Textes : encodage et recherche

Activité – Première NSI

1 Le code ASCII

Définition 1 **(Code ASCII)** Le code **ASCII** (*American Standard Code for Information Interchange*) associe à chaque caractère un nombre entier compris entre 0 et 127, codé sur 7 bits. Il permet de représenter les lettres latines non accentuées, les chiffres, les signes de ponctuation et quelques caractères de contrôle.

Table ASCII complète (codes 0 à 127). Chaque code est composé de 7 bits. Pour lire la table, on combine l'en-tête de colonne (3 bits de poids fort) et l'en-tête de ligne (4 bits de poids faible). Par exemple, `A` se trouve à la colonne `100` et à la ligne `0001`, soit $1000001_2 = 41_{16} = 65_{10}$.

		000	001	010	011	100	101	110	111
		0–	**1–**	**2–**	**3–**	**4–**	**5–**	**6–**	**7–**
0000	**–0**	NUL	DLE	SP	0	@	P	`	p
0001	**–1**	SOH	DC1	!	1	A	Q	a	q
0010	**–2**	STX	DC2	"	2	B	R	b	r
0011	**–3**	ETX	DC3	#	3	C	S	c	s
0100	**–4**	EOT	DC4	$	4	D	T	d	t
0101	**–5**	ENQ	NAK	%	5	E	U	e	u
0110	**–6**	ACK	SYN	&	6	F	V	f	v
0111	**–7**	BEL	ETB	'	7	G	W	g	w
1000	**–8**	BS	CAN	(	8	H	X	h	x
1001	**–9**	HT	EM	)	9	I	Y	i	y
1010	**–A**	LF	SUB	*	:	J	Z	j	z
1011	**–B**	VT	ESC	+	;	K	[	k	{
1100	**–C**	FF	FS	,	<	L	\	l	|
1101	**–D**	CR	GS	-	=	M	]	m	}
1110	**–E**	SO	RS	.	>	N	^	n	~
1111	**–F**	SI	US	/	?	O	_	o	DEL

Remarque 1. Les deux premières colonnes (codes 0 à 31) sont des **caractères de contrôle** non imprimables (retour à la ligne LF, tabulation HT, etc.). La colonne `010` commence par l'espace (SP). Le code 127 (DEL) est aussi un caractère de contrôle. On remarque que les majuscules et les minuscules ne diffèrent que par le bit 6 : `A` = $100\,0001_2$ et `a` = $110\,0001_2$.

Exercice 1 *Lire et comprendre la table ASCII*

1. En utilisant la table, donnez le code ASCII en décimal, en hexadécimal et en binaire de : `A`, `Z`, `a`, `z`, `0`, `9`.

2. Quel caractère correspond au code $4E_{16}$? Au code $0111\,0100_2$?
3. Quelle est la différence entre le code d'une lettre majuscule et celui de la même lettre en minuscule ? Quel bit change ?
4. Les chiffres `'0'` à `'9'` ont les codes 48 à 57. Quelle relation y a-t-il entre la valeur d'un chiffre et les 4 bits de poids faible de son code ?
5. En Python, la fonction `ord` renvoie le code ASCII d'un caractère et la fonction `chr` renvoie le caractère associé à un code. Que renvoient les expressions suivantes ?

   a) `ord("N")` b) `chr(33)` c) `ord("a") - ord("A")`

6. Combien de caractères différents peut-on représenter avec 7 bits ?
7. Pourquoi le code ASCII est-il insuffisant pour les textes en français ?

② Unicode et UTF-8

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 : latin, arabe, chinois, emojis, etc. On note un point de code `U+` suivi de sa valeur en hexadécimal.

Définition 3 **(UTF-8)** **UTF-8** est un encodage d'Unicode qui utilise un nombre variable d'octets :

— 1 octet pour les caractères ASCII (codes 0 à 127), compatibles avec ASCII ;

— 2 octets pour les caractères latins étendus (accents, etc.) ;

— 3 octets pour la plupart des écritures (chinois, arabe, etc.) ;

— 4 octets pour les emojis et caractères rares.

Exercice 2 *Comprendre UTF-8*

1. Le caractère `A` a pour point de code `U+0041` et le caractère `é` a pour point de code `U+00E9`. Combien d'octets faut-il pour encoder chacun d'eux en UTF-8 ?
2. Le mot « café » contient quatre caractères. Combien d'octets occupe-t-il en UTF-8 ?
3. Le mot « hello » contient cinq caractères. Combien d'octets occupe-t-il en UTF-8 ?
4. Pourquoi dit-on que UTF-8 est « rétrocompatible » avec ASCII ?
5. Un emoji comme « `U+1F600` » est encodé sur 4 octets. Quel est l'inconvénient d'utiliser beaucoup d'emojis dans un message ?

③ Manipuler les chaînes en Python

Propriété 1 **(Opérations sur les chaînes)** En Python, une chaîne de caractères est une séquence indexée à partir de 0. On peut accéder à un caractère par son indice (`mot[i]`), obtenir la longueur (`len(mot)`), et extraire une sous-chaîne (`mot[i:j]`).

Exercice 3 *Chaînes et indices*

Soit la chaîne `texte = "INFORMATIQUE"`.

1. Que vaut `len(texte)` ?
2. Que vaut `texte[0]` ? `texte[5]` ? `texte[-1]` ?
3. Que vaut `texte[2:6]` ?
4. Que renvoie le code suivant ?

```
resultat = ""
for c in "PYTHON":
    resultat = resultat + c + "-"
print(resultat)
```

5. Écrire une expression Python qui extrait les trois derniers caractères de `texte`.

4 Recherche d'un motif dans un texte

Exercice 4 *Recherche à la main*

On cherche le motif `"NSI"` dans le texte `"UN SI BEAU COURS DE NSI"`.

1. En commençant par la position 0, comparez le motif caractère par caractère avec le texte. Y a-t-il correspondance ?
2. Décalez le motif d'une position vers la droite (position 1). Comparez à nouveau. Y a-t-il correspondance ?
3. Continuez ainsi. À quelle position trouvez-vous le motif ?
4. Combien de positions avez-vous testées au total ?
5. Jusqu'à quelle position est-il nécessaire de tester ? Exprimer cette dernière position en fonction de la longueur du texte n et de la longueur du motif m.

Exercice 5 *Algorithme de recherche naïve*

La recherche naïve consiste à tester toutes les positions possibles du motif dans le texte. Pour chaque position, on compare le motif caractère par caractère.

1. Complétez la fonction Python suivante :

```
def recherche_naive(texte, motif):
    n = len(texte)
    m = len(motif)
    positions = []
    for i in range(..............):
        j = 0
        while j < m and texte[..........] == motif[..........]:
            j = j + 1
        if j == ..........:
            positions.append(i)
    return positions
```

2. Testez votre fonction à la main sur le texte `"ABCABC"` et le motif `"ABC"`. Quelles positions sont renvoyées ?
3. Testez sur le texte `"AAAA"` et le motif `"AA"`. Quelles positions sont renvoyées ?

⑤ Complexité de la recherche naïve

Exercice 6 *Compter les comparaisons*

1. **Meilleur cas.** On cherche le motif `"Z"` dans le texte `"ABCDEFGH"` (8 caractères). Combien de comparaisons sont effectuées ?
2. Généraliser : si le motif est de longueur $m = 1$ et qu'il n'apparaît pas dans un texte de longueur n, combien de comparaisons au total ?
3. **Pire cas.** On cherche le motif `"AAB"` dans le texte `"AAAAAAAAB"` (9 caractères). Complétez le tableau de trace :

Position i	Comparaisons	Résultat
0		
1		
2		
3		
4		
5		
6		

4. Dans le pire cas, pour un texte de longueur n et un motif de longueur m, combien de comparaisons sont effectuées au maximum ? En déduire la complexité.
5. Pourquoi dit-on que cet algorithme est « naïf » ?

⑥ Ce qu'il faut retenir

Propriété 2 **(L'essentiel sur les textes)**

— Le code ASCII encode 128 caractères sur 7 bits. Il est insuffisant pour les langues autres que l'anglais.

— Unicode attribue un numéro unique à chaque caractère. UTF-8 encode ces numéros sur 1 à 4 octets.

— La recherche naïve d'un motif de longueur m dans un texte de longueur n a une complexité en $O(n \times m)$ dans le pire cas.

Remarque 2. En pratique, la recherche naïve est souvent efficace car le pire cas (toutes les comparaisons avant échec) est rare sur des textes naturels. Python utilise en interne des algorithmes optimisés pour la méthode `str.find`.