Chaînes de caractères — TP10

Exercice 1 — Echauffement

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"`.

Exercice 2 — 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"`.

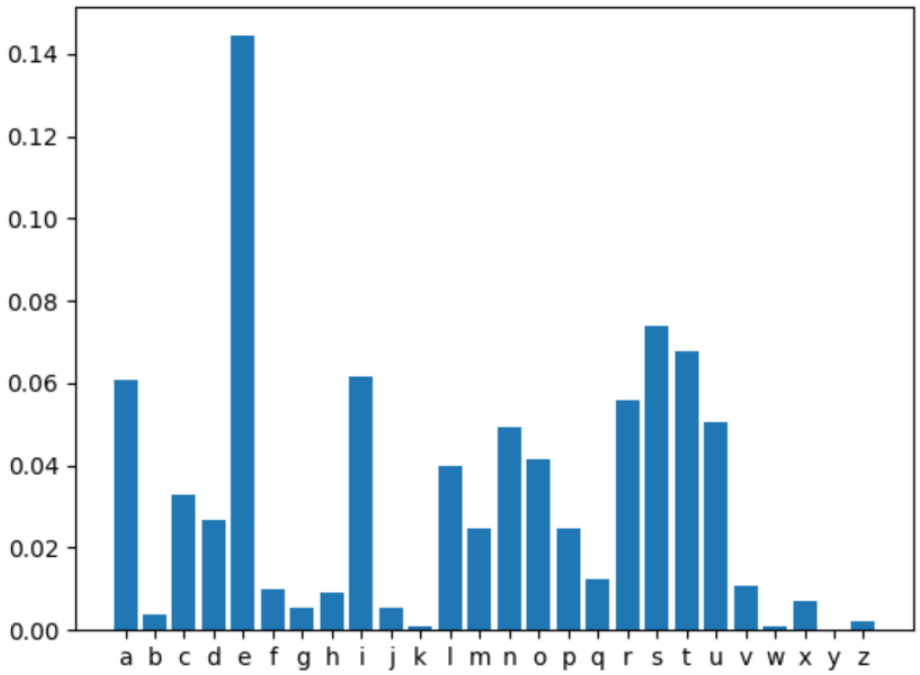
Récupérez une variable `alphabet` contenant une chaîne de caractères dont les éléments sont toutes les lettres de l'alphabet via la commande suivante :

```
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.

Q4 Tester votre fonction avec le texte fournit en exemple dans le fichier `TP10_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

Q5 *Question sans ordinateur.* Si on choisit un décalage de 5, comment est codé le message `"ave cesar"` ?

Q6 *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 ?

Q7 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.

Q8 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.

Q9 Décodez le message `mystere` donné dans le fichier `TP10_donnees.py`. *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.*

Exercice 3 Dernières Nouvelles d'Alsace

L'objectif de cet exercice est de manipuler les listes Python pour modéliser des brins d'ADN et les séquences d'acides aminés correspondants.

Pour rappel, l'ADN est une macromolécule constituée de brins formés d'un enchaînement de nucléotides contenant les bases de l'ADN : Adénine (A), Thymine (T), Guanine (G) et Cytosyne (C).

Dans tout l'exercice, on représentera un brin d'ADN par une liste dont les éléments sont les chaînes de caractères `'A'`, `'T'`, `'C'` ou `'G'`. Par exemple, `['A','C','A','T','A','G']` représente un brin d'ADN formé de 6 bases.

Q1 Écrire une fonction `adn` générant un brin d'ADN aléatoire de longueur `n`. Pour générer une base aléatoire, on utilisera la bibliothèque `random` dans laquelle la commande `random.choice(L)` permet de sélectionner de manière équiprobable un élément de la liste `L`.

Pour aboutir à la chaîne des acides aminés, un brin d'ADN est tout d'abord transcrit en ARNm. Il s'agit d'un autre brin, de même longueur et dont les bases sont : Adénine (A), Uracile (U), Guanine (G) et Cytosyne (C). La correspondance entre les bases de l'ADN et celles de l'ARNm est donnée ci-contre.

ADN	ARNm
Adénine (A)	Uracile (U)
Thymine (T)	Adénine (A)
Guanine (G)	Cytosyne (C)
Cytosyne (C)	Guanine (G)

Q2 Écrire une fonction `adn_to_arnm` réalisant la transcription d'une séquence d'ADN en la séquence d'ARNm correspondante. Cette fonction modifiera directement la séquence d'ADN fournie. Par exemple, si `L=['A','T','C','G','A','T']` alors après exécution de `adn_to_arnm(L)`, `L` vaut `['U','A','G','C','U','A']`.

Les bases de l'ARNm sont ensuite rassemblées en codons : un codon est un groupe de 3 bases consécutives. Chaque codon code alors un acide aminé selon les correspondances données ci-contre et disponibles dans le fichier `TP10_donnees.py`. Celle-ci est donnée sous la forme de deux listes : une liste `codons` et une liste `aa`. La correspondance se lit alors de la manière suivante : pour tout k, le codon `list_codons[k]` code l'acide aminé `list_aa[k]`. Notez que chaque codon et chaque acide aminé sont représentés par une seule chaîne de caractères.

codon de l'ARNm	acide aminé
AGG	Arginine (Arg)
AGA	Arginine (Arg)
AGC	Sérine (Ser)
AGU	Sérine (Ser)
AAG	Lysine (Lys)
...	...

```
1 list_codons = ['AGG', 'AGA', 'AGC', 'AGU', 'AAG', 'AAA', 'AAC',...]
2 list_aa = ['Arg','Arg','Ser','Ser','Lys','Lys','Asp',...]
```

Q3 Écrire une fonction `bases_to_codon` prenant en argument une liste à 3 éléments correspondant à 3 bases successives de l'ARNm et renvoyant la chaîne de caractères représentant le codon correspondant. Par exemple, `base_to_codon(['A','C','G'])` doit renvoyer `'ACG'`.

Q4 En déduire une fonction `arnm_to_codons` prenant en argument une séquence ARNm d'une longueur qu'on supposera divisible par 3, et renvoyant la séquence de codons correspondants. Par exemple, `arnm_to_codons(['A','C','G','T','U','C']` doit renvoyer `['ACG','TUC']`.

Q5 Écrire une fonction `arnm_to_aa` permettant de convertir une séquence d'ARNm en séquence d'acides aminés. Par exemple, `arnm_to_aa(['A','G','G','A','G','U'])` devra renvoyer la liste `['Arg','Ser']`. On utilisera les listes `list_codons` et `list_aa` téléchargées depuis cahier de prépa.