Chaînes de caractères — TP19

Exercice 1 — Le cours

Les chaînes de caractères sont un type de variables Python permettant de stocker du texte. Leur usage est très similaire à celui des listes. Voici un bref résumé des notions à connaître :

1. On place des chaînes de caractères entre guillemets ou apostrophes. Par exemple `c = "Salut !"` est une chaîne de caractères.
2. Majuscules et minuscules sont différentes en Python et les espaces comptent. Ainsi la chaîne `c` précédente est différente de `c_bis = "salut !"` ou de `c_ter = "Salut!"`.
3. On peut accéder au nombre d'éléments d'une chaîne de caractères avec la fonction `len`. Par exemple `len("Salut !")` vaut 7.
4. Les éléments d'une chaîne sont accessibles par la commande `chaine[k]` où `k` est l'indice de l'élément dans la chaîne. Comme pour les listes, si une chaîne compte n caractères alors ses éléments sont indicées de 0 à $n-1$.

Q1 Dans la console, définissez la chaîne `c = "le travail"`. Que valent `c[1]`, `c[2]`, `c[3]` ? `c[-1]` ?

Q2 Comme pour les listes, on peut concaténer les chaînes de caractères avec l'opération `+` ou les répéter avec l'opération `*`. Dans la console, définissez la chaîne `d = "vive "`. Faites ensuite des opérations sur les chaînes `c` et `d` pour obtenir la chaîne :

```
"vive le travail vive le travail vive le travail vive le travail"
```

Reprenons les notions à connaître sur les chaînes :

5. Les chaînes de caractères sont, comme les listes, des objets *itérables* : on peut les parcourir à l'aide d'une boucle `for`. Comme pour les listes, on peut envisager un parcours par éléments (`for x in chaine`) ou par indices (`for k in range(n)` où `n = len(chaine)`).

Q3 Écrire une fonction `nombre_a` prenant en argument une chaîne de caractères `c` et renvoyant le nombre de fois que `c` contient la lettre `a`. Par exemple `nombre_a("banane")` doit renvoyer 2. On utilisera un parcours par éléments.

Q4 Écrire une fonction `place_a` prenant en argument une chaîne de caractères `c` et renvoyant la liste des indices auxquels `c` contient la lettre `a`. Par exemple `place_a("banane")` doit renvoyer `[1,3]`.

Finissons ce mini-cours sur les chaînes de caractères en mettant en évidence une différence entre les listes et les chaînes :

6. Contrairement aux listes, les chaînes de caractères sont des objets *non mutables*. Cela signifie qu'on ne peut pas modifier une chaîne de caractères. Par exemple, si on essaye de modifier la chaîne `c = "bonzour"` en remplaçant le z par un j, alors en tapant `c[3]="j"` dans la console, on obtient (*compléter)* :

7. Ce caractère non mutable implique également qu'il n'y a ni commande `pop` ni commande `append` pour les chaînes de caractères. Pour créer une chaîne de caractères, on peut toutefois reprendre la méthode "liste vide puis `append`" qu'on utilisait pour les listes. Il suffit pour cela d'initialiser une chaîne `c` à la *chaîne vide* : `c = ""`, puis d'utiliser des concaténations successives `c = c + ...`. Une dernière remarque : attention dans ce contexte à ne pas confondre la chaîne vide `""` avec la chaîne contenant le caractère *espace* `" "`.

Q5 Écrire une fonction `espacer` prenant en argument une chaîne de caractères et renvoyant la chaîne similaire mais où tous les caractères sont suivis du symbole *espace* `" "`. Par exemple `esapcer("bcpst")` doit renvoyer la chaîne `"b c p s t "`.

Exercice 2 Pratiquons

Les questions ci-dessous sont indépendantes.

Q1 Écrire une fonction `premier_dernier` prenant en argument une chaîne de caractères et renvoyant son premier et son dernier caractères. Par exemple, `premier_dernier("banane")` doit renvoyer le couple `("b","e")`.

Q2 Écrire une fonction `presence` prenant en argument une chaîne de carcatères `texte` et un caractère `lettre` et renvoyant `True` si `lettre` apparaît dans `texte` et `False` sinon. Par exemple, `presence("banane","a")` doit renvoyer `True` et `presence("banane","z")` doit renvoyer `False`.

Q3 Écrire une fonction `trois_lettres` prenant en argument une chaîne de caractères `texte` de longueur supérieure à 5 et renvoyant la chaîne de caractères constituée des caractères de `texte` situés aux positions 1, 3 et 4. Par exemple, `trois_lettres("je suis Voldemort")` doit renvoyer `"esu"`.

Q4 Écrire une fonction `extrait` prenant en argument une chaîne de caractères `texte` et une liste d'entiers `L` et renvoyant la chaîne de caractères constituée des caractères de `texte` situés aux positions indiquées dans `L`. Par exemple, `extrait("je suis Voldemort",[0,1,2,8,9,10,12])` doit renvoyer la chaîne `"je Vole"`.

Q5 Écrire une fonction `premier_mot` prenant en argument une chaîne de caractères `texte` contenant potentiellement des espaces et renvoyant le premier mot qu'elle contient. Par exemple, si `texte = "bonjour à tous"` alors `premier_mot(texte)` doit renvoyer `"bonjour"`.

Q6 Écrire une fonction `miroir` prenant en argument une chaîne de caractères et renvoyant la chaîne inversée. Par exemple `miroir("plouf")` doit renvoyer `"fuolp"`.

Q7 En utilisant la fonction précédente, écrire une fonction `palindrome` prenant en argument une chaîne de caractères et renvoayant `True` si c'est un palindrome et `False` sinon. Par exemple `palindrome("kayak")` doit renvoyer `True`.

Exercice 3 Faj Hjxfw !

Lorsqu'on dispose d'un texte, il peut être intéressant d'étudier la fréquence d'apparition de chacune des lettres de l'alphabet dans celui-ci.

Q1 Écrire une fonction `frequence` prenant en argument un texte et une lettre et renvoyant la fréquence d'apparition de la lettre dans le texte. Par exemple, `frequence("banane","a")` doit renvoyer $\frac{2}{6} = 0,3333$ car il y a deux a parmi les six lettres du texte `"banane"`.

Récupérez une variable `alphabet` contenant une chaîne de caractères dont les éléments sont toutes les lettres de l'alphabet via la commande suivante :

```
1 import string
2 alphabet = string.ascii_lowercase
```

Q2 Vérifiez le contenu de la variable `alphabet`.

Q3 Écrire une fonction `list_frequence` prenant en argument un texte et renvoyant la liste dont les éléments sont les fréquences d'apparition de chacune des lettres de l'alphabet dans le texte. Ainsi, `list_frequence(texte)` doit renvoyer une liste dont le premier élément est la fréquence d'apparition de la lettre a dans `texte`, le deuxième celle de la lettre b, etc.

Pour tester votre fonction, on propose d'utiliser un texte fourni sous forme d'un fichier `.txt` sur cahier de prépa. Téléchargez le fichier `TP19donnees.txt` sur cahier de prépa, ouvrez-le avec le "bloc note" et enregistrez-le dans votre dossier personnel. Ensuite, recopiez et complétez la commande suivante dans le script pour ouvrir le fichier dans Spyder :

```
f = open(r"chemin du fichier à compléter")
```

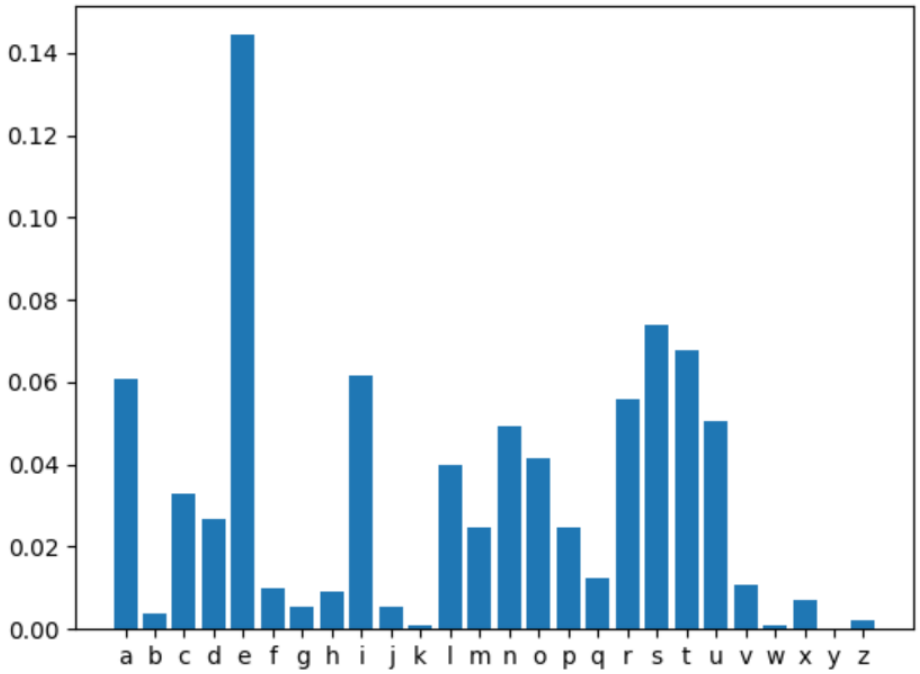
Dans la console, testez la commande `f.readline()` plusieurs fois : à quoi sert cette commande ?

Recopiez ensuite le code suivant dans le script :

```
ligne1 = f.readline()
ligne2 = f.readline()
ligne3 = f.readline()
ligne4 = f.readline()
```

Q4 On note la présence du caractère `\n` signalant un saut de ligne. On souhaite définir une nouvelle variable `test` contenant la chaîne de caractères de la ligne 2 mais sans le symbole `\n`. Pour cela, récupérez la longueur de la chaîne puis définissez une variable `test` grâce à une sous-chaîne.

Q5 Tester votre fonction avec le texte fourni en exemple dans le fichier `TP19_donnees.py` disponible sur cahier de prépa. Tracer alors le graphe présentant la fréquence d'apparition des lettres dans ce texte sous forme d'un histogramme. Pour tracer un histogramme "en bâtons" on utilisera la commande `plt.bar(absi,ordo)` à la place de `plt.plot(absi,ordo)`. *Vous devez obtenir le dessin ci-dessous.*



Sur ce graphe, on constate sans surprise que la lettre e est de loin la plus utilisée en Français. Cette remarque peut être utilisée pour décrypter un code appelé "chiffrement de César".

Un chiffrement de César est une méthode de cryptographie consistant à coder un texte en décalant chacune de ses lettres d'un nombre fixé de lettres vers la droite. Pour les lettres de la fin de l'alphabet, on reprend l'alphabet au début. Par exemple, si on décide de choisir un décalage de 5 alors tous les a du texte initial seront transformés en f, tous les b en g, tous les c en h, etc. En fin d'alphabet, les u sont transformés en z, puis les v en a, les w en b, etc. On peut représenter cela en complétant le tableau de correspondance des lettres :

message initial	A	B	C	D	...	T	U	V	W	X	Y	Z
message codé	F	G	H	I	...	Y	Z	A	B	C	D	E

Q6 *Question sans ordinateur.* Si on choisit un décalage de 5, comment est codé le message `"ave cesar"` ?

Q7 *Question sans ordinateur.* Toujours pour un décalage de 5, si une lettre du message initial est à la position k dans l'alphabet ($k \in [\![0, 25]\!]$ puisqu'il y a 26 lettres dans l'alphabet Français), quel sera, en fonction de k, la position de la lettre correspondante dans le message codé ? Comment écrire cela rapidement en Python grâce à une opération sur les entiers ?

Q8 Écrire une fonction `indice` prenant en argument une lettre de l'alphabet et renvoyant son indice dans l'alphabet (c'est-à-dire dans la chaîne de caractères `alphabet`). Par exemple, `indice("e")` doit renvoyer 4.

Q9 Écrire une fonction `codage` prenant en argument un texte et un nombre entier correspondant au décalage à effectuer et renvoyant le texte codé correspondant. Par exemple, `codage("ave cesar",5)` doit renvoyer `"faj hjxfw"`.

Supposons maintenant ne pas connaître le décalage utilisé pour coder le message. Pour décoder le message, on peut, si le texte est assez long, utiliser une approche fréquentielle. En analysant les fréquences d'apparition des lettres de l'alphabet dans le message codé, on peut repérer la lettre la plus utilisée. Si le message est écrit en Français, cette lettre est surement le e. Cela nous permet alors de trouver le décalage utilisé et donc de décoder le message.

Q10 Décodez le message mystère donné à la quatrième ligne du fichier `TP19donnees.text`. *On ne demande pas d'écrire une fonction permettant de décoder n'importe quel message, mais d'utiliser les questions précédentes pour décoder ce message en particulier.*