N° 05 · ARAŞTIRMA ÖDEVİ · KARAKTER KODLAMA

Aynı harf, *üç farklı* bayt dizisi.

UTF-8, UTF-16 ve UTF-32 aynı Unicode karakter kümesini bilgisayar belleğine yazmanın üç farklı yoludur. Hepsi her karakteri temsil edebilir; aralarındaki fark **kaç bayt kullandıkları**, **nasıl böldükleri** ve **nerede tercih edildikleri**dir.

Hazırlayan **Yusuf Tabak** Ders **Web Tasarımın Temelleri** Tarih **24 Kasım 2026**

UTF-8	UTF-16	UTF-32
1–4 bayt	2 veya 4 bayt	Her zaman 4 bayt
Değişken uzunluk · ASCII uyumlu · Web'in standardı	Değişken uzunluk · Windows, Java, JavaScript	Sabit uzunluk · Kod noktası = sayı

01 TEMEL KAVRAM — Önce Unicode: karakter ile bayt aynı şey değil

Bilgisayar yalnızca sayıları saklar. "A" harfini saklamak için önce ona bir numara vermek gerekir. 1963'te çıkan **ASCII** 7 bit ile sadece 128 karakter tanımlıyordu: İngiliz alfabesi, rakamlar ve noktalama. `ş`, `ğ`, `ı`, Arapça, Çince ya da emoji için yer yoktu. Her ülke kendi 8 bitlik tablosunu yaptı (Türkçe için `ISO-8859-9` / `Windows-1254`) ve aynı bayt farklı bilgisayarlarda farklı harf anlamına gelmeye başladı.

Unicode (1991) bu karmaşayı tek bir evrensel tabloyla çözdü: dünyadaki her karaktere benzersiz bir numara, yani bir **kod noktası** (*code point*) verdi. Kod noktaları `U+` ön ekiyle onaltılık yazılır: `A = U+0041`, `ş = U+015F`, `€ = U+20AC`, `😀 = U+1F600`. Toplam alan `U+0000` – `U+10FFFF` arasıdır, yani 1.114.112 olası kod noktası; bugün bunların 150.000'den fazlasına karakter atanmıştır.

> **Kilit nokta:** Unicode bir *tablodur*, bir dosya biçimi değildir. "U+015F" sayısını diske hangi baytlarla yazacağımızı söyleyen kurallara **kodlama** (*encoding*) denir. UTF-8, UTF-16 ve UTF-32 bu kurallardan üçüdür. UTF, *Unicode Transformation Format* demektir; sondaki sayı kullanılan en küçük birimin (*code unit*) bit genişliğidir: 8, 16 veya 32 bit.

Kod noktaları 17 **düzleme** (*plane*) ayrılır. İlki olan **BMP** (Basic Multilingual Plane, `U+0000` – `U+FFFF`) neredeyse tüm yaşayan dillerin harflerini içerir. Emojiler, tarihî yazılar ve nadir Çince karakterler ise BMP'nin dışındaki "ek düzlemlerdedir" (`U+10000` ve üstü). Bu ayrım UTF-16'yı anlamak için önemlidir.

02

DEĞİŞKEN · 1-4 BAYT

UTF-8: web'in dili

UTF-8, 1992'de Ken Thompson ve Rob Pike tarafından Plan 9 işletim sistemi için tasarlandı ve 2003'te **RFC 3629** ile bugünkü hâlini aldı. En küçük birimi 8 bittir (1 bayt) ve karakterin kod noktası büyüdükçe kullanılan bayt sayısı artar:

KOD NOKTASI ARALIĞI	BAYT	BİT KALIBI	ÖRNEK
U+0000 – U+007F	1	0xxxxxxx	A → 41
U+0080 – U+07FF	2	110xxxxx 10xxxxxx	ş → C5 9F
U+0800 – U+FFFF	3	1110xxxx 10xxxxxx 10xxxxxx	€ → E2 82 AC
U+10000 – U+10FFFF	4	11110xxx 10xxxxxx ×3	😀 → F0 9F 98 80

Adım adım: "ş" nasıl C5 9F olur?

`U+015F` ikilik tabanda `101 0101 1111` (11 bit). 2 baytlık kalıba yerleşir: ilk bayta `110` + ilk 5 bit, ikinci bayta `10` + son 6 bit.

`11000101 = C5` `10011111 = 9F`

Kırmızı bitler "işaret" bitleridir: `110` "ben 2 baytlık bir karakterin başıyım" der, `10` "ben devam baytıyım" der. Bu tasarım UTF-8'e üç süper güç verir:

- **ASCII ile birebir uyumlu:** İngilizce bir metin ASCII'de ve UTF-8'de bayt bayt aynıdır. Eski yazılımlar bozulmadan çalışır.
- **Kendini senkronize eder:** Bir dosyanın ortasından okumaya başlasanız bile `10` ile başlamayan ilk baytı bulup karakter sınırına dönebilirsiniz.
- **Bayt sırası sorunu yok:** Birim 1 bayt olduğu için "büyük uçlu / küçük uçlu" (endianness) ayrımı yoktur, BOM gerekmez.

Bugün web sitelerinin %98'inden fazlası UTF-8 kullanır. HTML5 de varsayılan olarak bunu önerir: `<meta charset="UTF-8">`. Linux, macOS, JSON, Python 3 kaynak dosyaları, Go ve Rust da UTF-8 tabanlıdır.

03

DEĞİŞKEN · 2 VEYA 4 BAYT

UTF-16: Windows ve JavaScript'in içi

Unicode'un ilk yıllarında 65.536 karakterin (16 bit) her şeye yeteceği düşünülüyordu. Bu yüzden ilk kodlama **UCS-2** her karaktere sabit 2 bayt veriyordu. Windows NT, Java ve JavaScript bu dönemde tasarlandığı için içlerinde 16 bitlik karakterler kullanır. 1996'da Unicode 16 biti aşınca UCS-2 genişletilip **UTF-16** oldu.

- **BMP içindeki karakterler** (`U+0000` – `U+FFFF`): tek 16 bitlik birim, yani **2 bayt**. `A` → `00 41` , `ş` → `01 5F` .
- **BMP dışındaki karakterler** (emoji vb.): iki birimden oluşan bir **vekil çifti** (*surrogate pair*), yani **4 bayt**.

Vekil çifti (surrogate pair) nasıl hesaplanır?

Unicode, `U+D800` – `U+DFFF` aralığını hiçbir karaktere atamayıp bu iş için ayırmıştır. 😀 örneği:

```
U+1F600 - 0x10000          = 0xF600   (20 bitlik sayı)
yüksek vekil = 0xD800 + (0xF600 >> 10)   = D83D
düşük vekil  = 0xDC00 + (0xF600 & 0x3FF) = DE00
UTF-16BE baytları: D8 3D DE 00
```

> **JavaScript tuzağı:** JS dizeleri UTF-16 birimleri sayar. Bu yüzden `"😀".length` sonucu **2**'dir, 1 değil. Gerçek karakter sayısı için `[..."😀"].length` veya `Array.from()` kullanılır.

Endianness ve BOM

16 bitlik bir birim belleğe iki şekilde yazılabilir: önce büyük bayt (**UTF-16BE**, `00 41`) veya önce küçük bayt (**UTF-16LE**, `41 00`). Okuyucunun hangisi olduğunu anlaması için dosyanın başına **BOM** (Byte Order Mark, `U+FEFF`) konur: `FE FF` = BE, `FF FE` = LE. Windows çoğunlukla UTF-16LE kullanır.

04

SABİT · 4 BAYT

UTF-32: en basit, en savurgan

UTF-32 (diğer adıyla UCS-4) her kod noktasını doğrudan 32 bitlik bir sayı olarak yazar. Hesap, bölme ya da vekil çifti yoktur: `ş` → `00 00 01 5F` , 😀 → `00 01 F6 00` .

- **Artısı, sabit genişlik:** n'inci karaktere doğrudan `n × 4` baytına giderek ulaşılır (O(1) erişim). Karakter saymak bayt sayısını 4'e bölmek kadar kolaydır.
- **Eksisi, bellek israfı:** Kod noktası en fazla 21 bit gerektirir, yani her karakterde en az 11 bit boşa gider. İngilizce bir metin UTF-8'e göre **4 kat** yer kaplar.
- **Kullanım alanı:** Dosya ve ağ aktarımında neredeyse hiç kullanılmaz. Program içi işleme için tercih edilir: Linux/macOS'ta C'nin `wchar_t` tipi, Python'un `str` iç gösterimi (gerektiğinde), metin işleme kütüphaneleri.
- UTF-16 gibi endianness sorunu vardır: `UTF-32BE` / `UTF-32LE` , BOM'u `00 00 FE FF` .

Not: Sabit genişlik bile her zaman "bir kod noktası = ekranda bir harf" demek değildir. 👨‍👩‍👧 aile emojisi 5 kod noktasından oluşur (3 kişi + 2 birleştirici), UTF-32'de bile 20 bayttır.

05

YAN YANA

Üçünün karşılaştırması

KARAKTER	KOD NOKTASI	UTF-8	UTF-16 (BE)	UTF-32 (BE)
A	U+0041	41 (1)	00 41 (2)	00 00 00 41 (4)
ş	U+015F	C5 9F (2)	01 5F (2)	00 00 01 5F (4)
€	U+20AC	E2 82 AC (3)	20 AC (2)	00 00 20 AC (4)
中	U+4E2D	E4 B8 AD (3)	4E 2D (2)	00 00 4E 2D (4)
😀	U+1F600	F0 9F 98 80 (4)	D8 3D DE 00 (4)	00 01 F6 00 (4)

ÖZELLİK	UTF-8	UTF-16	UTF-32
Birim (code unit)	8 bit	16 bit	32 bit
Karakter başına bayt	1 – 4	2 veya 4	Her zaman 4
ASCII uyumu	Tam uyumlu	Yok	Yok
Endianness / BOM	Gerekmez	Gerekir (BE/LE)	Gerekir (BE/LE)
n'inci karaktere erişim	Sıralı tarama	Sıralı tarama	Doğrudan (O(1))
İngilizce metin boyutu	En küçük	2 kat	4 kat
Çince/Japonca metin	3 bayt/karakter	2 bayt/karakter (en küçük)	4 bayt/karakter
Nerede kullanılır	Web, Linux, JSON, Git	Windows API, Java, JS, .NET	Program içi işleme

Mojibake: yanlış kodlamayla okunan metin

Kodlama yanlış tahmin edildiğinde ortaya çıkan anlamsız karakterlere **mojibake** denir. Türkçede en sık gördüğümüz örnek: UTF-8 ile kaydedilmiş `ş` (`C5 9F`) Windows-1254 ile okunursa her bayt ayrı bir harf sanılır ve ekranda `ÅŸ` çıkar. "Başlık" kelimesi "BaÅŸlÄ±k" olur. Çözüm basittir: dosyayı UTF-8 kaydetmek ve HTML'de `<meta charset="UTF-8">` satırını ilk sıraya koymak.

Hangisini seçmeliyim?

- **Dosya, web sayfası, API, veritabanı:** UTF-8. Tartışmasız varsayılan budur.
- **Windows API, Java, C#, JavaScript ile çalışırken:** Dil içeride zaten UTF-16 kullanır; dışarı yazarken UTF-8'e çevirin.
- **Karakter karakter yoğun işleme yapan algoritmalar:** Bellekte geçici olarak UTF-32.

07

ÖZET & KAYNAKÇA

Kısacası

Üç kodlama da aynı Unicode kod noktalarını taşır ve birbirine kayıpsız dönüştürülebilir. **UTF-8** ASCII uyumu ve küçük boyutuyla depolama ve iletimde, **UTF-16** tarihî nedenlerle Windows/Java/JavaScript'in içinde, **UTF-32** ise sabit genişliği sayesinde bellek içi işlemede tercih edilir.

- Unicode Consortium — UTF-8, UTF-16, UTF-32 & BOM SSS — https://www.unicode.org/faq/utf_bom.html
- RFC 3629 — UTF-8, a transformation format of ISO 10646 — https://www.rfc-editor.org/rfc/rfc3629
- RFC 2781 — UTF-16, an encoding of ISO 10646 — https://www.rfc-editor.org/rfc/rfc2781
- MDN — UTF-16 karakterler, kod noktaları ve grafem kümeleri — https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String#utf-16_characters_unicode_code_points_and_grapheme_clusters
- W3Techs — Web sitelerinde karakter kodlaması kullanımı — https://w3techs.com/technologies/overview/character_encoding