

# NSI Première

## TD : chaînes de caractères

qkzk

2021/05/23

### Exercice 1

1. Citer trois encodages différents permettant de représenter des chaînes de caractères en mémoire.

### Exercice 2 - Chiffrement de César

Le chiffrement de César est un des premiers algorithmes de chiffrement documenté. Soit-disant utilisé par Jules César lui-même, il consiste à décaler chaque lettre d'un texte de trois position dans l'alphabet.

Ainsi A devient D, B devient E etc. Arrivée à la fin de l'alphabet, on recommence au début : X devient A, Z devient C

Pour décoder un message, on applique la transformation inverse.

Il est simple d'écrire une fonction de chiffrement en 5 lignes de Python. La fonction de déchiffrement nécessite alors de changer un seul symbole !

Voici l'aide des fonctions `ord` et `chr`

Help on built-in function `ord` in module `builtins`:

```
ord(c, /)
    Return the Unicode code point for a one-character string.
```

Help on built-in function `chr` in module `builtins`:

```
chr(i, /)
    Return a Unicode string of one character with ordinal i; 0 <= i <= 0x10ffff.
```

1. Que produit l'instruction `ord("A")` ?
2. Enchaînons : `chr(ord("A"))` ?
3. Que faire pour obtenir un "D" quand on passe un "A" ?
4. Englober le tout dans une première fonction `cesar` qui prend une lettre et l'encode.
5. La version précédente a deux défauts :
  - Elle n'accepte qu'une lettre à la fois,
  - "X", "Y" et "Z" ne sont pas encodés par des lettres.

1. À l'aide d'une boucle, résoudre le premier problème.

Souvenons-nous que `"bonjour" + "Quentin" == "bonjourQuentin"`

2. Quelle opération Python permet d'obtenir le reste d'une division euclidienne ?

6. Version complète.

Afin de toujours obtenir une lettre, même pour "X", "Y" et "Z", nous allons :

1. Ramener chaque entier entre 0 et 25.

Combien faut-il soustraire à `ord(lettre)` pour qu'une majuscule renvoie toujours en entier entre 0 et 25 ?

2. Ajouter 3.
3. Prendre un reste modulo 26.
4. Ajouter le même entier qu'à l'étape 1.

Écrire l'opération complète réalisée sur chaque lettre.

7. Englober le tout dans une fonction `cesar` qui prend en paramètre une chaîne de caractère (seulement des lettres majuscules, aucune espace ni ponctuation) et renvoie sa version encodée.

```
>>> cesar("BONJOUR")
'ERQMRXU'
```

8. Écrire la fonction de décodage.

```
>>> decesar('ERQMRXU')
'BONJOUR'
```

## Exercice 2 - Se déplacer dans la table ASCII

Sous Linux, on consulte facilement une table ASCII à l'aide de la documentation :

```
$ man ascii
```

```
...
```

```
NAME
```

```
ascii - ASCII character set encoded in octal, decimal, and hexadecimal
```

Oct	Dec	Hex	Char	Oct	Dec	Hex	Char
000	0	00	NUL '\0' (null character)	100	64	40	@
001	1	01	SOH (start of heading)	101	65	41	A
002	2	02	STX (start of text)	102	66	42	B
003	3	03	ETX (end of text)	103	67	43	C
004	4	04	EOT (end of transmission)	104	68	44	D
005	5	05	ENQ (enquiry)	105	69	45	E
006	6	06	ACK (acknowledge)	106	70	46	F
007	7	07	BEL '\a' (bell)	107	71	47	G
010	8	08	BS '\b' (backspace)	110	72	48	H
011	9	09	HT '\t' (horizontal tab)	111	73	49	I
012	10	0A	LF '\n' (new line)	112	74	4A	J
013	11	0B	VT '\v' (vertical tab)	113	75	4B	K
014	12	0C	FF '\f' (form feed)	114	76	4C	L
015	13	0D	CR '\r' (carriage ret)	115	77	4D	M
016	14	0E	SO (shift out)	116	78	4E	N
017	15	0F	SI (shift in)	117	79	4F	O
020	16	10	DLE (data link escape)	120	80	50	P
021	17	11	DC1 (device control 1)	121	81	51	Q
022	18	12	DC2 (device control 2)	122	82	52	R
023	19	13	DC3 (device control 3)	123	83	53	S
024	20	14	DC4 (device control 4)	124	84	54	T
025	21	15	NAK (negative ack.)	125	85	55	U
026	22	16	SYN (synchronous idle)	126	86	56	V
027	23	17	ETB (end of trans. blk)	127	87	57	W
030	24	18	CAN (cancel)	130	88	58	X
031	25	19	EM (end of medium)	131	89	59	Y
032	26	1A	SUB (substitute)	132	90	5A	Z
033	27	1B	ESC (escape)	133	91	5B	[

034	28	1C	FS	(file separator)	134	92	5C	\	'\\'
035	29	1D	GS	(group separator)	135	93	5D	]	
036	30	1E	RS	(record separator)	136	94	5E	^	
037	31	1F	US	(unit separator)	137	95	5F	_	
040	32	20	SPACE		140	96	60	`	
041	33	21	!		141	97	61	a	
042	34	22	"		142	98	62	b	
043	35	23	#		143	99	63	c	
044	36	24	\$		144	100	64	d	
045	37	25	%		145	101	65	e	
046	38	26	&		146	102	66	f	
047	39	27	'		147	103	67	g	
050	40	28	(		150	104	68	h	
051	41	29	)		151	105	69	i	
052	42	2A	*		152	106	6A	j	
053	43	2B	+		153	107	6B	k	
054	44	2C	,		154	108	6C	l	
055	45	2D	-		155	109	6D	m	
056	46	2E	.		156	110	6E	n	
057	47	2F	/		157	111	6F	o	
060	48	30	0		160	112	70	p	
061	49	31	1		161	113	71	q	
062	50	32	2		162	114	72	r	
063	51	33	3		163	115	73	s	
064	52	34	4		164	116	74	t	
065	53	35	5		165	117	75	u	
066	54	36	6		166	118	76	v	
067	55	37	7		167	119	77	w	
070	56	38	8		170	120	78	x	
071	57	39	9		171	121	79	y	
072	58	3A	:		172	122	7A	z	
073	59	3B	;		173	123	7B	{	
074	60	3C	<		174	124	7C		
075	61	3D	=		175	125	7D	}	
076	62	3E	>		176	126	7E	~	
077	63	3F	?		177	127	7F	DEL	

1. Que désigne “l’octal” auquel la documentation se réfère ?
2. Donner la représentation binaire du “A”, représenté en mémoire par l’entier 65.
3. Recommencer avec le “a”.
4. Comment passer facilement d’une lettre majuscule à sa version minuscule quand on dispose de la représentation binaire ?
5. À quelle opération mathématique cela correspond-t-il ?

### Exercice 3

Nous allons étudier ce que fait l’instruction Python suivante :

```
open("ascii_table.txt", "wb").write(bytearray(range(32, 128)))
```

Elle est composée de trois parties :

1. `open( ... )`
2. `.write( ... )`
3. `bytearray(range(32, 128))`

### Questions

1. Que fait l’instruction `range(32, 128)` ?

2. L'instruction `bytearray( ... )` lorsqu'on lui passe une séquence itérable d'entiers entre 0 et 255 produit un tableau d'octets qu'on peut écrire dans un fichier. Que va produire `bytearray(range(32, 128))` ?
3. L'instruction `open( fichier, mode)` ouvre un fichier selon le mode fourni par `mode`. Le `mode` est donné par une chaîne de caractères et voici un extrait de la documentation python :

```
=====
Character Meaning
-----
'r'      open for reading (default)
'w'      open for writing, truncating the file first
'x'      create a new file and open it for writing
'a'      open for writing, appending to the end of the file if it exists
'b'      binary mode
```

Que fait le mode `wb` ?

4. Le premier paramètre de la fonction `open`, dans l'instruction plus haut est `ascii_table.txt`.

Voici un extrait de la documentation Python

```
open(file, mode='r', encoding=None, ... )
    Open file and return a stream. Raise OSError upon failure.

    file is either a text or byte string giving the name (and the path
    if the file isn't in the current working directory) of the file to
    be opened
```

Quel est le résultat de l'instruction `open("ascii_table.txt", "wb")` ?

4. Appliquée à un fichier ouvert en mémoire, la méthode `write` en mode accepte en paramètre soit une chaîne de caractère, soit une collection itérable d'octets (`bytearray`).

Maintenant que nous avons étudié toutes les parties, quel devrait-être le résultat de l'instruction complète ?

5. Vérifier en exécutant l'instruction.

Si vous utilisez *Basthon*, vous pouvez consulter le contenu du fichier avec :

```
>>> open("ascii_table.txt").read()
```

Qui va ouvrir le fichier en mode lecture et vous afficher son contenu sous la forme d'une chaîne de caractères.

## Exercice 4 - Un peu d'UTF-8

On l'a vu, l'encodage ASCII est limité à 127 caractères. Afin de dépasser cette limite assez de caractères pour toutes les langues il a fallu ruser. L'encodage UTF-8 permet :

- de représenter un texte ne comportant que des caractères ASCII sans le changer,
- de représenter jusqu'à  $2^{31}$  symboles,
- de reconnaître tous les symboles qu'il encode manière unique.

Afin d'éviter les confusions, certains bits sont réservés :

Voici un extrait de la documentation obtenue avec `$ man utf-8` :

```
The following byte sequences are used to represent a character.
The sequence to be used depends on the UCS code number of the character:
```

```
0x00000000 - 0x0000007F:
    0xxxxxxx
```

```
0x00000080 - 0x000007FF:
    110xxxxx 10xxxxxx
```

```
0x00000800 - 0x0000FFFF:
```

1110xxxx 10xxxxxx 10xxxxxx

0x00010000 - 0x001FFFFF:

11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

0x00200000 - 0x03FFFFFF:

111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

0x04000000 - 0x7FFFFFFF:

1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

The xxx bit positions are filled with the bits of the character code number in binary representation, most significant bit first (big-endian). Only the shortest possible multibyte sequence which can represent the code number of the character can be used.

Sur chaque paire de ligne, on lit d'abord les nombres encodés, représentés en hexadécimal puis les bits qu'ils peuvent occuper.

Ainsi pour tous les nombres entre 0x00000000 et 0x0000007F on utilise uniquement les bits 0xxxxxxx

1. En quoi cela-permet-t-il d'assurer la compatibilité avec ASCII ?
2. Comment-sait-on qu'un caractère est encodé sur 2 octets ? 3 octets ?
3. Donner les bits de la représentation utf-8 du caractère unicode 0xa9 (le signe copyright).
4. Un caractère est encodé par les bits : 11100010 10001001 10100000
  - Combien d'octets occupe-t-il en mémoire ?
  - Convertir chaque octet en hexadécimal et déterminer sur internet à quel symbole il correspond.