Chaines de caractères

I Chaines de caractères avec Python

Le type **str** de python permet de représenter du texte ; en anglais *character string* signifie *chaine de caractères*. On les délimite au choix avec ", ' ou '''.

```
>>> 'Il l'a faite'
SyntaxError: invalid syntax
>>> "Il l'a faite"
>>> print('Il a dit : "Il a fait quoi ?"')
Il a dit : "Il a fait quoi ?"
```
1

Les chaînes se comportent comme les listes sur différents points.

```
>>> c = "J'aime Python"
>>> type(c)
<class 'str'>
>>> len(c)
13
>>> c[0]
'J'
>>> c[-1]
'n'
>>> c[2: 4]
'ai'
```
2

```
>>> c + c
"J'aime PythonJ'aime Python"
>>> c[7] = 'p'

TypeError: 'str' object does
not support item assignment
>>> del c[7]

TypeError: 'str' object doesn't
support item deletion
```
3

Une différence fondamentale entre liste et chaîne de caractères est que la première est **modifiable** (**mutable**) mais pas la deuxième : on ne peut pas modifier une chaîne de caractères (on ne peut pas changer un caractère ni en ajouter un ou en supprimer un) ! Une chaîne de caractères est **non modifiable**, **immuable, non mutable**.

Une chaîne de caractères est itérable :

```
>>> for x in 'MPSI':
        print(x)


M
P
S
I
```
4

On conserve le test d'appartenance :

```
>>> 'I' in 'MPSI'
True
>>> 'MP' in 'MPSI'
True
>>> 'MSI' in 'MPSI'
False
```
5

On rappelle quelques fonctions de conversions de type :

```
>>> str(127)
'127'
>>> int('218')
218
>>> list('MPSI')
['M', 'P', 'S', 'I']
```
6

Pour passer d'un caractère à son code entier, on dispose des fonctions **ord**() et **chr**() :

```
>>> ord('A')
65
>>> ord("à")
224
>>> ord('#')
35
```
7

```
>>> chr(65)
'A'
>>> chr(64)
'@'
>>> chr(60)
'<'
```
8

Exercice 1

Écrire une fonction qui prend en paramètre une chaînes de caractères qui représente un nombre entier naturel en décimal et qui renvoie la valeur entière de ce nombre (on n'utilisera aucune méthode toute faite).

Entraînement 1

Le *chiffre de César* est une méthode très simple pour coder un message. On écrit le texte tout en majuscule et on supprime les espaces. On décale alors toutes les lettres d'un certain nombre n de rangs dans l'alphabet. Par exemple, pour $n = 19$ on a la situation de la figure ci-contre : A est codé par T, B par U, etc.

Écrire une fonction `chiffre_Cesar(message, n)` qui renvoie la chaîne obtenue par le décalage de n rangs dans l'alphabet.

Parmi les très nombreuses méthodes applicables aux chaînes de caractères en voici deux qui sont souvent bien pratiques :

```
>>> c = "Maman est en voyage d'affaires"
>>> c.split()
['Maman', 'est', 'en', 'voyage', "d'affaires"]
```
9

```
>>> L = ['Papa', 'est', 'a', 'la', 'maison']
>>> ' '.join(L)
'Papa est a la maison'
```
10

Deux caractères spéciaux sont souvent utiles :

`\n` provoque le passage à la ligne (nouvelle ligne) et `\t` le décalage pour une tabulation.

```
>>> c = "Voici une phrase\nun peu longue"
>>> print(c)
Voici une phrase
un peu longue
```
11

```
>>> c = "miam\n"
>>> print(c*3)
miam
miam
miam
```
12

```
>>> c = "Riri\t18\nFifi\t16\nLoulou\t12"
>>> print(c)
Riri    18
Fifi    16
Loulou  12
```
13

Exercice 2

Écrire une fonction qui prend en paramètre une liste (non vide) de chaînes de caractères et qui renvoie une chaîne de caractères contenant tous les mots de L séparés par un espace (on n'utilisera aucune méthode toute faite).

Par exemple :
`["Un", "python", "est", "un", "serpent"]` → `"Un python est un serpent"`

Exercice 3

Écrire une fonction qui prend en paramètres une liste de noms et une liste de notes de même longueur et qui réalise un affichage comme pour Riri, Fifi et Loulou.

Entraînement 2

Écrire une fonction prenant en paramètre une chaîne de caractères et renvoyant la liste des mots qui la constituent (on considère que deux mots sont séparés par un espace et qu'il n'y a pas de ponctuation). La méthode . split () est interdite !

Par exemple : `"Python est un langage"` → `['Python', 'est', 'un', 'langage']`

II Encodage des chaînes, de l'ASCII à l'Unicode

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.*

La première table de codage massivement utilisée est l'ASCII (American Standard Code for Information Interchange) qui fut définie dans les années 1960 et s'imposa rapidement. Il permet de coder 128 caractères sur 7 bits ($2^7 = 128$). Comme l'usage en informatique est de travailler sur 8 bits (un octet), il reste 128 autres possibilités. Ceci a permit de coder sur un octet les lettres accentuées (qui n'existent pas en anglais) et a donné naissance à l'encodage Latin-1 (ou ISO 8859-1 ou Europe Occidentale, 1986) qui est largement utilisé en France et prolonge le code ASCII.

Mais comme chaque langue possède ses propres caractères, il a fallu trouver un codage universel : l'unicode (1988). Celui-ci prolonge le code ASCII bien entendu mais permet également de coder les caractères d'énormément de langues (même le mandarin et ses milliers d'idéogrammes). Chaque symbole possède alors un numéro unique et l'encodage concret de ce numéro en une suite de bits se fait par exemple via l'UTF-8 (Universal Transformation Format) qui est le plus utilisé. L'encodage se fait sur un nombre d'octets variable (de un à quatre) alors que Latin-1 utilise invariablement un octet. Voir le site `unicode.org` pour se faire une idée des caractères que l'on peut utiliser.

Cette question d'encodage est fondamentale pour l'échange de fichiers et se pose dans de nombreuses circonstances : courrier électronique, site internet, éditeur, base de données etc. Qui n'a jamais vu d'étranges caractères s'afficher en lieu et place des accents ?

Par exemple, on peut régler Pyzo en UTF-8, ASCII, Latin-1 ...via les onglets Fichier et Encodage. On peut également placer en début de fichier la ligne suivante :

```
1 # -*- coding: utf-8 -*-
```

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

III Recherche d'un mot dans une chaîne de caractère

On considère deux chaines de caractères `phrase` et `mot`. On souhaite déterminer si `mot` est une sous-chaine de `phrase` et, si oui, à quelle position elle apparait pour la première fois. Ici, les chaînes `phrase` et `mot` peuvent n'avoir aucune signification et ne pas ressembler à des phrases ni des mots, le choix des noms ne sert qu'à fixer les rôles.

On va procéder en deux temps.

Exercice 4

Écrire une fonction `est_sous_chaine_position(phrase, mot, i)` qui teste si `mot` est une sous chaine de `phrase` à la position i dans `phrase`. On suppose que $i \leqslant$ **len**(`phrase`)−**len**(`mot`).

Exercice 5

En déduire une fonction `est_sous_chaine(phrase, mot)` qui utilise la précédente et répond au problème posé en préambule. Cette fonction doit renvoyer None si `mot` n'est pas une sous-chaîne de `phrase`.

Entraînement 3 

Fusionner les fonctions précédentes pour ne plus en faire qu'une seule.