What is Unicode?



www.yzokul.com.tr

What is Unicode?



You ask: What is Unicode? The short answer is that Unicode is a standard that gives every character a shared identity, so computers can store and exchange text reliably. This book explains that idea step by step, starting with why computers need numbers for letters and ending with how a single mark on screen can be more than one Unicode element. You will see examples and avoid common mistakes.



A computer cannot hold a letter directly; it can only store and process numbers. So every character, whether it is a letter, a digit, or a symbol, must be turned into a number before a machine can save it or send it. For example, to store the letter A, the computer saves a number that stands for A. This number-to-character mapping is the foundation of all text handling.



Before a common standard existed, two computers could use different number-to-letter tables. The same number might show as A on one machine and as something else on another. When text moved between systems, it could become garbled and unreadable. Without a shared rule for which number means which character, reliable exchange of text was difficult. A standard solves this by making the mapping the same everywhere.



The world has many writing systems: Latin, Turkish letters, Arabic, Cyrillic, Chinese, and many more. Each system has its own characters, and each needs to be represented. People also use punctuation, digits, currency signs, arrows, and emoji. All of these need to live together in one system so that text from any language can be stored and shared without conflict.



Unicode is a single big list where every character from every writing system gets its own unique identity. No two characters share the same identity number. This shared list means that a given character has the same identity everywhere Unicode is used. For example, the Latin capital letter A has a fixed identity, and so does the Turkish letter ş. The list covers all scripts and many symbols.



Turkish letters such as ş, ğ, ı, and İ are placed in the shared list with their own identities. They are not squeezed into other letters; each has a fixed place. For instance, the dotless ı is distinct from the dotted i, and each has its own identity number. This ensures that Turkish text is represented correctly and can be exchanged without losing meaning.



Unicode is not only about letters. Punctuation marks, digits, currency signs, arrows, and emoji also have identities in the same shared list. For example, the comma, the digit 7, the dollar sign, and the smiley emoji each have a unique identity. All of them sit together in one system, so a document can mix text and symbols seamlessly.



A character's identity is an abstract label, not the bytes stored on disk. The identity says what the character is, while the bytes say how it is written down in a file or sent over a network. For example, the identity of the letter A is a number, but the bytes used to store it depend on the encoding. Keeping identity separate from bytes is important for understanding Unicode.



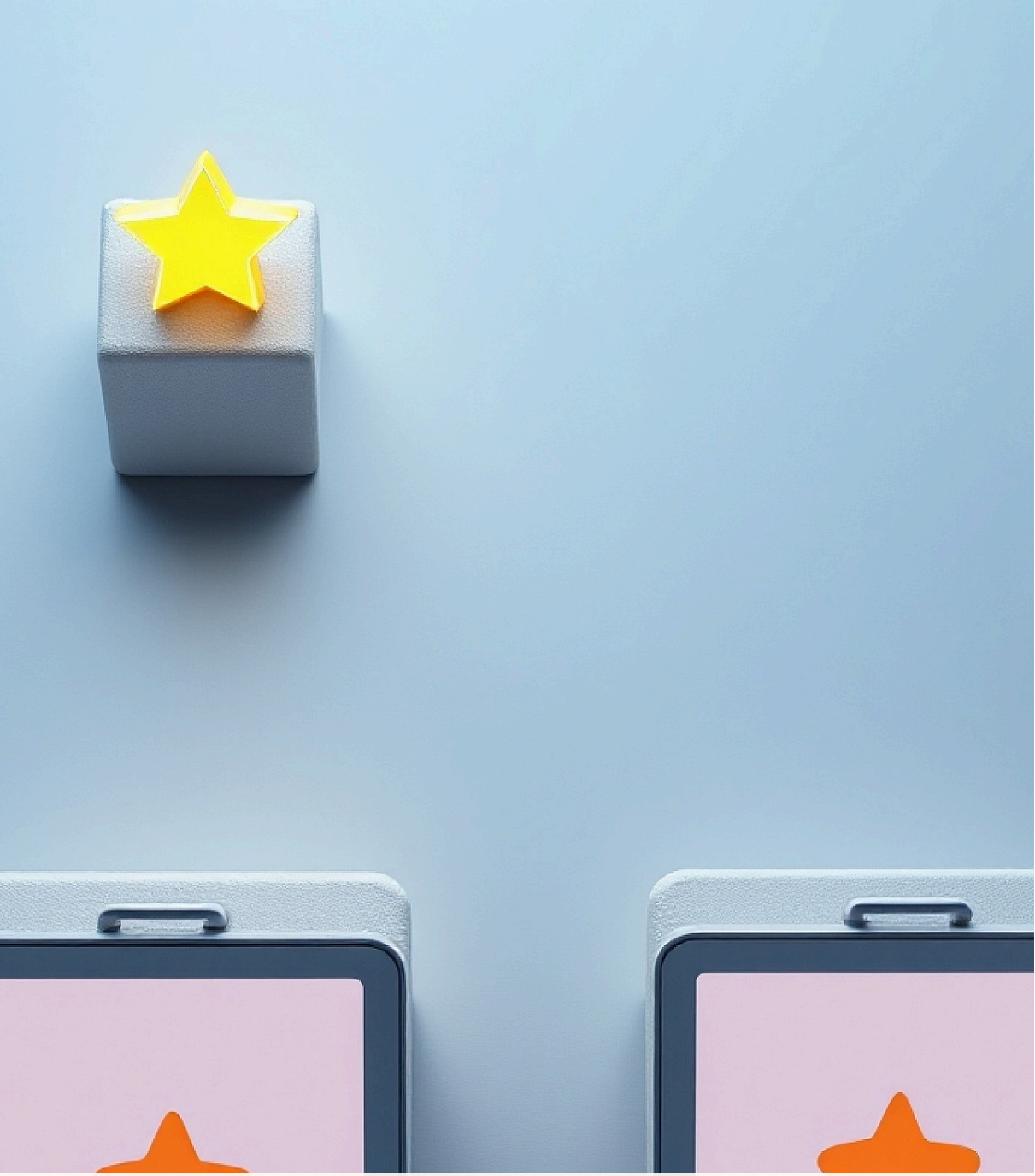
An encoding is a rule that converts a character's identity into a sequence of bytes. The identity stays the same, but the bytes depend on which encoding is used. For example, one encoding might turn the identity of A into a single byte, while another might use a different byte sequence. Encodings make it possible to store and transmit Unicode text on computers.



UTF-8 is a common encoding that turns Unicode identities into bytes. It uses one to four bytes per character, depending on the character. For example, the letter A becomes the single byte 0x41 in UTF-8. A character like ş takes two bytes. This variable length makes UTF-8 efficient for text that is mostly English while still supporting all Unicode characters.



Unicode and UTF-8 are not the same thing. Unicode is the list of identities, and UTF-8 is one way to store those identities as bytes. Confusing them is a common mistake. For example, saying a file is in Unicode is vague; saying it is in UTF-8 is precise. Keep the two ideas separate: Unicode defines characters, UTF-8 encodes them.



The same character can be stored as different byte sequences under different encodings. For example, the letter A might be one byte in UTF-8 but two bytes in UTF-16. This shows why the encoding must match on both sides when text is shared. If you send UTF-8 bytes but the receiver expects UTF-16, the text will be garbled. Always specify the encoding.



A single visible mark on screen can be built from more than one Unicode element combined. For example, the letter e followed by a combining acute accent can appear as é, a single shape. The Unicode list contains both the base letter and the accent mark as separate elements. Software combines them for display, but the underlying text has two elements.



Do not assume that every visible mark equals exactly one Unicode element. Some marks are combinations of a base character and one or more combining marks. For example, a letter with a cedilla might be two elements: the letter and the cedilla. Counting elements is not the same as counting what you see. This matters when you process text programmatically.



Do not assume that one byte always means one letter. In UTF-8, a single character can take one to several bytes. For example, the letter A is one byte, but ş is two bytes, and an emoji can be four bytes. So byte counts and letter counts differ. When you measure text size, remember that bytes are not characters.



Remember three key points: Unicode gives characters shared identities across all writing systems; encodings like UTF-8 store those identities as bytes; and the two are not the same. Also remember that one visible mark can be several Unicode elements. With this knowledge, you can understand how text is represented and avoid common mistakes when working with different languages and symbols.