Représentation des textes

De l'ASCII à l'Unicode

Résumé

Un ordinateur ne manipule que des nombres binaires. Pour représenter du texte, chaque caractère doit être associé à un nombre selon une convention appelée **encodage**. Ce cours retrace l'évolution des encodages, d'ASCII à Unicode et UTF-8, et montre comment Python manipule les caractères.

1 Le problème de la représentation

Un octet (8 bits) permet de stocker un nombre entre 0 et 255. Comment stocker la lettre « A » ? Il faut se mettre d'accord sur une **table de correspondance** entre les caractères et les nombres.

Exercice 1 *Découverte du problème*

1. Si l'on associe les lettres de l'alphabet aux nombres 1 à 26, quel nombre correspond à « C » ? Et quel caractère correspond au nombre 8 ?

2. Avec cette convention, combien de caractères différents peut-on représenter ? Est-ce suffisant pour un texte français ?

3. Que se passe-t-il si deux ordinateurs utilisent des conventions différentes ?

2 La table ASCII

Définition 1 **(ASCII)** L'**ASCII** (*American Standard Code for Information Interchange*), créé en 1963, est le premier encodage standardisé. Il utilise **7 bits** pour coder **128 caractères** (numérotés de 0 à 127).

Les 128 caractères de l'ASCII comprennent :

— les chiffres `0` à `9` (codes 48 à 57) ;

— les lettres majuscules `A` à `Z` (codes 65 à 90) ;

— les lettres minuscules `a` à `z` (codes 97 à 122) ;

— des signes de ponctuation et des caractères spéciaux ;

— 33 caractères de contrôle (codes 0 à 31 et 127), non affichables.

Extrait de la table :

Code	48	65	66	90	97	98	122
Caractère	0	A	B	Z	a	b	z

Exercice 2 *Explorer la table ASCII*

1. Quelle est la différence entre le code ASCII de « A » et celui de « a » ?

2. En déduire le code ASCII de « d » sachant que celui de « D » est 68.

3. Les codes des lettres majuscules se suivent (65, 66, 67…). Quel est le code de « M » ? Expliquez votre raisonnement.

4. Pourquoi l'ASCII est-il insuffisant pour écrire en français ? Donnez au moins trois exemples de caractères absents.

③ Les extensions : ISO 8859-1

Pour pallier les limites de l'ASCII, la norme **ISO 8859-1** (aussi appelée **Latin-1**) utilise **8 bits** pour coder **256 caractères**. Les 128 premiers sont identiques à l'ASCII (rétrocompatibilité). Les 128 suivants ajoutent les caractères accentués des langues d'Europe de l'Ouest : é, è, ê, à, ç, ñ, ü, etc.

Exercice 3 *ASCII étendu*

1. Avec 7 bits, on code 128 caractères. Avec 8 bits, combien en code-t-on ? Justifiez.

2. Pourquoi avoir gardé les mêmes codes que l'ASCII pour les 128 premiers caractères ?

3. L'ISO 8859-1 permet-il d'écrire en chinois, en arabe ou en japonais ? Pourquoi ?

④ Unicode

Définition 2 **(Unicode)** **Unicode** est une norme universelle qui attribue un numéro unique (appelé **point de code**) à chaque caractère de toutes les écritures du monde. Le point de code est noté `U+xxxx` en hexadécimal.

La version actuelle d'Unicode définit plus de 150 000 caractères, couvrant :

— toutes les écritures vivantes (latin, arabe, chinois, japonais, coréen, cyrillique, hébreu, etc.) ;

— les écritures historiques (hiéroglyphes, cunéiforme, etc.) ;

— les symboles mathématiques, musicaux, techniques ;

— les emojis.

Exemples de points de code :

Caractère	Point de code	Valeur décimale
A	U+0041	65
é	U+00E9	233
π	U+03C0	960

Remarque 1. Unicode n'est **pas** un encodage : c'est une table de correspondance entre caractères et numéros. L'encodage (la façon de stocker ces numéros en octets) est une question distincte, résolue par UTF-8, UTF-16, etc.

⑤ UTF-8 : l'encodage dominant

Définition 3 **(UTF-8)** **UTF-8** (*Unicode Transformation Format – 8 bits*) est un encodage **à longueur variable** qui utilise de 1 à 4 octets par caractère. Plus de 98 % des pages web l'utilisent.

Octets	Plage de codes	Caractères concernés
1	U+0000 à U+007F	ASCII (a-z, A-Z, 0-9, ponctuation)
2	U+0080 à U+07FF	Accents, cyrillique, arabe, hébreu…
3	U+0800 à U+FFFF	Chinois, japonais, coréen…
4	U+10000 à U+10FFFF	Emojis, écritures rares…

Avantages d'UTF-8 :

— **rétrocompatible** avec l'ASCII : un texte ASCII pur est identique en UTF-8 ;

— **économe** : les caractères courants (anglais) n'occupent qu'un octet ;

— **universel** : il peut représenter tous les caractères Unicode.

Le mécanisme

Les premiers bits de chaque octet indiquent son rôle : un octet qui commence par `0` est un caractère ASCII à lui seul ; un octet qui commence par `110`, `1110` ou `11110` ouvre une séquence de 2, 3 ou 4 octets ; les octets suivants de la séquence commencent tous par `10`. Les bits restants (marqués `x`) contiennent le point de code : `0xxxxxxx`, `110xxxxx 10xxxxxx`, `1110xxxx 10xxxxxx 10xxxxxx`, `11110xxx 10xxxxxx 10xxxxxx 10xxxxxx`.

Exemple 1 (Le caractère « é »). Son point de code U+00E9 vaut 233, soit `11101001` en binaire : il faut 2 octets. On répartit ses 11 bits (`000 1110 1001`, complétés par des zéros à gauche) dans les cases `x` de la forme `110xxxxx 10xxxxxx` :

```
110 00011   10 101001
  C   3       A   9
```

« é » s'écrit donc `C3 A9` en UTF-8, contre le seul octet `E9` en ISO 8859-1. Un emoji (U+1F600) occupe quatre octets : `F0 9F 98 80`.

Remarque 2 (Un texte mal décodé). Si un fichier écrit en UTF-8 est lu comme de l'ISO 8859-1, les deux octets `C3 A9` sont interprétés comme deux caractères, « Ã » et « © » : « été » s'affiche « Ã©tÃ© ». À l'inverse, un fichier ISO 8859-1 lu comme de l'UTF-8 produit des caractères de remplacement. Ces symptômes signalent toujours un décalage entre l'encodage d'écriture et l'encodage de lecture.

Exercice 4 *Taille d'un texte en UTF-8*

1. Combien d'octets occupe le mot `"Hello"` en UTF-8 ? Justifiez.

2. Combien d'octets occupe le mot `"café"` en UTF-8, sachant que « é » est codé sur 2 octets ?

3. Un texte contient 100 caractères ASCII et 20 caractères accentués. Quelle est sa taille en octets en UTF-8 ?

4. Pourquoi la taille en octets d'un texte UTF-8 ne correspond-elle pas toujours au nombre de caractères ?

6 En Python : `ord()` et `chr()`

Python offre deux fonctions pour passer d'un caractère à son point de code Unicode et inversement :

Fonction	Syntaxe	Description
`ord()`	`ord("A")` → `65`	Renvoie le point de code Unicode du caractère
`chr()`	`chr(65)` → `"A"`	Renvoie le caractère correspondant au point de code

```python
print(ord("A"))    # 65
print(ord("é"))    # 233
print(chr(65))     # A
print(chr(233))    # é
print(chr(960))    # π
```

`encode()` et `decode()`

La méthode `encode` d'une chaîne donne ses octets dans l'encodage choisi, et la méthode `decode` d'une suite d'octets reconstitue la chaîne :

```python
print("é".encode("utf-8"))      # b'\xc3\xa9'
print("é".encode("latin-1"))    # b'\xe9'
print(f'{len("été".encode("utf-8"))} {len("été".encode("latin-1"))}')  # 5 3
print(b"\xc3\xa9".decode("utf-8"))                # é
print("été".encode("utf-8").decode("latin-1"))    # Ã©tÃ©
```

Python affiche une suite d'octets entre `b'` et `'` : la séquence `\xc3` désigne un octet écrit en hexadécimal (ici l'octet `C3`), tandis qu'un octet qui correspond à un caractère ASCII affichable s'affiche directement sous la forme de ce caractère. Ainsi `b'\xc3\xa9'` est la suite des deux octets `C3 A9`.

Pour lire ou écrire un fichier texte, on précise toujours l'encodage ; convertir un fichier d'un encodage à l'autre tient en quelques lignes :

```python
with open("texte-latin1.txt", encoding="latin-1") as source:
    contenu = source.read()

with open("texte-utf8.txt", "w", encoding="utf-8") as destination:
    destination.write(contenu)
```

Exercice 5 *Manipuler ord() et chr()*

1. Donnez la valeur de `ord("Z")` et de `ord("z")`.

2. Écrivez une expression Python qui transforme une lettre minuscule en majuscule en utilisant `ord()` et `chr()`, sans utiliser `.upper()`.

3. Écrivez un programme qui affiche les 26 lettres majuscules et leur code ASCII en utilisant une boucle et `chr()`.

4. Écrivez une fonction `cesar(texte, decalage)` qui chiffre un texte en décalant chaque lettre majuscule de `decalage` positions dans l'alphabet (chiffre de César).

   Exemple : `cesar("ABC", 3)` renvoie `"DEF"`.

Synthèse

Encodage	Bits	Caractères	Particularités
ASCII	7	128	Caractères anglais uniquement
ISO 8859-1	8	256	Accents d'Europe de l'Ouest ; rétrocompatible ASCII
Unicode	–	150 000+	Table universelle de correspondance (pas un encodage)
UTF-8	8 à 32	Tous	Longueur variable ; rétrocompatible ASCII ; dominant sur le web

En Python :

— `ord(c)` renvoie le point de code Unicode du caractère `c` ;

— `chr(n)` renvoie le caractère correspondant au point de code `n` ;

— les chaînes Python 3 sont nativement en Unicode ;

— `chaine.encode("utf-8")` et `octets.decode("utf-8")` passent des caractères aux octets et réciproquement ; `open(..., encoding="utf-8")` précise l'encodage d'un fichier.