1 Manipulation de chaînes de caractères

Bloc-Note 1

☞ Une chaîne de caractères est une séquence éventuellement vide de caractères. 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 $0 \leqslant i < j$ ou `c[i:j:-1]` si $i < j \leqslant -1$.

```
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'
```

☞ Les chaînes de caractères possèdent aussi beaucoup de fonctions / méthodes spécifiques qu'on peut découvrir 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'
```

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
```

Bloc-Note 2

- Les caractères sont codés en machine par des nombres, la correspondance entre caractères et nombres s'appelle un codage. Le premier codage des caractères était le codage **ASCII**, il permettait de coder des caractères de mise en forme du texte hérités de la machine à écrire (retour chariot, espace ...), les chiffres, les alphabets romains minuscules et majuscules et les principaux caractères de ponctuation. Il était conçu pour coder des textes en Anglais. Le standard actuel **Unicode** permet désormais de coder 128 172 caractères.
- 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'
```

Exercice 2

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

1. Compléter le code de la fonction `histo(chaine)` qui prend en paramètre une chaîne de caractères (sans caractères accentués ou spéciaux) et qui retourne l'histogramme de la distribution dans la chaîne des 26 caractères de l'alphabet romain.

   ```
   In [11]: histo("Une hirondelle en ses voyages avait beaucoup appris.")
   Out[11]: [5, 1, 1, 1, 7, 0, 1, 1, 3, 0, 0, 2, 0, 3, 3, 3, 0, 2, 4, 1, 3, 2, 0,
        0, 1, 0]
   ```

   ```
   def histo(chaine):
       chaine = chaine.lower() #conversion en miuscules
       alphabet = ''.join([chr(ord('a') + k) for k in range(26)])
       t = [0] * 26          #liste a completer avec les effectifs
       for c in chaine:
           #a completer
       return t
   ```

2. Utiliser la fonction précédente pour écrire une fonction `anagrammeV1(mot1, mot2)` qui retourne un booléen indiquant si deux mots sans caractères accentués ou spéciaux, sont des anagrammes.

```
In [28]: anagrammeV1('Salavador Dali', 'Avida Dollars')
Out[28]: True
```

3. La fonction précédente ne peut pas déterminer si deux chaînes quelconque sont des anagrammes. Donnez un contre-exemple.

Pour déterminer si deux chaînes sont des anagrammes, il suffit de comparer leurs signatures selon un critère dépendant uniquement des caractères présents dans la chaîne. L'histogramme de la question **1.** est une signature valable mais on peut coder 128 172 caractères en Unicode, et ce n'est pas économe en espace mémoire de comparer des listes avec autant d'éléments.

Un algorithme plus économe en espace mémoire et de complexité temporelle raisonnable consiste à trier chaque chaîne dans l'ordre lexicographique ce qui constitue une signature valable.

a. Définir en quelques lignes les complexités d'un algorithme en espace mémoire ou en temps.
Ressources : https://interstices.info/jcms/c_5776/qu-est-ce-qu-un-algorithme

b. Qu'appelle-t-on ordre lexicographique ?
Un algorithme de tri permet-il uniquement de trier des nombres ou des caractères ?
Citer au moins trois algorithmes de tri.
Existe-t-il une complexité temporelle optimale pour un algorithme de tri ?
Ressources : https://interstices.info/jcms/c_6973/les-algorithmes-de-tri

c. En Python, la fonction permettant de trier une séquence d'objets comparables (nombres dans une liste ou caractères dans une chaîne) est `sorted`. Elle retourne la liste des objets dans l'ordre croissant :

```
In [16]: sorted('dceaj')
Out[16]: ['a', 'c', 'd', 'e', 'j']
```

En déduire l'écriture d'une fonction `anagrammeV2(chaine1, chaine2)` qui retourne un booléen indiquant si deux chaînes quelconques sont des anagrammes, sans tenir compte de la casse (distinction entre majuscule et minuscule).

```
In [30]: anagrammeV2('La crise économique', 'Le scénario comique')
Out[30]: True
```

2 Enjeux sociétaux et droit du numérique

Exercice 4 *Sujets d'exposés*

Par groupe de trois, traiter l'un des sujets suivants sous forme de présentation numérique (diaporama Impress, page web HTML, prezi . . .) que vous devrez présenter en dix minutes devant la classe au cours du premier trimestre.

Pour la réalisation de diaporama avec Impress, la ressource suivante est très bien : https://dane.ac-lyon.fr/spip/Video-Prise-en-main-d-Impress.

- Du codage au chiffrement, mathématiques et cybersécurité .

  Ressources : https://interstices.info/jcms/p_87300/skyfall-tombe-du-ciel

- Chiffrement et rançongiciels, faut-il s'inquiéter pour nos données ?

  Ressources : https://www.cybermalveillance.gouv.fr/experience/