarını gerektiği anda yükler.

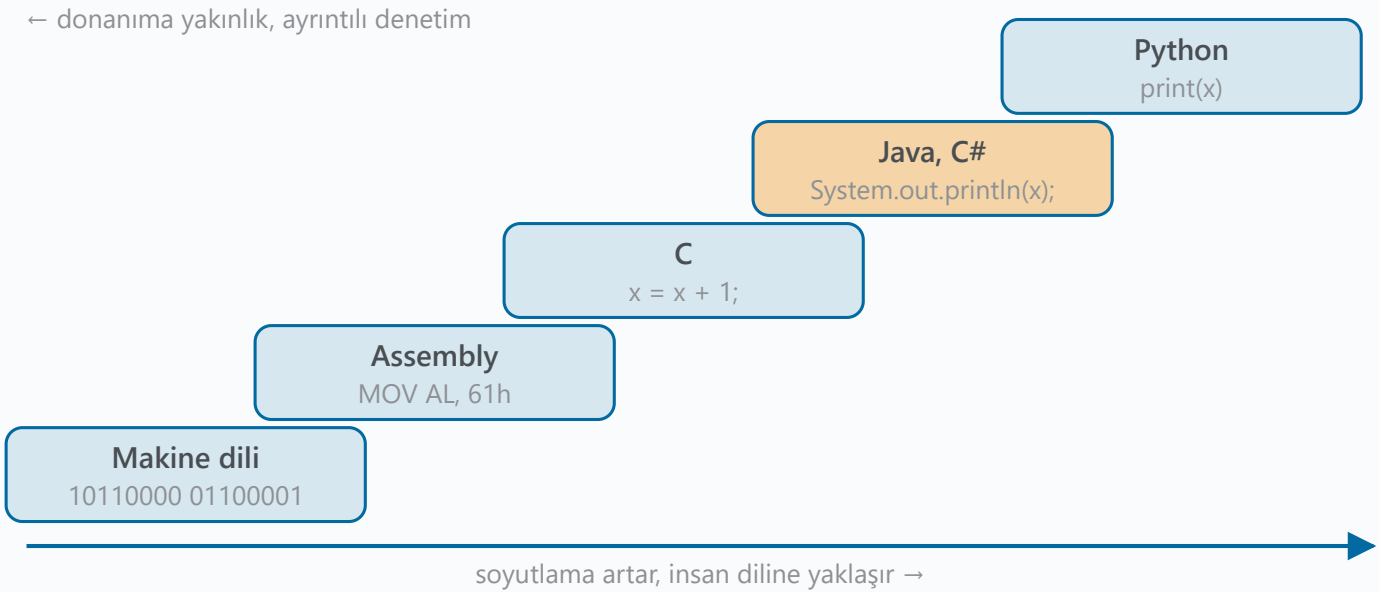
ÇALIŞTIRMA

Derleme mi Yorumlama mı?

Ölçüt	Derlenen (C)	Yorumlanan (Python)	Hibrit (Java)
Çalışma hızı	Genellikle en hızlı: doğrudan makine kodu	Daha yavaş: komutlar çalışırken çözülür	JIT sayesinde derlenene yakın
Hata bildirimi	Söz dizimi ve tip hataları çalıştırmadan önce	Tip hataları çoğunlukla ancak o satır çalışınca	Çoğu derlemede, kalanı çalışırken
Dağıtım	Her işletim sistemi ve işlemci için ayrı dosya	Kaynak kod + hedefte kurulu yorumlayıcı	Tek .class / .jar + her platformda JVM
Geliştirme döngüsü	Her değişiklikten sonra yeniden derleme	Kaydet ve hemen çalıştır	Derle, sonra çalıştır
Başlangıç	Hemen başlar, ek yazılım gerekmez	Önce yorumlayıcı yüklenir	Önce JVM başlatılır

Seçim bir değiş tokuştur: hız ve erken hata denetimi bir yanda, hızlı deneme ve esneklik öbür yanda. Bu yüzden aynı dil için hem derleyici hem yorumlayıcı yazılabilir.

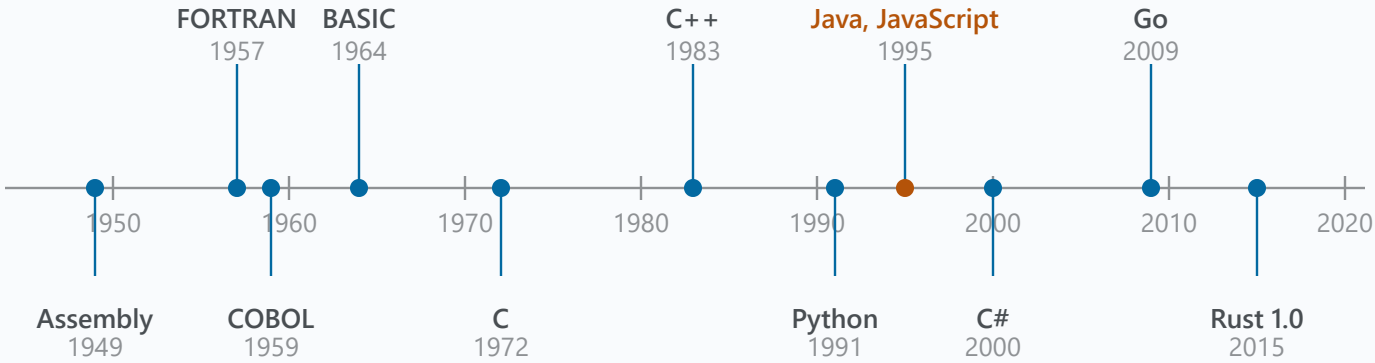
Programlama Dillerinin Düzeyi



- **Düşük düzeyli** diller donanıma yakındır; **yüksek düzeyli** diller problemi insanın ifade biçimine yakın yazdırır.
- Düzey, bir dilin iyi veya kötü olduğunu değil, **soyutlama düzeyini** anlatır.

DİLLER

Programlama Dillerinin Gelişimi



1940–1970

Makine dili, assembly; ilk yüksek düzeyli diller FORTRAN ve COBOL.

1970–1990

C ile yapısal ve taşınabilir sistem programlama; C++ ile nesne yönelimli yaklaşım.

1990 sonrası

Python, Java, JavaScript, C#: web, yönetilen ortamlar, açık kaynak ekosistemleri.

Kullanım Alanları

Web
İstemci, sunucu ve web servisleri.

Mobil
Telefon ve tablet uygulamaları.

Veri ve yapay zekâ
Analiz, modelleme ve otomasyon.

Sistem
İşletim sistemi, sürücü ve gömülü yazılım.

Bilimsel hesaplama
Simülasyon ve mühendislik çözümleri.

Oyun
Oyun motoru, grafik ve etkileşim.

DİLLER

Dil Türleri

Genel amaçlı

Her tür problem için değişken, karar, döngü ve fonksiyon sunar: Java, C, Python.

```
for (int i = 0; i < 3; i++)
```

Alana özgü

Tek bir işe odaklanır: SQL veri tabanını sorgular, Excel formülü hücrede hesap yapar.

```
SELECT ad FROM ogrenci WHERE ort >= 60;
```

Betik dilleri

Tekrarlanan işleri otomatikleştirir, genellikle yorumlanır: Python, JavaScript, Bash, PowerShell.

```
Copy-Item *.txt yedek\
```

İşaretleme dilleri

HTML, XML ve Markdown içeriğin yapısını tanımlar; karar ve döngü olmadığından programlama dili sayılmaz.

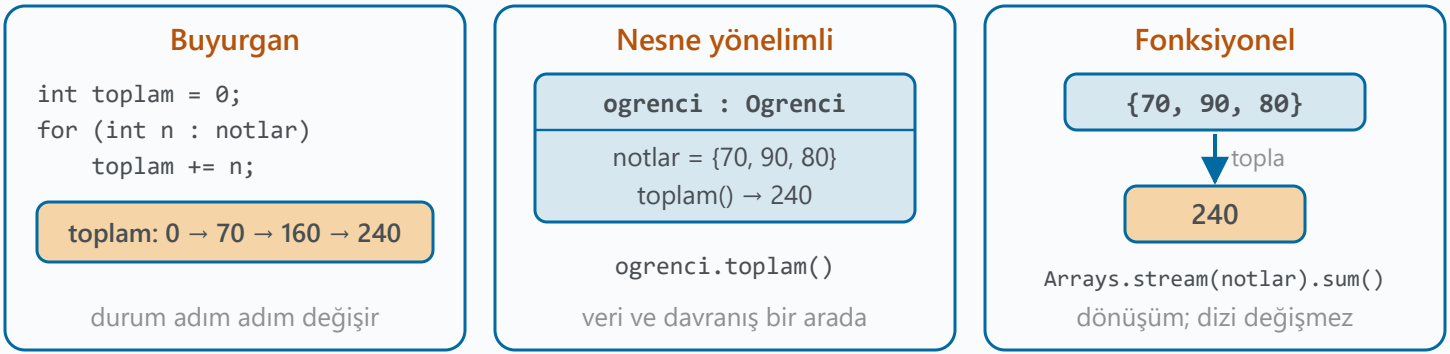
```
<h1>Başlık</h1>
```

Scratch gibi blok tabanlı görsel diller komutları sürüklenen bloklarla kurar; söz dizimi hatası yapılamadığından programlamayı öğretmek için kullanılır.

Programlama Paradigmaları

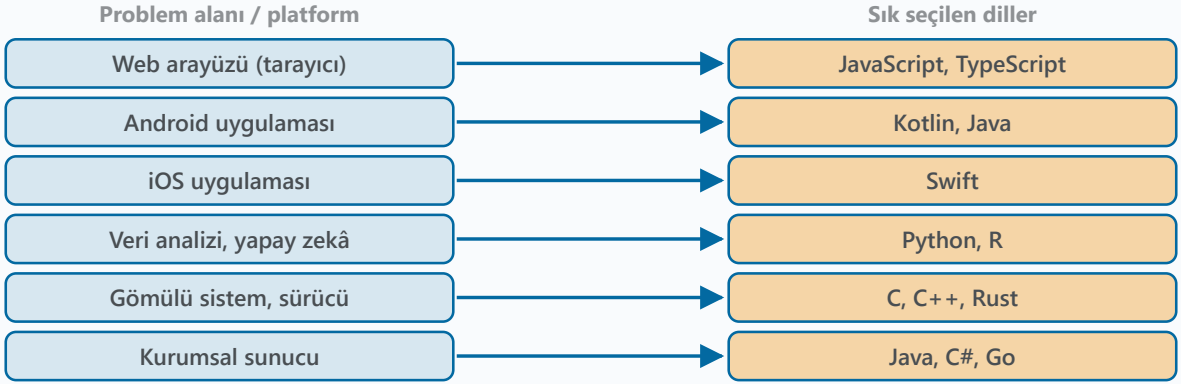
- Buyurgan yaklaşım (imperative) işlemlerin nasıl yapılacağını adım adım belirtir.
- Nesne yönelimli yaklaşım veri ve davranışı nesnelerde birleştirir.
- Fonksiyonel yaklaşım dönüşümleri ve değiştirilemez veriyi öne çıkarır.
- Java birden fazla yaklaşımı birlikte kullanabilir.

Aynı görev, üç yaklaşım: notlar = {70, 90, 80} dizisinin toplamı (240)



Dil Seçerken

- Problem alanı ve kullanılacak platform değerlendirilir.
- Kütüphane ekosistemi, topluluk ve bakım maliyeti dikkate alınır.
- Performans, güvenlik, ekip deneyimi ve dağıtım koşulları birlikte düşünülür.



Son kararı kütüphane ekosistemi, performans, güvenlik, ekip deneyimi ve bakım maliyeti verir.

Popülerlik tek başına doğru dil seçimi için yeterli değildir.

DİLLER

Statik ve Dinamik Tipleme

Java: statik

```java
int x = 5;
x = "beş";   // derlenmez
```

```
error: incompatible types:
String cannot be converted to int
```

Python: dinamik

```python
x = 5
x = "beş"    # geçerli
print("a" + 1)
```

```
TypeError: can only concatenate
str (not "int") to str
```

- **Statik tiplemede** değişkenin tipi kaynakta bildirilir ve **derleme sırasında** denetlenir: Java, C, C#. Hata program hiç çalışmadan bulunur.
- **Dinamik tiplemede** tip değerle birlikte taşınır ve **çalışırken** denetlenir: Python, JavaScript. Aynı ad önce sayı, sonra metin tutabilir; Python'daki hata ancak o satır çalışınca çıkar.
- Bazı diller tipleri sessizce dönüştürür: JavaScript'te `"5" * 2` → 10, ama `"5" + 2` → "52".

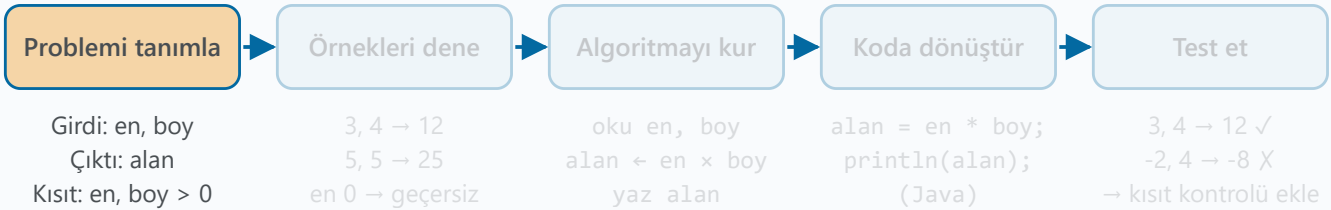
Sözde Koddan Dillere

Sözde kod	Java	Python
`oku x`	`x = giris.nextInt();`	`x = int(input())`
`yaz "Toplam: ", t`	`System.out.println("Toplam: " + t);`	`print("Toplam:", t)`
`x ← x + 1`	`x = x + 1;`	`x = x + 1`
`eğer x > 0 ise … değilse …`	`if (x > 0) { … } else { … }`	`if x > 0: … else: …`
`her i ← 1 … n için`	`for (int i = 1; i <= n; i++)`	`for i in range(1, n + 1):`
`koşul doğru iken tekrarla`	`while (kosul) { … }`	`while kosul:`

- Java blokları süslü parantezle, Python ise **girintiyle** belirler; Python'da yanlış girinti programın anlamını değiştirir.
- Java her deyimi `;` ile bitirir ve değişkenin tipini bildirmeyi ister; Python ikisini de istemez.
- Sözde kodda atama için ← kullanılır; böylece eşitlik karşılaştırmasıyla karışmaz. Java'daki `giris` klavyeden okuyan bir `Scanner` nesnesidir.

PROBLEM ÇÖZME · ADIM ADIM

Problemden Programa



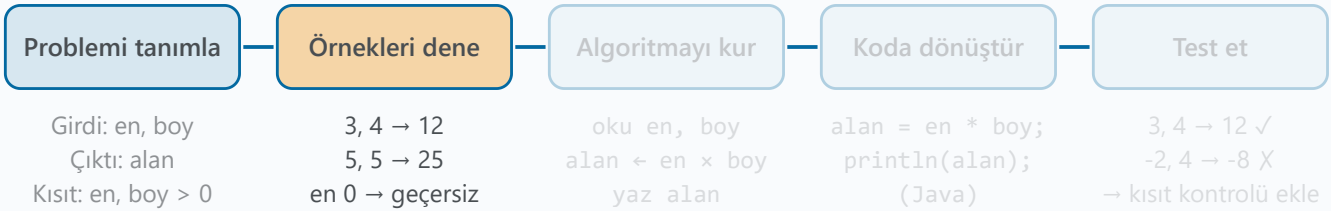
Önce **ne** istendiği netleşir: dikdörtgenin alanı. Girdi, çıktı ve kısıt yazılır.

Adım 1 / 5

- Her aşamanın çıktısı bir sonraki aşamanın girdisidir.
- Yanlış tanımlanan problem, doğru çalışan ama yanlış işi yapan programa dönüşebilir.

PROBLEM ÇÖZME · ADIM ADIM

Problemden Programa



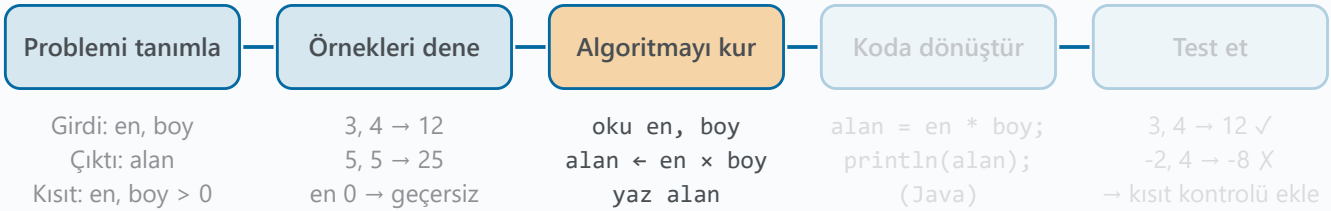
Elle birkaç **örnek** çözülür; sınır durumu (0) da düşünülür.

Adım 2 / 5

- Her aşamanın çıktısı bir sonraki aşamanın girdisidir.
- Yanlış tanımlanan problem, doğru çalışan ama yanlış işi yapan programa dönüşebilir.

PROBLEM ÇÖZME · ADIM ADIM

Problemden Programa



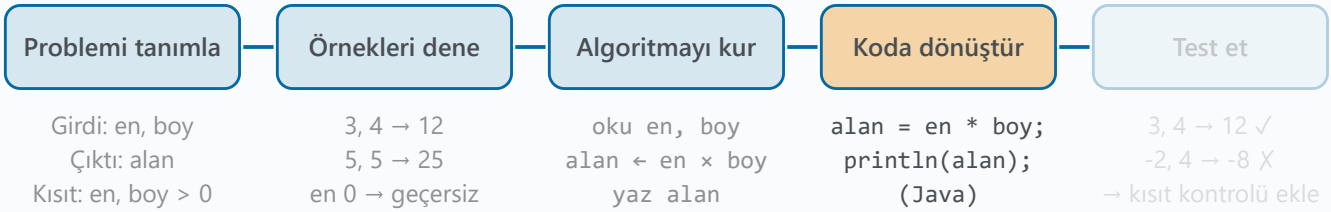
Dilden bağımsız **algoritma** (sözde kod) yazılır.

Adım 3 / 5

- Her aşamanın çıktısı bir sonraki aşamanın girdisidir.
- Yanlış tanımlanan problem, doğru çalışan ama yanlış işi yapan programa dönüşebilir.

PROBLEM ÇÖZME · ADIM ADIM

Problemden Programa



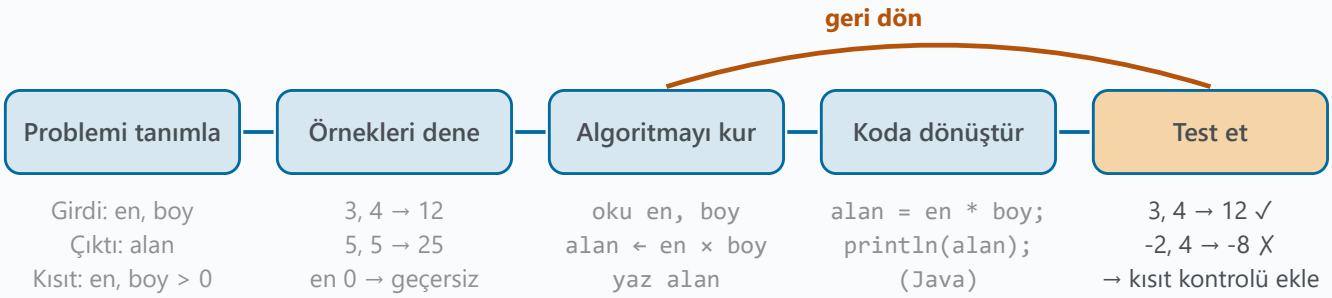
Algoritma seçilen dile (Java) **çevrilir**.

Adım 4 / 5

- Her aşamanın çıktısı bir sonraki aşamanın girdisidir.
- Yanlış tanımlanan problem, doğru çalışan ama yanlış işi yapan programa dönüşebilir.

PROBLEM ÇÖZME · ADIM ADIM

Problemden Programa



Test: negatif girdi yanlış sonuç verdi → turuncu ok: algoritmaya dönülür, "en, boy > 0 değilse hata" adımı eklenir.

Adım 5 / 5

- Her aşamanın çıktısı bir sonraki aşamanın girdisidir.
- Yanlış tanımlanan problem, doğru çalışan ama yanlış işi yapan programa dönüşebilir.

PROBLEM ÇÖZME · SÖZDE KODU ADIM ADIM ÇALIŞTIR

Algoritmayı İzleme: Dikdörtgen Alanı



1. çalıştırma: **en = 3** okunur; bellekte "en" adlı kutuya yazılır.

Adım 1 / 9

```
▶ oku en
2 oku boy
3 eğer en > 0 ve boy > 0 ise
4     alan ← en × boy
5     yaz "Alan: ", alan
6 değilse
7     yaz "Geçersiz ölçü"
```

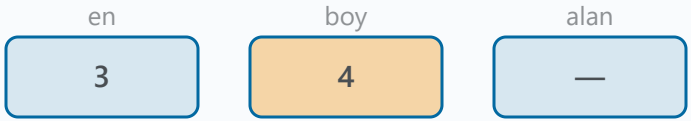
DEĞİŞKENLER

en = 3

ÇIKTI

PROBLEM ÇÖZME · SÖZDE KODU ADIM ADIM ÇALIŞTIR

Algoritmayı İzleme: Dikdörtgen Alanı



boy = 4 okunur.

Adım 2 / 9

```
1 oku en
▶ oku boy
3 eğer en > 0 ve boy > 0 ise
4     alan ← en × boy
5     yaz "Alan: ", alan
6 değilse
7     yaz "Geçersiz ölçü"
```

DEĞİŞKENLER

en = 3 boy = 4

ÇIKTI

PROBLEM ÇÖZME · SÖZDE KODU ADIM ADIM ÇALIŞTIR

Algoritmayı İzleme: Dikdörtgen Alanı



Koşul: **3 > 0 ve 4 > 0 → doğru**.

Adım 3 / 9

```
1 oku en
2 oku boy
▶ eğer en > 0 ve boy > 0 ise
4     alan ← en × boy
5     yaz "Alan: ", alan
6 değilse
7     yaz "Geçersiz ölçü"
```

DEĞİŞKENLER

`en = 3` `boy = 4`

ÇIKTI

PROBLEM ÇÖZME · SÖZDE KODU ADIM ADIM ÇALIŞTIR

Algoritmayı İzleme: Dikdörtgen Alanı



alan = 3 × 4 = 12.

Adım 4 / 9

```
1 oku en
2 oku boy
3 eğer en > 0 ve boy > 0 ise
▶       alan ← en × boy
5       yaz "Alan: ", alan
6 değilse
7       yaz "Geçersiz ölçü"
```

DEĞİŞKENLER

en = 3 | boy = 4 | alan = 12

ÇIKTI

Algoritmayı İzleme: Dikdörtgen Alanı

en	boy	alan
3	4	12

Sonuç yazılır. "değilse" dalı bu çalıştırmada atlanır.

Adım 5 / 9

```
1 oku en
2 oku boy
3 eğer en > 0 ve boy > 0 ise
4     alan ← en × boy
▶     yaz "Alan: ", alan
6 değilse
7     yaz "Geçersiz ölçü"
```

DEĞİŞKENLER

`en = 3` `boy = 4` `alan = 12`

ÇIKTI

```
Alan: 12
```

Algoritmayı İzleme: Dikdörtgen Alanı



2. çalıştırma: **en = -2** okunur; bellekte "en" adlı kutuya yazılır.

Adım 6 / 9

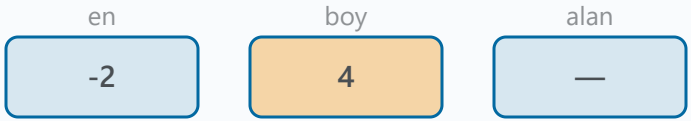
```
▶ oku en
2 oku boy
3 eğer en > 0 ve boy > 0 ise
4     alan ← en × boy
5     yaz "Alan: ", alan
6 değilse
7     yaz "Geçersiz ölçü"
```

DEĞİŞKENLER

en = −2

ÇIKTI

Algoritmayı İzleme: Dikdörtgen Alanı



boy = 4 okunur.

Adım 7 / 9

```
1 oku en
▶ oku boy
3 eğer en > 0 ve boy > 0 ise
4     alan ← en × boy
5     yaz "Alan: ", alan
6 değilse
7     yaz "Geçersiz ölçü"
```

DEĞİŞKENLER

en = −2 | boy = 4

ÇIKTI

PROBLEM ÇÖZME · SÖZDE KODU ADIM ADIM ÇALIŞTIR

Algoritmayı İzleme: Dikdörtgen Alanı

en	boy	alan
-2	4	—

Koşul: **-2 > 0 ve 4 > 0** → **yanlış** (en negatif) → değilse dalına geçilir.

Adım 8 / 9

```
1 oku en
2 oku boy
▶ eğer en > 0 ve boy > 0 ise
4     alan ← en × boy
5     yaz "Alan: ", alan
6 değilse
7     yaz "Geçersiz ölçü"
```

DEĞİŞKENLER

`en = -2` `boy = 4`

ÇIKTI

Algoritmayı İzleme: Dikdörtgen Alanı

en	boy	alan
-2	4	—

Uyarı yazılır; alan hiç hesaplanmadı. Test aşamasında bulunan hata böylece düzeltildi.

Adım 9 / 9

```
1 oku en
2 oku boy
3 eğer en > 0 ve boy > 0 ise
4     alan ← en × boy
5     yaz "Alan: ", alan
6 değilse
      yaz "Geçersiz ölçü"
```

DEĞİŞKENLER

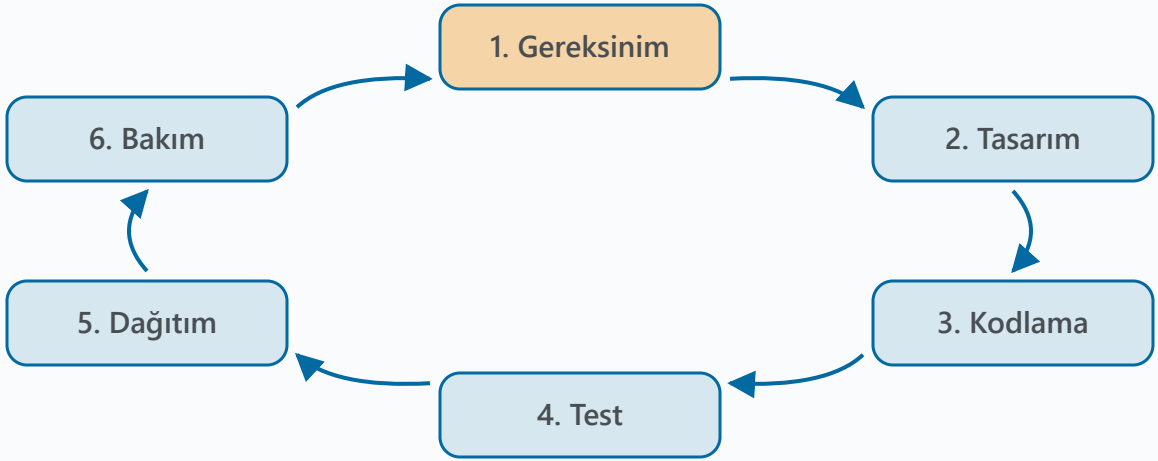
`en = -2` `boy = 4`

ÇIKTI

```
Geçersiz ölçü
```

YAZILIM · ADIM ADIM

Yazılım Geliştirme Süreci

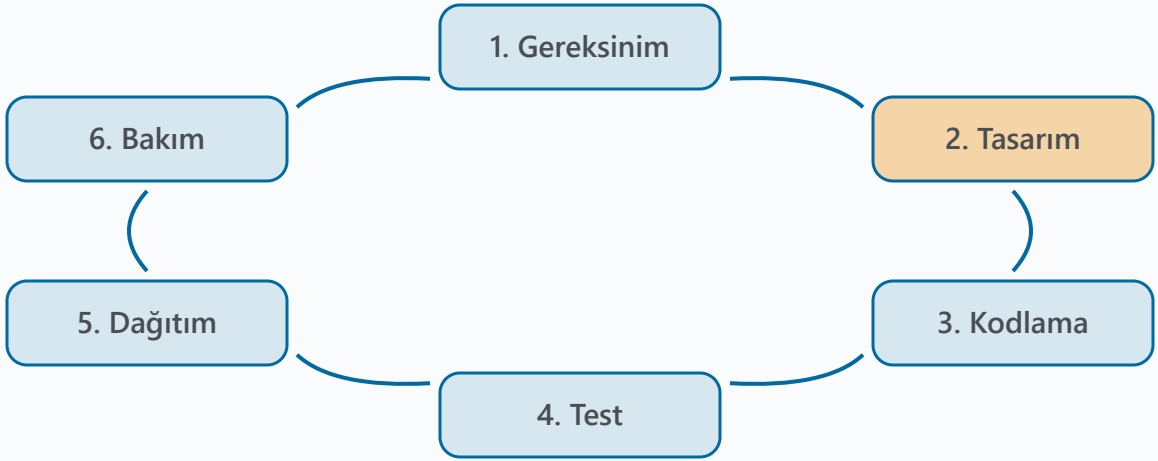


Gereksinim: kullanıcı ne istiyor? Örn. "öğrenci not ortalamasını hesaplayan uygulama".

Adım 1 / 6

- Aşamalar projeye göre ardışık veya yinelemeli yürütülebilir; kullanıcı geri bildirimi gereksinimi ve tasarımı değiştirebilir.

Yazılım Geliştirme Süreci

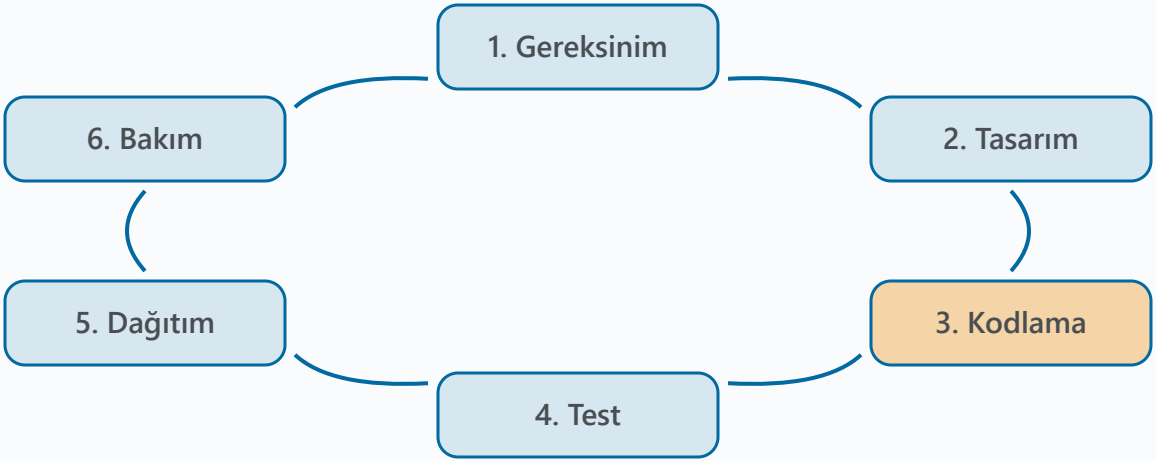


Tasarım: ekranlar, veri ve algoritmalar planlanır (akış diyagramı burada kullanılır).

- Aşamalar projeye göre ardışık veya yinelemeli yürütülebilir; kullanıcı geri bildirimi gereksinimi ve tasarımı değiştirebilir.

YAZILIM · ADIM ADIM

Yazılım Geliştirme Süreci

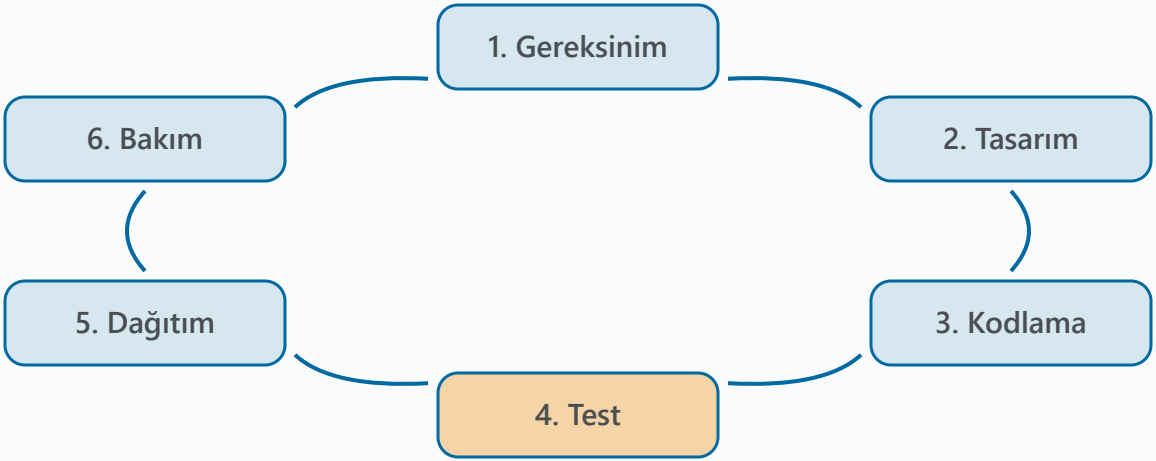


Kodlama: tasarım bir programlama diliyle yazılır.

Adım 3 / 6

- Aşamalar projeye göre ardışık veya yinelemeli yürütülebilir; kullanıcı geri bildirimi gereksinimi ve tasarımı değiştirebilir.

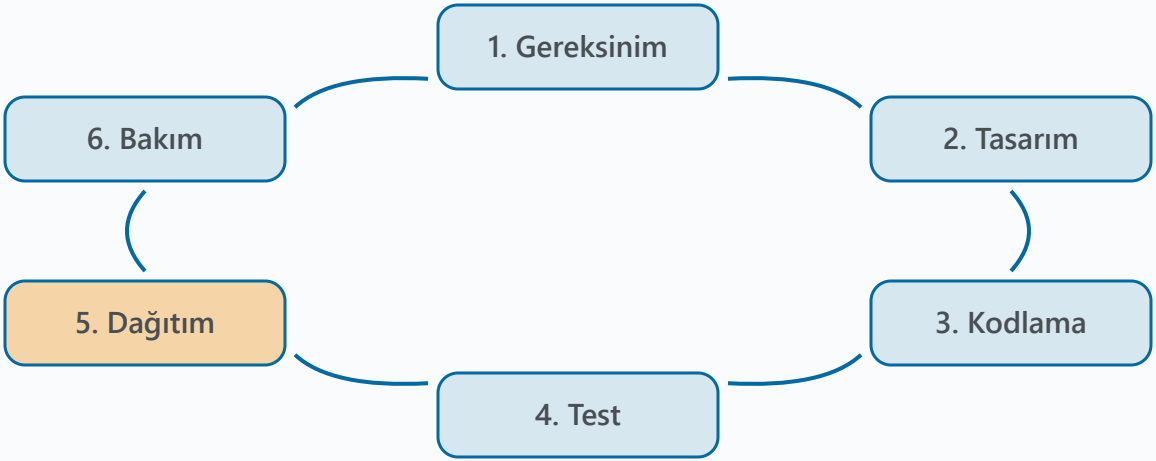
Yazılım Geliştirme Süreci



Test: beklenen ve gerçek sonuçlar karşılaştırılır; hata varsa kodlamaya geri dönülür.

- Aşamalar projeye göre ardışık veya yinelemeli yürütülebilir; kullanıcı geri bildirimi gereksinimi ve tasarımı değiştirebilir.

Yazılım Geliştirme Süreci

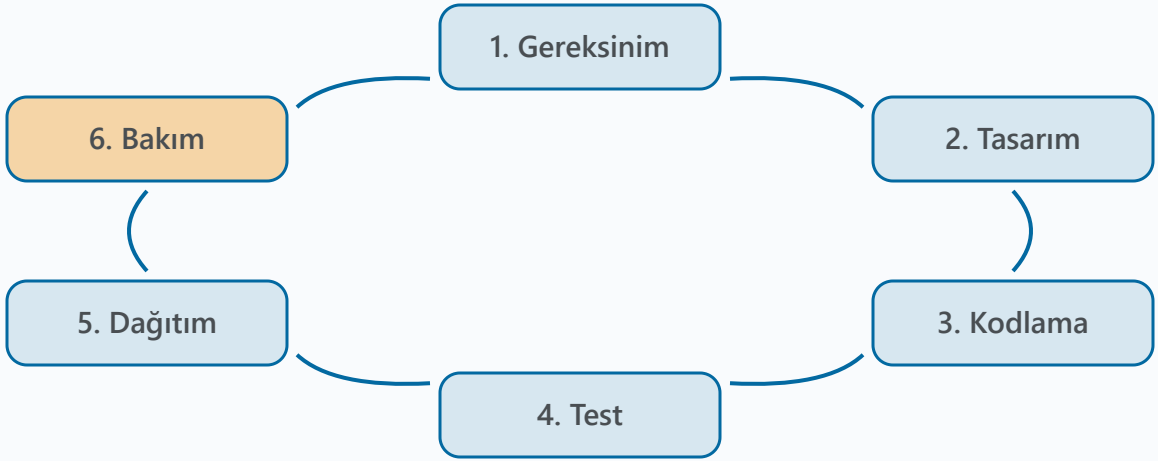


Dağıtım: program kullanıcıya ulaştırılır (kurulum, yayın).

- Aşamalar projeye göre ardışık veya yinelemeli yürütülebilir; kullanıcı geri bildirimi gereksinimi ve tasarımı değiştirebilir.

YAZILIM · ADIM ADIM

Yazılım Geliştirme Süreci



Bakım: hatalar giderilir, yeni istekler gelir → yeni gereksinimlerle döngü yeniden başlar.

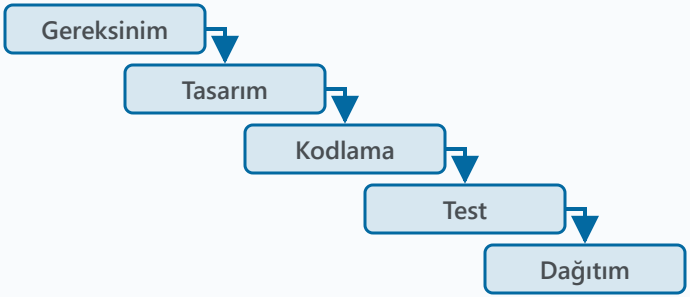
Adım 6 / 6

- Aşamalar projeye göre ardışık veya yinelemeli yürütülebilir; kullanıcı geri bildirimi gereksinimi ve tasarımı değiştirebilir.

YAZILIM

Geliştirme Yaklaşımları

Şelale



Çevik



her döngü sonunda çalışan sürüm + geri bildirim

Yaklaşım	Güçlü yön	Dikkat
Şelale modeli	Aşamalar ve belgeler nettir	Değişikliğe dönüş maliyetlidir
Çevik	Kısa döngülerle geri bildirim alır	Sürekli iletişim gerektirir
Prototipleme	Belirsiz gereksinimi görünür kılar	Prototip ürün sanılmamalıdır

YAZILIM

Geliştirme Araçları

Kod düzenleyici / IDE
Kod yazma, tamamlama ve proje yönetimi.

Derleyici / çalışma ortamı
Programı çevirme ve çalıştırma.

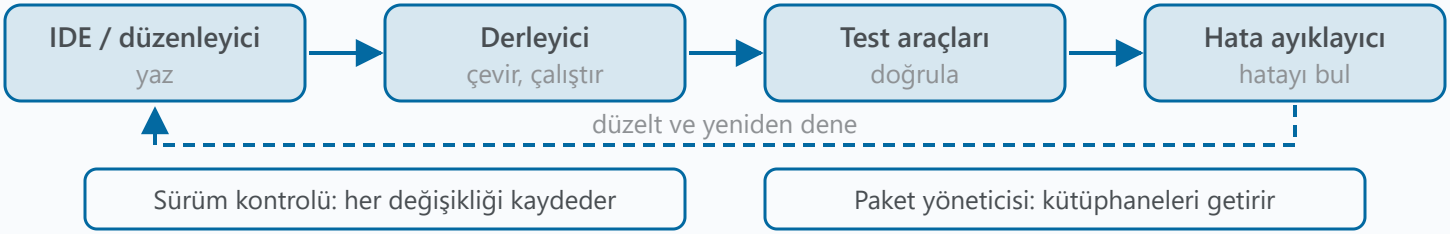
Hata ayıklayıcı
Program durumunu adım adım inceleme.

Sürüm kontrolü
Değişiklik geçmişi ve ekip çalışması.

Test araçları
Beklenen davranışı otomatik doğrulama.

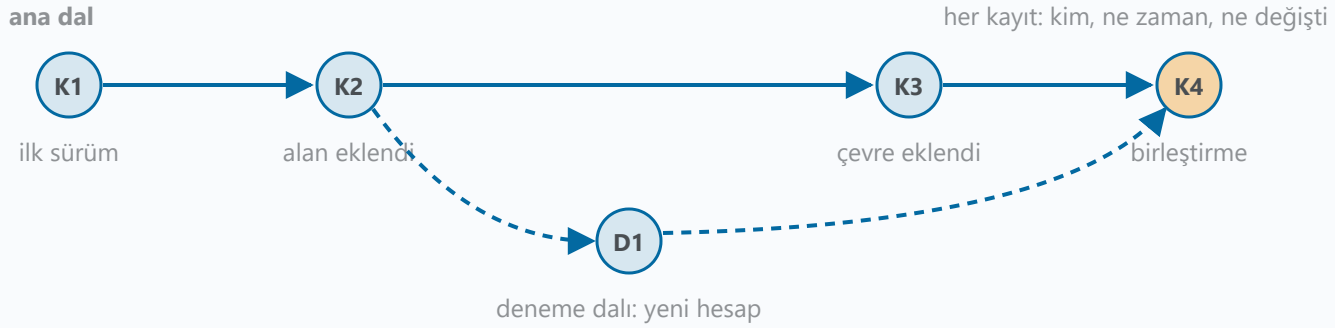
Paket yöneticisi
Bağımlılıkları yönetme.

Bir geliştirme turu: her araç döngünün farklı adımında devreye girer



YAZILIM

Sürüm Kontrolü

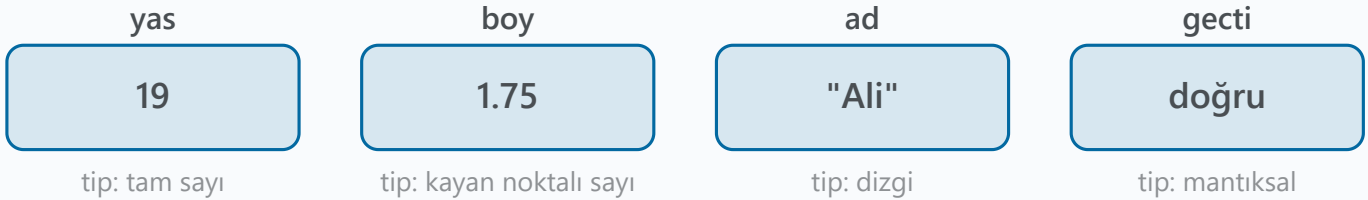


- Her **kayıt** (commit) dosyaların o anki hâlini; kimin, ne zaman, neyi değiştirdiğiyle birlikte saklar. Hatalı bir değişiklikten sonra eski bir sürüme dönülebilir.
- **Dal** (branch) ana kodu bozmadan deneme yapmayı sağlar; iş bitince dal ana dala **birleştirilir** (merge).
- Ekip üyeleri aynı dosyalar üzerinde çalışır; aynı satırı iki kişi değiştirirse araç bu **çakışmayı** gösterir ve birinin seçmesini ister.
- En yaygın araç **Git**'tir; GitHub ve GitLab depoları ağ üzerinde saklayan hizmetlerdir.

TEMEL YAPILAR

Değişkenler ve Veri Tipleri

- Değişken, bellekte bir değeri tutan **adlandırılmış kutudur**; program boyunca değeri değişebilir.
- Veri tipi değerin temsilini ve üzerinde yapılabilecek işlemleri sınırlar.
- Anlamlı adlar programın amacını görünür kılar.

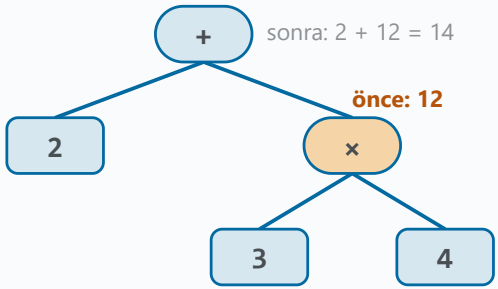


Java'daki karşılıkları: int, double, String, boolean.

İşleç Önceliği

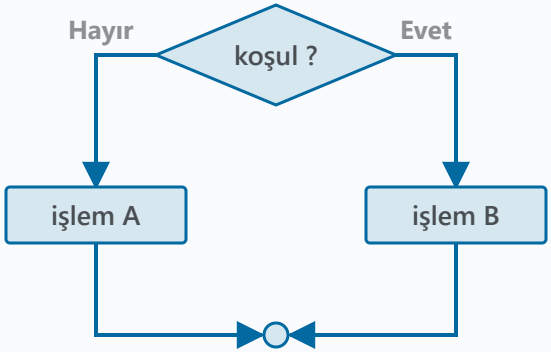
1. Parantez içi
2. Çarpma, bölme, mod (kalan)
3. Toplama, çıkarma
4. Aynı düzeydekiler soldan sağa

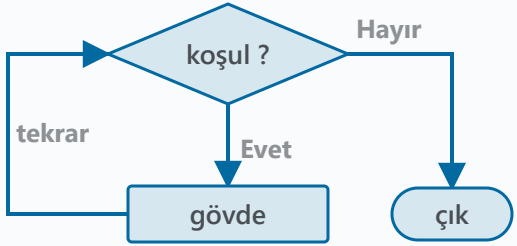
İfade	Sonuç
`2 + 3 × 4`	14
`(2 + 3) × 4`	20
`20 / 4 × 2`	10 (soldan sağa)
`10 − 4 − 3`	3
`7 mod 3 + 1`	2



Formüldeki parantez unutulursa program hata vermez, yanlış sonuç üretir: `ort ← a + b / 2` önce b'yi böler. Bu bir **mantık hatasıdır**; ancak test ederek bulunur.

Karar ve Döngü





Karar: iki yoldan yalnızca biri

Döngü: geri dönen ok

- Karar yapısı koşula göre farklı işlem yollarından birini seçer; döngü bir işlem grubunu koşul boyunca tekrarlar.
- Her ikisinde de koşulun ne zaman değiştiği açık olmalıdır (yoksa sonsuz döngü).

Döngü İzleme: Sonsuz Döngü

İlk sürüm (4. satır yok)

	koşul	gövde çalışma sayısı
i = 1	i ≤ 3	0

İlk sürüm: 4. satır unutulmuş. i ← 1 ile döngü değişkeni hazırlanır.

Adım 1 / 12

```
▶ i ← 1
2 i ≤ 3 iken tekrarla
3     yaz i
4     i ← i + 1      // ilk sürümde yoktu
5 yaz "bitti"
```

DEĞİŞKENLER

`i = 1`

ÇIKTI

Döngü İzleme: Sonsuz Döngü

İlk sürüm (4. satır yok)

	koşul	gövde çalışma sayısı
i = 1	1 ≤ 3 → doğru	0

Koşul **1 ≤ 3 → doğru**: döngü gövdesine girilir.

Adım 2 / 12

```
1 i ← 1
▶ i ≤ 3 iken tekrarla
3     yaz i
4     i ← i + 1      // ilk sürümde yoktu
5 yaz "bitti"
```

DEĞİŞKENLER

`i = 1`

ÇIKTI

Döngü İzleme: Sonsuz Döngü

İlk sürüm (4. satır yok)

	koşul	gövde çalışma sayısı
i = 1	i ≤ 3	1

yaz i → 1. Gövde bitti; koşula geri dönülür.

Adım 3 / 12

```
1 i ← 1
2 i ≤ 3 iken tekrarla
      yaz i
4     i ← i + 1      // ilk sürümde yoktu
5 yaz "bitti"
```

DEĞİŞKENLER

`i = 1`

ÇIKTI

```
1
```

Döngü İzleme: Sonsuz Döngü

İlk sürüm (4. satır yok)

	koşul	gövde çalışma sayısı
i = 1	1 ≤ 3 → doğru	1

Koşul yine **1 ≤ 3 → doğru**: gövdede i'yi değiştiren hiçbir satır yok.

Adım 4 / 12

```
1 i ← 1
▶ i ≤ 3 iken tekrarla
3     yaz i
4     i ← i + 1      // ilk sürümde yoktu
5 yaz "bitti"
```

DEĞİŞKENLER

`i = 1`

ÇIKTI

```
1
```

Döngü İzleme: Sonsuz Döngü

İlk sürüm (4. satır yok)

	koşul	gövde çalışma sayısı
i = 1	i ≤ 3	2

Aynı çıktı bir daha: 1. Durum bir önceki turla birebir aynı.

Adım 5 / 12

```
1 i ← 1
2 i ≤ 3 iken tekrarla
▶     yaz i
4     i ← i + 1      // ilk sürümde yoktu
5 yaz "bitti"
```

DEĞİŞKENLER

`i = 1`

ÇIKTI

```
1
1
```

Döngü İzleme: Sonsuz Döngü

İlk sürüm (4. satır yok)

	koşul	gövde çalışma sayısı	
i = 1	1 ≤ 3 → doğru	...	↻ ∞

Koşulu yanlış yapacak bir değişiklik olmadığı için döngü hiç bitmez: **sonsuz döngü**. Konsol programı ancak Ctrl+C ile durdurulur.

Adım 6 / 12

```
1 i ← 1
▶ i ≤ 3 iken tekrarla
3     yaz i
4     i ← i + 1      // ilk sürümde yoktu
5 yaz "bitti"
```

DEĞİŞKENLER

`i = 1`

ÇIKTI

```
1
1
...
```

Döngü İzleme: Sonsuz Döngü

Düzeltilmiş sürüm

	koşul	gövde çalışma sayısı
i = 1	i ≤ 3	0

Düzeltilmiş sürüm: gövdeye **i ← i + 1** eklendi. Yeniden başlanır: i = 1.


```
▶ i ← 1
2 i ≤ 3 iken tekrarla
3     yaz i
4     i ← i + 1      // ilk sürümde yoktu
5 yaz "bitti"
```

DEĞİŞKENLER

`i = 1`

ÇIKTI

Döngü İzleme: Sonsuz Döngü

Düzeltilmiş sürüm

	koşul	gövde çalışma sayısı
i = 2	1 ≤ 3 → doğru	1

1 ≤ 3 doğru: 1 yazılır, ardından i ← 2. Döngü değişkeni artık her turda ilerliyor.

Adım 8 / 12

```
1 i ← 1
2 i ≤ 3 iken tekrarla
▶     yaz i
▶     i ← i + 1      // ilk sürümde yoktu
5 yaz "bitti"
```

DEĞİŞKENLER

`i = 2`

ÇIKTI

```
1
```

Döngü İzleme: Sonsuz Döngü

Düzeltilmiş sürüm

i	koşul	gövde çalışma sayısı
3	2 ≤ 3 → doğru	2

2 ≤ 3 doğru: 2 yazılır, i ← 3.


```
1 i ← 1
2 i ≤ 3 iken tekrarla
▶     yaz i
▶     i ← i + 1      // ilk sürümde yoktu
5 yaz "bitti"
```

DEĞİŞKENLER

`i = 3`

ÇIKTI

```
1
2
```

Döngü İzleme: Sonsuz Döngü

Düzeltilmiş sürüm

i	koşul	gövde çalışma sayısı
4	3 ≤ 3 → doğru	3

3 ≤ 3 de doğrudur (eşitlik): 3 yazılır, i ← 4.


```
1 i ← 1
2 i ≤ 3 iken tekrarla
▶     yaz i
▶     i ← i + 1      // ilk sürümde yoktu
5 yaz "bitti"
```

DEĞİŞKENLER

`i = 4`

ÇIKTI

```
1
2
3
```

Döngü İzleme: Sonsuz Döngü

Düzeltilmiş sürüm

i	koşul	gövde çalışma sayısı
4	4 ≤ 3 → yanlış	3

Koşul **4 ≤ 3 → yanlış**: döngüden çıkılır.


```
1 i ← 1
▶ i ≤ 3 iken tekrarla
3     yaz i
4     i ← i + 1      // ilk sürümde yoktu
5 yaz "bitti"
```

DEĞİŞKENLER

`i = 4`

ÇIKTI

```
1
2
3
```

Döngü İzleme: Sonsuz Döngü

Düzeltilmiş sürüm

i	koşul	gövde çalışma sayısı
4	i ≤ 3	3

Döngüden sonraki satır: **bitti**. Gövde 3 kez çalıştı, koşul 4 kez denetlendi.

Adım 12 / 12

```
1 i ← 1
2 i ≤ 3 iken tekrarla
3     yaz i
4     i ← i + 1      // ilk sürümde yoktu
▶ yaz "bitti"
```

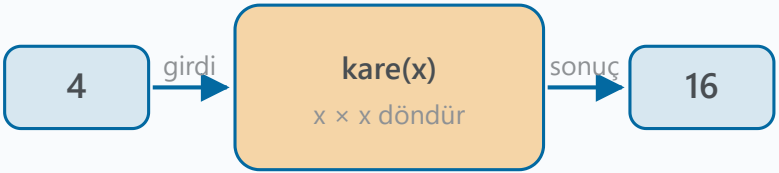
DEĞİŞKENLER

`i = 4`

ÇIKTI

```
1
2
3
bitti
```

Fonksiyonlar ve Diziler





- Fonksiyon belirli görevi adlandırır, girdi alabilir ve sonuç döndürebilir.
- Dizi aynı tipteki çok sayıda değeri indisle saklar.
- Bu yapılar programı küçük, test edilebilir parçalara ayırmaya yardımcı olur.

Fonksiyon İzleme: Çağrı ve Dönüş

Tanım atlanır; çalışma 5. satırdan başlar.

alan(3, 4) çağrılacak

ana program a = – b = –

çağrı yığını (üstteki çerçeve çalışıyor)

Fonksiyon **tanımı** yalnızca kaydedilir, çalışmaz. Program 5. satırdan başlar ve **alan(3, 4)** çağrısını görür.

Adım 1 / 10

```
1 fonksiyon alan(en, boy)
2     sonuc ← en × boy
3     döndür sonuc
4 son
▶ a ← alan(3, 4)
6 b ← alan(2, 5)
7 yaz a + b
```

DEĞİŞKENLER

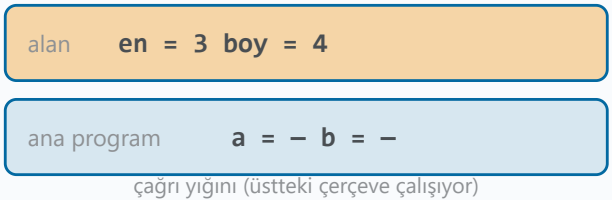
—

ÇAĞRI YIĞINI

ana program

ÇIKTI

Fonksiyon İzleme: Çağrı ve Dönüş



Yeni çerçeve açıldı.

argümanlar parametrelere kopyalandı

Çağrı yığınına **alan** için yeni bir çerçeve eklenir: 3 → en, 4 → boy.

Adım 2 / 10

```
▶ fonksiyon alan(en, boy)
2     sonuc ← en × boy
3     döndür sonuc
4 son
5 a ← alan(3, 4)
6 b ← alan(2, 5)
7 yaz a + b
```

DEĞİŞKENLER

`en = 3` `boy = 4`

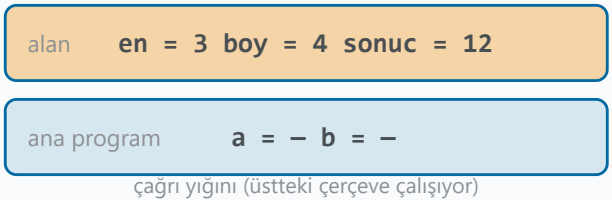
ÇAĞRI YIĞINI

`alan(3, 4)`

`ana program`

ÇIKTI

Fonksiyon İzleme: Çağrı ve Dönüş



sonuc ← 3 × 4 = 12

çağrı yığını (üstteki çerçeve çalışıyor)

Gövde çalışır: **sonuc = 12**. Bu değişken yalnızca bu çerçevede yaşar.

Adım 3 / 10

```
1 fonksiyon alan(en, boy)
▶     sonuc ← en × boy
3     döndür sonuc
4 son
5 a ← alan(3, 4)
6 b ← alan(2, 5)
7 yaz a + b
```

DEĞİŞKENLER

`en = 3` `boy = 4` `sonuc = 12`

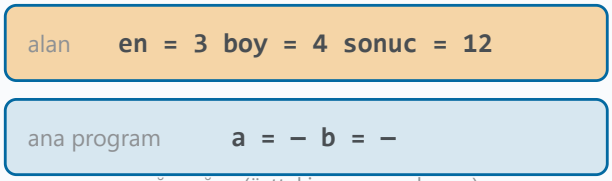
ÇAĞRI YIĞINI

`alan(3, 4)`

`ana program`

ÇIKTI

Fonksiyon İzleme: Çağrı ve Dönüş



çağrı yığını (üstteki çerçeve çalışıyor)

döndür sonuc: 12 çağıran yere gönderilir ve fonksiyon biter.

Adım 4 / 10

```
1 fonksiyon alan(en, boy)
2     sonuc ← en × boy
▶     döndür sonuc
4 son
5 a ← alan(3, 4)
6 b ← alan(2, 5)
7 yaz a + b
```

DEĞİŞKENLER

en = 3 | boy = 4 | sonuc = 12

ÇAĞRI YIĞINI

alan(3, 4)

ana program

ÇIKTI

Fonksiyon İzleme: Çağrı ve Dönüş

a ← 12

en, boy, sonuc artık yok

ana program a = 12 b = –

çağrı yığını (üstteki çerçeve çalışıyor)

Çerçeve yığından kalkar; dönen değer **a**'ya atanır: a = 12.

Adım 5 / 10

```
1 fonksiyon alan(en, boy)
2     sonuc ← en × boy
3     döndür sonuc
4 son
▶ a ← alan(3, 4)
6 b ← alan(2, 5)
7 yaz a + b
```

DEĞİŞKENLER

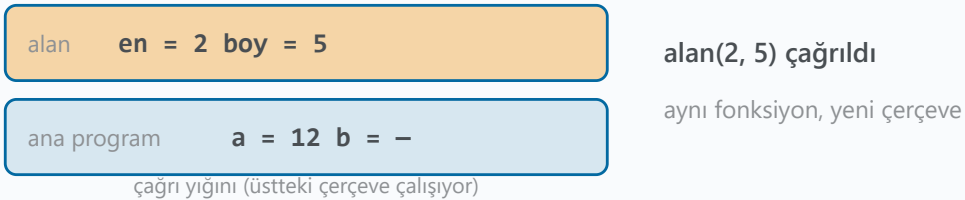
a = 12

ÇAĞRI YIĞINI

ana program

ÇIKTI

Fonksiyon İzleme: Çağrı ve Dönüş



Aynı fonksiyon farklı argümanlarla çağrılır: yepyeni bir çerçeve, en = 2, boy = 5.

Adım 6 / 10

```
▶ fonksiyon alan(en, boy)
2     sonuc ← en × boy
3     döndür sonuc
4 son
5 a ← alan(3, 4)
6 b ← alan(2, 5)
7 yaz a + b
```

DEĞİŞKENLER

en = 2 boy = 5

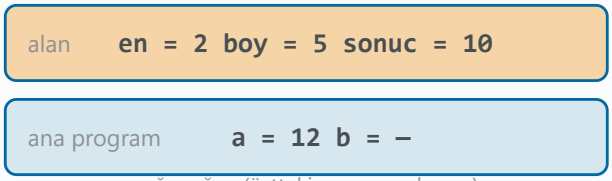
ÇAĞRI YIĞINI

alan(2, 5)

ana program

ÇIKTI

Fonksiyon İzleme: Çağrı ve Dönüş



sonuc ← 2 × 5 = 10

çağrı yığını (üstteki çerçeve çalışıyor)

sonuc = 10. Önceki çağrının 12 değeri bu çerçevede görünmez.

Adım 7 / 10

```
1  fonksiyon alan(en, boy)
▶      sonuc ← en × boy
3      döndür sonuc
4  son
5  a ← alan(3, 4)
6  b ← alan(2, 5)
7  yaz a + b
```

DEĞİŞKENLER

en = 2 | boy = 5 | sonuc = 10

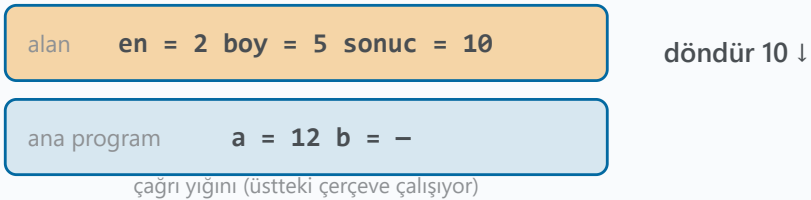
ÇAĞRI YIĞINI

alan(2, 5)

ana program

ÇIKTI

Fonksiyon İzleme: Çağrı ve Dönüş



döndür 10 ↓

çağrı yığını (üstteki çerçeve çalışıyor)

10 döndürülür, çerçeve kapanır.

Adım 8 / 10

```
1 fonksiyon alan(en, boy)
2     sonuc ← en × boy
▶     döndür sonuc
4 son
5 a ← alan(3, 4)
6 b ← alan(2, 5)
7 yaz a + b
```

DEĞİŞKENLER

en = 2 | boy = 5 | sonuc = 10

ÇAĞRI YIĞINI

alan(2, 5)

ana program

ÇIKTI

Fonksiyon İzleme: Çağrı ve Dönüş

b ← 10

ana program a = 12 b = 10

çağrı yığını (üstteki çerçeve çalışıyor)

Dönen değer **b**'ye atanır: b = 10.

Adım 9 / 10

```
1 fonksiyon alan(en, boy)
2     sonuc ← en × boy
3     döndür sonuc
4 son
5 a ← alan(3, 4)
▶ b ← alan(2, 5)
7 yaz a + b
```

DEĞİŞKENLER

a = 12 b = 10

ÇAĞRI YIĞINI

ana program

ÇIKTI

Fonksiyon İzleme: Çağrı ve Dönüş

yaz 12 + 10 → 22

ana program a = 12 b = 10

çağrı yığını (üstteki çerçeve çalışıyor)

22 yazılır. Bir kez yazılan fonksiyon iki kez kullanıldı; alan formülü tek yerde duruyor.

Adım 10 / 10

```
1 fonksiyon alan(en, boy)
2     sonuc ← en × boy
3     döndür sonuc
4 son
5 a ← alan(3, 4)
6 b ← alan(2, 5)
▶ yaz a + b
```

DEĞİŞKENLER

a = 12 b = 10

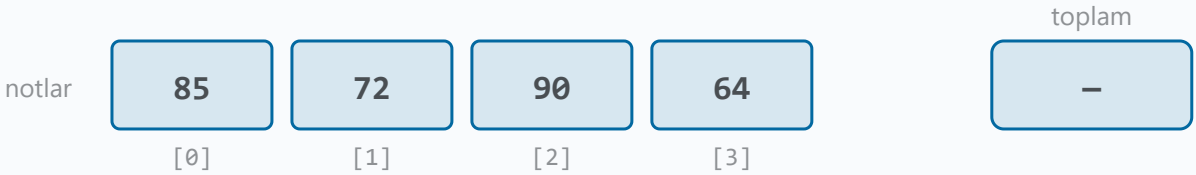
ÇAĞRI YIĞINI

ana program

ÇIKTI

```
22
```

Dizi İzleme: Notların Ortalaması



Dizi dört notu **tek bir adla** tutar; hücreler 0, 1, 2, 3 indisleriyle seçilir.

Adım 1 / 8

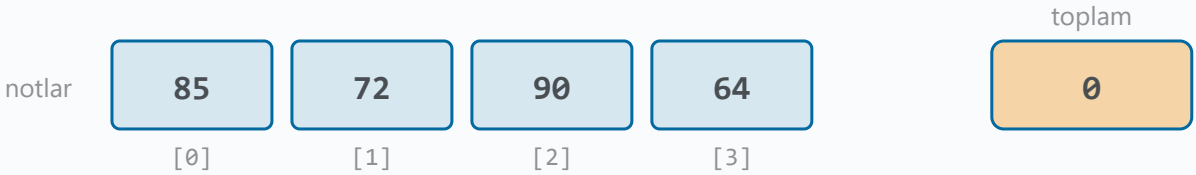
```
notlar ← [85, 72, 90, 64]
toplam ← 0
her i ← 0, 1, 2, 3 için
    toplam ← toplam + notlar[i]
ort ← toplam / 4
yaz "Ortalama: ", ort
```

DEĞİŞKENLER

`notlar = [85, 72, 90, 64]`

ÇIKTI

Dizi İzleme: Notların Ortalaması



toplam ← 0: biriktirici sıfırdan başlar. Başlangıç değeri verilmeseydi toplamın ne olduğu belli olmazdı.

Adım 2 / 8

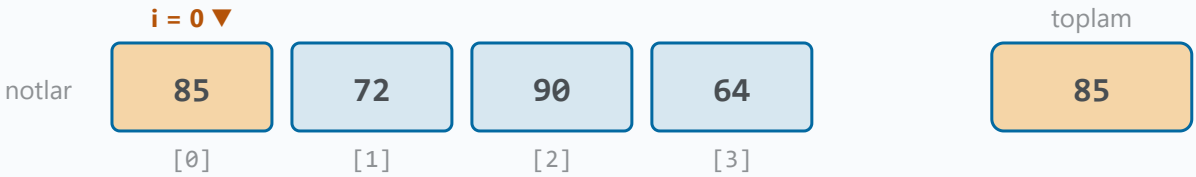
```
1 notlar ← [85, 72, 90, 64]
▶ toplam ← 0
3 her i ← 0, 1, 2, 3 için
4     toplam ← toplam + notlar[i]
5 ort ← toplam / 4
6 yaz "Ortalama: ", ort
```

DEĞİŞKENLER

`toplam = 0`

ÇIKTI

Dizi İzleme: Notların Ortalaması



i = 0: notlar[0] = 85 → toplam = 0 + 85 = **85**.

Adım 3 / 8

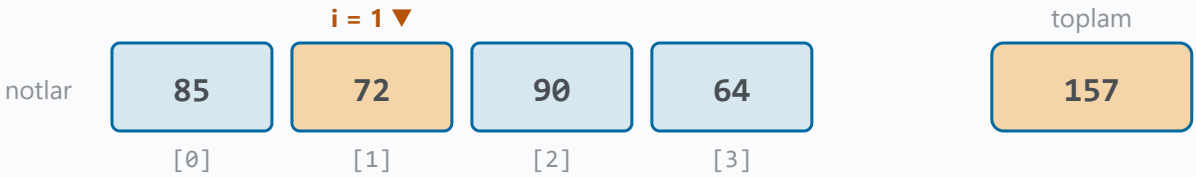
```
1 notlar ← [85, 72, 90, 64]
2 toplam ← 0
▶ her i ← 0, 1, 2, 3 için
▶     toplam ← toplam + notlar[i]
5 ort ← toplam / 4
6 yaz "Ortalama: ", ort
```

DEĞİŞKENLER

`i = 0` `notlar[i] = 85` `toplam = 85`

ÇIKTI

Dizi İzleme: Notların Ortalaması



i = 1: notlar[1] = 72 → toplam = 85 + 72 = **157**.

Adım 4 / 8

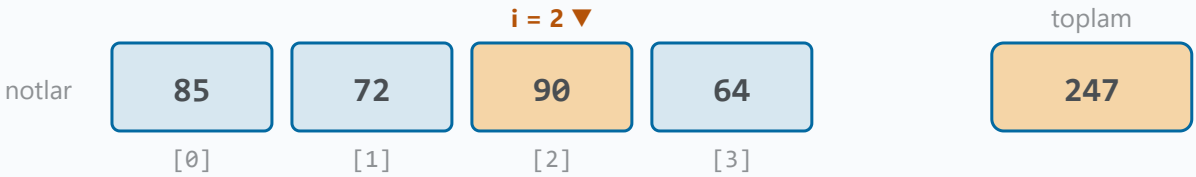
```
1 notlar ← [85, 72, 90, 64]
2 toplam ← 0
▶ her i ← 0, 1, 2, 3 için
▶     toplam ← toplam + notlar[i]
5 ort ← toplam / 4
6 yaz "Ortalama: ", ort
```

DEĞİŞKENLER

`i = 1` `notlar[i] = 72` `toplam = 157`

ÇIKTI

Dizi İzleme: Notların Ortalaması



i = 2: notlar[2] = 90 → toplam = 157 + 90 = **247**.

Adım 5 / 8

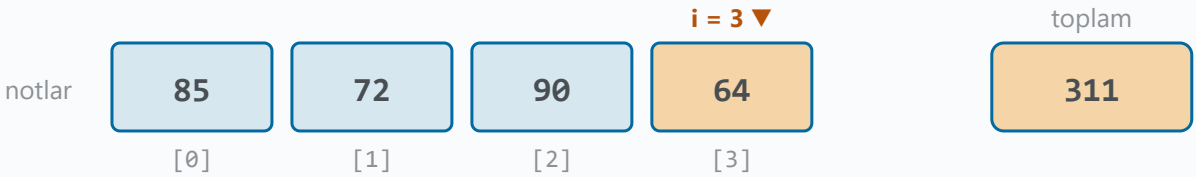
```
1 notlar ← [85, 72, 90, 64]
2 toplam ← 0
▶ her i ← 0, 1, 2, 3 için
▶     toplam ← toplam + notlar[i]
5 ort ← toplam / 4
6 yaz "Ortalama: ", ort
```

DEĞİŞKENLER

`i = 2` `notlar[i] = 90` `toplam = 247`

ÇIKTI

Dizi İzleme: Notların Ortalaması



i = 3: notlar[3] = 64 → toplam = 247 + 64 = **311**.

Adım 6 / 8

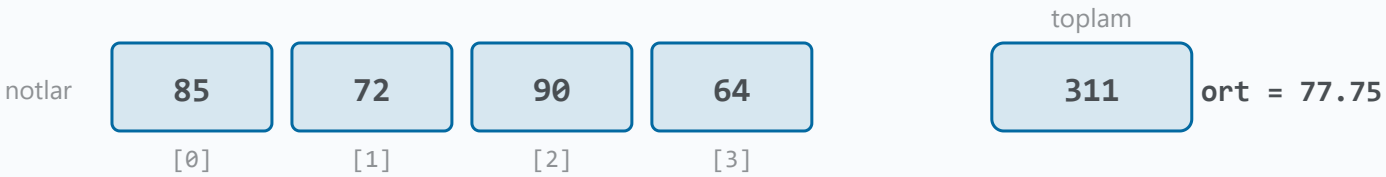
```
1 notlar ← [85, 72, 90, 64]
2 toplam ← 0
▶ her i ← 0, 1, 2, 3 için
▶     toplam ← toplam + notlar[i]
5 ort ← toplam / 4
6 yaz "Ortalama: ", ort
```

DEĞİŞKENLER

`i = 3` `notlar[i] = 64` `toplam = 311`

ÇIKTI

Dizi İzleme: Notların Ortalaması



Döngü 4 tur sonra biter. **ort = 311 / 4 = 77.75**. Son indis 3'tür: uzunluk − 1.

Adım 7 / 8

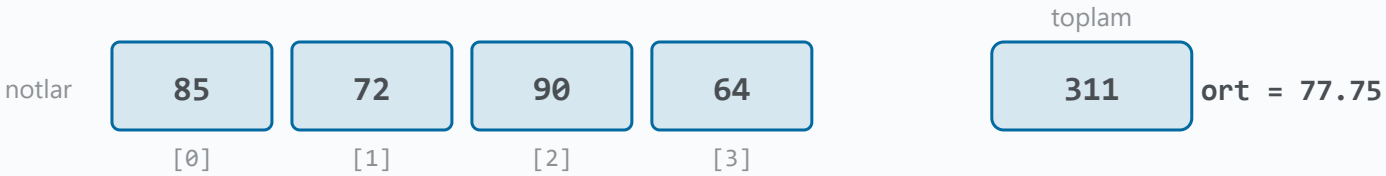
```
1 notlar ← [85, 72, 90, 64]
2 toplam ← 0
3 her i ← 0, 1, 2, 3 için
4     toplam ← toplam + notlar[i]
▶ ort ← toplam / 4
6 yaz "Ortalama: ", ort
```

DEĞİŞKENLER

`toplam = 311` `ort = 77.75`

ÇIKTI

Dizi İzleme: Notların Ortalaması



Sonuç yazılır. Döngü **notlar[4]**'e uzansaydı dizinin dışına çıkılırdı: 4 öğeli dizide 4 numaralı hücre yoktur.

Adım 8 / 8

```
1 notlar ← [85, 72, 90, 64]
2 toplam ← 0
3 her i ← 0, 1, 2, 3 için
4     toplam ← toplam + notlar[i]
5 ort ← toplam / 4
▶ yaz "Ortalama: ", ort
```

DEĞİŞKENLER

`toplam = 311` `ort = 77.75`

ÇIKTI

```
Ortalama: 77.75
```

Kütüphaneler ve API

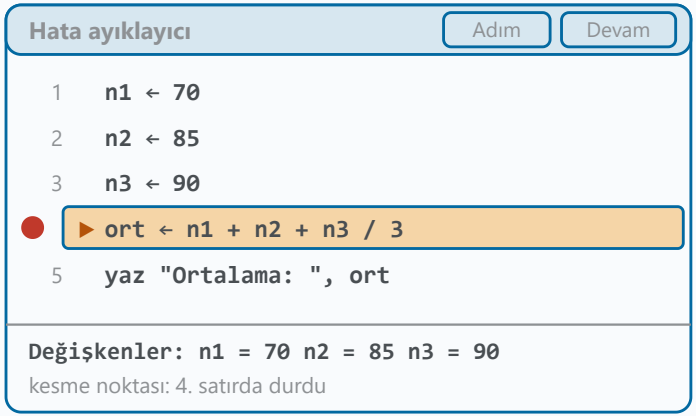


- **Kütüphane**, başkalarının yazıp test ettiği hazır fonksiyonların topluluğudur; aynı işi her programda yeniden yazmayı önler.
- **API** (uygulama programlama arayüzü) bir kütüphanenin ya da hizmetin dışarıya sunduğu fonksiyonları, girdilerini ve dönüş değerlerini tanımlar. Kullanan, içini bilmek zorunda değildir.

Çağrı	Dil	Sonuç
`Math.sqrt(16)`	Java	4.0
`Math.max(3, 8)`	Java	8
`len("Giresun")`	Python	7

Hata Ayıklama Teknikleri

- **Hatayı yeniden üret**: aynı girdiyle hata her seferinde çıkıyor mu?
- **Ara değerleri yazdır**: şüpheli satırdan önce ve sonra değişkenleri yaz.
- **Kesme noktası** koy: program o satıra gelince durur; değişkenler incelenir.
- **Adım adım** ilerle: her satırdan sonra hangi değerin değiştiğine bak.
- **Alanı daralt**: kodun yarısını devre dışı bırakıp hatanın hangi yarıda olduğunu bul.
- **Anlatarak düşün**: kodu satır satır birine (ya da masadaki bir oyuncağa) anlatmak çoğu zaman hatayı gösterir.



Şekildeki program durdurulduğunda n3 / 3 = 30 hesaplanacağı görülür: ortalama 185 çıkar, beklenen 81.67'dir.

İzleyerek Hata Bulma: En Büyük Sayı

Test 1: 7, 12, 5

a	b	c	eb
7	12	5	–

Test 1: girdiler 7, 12, 5; beklenen çıktı 12.

Adım 1 / 10

```
▶ oku a, b, c
2 eb ← 0
3 eğer a > eb ise eb ← a
4 eğer b > eb ise eb ← b
5 eğer c > eb ise eb ← c
6 yaz "En büyük: ", eb
```

DEĞİŞKENLER

a = **7** b = **12** c = **5**

ÇIKTI

İzleyerek Hata Bulma: En Büyük Sayı

Test 1: 7, 12, 5

a	b	c	eb
7	12	5	0

eb ← **0**: en büyük için başlangıç değeri.

Adım 2 / 10

```
1 oku a, b, c
▶ eb ← 0
3 eğer a > eb ise eb ← a
4 eğer b > eb ise eb ← b
5 eğer c > eb ise eb ← c
6 yaz "En büyük: ", eb
```

DEĞİŞKENLER

a = 7 | b = 12 | c = 5 | eb = 0

ÇIKTI

İzleyerek Hata Bulma: En Büyük Sayı

Test 1: 7, 12, 5

a	b	c	eb
7	12	5	7

`7 > 0`

doğru → eb = 7

7 > 0 doğru: eb = 7.

Adım 3 / 10

```
1 oku a, b, c
2 eb ← 0
▶ eğer a > eb ise eb ← a
4 eğer b > eb ise eb ← b
5 eğer c > eb ise eb ← c
6 yaz "En büyük: ", eb
```

DEĞİŞKENLER

`a = 7` `b = 12` `c = 5` `eb = 7`

ÇIKTI

İzleyerek Hata Bulma: En Büyük Sayı

Test 1: 7, 12, 5



12 > 7

doğru → eb = 12

12 > 7 doğru: eb = 12.

Adım 4 / 10

```
1 oku a, b, c
2 eb ← 0
3 eğer a > eb ise eb ← a
▶ eğer b > eb ise eb ← b
5 eğer c > eb ise eb ← c
6 yaz "En büyük: ", eb
```

DEĞİŞKENLER

a = **7** | b = **12** | c = **5** | eb = **12**

ÇIKTI

İzleyerek Hata Bulma: En Büyük Sayı

Test 1: 7, 12, 5

a	b	c	eb
7	12	5	12

5 > 12

yanlış

5 > 12 yanlış: eb değişmez. Çıktı **12** ✓. Tek bir başarılı test programın doğru olduğunu kanıtlamaz.

Adım 5 / 10

```
1 oku a, b, c
2 eb ← 0
3 eğer a > eb ise eb ← a
4 eğer b > eb ise eb ← b
▶ eğer c > eb ise eb ← c
▶ yaz "En büyük: ", eb
```

DEĞİŞKENLER

a = **7** | b = **12** | c = **5** | eb = **12**

ÇIKTI

```
En büyük: 12
```

İzleyerek Hata Bulma: En Büyük Sayı



Test 2: hepsi negatif, −5, −2, −9; beklenen çıktı −2.

Adım 6 / 10

```
▶ oku a, b, c
2 eb ← 0
3 eğer a > eb ise eb ← a
4 eğer b > eb ise eb ← b
5 eğer c > eb ise eb ← c
6 yaz "En büyük: ", eb
```

DEĞİŞKENLER

a = −5 b = −2 c = −9

ÇIKTI

İzleyerek Hata Bulma: En Büyük Sayı

Test 2: −5, −2, −9

a	b	c	eb
-5	-2	-9	0

eb ← 0. Bu değer hiçbir girdiden gelmiyor.

Adım 7 / 10

```
1 oku a, b, c
▶ eb ← 0
3 eğer a > eb ise eb ← a
4 eğer b > eb ise eb ← b
5 eğer c > eb ise eb ← c
6 yaz "En büyük: ", eb
```

DEĞİŞKENLER

a = −5 | b = −2 | c = −9 | eb = 0

ÇIKTI

İzleyerek Hata Bulma: En Büyük Sayı

Test 2: −5, −2, −9

a	b	c	eb
-5	-2	-9	0

`-5 > 0 · -2 > 0 · -9 > 0`

üçü de yanlış

Üç karşılaştırmanın üçü de yanlış: hiçbir girdi 0'dan büyük değil, eb hiç değişmez.

Adım 8 / 10

```
1 oku a, b, c
2 eb ← 0
▶ eğer a > eb ise eb ← a
▶ eğer b > eb ise eb ← b
▶ eğer c > eb ise eb ← c
6 yaz "En büyük: ", eb
```

DEĞİŞKENLER

`a = −5` `b = −2` `c = −9` `eb = 0`

ÇIKTI

İzleyerek Hata Bulma: En Büyük Sayı

Test 2: −5, −2, −9

a	b	c	eb
-5	-2	-9	0

çıktı 0, beklenen -2

✗ mantık hatası

Çıktı **0** ✗. İzleme suçluyu gösteriyor: 2. satırdaki başlangıç değeri girdilerin dışından geliyor.

Adım 9 / 10

```
1 oku a, b, c
2 eb ← 0
3 eğer a > eb ise eb ← a
4 eğer b > eb ise eb ← b
5 eğer c > eb ise eb ← c
▶ yaz "En büyük: ", eb
```

DEĞİŞKENLER

a = −5 | b = −2 | c = −9 | eb = 0

ÇIKTI

```
En büyük: 0
```

İzleyerek Hata Bulma: En Büyük Sayı

Düzeltme: eb ← a

a	b	c	eb
-5	-2	-9	-2

`eb ← a ile yeniden`

✓ En büyük: -2

Düzeltme: **eb ← a**. eb = −5 → −2 > −5 doğru, eb = −2 → −9 > −2 yanlış. Çıktı −2 ✓. Test 1 de yeniden denenir: yine 12.

Adım 10 / 10

```
1 oku a, b, c
▶ eb ← 0
3 eğer a > eb ise eb ← a
4 eğer b > eb ise eb ← b
5 eğer c > eb ise eb ← c
6 yaz "En büyük: ", eb
```

DEĞİŞKENLER

a = **−5** | b = **−2** | c = **−9** | eb = **−2**

ÇIKTI

```
En büyük: -2
```

HATA AYIKLAMA

Sık Yapılan Hatalar

Hata	Örnek	Ne olur?
Başlangıç değeri yok ya da yanlış	`toplam` başlatılmadan `toplam ← toplam + x`	Java derlemez; bazı dillerde rastgele bir değerle hesaplar
Sınır hatası (bir eksik / bir fazla)	`her i ← 1 … n − 1 için`	son öğe atlanır ya da bir fazla tur döner
Sonsuz döngü	döngü değişkeni gövdede güncellenmez	program bitmez
Sıfıra bölme	`ort ← toplam / n`, n = 0	çalışma zamanı hatası
Dizinin dışına çıkmak	4 öğeli dizide `notlar[4]`	çalışma zamanı hatası
Parantez unutmak	`ort ← a + b / 2`	yanlış sonuç (mantık hatası)
Girdiyi doğrulamamak	yaş olarak −3 kabul etmek	anlamsız sonuç, sonraki adımlarda hata

Sınır durumları (0, boş liste, en küçük ve en büyük değer, negatif sayı) bu hataların çoğunu ortaya çıkarır.

HATA AYIKLAMA

Yazılım Hatalarının Bedeli

Ariane 5 (1996)

64 bitlik kayan noktalı bir hız değeri 16 bitlik tam sayıya dönüştürülürken **taşma** oldu. Seyrüsefer yazılımı durdu; roket kalkıştan yaklaşık 37 saniye sonra rotasından saptı ve birkaç saniye içinde parçalandı.

Mars Climate Orbiter (1999)

Bir yazılım itme verisini İngiliz birimiyle (pound-kuvvet·saniye) üretti, diğeri metrik birim (newton·saniye) bekledi. Sonda Mars'a fazla yaklaştı ve kayboldu.

Therac-25 (1985–1987)

Radyoterapi cihazının yazılımındaki bir zamanlama hatası, operatör hızlı veri girdiğinde güvenlik denetimini atlattı; hastalar aşırı radyasyon dozu aldı.

- Üçü de derleyicinin yakalayamayacağı hatalardır: biri sınır durumu, biri birim tutarsızlığı, biri zamanlamaya bağlı mantık hatası.
- Ortak ders: test, sınır durumlarını denemek, birimleri açıkça yazmak ve kodun başka biri tarafından incelenmesi.

Okunabilir Kod

Çalışır ama okunmaz

```
a ← oku()
b ← oku()
c ← a*b/2
yaz c
```

Aynı program, okunur

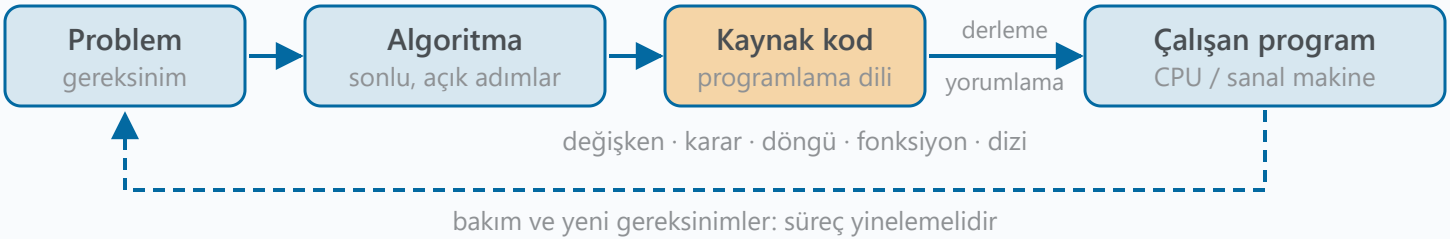
```
// üçgenin alanı = taban × yükseklik / 2
taban ← oku()
yukseklik ← oku()
alan ← taban × yukseklik / 2
yaz "Alan: ", alan
```

- **Anlamlı adlar**: `c` yerine `alan`; okuyan kişi değişkenin rolünü addan anlar.
- **Tutarlı girinti ve boşluk**: iç içe blokların sınırı bir bakışta görünür.
- **Sihirli sayı** yerine adlandırılmış sabit: `0.20` yerine `KDV_ORANI ← 0.20`; oran değişince tek satır düzeltilir.
- **Tekrarı fonksiyona taşı**: aynı formül üç yerde yazılıysa bir düzeltme üç kez yapılmak zorunda kalır.
- Açıklama kodun **ne** yaptığını değil **neden** yaptığını anlatır. Kod bir kez yazılır, defalarca okunur.

ÖZET

Bölüm Özeti

- Programlama dili algoritmayı bilgisayarın yürütebileceği biçimde ifade eder.
- Kaynak kod derleme, yorumlama veya hibrit bir süreçle çalıştırılır.
- Diller düzey, tür ve paradigmaya göre sınıflandırılır.
- Yazılım geliştirme gereksinimden bakıma uzanan yinelemeli bir süreçtir.
- Değişken, karar, döngü, fonksiyon ve dizi temel programlama yapılarını oluşturur.
- Hatalar sınır durumlarıyla test ederek ve programı adım adım izleyerek bulunur.



ÖZET

Kontrol Soruları

1. Kaynak kod ile makine dili arasındaki fark nedir?
2. Derlenen ve yorumlanan programlar nasıl ayrılır?
3. Bir algoritma neden programlama dilinden bağımsız düşünülebilir?
4. Yazılım testinin amacı nedir?
5. Karar yapısı ile döngünün görevleri nasıl farklıdır?
6. Bağlayıcı ve yükleyici programın çalışmasında hangi işi yapar?
7. Statik ve dinamik tiplemede tip hatası ne zaman fark edilir?
8. Buyurgan ve fonksiyonel yaklaşım bir dizinin toplamını nasıl farklı ifade eder?
9. Bir döngü neden sonsuz döngüye dönüşür?

ÖZET

Alıştırmalar

1. Aşağıdaki hataları derleme, çalışma zamanı ve mantık hatası olarak sınıflandırın: noktalı virgül unutmak; ortalama için toplamı n yerine n + 1 ile bölmek; sıfıra bölmek.
2. "Bir öğrencinin üç sınav notunun ortalamasını hesapla" problemi için girdi, çıktı ve kısıtları yazın.
3. Aynı problem için sözde kod yazın ve (70, 85, 90) girdisiyle izleyin; sonuç 81.67 olmalı.
4. Dikdörtgen alanı algoritmasına çevre hesabını ekleyin: cevre = 2 × (en + boy). en = 3, boy = 4 için beklenen çıktılar nedir?
5. Bir telefon uygulaması için yazılım geliştirme sürecinin her aşamasında yapılacak birer işi yazın.
6. Java, Python ve C dillerini derlenen / yorumlanan / hibrit olarak sınıflandırın ve çalıştırma hatlarını çizin.

▶ **İpuçları**

ÖZET

Kaynaklar

- Sebesta, Concepts of Programming Languages, 12. baskı, Pearson, 2019.
- Aho, Lam, Sethi ve Ullman, Compilers: Principles, Techniques, and Tools, 2. baskı, Addison-Wesley.
- Sedgewick ve Wayne, Computer Science: An Interdisciplinary Approach, Addison-Wesley, 2016.
- Downey ve Mayfield, Think Java, 2. baskı, O'Reilly, 2019 (**ücretsiz çevrim içi sürüm**).
- Oracle, The Java Tutorials: **Getting Started**.

Son

Programlamaya Giriş

Sonraki sunum: Java'ya Giriş ve İlk Program · Çalışma soruları · Terim sözlüğü