Engineer

# Unicode

⌨ norme donnant un numéro propre à chaque caractère

## Ce que c’est

Unicode est la norme qui attribue un numéro propre à chaque lettre, chiffre, signe de ponctuation et symbole des systèmes d’écriture du monde. Ses responsables parlent du codage universel des caractères : il permet d’échanger, de traiter et d’afficher partout les textes des langues et des domaines techniques du monde moderne, tout en représentant aussi des écritures anciennes et des textes historiques. Un même répertoire réunit le å danois, le ґ ukrainien, le signe euro et les lettres d’écritures que personne n’utilise depuis des siècles : un même document ou une même colonne de base de données peut donc tous les contenir.

À proprement parler, une version d’Unicode comprend plusieurs parties coordonnées : la spécification centrale, les tableaux de caractères, les annexes et une base de données des caractères, des fichiers lisibles par machine qui consignent les propriétés de chaque caractère. Le Comité technique Unicode la maintient en continu, le Consortium Unicode la publie et la version 18.0 a été annoncée le 16 septembre 2026.

Le numéro attribué à un caractère est son point de code. L’espace de codes va de 0 à 10FFFF en hexadécimal, soit 1 114 112 points de code au total ; un point de code s’écrit U+ suivi d’au moins quatre chiffres hexadécimaux : å est U+00E5, ґ est U+0490 et le signe euro est U+20AC. La politique de stabilité du Consortium garantit qu’un caractère codé conserve définitivement sa place : il n’est jamais déplacé ni retiré, et son nom ne change jamais ; un texte codé aujourd’hui garde donc le même sens dans toutes les versions ultérieures.

Deux distinctions sont au cœur de la norme. La première sépare le caractère du glyphe. Unicode code le caractère, la lettre abstraite, et laisse sa forme à la police : la spécification centrale indique que la forme des glyphes relève des

fournisseurs de polices et ne fait pas partie de la norme. Le même point de code peut avoir deux apparences dans deux polices, et une police qui ne possède aucune forme pour lui ne peut pas le dessiner. La seconde distinction sépare le point de code des octets qui le stockent. Les trois formes de codage Unicode sont UTF-8, UTF-16 et UTF-32 ; elles représentent chaque point de code par une ou plusieurs unités de 8, 16 ou 32 bits. La spécification qualifie de piège la tentative d'opposer « UTF-8 à Unicode » : les trois sont des façons également valides de représenter les mêmes caractères.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce qu'Unicode ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Un logiciel qui se lit naturellement sur chaque marché où vous vendez, y compris les dates, les devises et le classement des noms.

**Découvrir nos prestations** Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. Système i18n anglais/danois — 12,027 messages par langue, en anglais et en danois

**Système i18n anglais/danois** Contactez-nous →

## Pourquoi c'est important

Avant Unicode, chaque écriture avait ses pages de codes, et un octet ne signifiait rien sans savoir quelle page était utilisée. Dans la page ouest-européenne windows-1252, l'octet E5 est å ; dans la page cyrillique windows-1251, ce même octet est la lettre е. Aucune ne pouvait contenir les lettres de l'autre : windows-1251 contient Ґ, mais pas å, æ ni ø ; windows-1252 fait l'inverse. Un produit en danois et en ukrainien exigeait deux codages et une indication fiable du codage de chaque texte ; une phrase citant les deux langues ne tenait dans aucun des deux.

Unicode a supprimé cette gestion. Toutes les langues publiées par un produit et tous les noms saisis dans un formulaire par un client tiennent dans un seul jeu de caractères ; le même numéro désigne le même caractère sur tous les systèmes qui le lisent. Une base de données, une API et un navigateur peuvent ainsi échanger du texte sans convenir d'abord d'une page de codes.

La norme tranche aussi des questions que chaque programme réglerait autrement à sa façon. Elle unifie les caractères entre les langues : la virgule est un même caractère, qu'elle sépare les milliers, comme en anglais, ou les décimales, comme en français ; le sens vient donc de la locale et non du point de code. Elle n'unifie pas les écritures : le o latin, le о cyrillique et le ο grec se ressemblent dans

la plupart des polices, mais sont trois caractères, parce qu'ils appartiennent à trois écritures et que les anciens codages les distinguaient déjà. Elle consigne aussi les séquences équivalentes. Å peut être stocké comme un caractère précomposé ou comme A suivi d'un diacritique rond en chef ; la norme définit les deux comme canoniquement équivalents, à afficher et à interpréter de la même manière, et propose des formes de normalisation qui les ramènent à une représentation convenue.

Cette généralité a un prix, payé par le programmeur. Un caractère n'est pas un octet, pas toujours une seule unité d'une chaîne, et pas toujours ce qu'un lecteur compte comme une lettre. Le code qui confond ces trois notions réussit ses tests sur des données anglaises et échoue sur les noms, adresses et messages des personnes auxquelles il est destiné.

## À quoi cela ressemble en pratique

JavaScript rend ces couches visibles. Une chaîne est une suite d'unités de code UTF-16 ; la décomposer dans un tableau la sépare en points de code :

```
const hex = (n) => n.toString(16).toUpperCase().padStart(4, "0");
for (const text of ["a", "å", "Ґ", "€", "𐌂"]) {
  const points = [...text].map((c) => "U+" + hex(c.codePointAt(0)));
  console.log(text, points.join(" "), text.length, [...text].length);
}
```

Exécuté avec Node.js 25, ce code affiche chaque caractère, son point de code, sa longueur en unités UTF-16 et sa longueur en points de code :

```
a U+0061 1 1
å U+00E5 1 1
Ґ U+0490 1 1
€ U+20AC 1 1
𐌂 U+10302 2 1
```

La dernière ligne est une lettre du vieil italique. Son point de code est au-delà de U+FFFF : UTF-16 la stocke donc en deux unités, une paire de substituts, et `length` annonce 2 pour un seul caractère.

L’équivalence se voit de la même manière. Voici trois chaînes qui s’affichent toutes comme Å :

```
const forms = ["\u00C5", "A\u030A", "\u212B"];
console.log("form length equal NFC");
for (const [i, s] of forms.entries()) {
  console.log(i + 1, s.length, s === forms[0], s.normalize("NFC") === forms[0]);
}
```

```
form length equal NFC
1 1 true true
2 2 false true
3 1 false true
```

La première est la lettre précomposée, U+00C5. La deuxième est un simple A suivi de U+030A, le diacritique rond en chef. La troisième est U+212B, le symbole angström, que la base de données des caractères Unicode fait correspondre à U+00C5. Comparées telles quelles, ce sont trois chaînes différentes ; après normalisation en NFC, elles sont identiques. Rien à l’écran ne les distingue : une recherche, un index unique ou une connexion qui compare les chaînes brutes voit donc trois noms différents.

Ce site dépend des points de code à un endroit moins visible. La note sur le soulignement qui traversait chaque lettre décrit les six variantes latines et cyrilliques de la police Ubuntu utilisées pour le texte courant : trois graisses pour le latin et trois pour le cyrillique, afin qu’une page ukrainienne utilise la même police qu’une page anglaise. Chaque variante déclare la plage de caractères qu’elle couvre. Ces plages guident le choix de la police pour les caractères de la page ; un préchargement explicite peut aussi récupérer un fichier avant la découverte normale des polices. La déclaration cyrillique donne sa plage en points de code :

```
unicode-range: U+0400-045F, U+0490-0491;
```

La première plage va de È à џ et contient les caractères ukrainiens Є, І et Ї ainsi que le reste de l’alphabet cyrillique de base. La seconde ne contient que Ґ et ґ, situés plus loin dans le bloc cyrillique. Les plages déclarent quelle variante de police peut être choisie pour un caractère, mais la police doit aussi contenir son

glyphe. Elles ne garantissent pas qu'une page ne récupère que des fichiers pour une seule écriture. Les textes anglais et danois utilisent les variantes latines, mais le libellé de la langue ukrainienne peut nécessiter une variante cyrillique ; le site précharge aussi explicitement la variante cyrillique légère du texte courant sur les pages non cyrilliques pour l'affichage à l'écran.

## Où les problèmes surviennent

**La longueur est comptée en unités de code.** Le `length` de JavaScript compte les unités UTF-16 : 𐐷 en compte donc deux, et un A avec un diacritique rond en chef en compte deux là où le lecteur voit une lettre. Une limite exprimée en unités refuse du texte qui tient, et une coupure à une position fixe peut scinder un caractère : `"A\u030A".slice(0, 1)` garde le A et perd le rond ; couper 𐐷 en deux laisse un substitut isolé que l'encodeur UTF-8 remplace par U+FFFD, le caractère de remplacement. Décidez ce que compte chaque limite : des octets lorsque le stockage l'impose, des caractères tels que les voit le lecteur lorsque la limite est destinée à une personne. `Intl.Segmenter` compte ces derniers : pour Århus écrit avec un diacritique rond en chef, il renvoie 5 là où `length` renvoie 6. Ne coupez le texte qu'aux frontières qu'il indique.

**Des textes égaux sont comparés unité par unité.** Les trois formes de Å ci-dessus sont trois chaînes différentes pour `===`, pour un hachage et pour un index unique : un client peut donc inscrire deux fois le même nom ou ne pas retrouver un document sous le titre qu'il a saisi. Normalisez le texte en NFC à son entrée dans le système, stockez-le normalisé et comparez les formes normalisées. La politique de stabilité garantit qu'une chaîne normalisée avec une version le reste dans toutes les versions ultérieures, à condition qu'elle ne contienne que des caractères connus de cette version : le code de normalisation doit donc être aussi récent que le texte.

**Des caractères ressemblants passent pour identiques.** Le o latin, le o cyrillique et le o grec sont des points de code différents que la plupart des polices dessinent de la même manière : deux noms de compte ou deux codes de produit peuvent donc sembler identiques tout en différant d'un caractère. Unicode les distingue volontairement, et sa norme sur les mécanismes de sécurité, UTS #39, explique comment limiter ces usurpations. Lorsqu'un nom identifie une personne ou un bénéficiaire, autorisez une seule écriture par nom ou vérifiez-le avec les données de sécurité de la norme avant de l'accepter.

**Les octets sont lus avec le mauvais codage.** Un texte stocké en UTF-8 et lu en windows-1252 transforme blåbær en blÃ¥bÃ¦r, car chaque lettre de deux octets est lue comme deux lettres d'un octet. Le mot Unicode ne dit pas comment les octets ont été écrits ; seule la forme de codage l'indique. Déclarez-la à chaque frontière franchie par le texte : dans l'en-tête HTTP, dans la page, sur la connexion à la base de données et dans le format du fichier. Traitez un fichier qui arrive sans cette indication comme une question pour une personne plutôt qu'une devinette pour le code.

**Un glyphe manquant est imputé au texte.** Lorsqu'une lettre apparaît comme une case vide, la cause habituelle est une police sans forme pour ce point de code ; le texte lui-même est intact. Unicode laissant les formes aux polices, chaque police utilisée par un produit doit couvrir tous les caractères nécessaires à ses langues, Є, I, Ї et Ґ comme A à я ; sinon, le navigateur les dessine avec la police de remplacement qu'il trouve. La note citée explique pourquoi le site sert du cyrillique : sans lui, une page ukrainienne utiliserait ce que propose le système. Testez chaque langue avec tout son alphabet plutôt qu'une phrase d'exemple.

**Les noms sont vérifiés uniquement de A à Z.** Une expression comme `^[A-Za-z' -]+$` accepte O'Neill et rejette Søren Ågård, Zoë et Ґалаґан, précisément les noms qu'un produit danois ou ukrainien doit prendre en charge. Vérifiez plutôt les lettres par leurs propriétés Unicode : en JavaScript, `/^[\p{L}\p{M}' -]+$/u` accepte les quatre. Ces propriétés viennent de la copie de la base de données des caractères Unicode intégrée au moteur d'exécution. Par exemple, le moteur Node.js 25.2.1 utilisé pour ces exemples annonce Unicode 16.0 ; il s'agit d'une compilation précise, sans garantie pour toutes les versions 25.x. Les lettres ajoutées dans des versions Unicode ultérieures échouent au test jusqu'à la mise à jour des données de caractères du moteur. Validez les noms avec souplesse et laissez une personne plutôt qu'une expression décider des cas difficiles.

## Pour aller plus loin

- À propos du standard Unicode — la description de la norme par le Consortium, les éléments d'une version et l'endroit où la plus récente est publiée.
- Le standard Unicode, chapitre 2 : structure générale — les dix principes de conception, l'espace de codes, les trois formes de codage et les règles d'équivalence des séquences.
- Politiques de stabilité du codage des caractères Unicode — ce qui ne peut jamais changer une fois un caractère codé, de son point de code et de son nom à sa normalisation.

- Unicode 18.0.0 — les ajouts de la version actuelle et les normes techniques versionnées avec elle, dont le classement et la sécurité.
- Node.js : process.versions — le relevé des versions des composants du moteur, utilisé ici pour rattacher l'exemple des données Unicode à une compilation de Node.js plutôt qu'à une lignée entière de versions.
- CSS Fonts : plages de caractères — comment les plages déclarées et la couverture réelle des glyphes de la police déterminent ensemble quels caractères une variante de police peut afficher.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le processus de travail, les interfaces existantes et les utilisateurs qui ont besoin du changement. Nous pouvons définir une livraison testable et une transmission permettant à votre équipe d'assurer la maintenance des éléments livrés.

**Consulter les tarifs** Contactez-nous →

## Rendez la terminologie concrète

Demandez comment « Qu'est-ce qu'Unicode ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

**Découvrir nos prestations**

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

**Créé avec soin et amour par D.**

**CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N**

**Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.**

**Imprimé depuis https://engineer.company/fr/glossary/unicode/**