**Derleyici, Yorumlayıcı, Bytecode ve Programlama Dilleri Üzerine Temel Bir Rehber**

Bilgisayarlar bizimle aynı dili konuşmaz. Biz insanlar kelimelerle, anlamlarla düşünürken, bilgisayarlar yalnızca iki rakamla var olur: **0 ve 1**. Peki bu kadar "basit" bir sistemle böylesine kompleks uygulamalar nasıl çalışıyor? Gelin, birlikte keşfedelim.

**1. Bilgisayarların Anadilinde Bilgi: Bitler ve Transistörler**

Bilgisayarda her veri parçası — ister bir fotoğraf olsun, ister bir müzik dosyası ya da kod — **0 ve 1**’lerden oluşur. Bu ikili sistem, bilgisayar donanımındaki **transistörlerin açık (1) veya kapalı (0)** olma durumuyla temsil edilir.

Yani bir bakıma bilgisayarlar kendilerini **binary (ikili) sayılarla ifade eden makineler**dir. Ancak bu verilerin anlam kazanabilmesi için insan-dostu dillerle bilgisayar arasında bir köprü kurulması gerekir.

**2. Yüksek Seviye (High Level Programming) ve Düşük Seviye (Low level Programming) Programlama Dilleri**

Bilgisayarın anlayabildiği **en düşük seviye** dil, doğrudan 0 ve 1’lerden oluşan **makine kodudur**. Ancak insanlar için okunması ve yazması imkansıza yakındır.

metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Bu nedenle, programcılar **yüksek seviye diller** kullanır:

* **Python**, **Java**, **C#**, **JavaScript** gibi.

Ancak yüksek seviye bir dil, bilgisayarın anlayacağı dile **dönüştürülmeden** çalıştırılamaz. İşte bu dönüşüm sürecinde karşımıza şu kavramlar çıkar:

**High Level Languages:**

* Yüksek seviye diller, **insanların anlayabileceği** şekilde yazılmıştır.  
  **Abstraksiyon** (soyutlama) yüksektir, donanım detayları gizlenmiştir.
* Bellek yönetimi, donanım adreslemesi gibi detayları **otomatik olarak** halleder.
* Derleyici (compiler) ya da yorumlayıcı (interpreter) kullanılarak çalıştırılır.

**Low-Level Language (Düşük Seviye Programlama Dili)**

* Düşük seviye diller, **donanım ile çok daha yakından çalışır**.  
  Makinenin nasıl çalıştığına dair daha fazla kontrol sağlar, ama yazımı daha zordur.

**Özellikleri:**

* Donanım odaklıdır, **hızlı ve verimlidir**.
* Programcıdan daha fazla detay ister (örneğin: bellek yönetimi).
* **İşlemci mimarisine özgü** olabilir (taşınabilirlik düşük).
* Genelde daha az soyutlama içerir.

**İki alt sınıfa ayrılır:**

**1. Assembly Language (Orta seviye/Low-level) :** Assembly Language, bilgisayar donanımına çok yakın, ancak tamamen 0 ve 1 olmayan, insan tarafından yazılabilir bir düşük seviye programlama dilidir. Her işlemci (CPU) yalnızca kendine özel **makine komutlarını** anlar. **Her işlemcinin kendi komut seti (Instruction Set Architecture – ISA)** vardır. Assembly dili de bu komut setine göre yazılır.Bu yüzden **bir işlemcide çalışan Assembly kodu, başka bir işlemcide çalışmayabilir.** **ISA, işlemcinin hangi komutları anlayabildiğini belirler.**

* İnsan tarafından okunabilir bir versiyondur.
* Ama her komut doğrudan işlemci komutlarına karşılık gelir.
* Her işlemci için farklıdır (x86, ARM vs.)
* Bellek ve donanım üzerinde tam kontrol sağlar

**2. Machine Language (Makine dili)**

* Bilgisayarın anladığı **saf 0 ve 1'lerden** oluşur.

**3. Compiler (Derleyici) Nedir?**

**Derleyici (compiler)**, yazdığınız kodu **tamamıyla alır**, kontrol eder ve **makineye özel bir çıktı** üretir.

**Özellikleri:**

* Tüm kod **önceden derlenir.**
* Eğer hata varsa, program hiç çalışmaz.
* Derlenen çıktı genellikle .exe, .out veya Java’da .class dosyasıdır.

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**4. Bytecode Nedir?**

Java gibi dillerde, compiler çıktısı doğrudan makine kodu değil, arada bir **ara format** olan **bytecode**’dur.

* **Makineye özel değildir.**
* **Java Virtual Machine (JVM)** adlı yorumlayıcı tarafından satır satır çevrilerek çalıştırılır.
* Bu sayede Java programları, "bir kez yaz, her yerde çalıştır" mottosunu gerçekleştirebilir.
* Java gibi dillerde, yazılan kod **önce bytecode** denen özel bir formata dönüştürülür.
* Bu kodu **doğrudan işlemci çalıştıramaz**.
* Bunun amacı **platform bağımsızlık** sağlamaktır: aynı bytecode her işletim sisteminde çalışabilir (Windows, Linux, macOS vs.)

**5. Interpreter (Yorumlayıcı) Nedir?**

**Yorumlayıcı (interpreter)**, yazdığınız kodu **satır satır çalıştırır.**

**JVM nedir?**

* **Java Virtual Machine**, Hello.class gibi bytecode dosyalarını **satır satır** yorumlayarak (interpreter gibi) **makine koduna dönüştürür** ve çalıştırır.

**Özellikleri:**

* Kodu **anında çalıştırır**, önceden bir çıktı üretmez.
* Hatalar çalışırken ortaya çıkar.
* Genellikle Python, JavaScript gibi dillerde kullanılır.

**Interpreter nasıl çalışır?**

* Kodu **bir satır okur, anlar ve çalıştırır**.
* **Tüm programı önceden çevirmek yerine** çalıştıkça çevirir.
* Hata varsa çalıştırdığı sırada verir (derleme sırasında değil).

metin, yazı tipi, çizgi, ekran görüntüsü içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**6. Machine Code (Makine Kodu) Nedir?**

Makine kodu, işlemcinin **doğrudan anlayabildiği tek dildir**.  
Tamamen 0 ve 1’lerden oluşur ve her işlemci mimarisi (x86, ARM vb.) için özeldir.

Programınız son adımda bu **ikili sinyaller** haline gelir ve bilgisayar donanımı üzerinden çalıştırılır.

İşlemci (CPU), bu dili okur ve çalıştırır.

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**7. High-Level vs Low-Level Diller**

| **Özellik** | **High-Level Language** | **Low-Level Language** |
| --- | --- | --- |
| Okunabilirlik | İnsanlar için uygundur | Makineye yakındır |
| Yazım kolaylığı | Kolay | Zordur |
| Hız / Performans | Daha yavaş | Çok hızlı |
| Bellek yönetimi | Otomatik (GC) | Elle yapılır |
| Taşınabilirlik | Yüksek | Düşük |
| Örnek diller | Python, Java, C# | Assembly, Machine Code |
|  |  |  |

* **JDK**, Java programları geliştirmek ve çalıştırmak için gerekli **tüm araçları içeren yazılım paketidir**.

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**IDE**, yazılım geliştirme sürecini kolaylaştıran, birçok aracı tek arayüzde sunan bir **geliştirme ortamıdır**.

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**Tüm sayısal olmayan ifadeler, bilgisayarda önce sayılara dönüştürülür; sonra bu sayılar 0 ve 1’lere çevrilerek saklanır.**

**1. Metin (Harfler, Karakterler) Nasıl Saklanır?**

**👉 Karakter kodlama sistemleri kullanılır:**

**✅ ASCII (American Standard Code for Information Interchange)**

* Her karaktera bir sayı atanır.
* Örneğin:
  + 'A' → **65**
  + 'a' → **97**
  + ' ' (boşluk) → **32**

Bilgisayar bu sayıları **binary** (örneğin: 65 → 01000001) şeklinde saklar.

**✅ Unicode (UTF-8, UTF-16)**

* ASCII yetersiz kaldığında tüm dünya dillerini, emojileri vs. kapsayan sistem.
* Örneğin:
  + 'Ş' → 350
  + '你' → 20320
  + '😀' → 128512

**🖼️ 2. Görsel (Resim) Veriler Nasıl Saklanır?**

Resimler; piksellerden oluşur ve her pikselin **renk değeri** vardır.

* Örneğin: bir pikselin rengi **RGB (Kırmızı, Yeşil, Mavi)** olarak ifade edilir.
  + (255, 0, 0) = kırmızı
* Bu değerler **sayılardır**, her biri 0–255 arasında olur.
* Tüm görsel bu sayı matrisleri olarak saklanır → sonra **bit’lere çevrilir**.

**🔊 3. Ses Verileri Nasıl Saklanır?**

* Ses, bir **analog sinyaldir**, bilgisayar ise sadece dijital veri anlayabilir.
* Bu nedenle:
  + Ses dalgaları belirli aralıklarla **örneklenir (sampling)**
  + Her örnek bir sayıdır (örneğin: -32768 ila +32767 arası)
  + Tüm bu sayılar **binary** olarak diskte saklanır

🎧 Örneğin bir MP3 dosyası, aslında sıkıştırılmış ses örneklerinden oluşan sayısal bir listedir.

**🎥 4. Video Verileri Nasıl Saklanır?**

* Video = **Görsel + Ses + Zaman**
* Her kare (frame) bir resim gibidir → pikseller sayısal olarak tutulur
* Araya zaman bilgisi (frame rate) eklenir
* Ses kanalı da eş zamanlı tutulur
* Tüm bu veriler sıkıştırılarak (codec ile) dosya formatına çevrilir (örneğin: .mp4)

**🧠 Özetle**

| **Veri Türü** | **Bilgisayarda Nasıl Saklanır?** |
| --- | --- |
| Harf, Metin | Karakter kodları (ASCII, Unicode) → Sayılar → 0 ve 1 |
| Görsel | Piksel renk değerleri (RGB) → Sayılar → 0 ve 1 |
| Ses | Ses örnekleme değerleri → Sayılar → 0 ve 1 |
| Video | Kareler (resim) + Ses + Zaman → Sayılar → 0 ve 1 |

**Bit, Byte ve Diğer Veri Birimleri: Bilgisayarlar Veriyi Nasıl Saklar?**

Günlük hayatta kullandığımız bilgisayarlar, telefonlar ve diğer dijital cihazlar sayısal verilerle çalışır. Ancak bu verinin arka planda nasıl temsil edildiği çoğu zaman göz ardı edilir. "1 TB depolama alanı" ya da "dosya boyutu 3 MB" gibi ifadeler ne anlama geliyor? Bu yazıda veri birimlerinin temellerini, bit’ten terabyte’a kadar olan yapıyı sade ve teknik bir dille ele alacağız.

**Bit Nedir?**

Bit (Binary Digit), bilgisayarların anlayabildiği **en küçük veri birimidir**. Sadece iki değerden birini alabilir: **0** veya **1**. Bilgisayarlar tüm işlemlerini bu iki sembol üzerinden yapar. Her bir 0 veya 1, bir elektrik sinyalinin açık ya da kapalı olma durumunu temsil eder.

**Byte Nedir?**

Bir **byte**, **8 bit’ten** oluşur. Yani, 8 tane 0 veya 1'in bir araya gelmesiyle bir byte oluşur. Örneğin:

css

KopyalaDüzenle

01000001 → Bu bit dizisi, ASCII sistemine göre büyük A harfini temsil eder.

Bir karakterin (örneğin harf, rakam, sembol) bilgisayarda tutulması için genellikle 1 byte yeterlidir.

**Veri Birimleri ve Katları**

Byte, veri ölçümünün temel taşıdır. Ancak modern uygulamalarda daha büyük birimlere ihtiyaç duyulur. Bu nedenle byte’ın katları kullanılır. Aşağıda en yaygın veri birimlerini ve yaklaşık büyüklüklerini bulabilirsin:

| **Birim** | **Değer (İkili)** | **Açıklama** |
| --- | --- | --- |
| 1 Bit | 0 veya 1 | En küçük birim |
| 1 Byte | 8 Bit | 1 karakter bilgisi |
| 1 Kilobyte | 1,024 Byte | Küçük bir metin dosyası (~1 sayfa) |
| 1 Megabyte | 1,024 KB | Bir şarkı veya küçük resim dosyası |
| 1 Gigabyte | 1,024 MB | Video veya büyük yazılım dosyaları |
| 1 Terabyte | 1,024 GB | Harici disk kapasitesi |
| 1 Petabyte | 1,024 TB | Devasa veri merkezlerinde kullanılır |
| 1 Exabyte | 1,024 PB | Tüm internet trafiği düzeyinde veri |

Not: Bu değerler **ikili sistem**e (2 tabanlı) göredir. Pazarlama terimlerinde 1 KB genellikle 1,000 byte olarak kullanılsa da, teknik olarak 1 KB = 1,024 byte’tır.

**Sayısal Olmayan Veriler Bilgisayarda Nasıl Tutulur?**

Bilgisayar yalnızca sayılarla çalıştığına göre, resim, ses, metin gibi sayısal olmayan veriler nasıl temsil edilir?

**Metin Verisi**

Karakterler, ASCII veya Unicode gibi karakter kodlama sistemleriyle belirli sayılara karşılık gelir. Örneğin:

* 'A' → 65 (ASCII)
* 'ş' → 351 (Unicode)

Bu sayılar daha sonra 0 ve 1'lerle ifade edilerek saklanır.

**Görsel Verisi**

Resimler, milyonlarca pikselden oluşur. Her pikselin rengi sayısal bir değerle (örneğin RGB: 255, 0, 0) temsil edilir. Bu renk değerleri sayılara çevrilir ve bit dizisi olarak saklanır.

**Ses Verisi**

Analog ses dalgaları belirli aralıklarla örneklenir (sampling). Her örnek bir sayıya karşılık gelir ve bu sayılar dijital olarak saklanır. Örneğin, 44.1 kHz örnekleme oranı, saniyede 44,100 ses örneği demektir.

**Video Verisi**

Video, karelerden (frame) ve seseşzamanlı ses verilerinden oluşur. Her kare bir görüntü, her saniye belirli sayıda kare ve ses örneği içerir. Bu veriler sıkıştırılarak dosya biçimlerine (MP4, AVI) dönüştürülür.

**Günlük Hayattan Örnekler**

* 1 SMS (160 karakter) → yaklaşık 160 byte
* 1 MP3 şarkı (3 dakika) → ~3 MB
* 1 HD fotoğraf → ~2-6 MB
* 1 film (HD) → ~1-4 GB
* 1 TB harici disk → ~500 HD film veya ~250.000 şarkı

**Sonuç**

Bilgisayar dünyasında her şey, en temelde 0 ve 1’lerden oluşur. Bit, byte ve onların katları; bilgisayarların veriyi nasıl sakladığını, işlediğini ve ilettiğini anlamak için temel yapı taşlarıdır. Sayısal olmayan her veri, önce sayıya dönüştürülür ve ardından bu sayıların ikili temsilleri üzerinden bilgisayarın anlayabileceği forma sokulur.

Bu temel bilgileri anlamak, daha derin konulara geçerken sağlam bir zemin oluşturacaktır.

**Recursion**, bir problemin **daha küçük alt problemlerine bölünerek**, bu alt problemler çözülene kadar **aynı fonksiyonun kendi kendini çağırmasıdır.**