CHAPITRE

5 Les chaînes de caractères

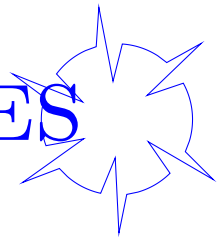

Table des matières

Nous avons vu comment ***coder de l'information*** à partir de deux bits. Un octet peut par exemple, coder un nombre entier, positif ou négatif mais ce même octet peut aussi coder une couleur. Comment l'information ***alphabétique*** est-elle codée, stockée ? C'est en partie la réponse que va donner ce modeste cours sur les chaînes de caractères...

1 Encodage des caractères

Au commencement, chaque caractère était identifié par un code unique qui est un entier naturel et la correspondance entre le caractère et son code était appelée un Charset. Le code n'étant pas utilisable tel quel par un ordinateur qui ne comprend que le binaire, il fallut donc représenter les codes par des octets, et cela fut appelé Encoding.

1.1 La table ASCII

Dans de nombreux grimoires anciens on découvre le code ASCII qui était utilisé pour représenter du texte en informatique.

ASCII signifiait *American Standard Code for Information Interchange*. Il paraît que ce code est toujours en usage...

Exercice n°1
Pour représenter les lettres minuscules, majuscules et les chiffres, combien de bits suffiraient ?

Le code ASCII se base sur un tableau contenant les caractères les plus utilisés en langue anglaise : les lettres de l'alphabet en majuscule (de A à Z) et en minuscule (de a à z), les dix chiffres arabes (de 0 à 9), des signes de ponctuation (point, virgule, point-virgule, deux points, points d'exclamation et d'interrogation, apostrophe ou quote, guillemet ou double quotes, parenthèses, crochets etc.), quelques symboles et certains caractères spéciaux invisibles (espace, retour-chariot, tabulation, retour-arrière, etc.). Les créateurs de ce code limitèrent le nombre de ses caractères à 128, pour qu'ils puissent être codés avec seulement 7 bits : les ordinateurs utilisaient des cases mémoires de un octet, mais ils réservaient toujours le huitième bit pour le contrôle de parité (c'est une sécurité pour éviter les erreurs, qui étaient très fréquentes dans les premières mémoires électroniques).

Exercice n°2

① À l'aide de la table ASCII ci-dessous, coder en binaire la phrase suivante : « L'an qui vient ! » ;
② Quelle est cette exclamation :
01000010 01110010 01100001 01110110 01101111 00101100 ?
③ Peut-on coder la phrase suivante : « Un âne est-il passé par là ? »

ASCII control characters		
00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters					
32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Il a donc fallu étendre la table ASCII pour pouvoir coder les nouveaux caractères. Les mémoires devenant plus fiables et, de nouvelles méthodes plus sûres que le contrôle de parité ayant été inventées, le huitième bit a pu être utilisé pour coder plus de caractères.

Exercice n°3
Avec un huitième bit, combien de caractères nouveaux peut-on coder ?

Le fait d'utiliser un bit supplémentaire a bien entendu ouvert des possibilités mais malheureusement tous les caractères ne pouvaient être pris en charge.

La norme ISO 8859-1 appelée aussi Latin-1 ou Europe occidentale est la première partie d'une norme plus complète appelée ISO 8859 (qui comprend 16 parties) et qui permet de coder tous les caractères des langues européennes.

Cette norme ISO 8859-1 permet de coder 191 caractères de l'alphabet latin qui avaient à l'époque été jugés essentiels dans l'écriture, mais omet quelques caractères fort utiles (ainsi, la ligature n'y figure pas).Dans les pays occidentaux, cette norme est utilisée par de nombreux systèmes d'exploitation, dont Linux et Windows. Elle a donné lieu à quelques extensions et adaptations, dont Windows-12527 (appelée ANSI) et ISO 8859-158 (qui prend en compte le symbole € créé après la norme ISO 8859-1). C'est source de grande confusion pour les développeurs de programmes informatiques car un même caractère peut être codé différemment suivant la norme utilisée.

1.2 Et survint Unicode...

La globalisation des échanges culturels et économiques a mis l'accent sur le fait que les langues européennes coexistent avec de nombreuses autres langues aux alphabets spécifiques voire sans alphabet. La généralisation de l'utilisation d'Internet dans le monde a ainsi nécessité une prise en compte d'un nombre beaucoup plus important de caractères (à titre d'exemple, le mandarin possède plus de 5000 caractères !). Une autre motivation pour cette évolution résidait dans les possibles confusions dues au trop faible nombre de caractères pris en compte ; ainsi,les symboles monétaires des différents pays n'étaient pas tous représentés dans le système ISO 8859-1, de sorte que les ordres de paiement

internationaux transmis par courrier électronique risquaient d'être mal compris. La norme Unicode a donc été créée pour permettre le codage de textes écrits quel que soit le système d'écriture utilisé. On attribue à chaque caractère un nom, une position normative et un bref descriptif qui seront les mêmes quelle que soit la plate-forme informatique ou le logiciel utilisés. Un consortium composé d'informaticiens, de chercheurs, de linguistes et de personnalités représentant les États ainsi que les entreprises s'occupe donc d'unifier toutes les pratiques en un seul et même système : **l'Unicode**.

L'Unicode est une table de correspondance Caractère-Code (Charset), et l'***UTF-8*** est l'encodage correspondant(Encoding) le plus répandu. Maintenant, par défaut, les navigateurs Internet utilisent le codage UTF-8 et les concepteurs de sites pensent de plus en plus à créer leurs pages web en prenant en compte cette même norme ; c'est pourquoi il y a de moins en moins de problèmes de compatibilité.

Exercice n°4

On considère la phrase : « Enfin ! J'ai tout compris à l'encodage. »

① Combien d'octets sont nécessaires pour écrire cette phrase ?

② Tapez ce texte dans le bloc notes puis dans un logiciel de traitement de texte comme Word. Sauvegardez les fichiers correspondants et comparez leur taille. Expliquez la différence observée.

UTF-8 est une extension mondiale de la table ASCII.

2 Les chaînes de caractères en Python

Tout programme informatique peut manipuler des chaînes de caractères. Nous dressons ici l'inventaire non exhaustif des routines Python.

2.1 Déclarer des variables de type string

En python , les variables qui contiennent une information alphabétique sont de types `str` : elles se déclarent avec des ***doubles quotes*** `" "`.

```
1 nom = "De Baudre"
2 prenom = "Jean Baptiste"
```

Les variables `nom` et `prenom` sont des « string ». Nous avons déjà rencontré l'instruction `input` qui permet d'affecter à une variable une information de type `str`, sans utilise les `""`.

```
1 lieu = input("Où habitez-vous?")
```

2.2 Nombre d'éléments d'une chaîne :

Considérons la variable `message` ci-dessous :

```
message = "Je suis une belle phrase de test..."
```

La chaîne de caractères `message` contient ***35 caractères*** : des lettres, des signes de ponctuations bien visibles, mais aussi des espaces !

La méthode `len` permet de retourner le nombre de caractères qui composent une chaîne de caractères.

Il suffit de taper l'instruction suivante pour confirmer le nombre d'élément de la variable `message` :

```
print(len(message))
```

2.3 Accès aux éléments de la chaîne de caractères

Comme son nom l'indique, une chaîne de caractères est constitué de caractères, ***ordonnés*** selon le ***rang*** qu'ils occupent dans la chaîne.

Le premier élément d'une chaîne est au rang 0.

Pour accéder en ***lecture*** au premier caractère de la variable `message`, il suffit de saisir l'instruction suivante :

```
print(message[0])
```

De même, l'instruction `print(message[10])` afficherait le caractère situé au rang 10 de la variable `message`.

Il ne faut pas donner un rang qui n'existe pas !

En effet, si le premier élément est au rang `0`, le ***dernier*** élément de la chaîne `message` est au rang `len(message) - 1`.
Donc `print(message[34])` affichera le dernier élément qui constitue la chaîne `message` mais `print(message[35])` retournera une erreur bien connue, `index out of range`.

2.4 Modification d'une chaîne de caractères

Une chaîne de caractères est un objet ***non mutable*** !

Cela signifie qu'une fois déclarée, on ne peut plus la modifier... contrairement à d'autres types de variables dont on peut changer la ***valeur***.

2.5 Accès à tous les éléments de la chaîne.

Le paragraphe précédent montre comment accéder à un élément de la chaîne, à partir de son ***rang***, son ***indice***.Par exemple, l'instruction `print(message[5])` permet d'afficher (s'il existe !), le caractère situé au rang `5` de la chaîne `message`.

Mais comment accéder à tous les éléments ?

C'est simple : pour chaque indice ou rang `i`, il faut faire afficher `message[i]`. On va donc ***boucler*** pour répéter l'instruction.
Une chaîne de caractères est un objet ***itérable***, ce qui signifie qu'on peut le parcourir élément par élément avec une ***boucle*** !
Il y a deux façons de procéder.

2.5.1 Parcourir les indices, les rangs

```
message = "Je suis une belle phrase de test..."
for i in range(len(message)):
    print(f"le caractère situé au rang {i} est {message[i]}")
```

Retenez que `i` désigne le rang, de `0` le plus petit à `len(message) - 1` le plus grand, et que `message[i]` est l'élément situé au rang `i` de la variable `message`.

2.5.2 Parcourir les éléments

```
message = "Je suis une belle phrase de test..."
for let in message:
    print(let.upper())
```

La variable `let` parcourt la chaîne de caractères `message` : sa première valeur sera donc `J` et sa dernière, le dernier point des trois points de suspensions `..`
La méthode `upper()` permet de transformer un minuscule en majuscule...

> **Exercice n°5**
> Que fait le code Python ci-dessous appliqué à une chaîne de caractères `texte` quelconque ?

```
cpt = 0
for elt in texte:
    if elt == " ":
        cpt = cpt + 1
print(cpt)
```

Lorsqu'on veut modifier une chaîne, par exemple pour changer les éventuelles majuscules en minuscules, on sait que l'opération est ***impossible***(objet non mutable).
En général, on en construit une nouvelle, vide au départ, en s'inspirant de la première. Un exemple :

```
message_en_majuscule = "Regardez l'AVION!"
message_en_minuscule = "" #chaine vide au départ
for let in message_en_majuscule:
    message_en_minuscule = message_en_minuscule + let.lower()
```

On « ajoute » ici des chaînes de caractères : cette opération porte un nom spécial :

Le $+$ des string ***concatène*** ces chaînes de caractères.

Par exemple, `"bon" + "jour"` donnerait `"bonjour"`...

Exercice n°6
Que fait le code suivant ?

```
mot = "anticonstitutionnellement"
tom = ""
for lettre in mot:
    tom = mot + tom
print(tom)
```

2.6 Encodage

La première partie de ce cours rappelle qu'un ***caractère*** est un ***nombre***.

Mais comment passer de l'un à l'autre ?

Nous allons utiliser les méthodes ***ord*** et ***chr*** qui permettent de convertir un caractère en nombre pour le premier et un nombre en caractère pour le second.

Exercice n°7

❶ (a) Dans la table ASCII ci-dessus, quel est le code ASCII du A ?
(b) Tapez `print(ord('A'))` pour vérifier.
❷ (a) Dans la table ASCII, ci-dessus quelle lettre est encodée par 112 ?
(b) Vérifiez en tapant `print(chr(112))` dans une console Python.

L'usage courant consiste à déclarer avec une ***seule quote*** une chaîne lorsqu'elle est réduite à un seul élément !

D'où le `'A'` au lieu de `"A"`...
Vous avez peut être déjà remarqué comment passer des codes des lettres minuscules aux codes des lettres majuscules : il suffit d'ajouter ou de soustraire 32 !

Exercice n°8

❶ Montrez que pour ajouter 32 à un nombre, il suffit de changer la valeur de l'un

de ses bits dans sa représentation binaire.

❷ Expliquez alors comment passer de la représentation binaire d'un minuscule à un majuscule et réciproquement

❸ Complétez le code ci-dessous pour passer des minuscules aux majuscules, la transformation n'étant applicables qu'aux... minuscules

```
mes_en_majus = ""
for i in range(len(mes_en_minus)):
    if ... < ord(mes_en_minus[i]) < ...:
        mes_en_majus = mes_en_majus + ...(...(mes_en_minus[i]) - 32)
    else:
        mes_en_majus = mes_en_majus + ....
print(mess_en_majus)
```

3 Le coin des exercices :

Exercice n°9

1. Écrire des instructions python qui permettent d'afficher le rang ou l'indice du premier caractère dans la chaîne si il est présent et -1 sinon. Par exemple `recherche('a', "vacances")` donne 1 et `recherche('b', "vacances")` retourne -1.
2. Même consigne avec une fonction qui renvoie le dernier rang si il existe... `recherche('a', "vacances")` donne alors 3.

Exercice n°10

Le codage de César consiste à crypter des lettres en les décalant de 3 vers la droite : ainsi le mot « bac » devient-il le mot « edf ».

① Écrire l'algorithme qui permet cette transformation. On ne transformera que les minuscules contenues dans le mot à crypter. Au mot 'Voyage' correspondra le mot 'Vrbdjh'.

② En s'aidant de routines vues dans les exercices précédents, construire un code qui transforme une chaîne de caractères selon le principe du codage de César.