



TP2 : Chaines de caractères



Copiez le répertoire du TP dans votre espace personnel. Lancez spyder(anaconda3).

Un script est fourni avec ce TP.

Tout programme sera testé avec des paramètres judicieusement choisis.

I. Présentation

Les chaînes de caractères sont utilisées pour la représentation des textes. Les chaînes de caractères sont constituées de caractères. Ce sont des valeurs au même titre que les nombre entiers ou flottants (réels) au sens où elles peuvent être affectées à des variables, et que des opérations spécifiques peuvent leur être appliquées.

Le type informatique correspondant aux chaînes de caractères est **string** (chaîne en anglais), apparaissant en python sous **str**. Pour que l'interpréteur python identifie une chaîne de caractères, celle-ci doit être encadrée par des guillemets ou des doubles guillemets :

- 'u' (on utilise ' et ' pour encadrer la chaîne de caractères 4)
- "une chaîne" (on utilise pour " pour encadrer la chaîne de caractères 3)

Exemple :

```
une_chaine="ceci est une chaîne de caractères"
print( type(une_chaine) )
```

Le code ci-dessus provoque l'affichage, ce qui signifie que **une_chaine** est une variable de type.....

Remarque : une chaîne de caractère peut n'être constituée que d'un unique caractère : "u"

II. Manipulation des caractères et chaînes de caractères

➤ Affectation

Exécutez le code suivant, et utilisez l'explorateur de variables pour vérifier les affectations

```
g = " tagada"  #notez que le premier caractère est un espace !
h = "fraise"
```

➤ Opérateur +

L'opérateur + permet la concaténation de deux chaînes de caractères :

```
13 print( h + g )#provoque l'affichage.....
```

h+g correspond à une chaîne de caractères obtenue en

➤ Opérateur *

L'opérateur * permet de répéter la concaténation d'une chaîne de caractères avec elle-même :

```
15 print( g*3 ) #provoque l'affichage.....
16
17 print( 3*g ) #provoque l'affichage.....
```

➤ Accès à un caractère

synthaxe : `nom_de_la_chaine[indice du caractère]`

testez les instructions suivantes, et observez les valeurs affectées aux variables dans l'explorateur de variables :

```
19 caractere1= g[0]    # caractere1 est affectée de .....
20
21 caractere2= h[2]    # caractere2 est affectée de .....
```

➤ Accès à un sous-ensemble d'une chaîne (slicing)

Il est possible d'accéder à un sous-ensemble d'une chaîne de caractères:

`nom_de_la_variable [ indice_du_premier_caractère : indice_du_dernier_caractère+1 ]`

exemple : testez

```
22 sous_ensemble=h[0:3] # sous_ensemble est .....
```

Sous chaque caractère de la chaîne obtenue, indiquez son indice

➤ Modification d'une chaîne de caractère

Il n'est pas possible de modifier une chaîne de caractère.

Exemple :

```
27 mon_message= "je suis en terminale"
28 mon_message[10] = 'a' #provoque l'affichage.....
```

Par contre, on peut mettre à profit le slicing et la concaténation : je peux produire une nouvelle chaîne de caractère, pour afficher 'je suis en CPGE'

```
mon_message = "je suis en terminale"
mon_nouveau_message = mon_message[.....] + "....."
print( mon_nouveau_message ) #provoque l'affichage.....
```

➤ Longueur

La longueur d'une chaîne de caractère est obtenue au moyen de la fonction `len()` :

exemple :

```
print( len(h)) #provoque l'affichage.....
```

III. Parcourir une chaîne de caractères

Il est fréquent que l'on ait besoin de parcourir une chaîne de caractères, c'est-à-dire utiliser successivement chacun des caractères d'une chaîne de caractères. Pour cela, on utilise une structure itérative. Exemples, pour afficher un à un chacun des caractères de la chaîne **ma_chaine** :

Avec une boucle *for* :

```
37 g = " tagada "
38 h = "fraise"
39 ma_chaine = h + g
40
41 for i in range(len(ma_chaine)):
42     print(ma_chaine[i])
```

Avec une boucle *while* :

```
44 g = " tagada "
45 h = "fraise"
46 ma_chaine = h + g
47
48 i=0
49 while i<len(ma_chaine):
50     print(ma_chaine[i])
51     i=i+1
```

Nous le reverrons plus tard, mais la boucle **for** accepte différents éléments après son **in**, pourvu que celui-ci soit un itérateur. C'est le cas de **range**, et c'est le cas des chaînes de caractères. Ainsi, une autre écriture équivalente aux programmes précédents peut prendre la forme ci-contre:

```
g = " tagada "
h = "fraise"
ma_chaine = h + g
for caractere in ma_chaine:
    print(caractere)
```

IV. Exercices

1. Parcourir une chaîne de caractères



Ouvrir le script python **chaîne_de_caractères.py**



Ex1.1 à la suite de **chaîne_de_caractères.py**, proposez un programme parcourant la chaîne de caractères **ma_chaine**, et n'affichant que les caractères d'indice pair. On utilisera une boucle **for**. On rappelle que pour le test de parité, on pourra utiliser le reste de la division euclidienne : $i\%2$ vaut 0 si i est pair, 1 sinon.



Ex1.2 reprendre la question précédente avec une boucle **while**.



Ex2 à la suite de **chaîne_de_caractères.py**, on propose une ébauche de programme. Complétez ce programme de sorte à ce que, en fin de programme, **copie** soit une chaîne de caractères identique à **ma_chaine**

Palindrome :



Ex3 inspirez-vous du programme précédent pour proposer un programme qui crée une nouvelle chaîne de caractères, appelée **palindrome**, qui contient les mêmes caractères que **ma_chaine**, mais ordonnés en sens inverse. On rappelle que 'E'+ 'n' donne 'En', et que 'n'+ 'E' donne 'nE'.

2. Recherche dans une chaîne de caractères



Ex4 Proposez un programme effectuant la recherche du caractère 'x' dans la chaîne de caractères **ma_chaine** : il s'agit de parcourir la chaîne de caractère **ma_chaine** en utilisant une boucle **while** le programme pourra terminer en indiquant l'indice où se trouve le caractère recherché dans **ma_chaine**.



Ex4.1 Reprendre le programme précédent de sorte à ce que la recherche s'arrête dès qu'on trouve le caractère 'x' : on utilise toujours une boucle **while**, mais qui parcourt la chaîne tant qu'on est pas arrivé à la fin de la chaîne, et que le i -ème caractère est différent de 'x'.

On peut effectuer la recherche d'un mot en comparant ce mot à un sous ensemble du texte dans lequel on recherche.

On rappelle qu'on peut obtenir un sous-ensemble d'une chaîne de caractères en utilisant le slicing (voir page précédente)



Ex5 Inspirez-vous du programme précédent pour proposer un programme effectuant la recherche du mot '**creux**' dans la chaîne de caractères '**ma_chaine**'.

V. Conversion

Les chaînes de caractères peuvent utiliser des caractères représentant des chiffres ou des valeurs booléennes. Ainsi, il est possible d'effectuer des conversions des chaînes de caractères vers les nombres ou booléens, et inversement :

- *Conversion en chaîne de caractère :*

```
a = str(1.2)
print( a )      #.....
print( type(a) ) #.....
```

- *Conversion vers un autre type :* il est possible de convertir une chaîne de caractères vers un type, si les caractères ont un sens dans le type de destination :

```
un_entier = int('3')
print( un_entier )      #.....
print( type(un_entier) ) #.....

un_flottant = float('3.2')
print( un_flottant )    #.....
print( type(un_flottant) )#.....

un_autre_flottant = float('3')
print( un_autre_flottant ) #.....
print( type(un_autre_flottant) )#.....
```

Si les caractères n'ont pas de sens dans le type de destination, la conversion est impossible.

Par exemple :

```
un_essai = int('3.1') #.....

un_autre_essai = float('3,1') #.....
```

VI. Exercices



Ex6 Proposez un programme prenant en paramètre une chaîne de caractères **s**, et qui compte le nombre de caractères de **s**. Une solution évidente que nous n'utiliserons pas est l'utilisation de la fonction **len()**.



Ex7 Proposez un programme prenant en paramètre une chaîne de caractères **s**, et un caractère **c**, qui compte le nombre d'occurrences de **c** dans la chaîne **s**.



Ex8 Proposez un programme prenant en paramètre une chaîne de caractères, et produisant une nouvelle chaîne de caractères ne contenant que les voyelles de la chaîne initiale.



VII. Correcteur d'orthographe : détection d'oubli de majuscule

Les éditeurs de texte disposent d'outils d'aide à la rédaction tels que les correcteurs d'orthographe et correcteurs grammaticaux. On se propose ici de concevoir un programme permettant la détection et la correction des erreurs suivantes :

- oubli d'espace entre un point et le début de la phrase suivante,
- oubli de majuscule en début de phrase.

Le fichier **correcteur.py** contient pour le moment le code suivant, permettant d'importer le texte dans la variable **texte** :

```
7 #-----acquisition du texte-----
8
9 fichier = open("fichierutf8.txt", "r") #ouverture du fichier contenant le texte
10 contenu=fichier.readlines()
11 texte=""
12 for ligne in contenu:
13     texte=texte+ligne.rstrip("\n\r")
14 fichier.close()
15 texte=texte.decode("utf-8")
16 #la variable texte contient maintenant le texte sur lequel vous allez travailler
17 #-----
18 #----- espace dédié à votre code -----
19 #          V V V V V V V V V V
```

1. Détection de l'oubli d'un espace en début de phrase

-  6) Proposer un code python qui parcourt la chaîne de caractères '**texte**', et qui :
- teste pour chaque caractère s'il s'agit du caractère '.',
 - dans l'affirmative, teste si le caractère suivant est bien un espace.
 - si ce n'est pas le cas, mémorise l'indice correspondant en affectant à une variable qu'on appellera **indice_erreur_espace**

On fera attention à ne pas tester le dernier point du texte !

-  7) Que se passe-t-il si vous souhaitez utiliser la méthode décrite ci-dessus, jusqu'au dernier point du texte ?

.....

.....

.....

.....

-  8) exécutez votre script. Observez **indice_erreur_espace** dans l'explorateur de variables. Pour vérifier le résultat, vous pouvez utiliser l'indice affecté à **indice_erreur_espace** et les utiliser pour afficher un sous-ensemble du texte ; par exemple : `print( texte[indice-20 : indice+20] )`

2. Détection d'oubli de majuscule

a. En codage des caractères

La fonction `ord( mon_caractère )` permet d'obtenir le codage du caractère.



9) Dans l'interpréteur, testez les commandes suivantes :

```
>>>ord('a')
```

```
>>>ord('A')
```

```
>>> ord(' ') # pour obtenir le codage d'un espace
```

On rappelle la table d'encodage 'Latin-1' :

ISO-8859-1																
	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1x	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2x	SP	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
8x	PAD	HOP	BPH	NBH	IND	NEL	SSA	ESA	HTS	HTJ	VTS	PLD	PLU	RI	SS2	SS3
9x	DCS	PU1	PU2	STS	CCH	MW	SPA	EPA	SOS	SGCI	SCI	CSI	ST	OSC	PM	APC
Ax	NBSP	ı	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	­	®	¯
Bx	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
Fx	ø	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Dans ce tableau, on observe que le codage d'un caractère est obtenu en utilisant les digits des 2 entrées du tableau. Ces digits correspondent à des valeurs en base 16.

Ainsi le caractère B a pour code la valeur $(42)_{16}$

Le codage obtenu à partir de ce tableau est en base Hexadécimale. On rappelle que la base hexadécimale utilise les digit suivants

Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Valeur decimale correspondante	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

En python, pour obtenir la représentation en base 10 d'un nombre exprimé en base 16, il suffit de le faire précéder de 0x.

Exemple :

Soit un nombre en base hexadécimale : 6A

```
>>> 0x6A
```

```
106
```

```
>>> hex(106)
```

```
'0x6a'
```

hex(mon_nombre_en_base_dix) donne la conversion en base 16 d'un nombre en base 10.

 10) En utilisant les fonction **ord()** et **hex()**, vérifiez dans l'interpréteur python le code hexadécimal obtenu pour quelques caractères.

b. Majuscules et minuscules

11) D'après cette table, que peut-on remarquer quant à la disposition d'une lettre minuscule par rapport à sa majuscule ?

Remarque : base de représentation des nombres

Un nombre représenté en utilisant la base peut être décomposé en fonction des puissances de 10. Exemple : $(125)_{10} = 1 \times 10^2 + 2 \times 10^1 + 5 \times 10^0$

De même, un nombre représenté en base 16 peut être décomposé en fonction des puissances de 16. Exemple : $(C1)_{16} = 12_{10} \times 16_{10}^1 + 1_{10} \times 16_{10}^0$ (on rappelle que C_{16} est représenté en base 10 par 12_{10})

On peut utiliser cette décomposition pour obtenir la représentation en base 10 d'un nombre exprimé en base 16 : $(C1)_{16} = 12_{10} \times 16_{10}^1 + 1_{10} \times 16_{10}^0 = (193)_{10}$



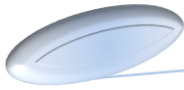
12) Quel est le résultat de l'opération $\text{ord}('L') + 3$? de l'opération $\text{ord}('L') + 16$?



13) vérifiez avec l'interpréteur python



14) D'après la table d'encodage, quelle opération mathématique simple permet de faire passer du code d'un caractère minuscule au code du caractère majuscule correspondant ? Et pour l'opération inverse ?



10) Pour le code que nous voulons construire, comment détecter si un caractère est un caractère majuscule ou minuscule ? Comment traduire cela en expression logique codable ?

.....

.....

.....

.....

c. Détection d'oubli de majuscule



15) En modifiant légèrement le code que vous avez écrit pour la détection d'oubli d'espace, proposez un code qui permet la détection de l'oubli de majuscule en début de phrase.

VIII. Amélioration de la recherche de mot dans un texte

Un mot peut apparaître plusieurs fois dans un texte. Le programme tel que vous l'avez écrit dans la partie I ne gère pas cette éventualité.



16) modifiez votre programme `correcteur.py` afin qu'il renvoie la liste des indices où se trouve le mot recherché. On pensera à initialiser une liste nommée ***liste*** avant la boucle de recherche, puis, à chaque fois que l'élément recherché est trouvé, on insérera l'indice dans la liste à l'aide de la commande ***liste.append(élément_à_insérer)***