

Chapitre 6 : Représentation des textes

ISN

Année scolaire 2019/2020

Introduction

Le but de ce chapitre est de comprendre comment les textes sont représentés dans un ordinateur et d'apprendre à manipuler le type `str` de Python qui leur est dédié.

1 Chaînes de caractères en Python

Méthode

- ☞ Une chaîne de caractères est une séquence éventuellement vide de caractères. Chaque caractère est un chaîne de longueur 1.

En Python, les chaînes de caractères définissent le type `str`. La longueur d'une chaîne de caractères s'obtient avec la fonction `len`.

```
In [6]: a = 1

In [7]: type(a)
Out[7]: int

In [8]: b = str(a)

In [9]: b, type(b)
Out[9]: ('1', str)

In [10]: len(b), len('un'), len('')
Out[10]: (1, 2, 0)
```

- ☞ Les chaînes de caractères partagent certaines propriétés avec les listes :

- Une chaîne de caractères `c` peut être vue comme un tableau de caractères indexés de 0 à `len(c) - 1` de gauche à droite et de `-1` à `-len(c)` de droite à gauche. L'opérateur crochet permet d'accéder au caractère d'index `i` avec la syntaxe `c[i]` ou à la tranche de caractères d'index appartenant à l'intervalle `[i;j[` avec la syntaxe `c[i:j]` si $0 \leq i < j$ ou `c[i:j:-1]` si $i < j \leq -1$.

```
In [26]: c[0], c[1], c[len(c)-1]
Out[26]: ('X', 'Y', 'T')
```

```
In [27]: c[-1], c[-2], c[-len(c)]
Out[27]: ('T', 'Z', 'X')

In [28]: c[1:3], c[-1:-len(c):-1]
Out[28]: ('YZ', 'TZY')
```

- On peut itérer sur une chaîne de caractères avec une boucle for.

```
In [18]: for c in 'XY':
...:     print(c)
...:
X
Y
```

- On peut concaténer deux chaînes pour créer une nouvelle chaîne avec l'opérateur +.

```
In [19]: a, b, c = 'belle', '-', 'ile'

In [20]: a + b + c
Out[20]: 'belle-ile'
```

- ☞ Une chaîne de caractères d'identifiant `chaine` est un objet Python qui intègre de nombreuses méthodes accessibles avec la notation pointée `chaine.methode()`. On peut les découvrir en saisissant `dir(chaine)` ou dans la documentation en ligne :

<https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str>

```
In [24]: a = 'un beau marin'

In [25]: a.find('beau')
Out[25]: 3

In [26]: a.replace('b', 'v')
Out[26]: 'un veau marin'

In [27]: a.upper()
Out[27]: 'UN BEAU MARIN'

In [28]: dir(a)
Out[28]: ['__add__',
 '__class__',
 '__contains__',
 ...]
```

Exercice 1

1. Recopier et compléter le code de la fonction `miroir` qui prend en paramètre une chaîne de caractères, pour qu'elle retourne la chaîne de caractères renversée :

```
def miroir(chaine):
```

```
res = ''
for c in chaine:
    #a completer
return res
```

```
In [33]: miroir('Suis-je toujours la plus belle?')
Out[33]: '?elleb sulp al sruojuot ej-siuS'
```

2. Écrire une fonction `palindrome(chaine)` qui retourne un booléen indiquant si la chaîne de caractères est un palindrome. Cette fonction ne doit pas tenir compte des espaces.

```
In [38]: palindrome('caser vite ce palindrome ne mord ni lape cet
ivre sac')
Out[38]: True
```

2 Du code ASCII à l'Unicode

2.1 Table de codage ASCII

Pour représenter de manière informatique un texte, il faut commencer par se donner une correspondance entre lettres et nombres. C'est ici qu'interviennent les tables de codages.



Définition 1 (*Table de codage*)

Une **table de codage** ou **jeu de caractères** ou **charset** ou **code point** permet de faire correspondre les caractères qu'on souhaite coder à un unique code.

Au début des années 1960, devant la multiplication des encodages de caractères et leurs incompatibilités mutuelles, l'ANSI (*American National Standards Institute*), définit une norme appelée **ASCII** (*American Standard Code for Information Interchange*).

Ce standard permet de coder 128 caractères sur 7 bits ($2^7 = 128$), le bit de poids fort étant réservé à des contrôles d'erreur : ce bit de parité est choisi pour que le nombre total de bits à 1 dans l'octet soit toujours pair.

La table de codage **ASCII** représentée ci-dessous contient :

- les lettres de l'alphabet latin en majuscule et minuscule et les chiffres de 0 à 9 ;
- des signes de ponctuation et es opérateurs arithmétiques ;
- des caractères spéciaux (tabulation, retour chariot, nouvelle ligne ...) et des caractères de contrôle non imprimables pour des protocole de communication et des contrôles de périphériques (Fin de transmission, EOT, demande de transmission ENQ ...).

La table de codage ASCII :

Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT End of transmission	0x24	36	\$	0x44	68	D	0x64	100	d
0x05	5	ENQ Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS Backspace	0x28	40	(0x48	72	H	0x68	104	h
0x09	9	TAB Horizontal tab	0x29	41)	0x49	73	I	0x69	105	i
0x0A	10	LF New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1 Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2 Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3 Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4 Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC Escape	0x3B	59	;	0x5B	91	[0x7B	123	{
0x1C	28	FS File separator	0x3C	60	<	0x5C	92	\	0x7C	124	
0x1D	29	GS Group separator	0x3D	61	=	0x5D	93]	0x7D	125	}
0x1E	30	RS Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

Exercice 2

En Python, la fonction `ord` associe à un caractère son code Unicode et la fonction `chr` est sa réciproque :

```
In [39]: ord('a'), ord('z'), ord('A'), ord('Z')
Out[39]: (97, 122, 65, 90)

In [40]: chr(65), chr(90)
Out[40]: ('A', 'Z')

In [41]: [chr(ord('a') + k) for k in range(26)]
Out[41]: ['a', 'b', 'c', ..., 'y', 'z']

In [42]: ''.join([chr(ord('a') + k) for k in range(26)])
Out[42]: 'abcdefghijklmnopqrstuvwxyz'
```

1. Définir en quelques lignes le chiffrement d'une chaîne de caractères par l'algorithme rot13.

Ressources : <https://fr.wikipedia.org/wiki/ROT13>

2. L'image par `rot13` du caractère 'H' est 'E'.

- on calcule d'abord le rang alphabétique d'un caractère :

```
In [56]: ord('R') - ord('A')
Out[56]: 17
```

- puis on ajoute 13 à ce rang et on prend le reste dans la division euclidienne par 26

```
In [59]: (ord('R') - ord('A') + 13) % 26
Out[59]: 4
```

- enfin on retrouve le rang du caractère associé au rang alphabétique calculé :

```
In [60]: chr(ord('A') + 4)
Out[60]: 'E'
```

Écrire une fonction `rot13(chaine)` qui chiffre ou déchiffre la chaîne (en majuscules ou convertie en majuscules avec `chaine.upper()`) passée en paramètre, avec l'algorithme `rot13`.

Exercice 3 Défi Turing problème 165

Le fichier `dico.txt` contient 323 471 mots français non accentués. En associant à chaque lettre son rang dans l'alphabet ($a=1, b=2, c=3, \dots, z=26$), puis en multipliant entre elles toutes les lettres d'un mot, on obtient un nombre.

Par exemple, "chaud" = $3 \times 8 \times 1 \times 21 \times 4 = 2016$.

Si l'on applique ce procédé à tous les mots du fichier `dico.txt`, quelle est l'année du troisième millénaire qui sera atteinte le plus souvent ?

Plus de défis sur <http://apprendre-en-ligne.net/turing/enonces.php>.

2.2 Tables de codages ISO-8859

Les caractères de la table **ASCII** se sont vite avérés insuffisants pour l'encodage de textes dans d'autres langues que l'Anglais qui ne comprend pas de caractères accentués par exemple. Pour y remédier, l'**ISO** (*Organisation internationale de normalisation*) a proposé la norme **ISO-8859** qui étend la table **ASCII** en utilisant le huitième bit. Sur un octet, on peut ainsi représenter deux fois plus de caractères, soit $2^8 = 256$.

La norme **ISO 8859** comprend seize tables de codages, certes compatibles entre elles et avec **ASCII** sur les 128 premiers caractères, mais avec 256 caractères il n'est pas possible d'écrire un texte avec un mélange de caractères arabes et russes.

De plus, avec la mondialisation des échanges numériques et la multiplication des encodages, les problèmes d'incompatibilité d'encodages se sont multipliés. Qui n'a jamais reçu de courriel avec d'étranges caractères comme ci-dessous ?

```
Je transf??re le message ?? Valentin
```

L'encodage du message n'étant pas reconnu par le client de messagerie, les caractères accentués è et à ont été remplacés par des signes ?.

2.3 Table de codage universelle et encodages Unicode

Pour assurer l'universalité de la représentation des caractères, l'ISO a défini une table de codage universelle, l'*Universal Character Set* sous la norme **ISO-10646**. Cette norme associe à chaque caractère un unique nombre appelé **point de code** et permet de représenter $2^{32} = 4\,294\,967\,296$ caractères. Actuellement, 110 000 caractères sont recensés.

Chaque point de code est de la forme $U+xxxx$ où chaque chiffre x est un caractère hexadécimal (base 16) avec au moins quatre chiffres.

Il est possible d'encoder chaque point de code sur 4 octets mais des encodages plus compacts ont été imaginés dans la norme **Unicode**. En effet, il suffit d'un octet pour stocker les caractères **ASCII** et de deux octets pour les plus courants dont le point de code est compris entre 0 et $2^{16} - 1 = 65\,535$.

Les encodages **Unicode** sont notés **UTF- n** où n est le nombre minimal de bits pour représenter un point de code.

L'encodage **Unicode** le plus courant est **UTF-8**.

Exercice 4

Sur la page <https://unicode-table.com/fr/> se trouve un moteur de recherche de caractère par **point de code**.

1. Déterminer la valeur en décimal (base 10) du point de code $U+ABCD$.
2. Rechercher le caractère dont le point de code est $U+263A$.
3. Tester puis expliquer le code ci-dessous :

```
pointcode = 0x263A
print(pointcode)
for k in range(10):
    print(chr(pointcode))
    pointcode = pointcode + 1
```

4. En Python, pour saisir directement un caractère à partir de son point de code, on peut utiliser des séquences d'échappement spéciales : `'\uxxxx'` si le point de code peut s'écrire avec 4 chiffres hexadécimaux, ou `'\Uxxxxxxxx'` s'il faut plus de quatre chiffres, en remplissant par des 0 à gauche les positions vides sur les huit possibles. Néanmoins il est plus pratique d'utiliser `chr` avec comme argument le point de code en décimal ou en hexadécimal.

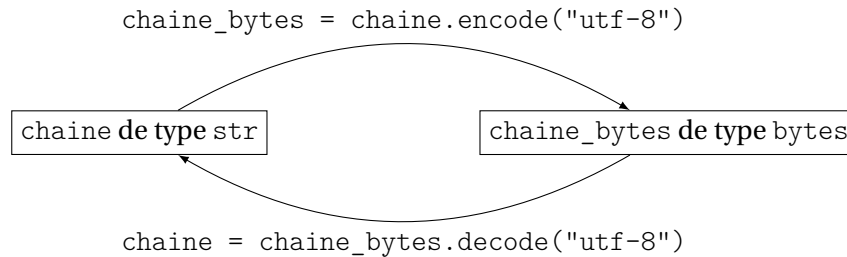
- a. Tester l'instruction ci-dessous dans une console Python

```
In [11]: print("\U0001f600")
```

- b. Écrire un code Python qui affiche tous les caractères dont le point de code est compris entre $U+1F600$ et $U+Ff64F$.

Méthode

Les méthodes `encode` et `decode` permettent de transformer une chaîne de caractères de type `'str'` en séquence d'octets de type `'bytes'` et réciproquement. Il faut préciser l'encodage.



Les octets d'une séquence d'octets de type `bytes` sont représentés par le caractère ASCII correspondant s'ils ont une valeur inférieure ou égale à 127 (7F en hexadécimal) ou par leur valeur hexadécimale préfixée de la séquence `\x`. Par exemple le caractère `'é'` est codé en **UTF-8** sur deux octets de valeurs hexadécimales C3 et A9. En **ISO-8859-1** ou **Latin1**, le caractère `'é'` est codé sur un seul octet. On peut mettre en évidence les incompatibilités de ces deux encodages pour les caractères non ASCII.

```
In [35]: "lycée".encode("utf-8")
Out[35]: b'lyc\xc3\xa9e'

In [36]: b'lyc\xc3\xa9e'.decode("utf-8")
Out[36]: 'lycée'

In [37]: "lycée".encode("latin1")
Out[37]: b'lyc\xe9e'

In [38]: b'lyc\xe9e'.decode("utf-8")
```

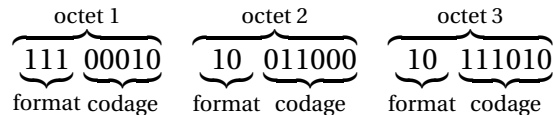
```
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xe9 in position 3:
invalid continuation byte
```

Exercice 5

Le principe de l'encodage **UTF-8** est le suivant :

- Si le bit de poids fort du premier octet lu est 0 alors il s'agit d'un caractère ASCII codé sur les 7 bits restants.
- Sinon, le nombre de bits de poids forts successifs du premier octet qui sont à 1 donne le nombre d'octets sur lequel le caractère est encodé. Chaque octet suivant commence par la séquence de deux bits 10 et le point de code en binaire est encodé sur les bits qui ne sont pas utilisés pour le formatage.

Par exemple le caractère de point de code U+263A soit 9786 en décimal et $\overbrace{10}^{\text{octet 1}} \overbrace{011000111010}^{\text{octet 2}} \overbrace{10}^{\text{octet 3}}$ ² en binaire est encodé sur trois octets par :



Plage de points de code	Séquence d'octets (en binaire)	bits codants
U+0000 (0) à U+007F (127) (ASCII)	0xxxxxxx	7 bits
U+008F (128) à U+07FF (2047)	110xxxxx 10xxxxxx	11 bits
U+0800 (2048) à U+FFFF (65535)	1110xxxx 10xxxxxx 10xxxxxx	16 bits
U+10000 (65536) à U+10FFF (1114111)	110xxxxx 10xxxxxx 10xxxxxx 10xxxxxx	21 bits

- Combien de caractères peut-on encoder en **UTF-8** ?
- Sur combien de caractères sont encodés les caractères accentués en **UTF-8** ?
- Écrire une fonction `unicode(s)` qui affiche pour chaque caractère de la chaîne `s`, le caractère `s`, son point de code et son encodage UTF-8 sous forme d'octets en hexadécimal puis en binaire.
- Déterminer l'encodage UTF-8 en donnant les octets en décimal, en binaire et en hexadécimal des caractères suivants :
 - le caractère @ de point de code U+0040
 - le caractère æ de point de code U+00E6
 - le caractère ℚ de point de code U+211A
- Écrire en Python une fonction `longueur(b)` qui parcourt une chaîne d'octets de type `bytes` encodée en **UTF-8** et qui retourne le nombre de caractères représentés.



Il est interdit de convertir la séquence d'octets en chaîne de caractères avec la méthode `decode`.

Sources :

Le contenu de ce cours et certains exercices sont directement inspirés du chapitre 21 du Manuel de première NSI chez Ellipses.