Chaînes de caractères

Étienne Lozes – Université Nice Sophia Antipolis

Formation ISN Python 2018

Exercice 1 (Un exercice renversant, ⋆)

1. *Écrivez une fonction* `show_mirror(s)` *qui prend une chaîne de caractères* `s` *et qui affiche* `s` *et son image miroir (*`s` *à l'envers) ; vous proposerez deux solutions, une avec un pas de boucle de* -1*, et l'autre avec un pas de boucle de* 1*.*

```
>>> show_mirror('abc')
abc cba
```

2. *Écrivez une fonction* `mirror(s)` *qui cette fois-ci retourne la chaîne de caractères* `s` *à l'envers, de sorte que l'on puisse répondre à la question 1 par* `def show_mirror(s): print(s,mirror(s))`*.*
3. *Écrivez une fonction* `is_palindrome(s)` *qui retourne* `True` *si* `s` *est un palindrome, autrement dit un mot qui est égal à son image miroir. Proposez une solution qui n'utilise pas la fonction* `mirror` *et qui ne crée aucune nouvelle chaîne de caractères.*

□

Exercice 2 (La disparition, ⋆⋆)

1. *Écrivez une fonction* `nb_occurs(c,s)` *qui prend en paramètre un caractère* `c` *et une chaîne de caractères* `s` *et qui renvoie le nombre de fois où* `c` *apparait dans* `s`*. Par exemple,* `nb_occurs('e','les revenentes')` *renvoie 5.*
2. *En déduire une fonction* `has_no_e(s)` *qui prend en paramètre une chaîne de caractères* `s` *et renvoie* `True` *si* `s` *ne contient ni le caractère* `e` *ni* `E`*.*
3. *Si* n *est le nombre de caractères de* `s`*, quelle est la complexité*[1] *de votre solution ?*
4. *Proposez une autre solution qui n'a cette complexité que dans le cas où la fonction renvoie* `True`*, mais potentiellement une meilleure complexité quand elle renvoie* `False`*.*

□

Exercice 3 (Pangrammes, ⋆⋆)

Écrivez une fonction `est_pangramme(s)` *qui prend en paramètre une chaîne de caractères* `s` *et qui renvoie* `True` *si* `s` *contient toutes les lettres de l'alphabet (on ne tient pas compte des caractères accentués).*

```
>>> alphabet = 'abcdefghijklmnopqrstuvwxyz'
>>> est_pangramme(alphabet)
True
>>> est_pangramme('Portez ce vieux whisky au juge blond qui fume.')
True
```

Indication : vous pourrez utiliser la chaîne de caractères contenue dans la variable alphabet ci-dessus, ainsi que la méthode s.lower(), ou (plus compliqué) vous chercherez à générer la chaîne `alphabet` *à l'aide de* `chr` *et* `ord`*.*

1. Pour évaluer la complexité, on évaluera le nombre d'accès mémoire : lecture/écriture de variable, lecture d'un caractère dans une chaîne de caractères, etc.

□

Exercice 4 (Cryptographie antique, ⋆⋆)
Un système cryptographique ancien, souvent appelé code de César, consiste à choisir une clé entière k entre 1 et 25 pour fabriquer, à partir d'un message msg, un nouveau message codé avec la technique suivante. Chaque lettre majuscule de msg est décalée de k positions vers la droite (l'alphabet est circulaire : après 'Z' on revient sur 'A'). Les autres caractères du message sont laissés intacts !

1. *Programmez la fonction* `code_cesar(msg,k)` *qui retourne le message codé.*

```
>>> code_cesar('ENVOYEZ 36 HOMMES !',3)
'HQYRBHC 36 KRPPHV !'
```

2. *Sur le même modèle, programmez la fonction* `decode_cesar(msg,k)` *qui prend un message codé et retourne le message en clair.*
3. *Défi urgent : décodez le message 'JLGVI XRJFZC' dont César a perdu la clé !*

□

Exercice 5 (Table ASCII, ⋆⋆)
Écrivez un programme qui affiche la table ASCII ci-dessous en respectant la mise en page.

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

□

Exercice 6 (Print Proust, ⋆⋆⋆)
Écrivez une fonction `justify_left(s,n)` *qui prend une chaîne de caractères* `s` *contenant une phrase (sans aucun caractère de contrôle) et qui l'affiche à l'écran en mettant au plus* `n` *caractères par ligne, en justifiant à gauche.*

```
>>> justify_left('Mais au lieu de la simplicité, c'est le faste que...' , 12)
Mais au lieu
de la
simplicité
c'est le
faste que...
```

Testez votre programme en affichant cette phrase de Proust en entier sur 80 caractères par ligne (demandez la phrase à Google). □