

TD n° 5

1 Découpage de chaînes de caractères

Exercice 1. Découpage en deux sous-chaînes

Écrivez une fonction ***decoupage2(ch, ch1, ch2)*** qui teste si la chaîne de caractère *ch* peut être découpée selon les deux chaînes *ch1* et *ch2*.

Par exemple :

- *decoupage2("tartuffe", "tarte", "truffe")* renverra False,
- *decoupage2("bateau", "bat", "eau")* renverra True,
- *decoupage2("bateau", "eau", "bat")* renverra True également.

Exercice 2. Découpage en trois sous-chaînes

Écrivez une fonction ***decoupage3(ch, ch1, ch2, ch3)*** qui teste si la chaîne de caractère *ch* peut être découpée selon les trois chaînes *ch1*, *ch2* et *ch3*.

Par exemple :

- *decoupage3("abcdef", "ab", "f", "cde")* renverra True,
- *decoupage3("abcdef", "abc", "cde", "f")* renverra False.

Exercice 3. Découpage en *n* sous-chaînes

Supposons qu'on nous demande d'écrire une fonction ***decoupage(ch, liste)*** qui testerait si la chaîne *ch* est une concaténation des *n* chaînes de la liste passée en paramètres.

1. Combien de tests d'égalité devra-t-on réaliser, dans le pire des cas, pour $n = 4$?
2. Combien de tests d'égalité devra-t-on réaliser, dans le pire des cas, pour *n* quelconque ?
3. Que penser d'un tel algorithme ?

2 Comptage dans une chaîne, une liste ou un tuple

La méthode *count* est à disposition des chaînes de caractère, tuples et listes. Elle retourne le nombre d'occurrences de l'objet passé en paramètre dans la structure qui appelle la méthode.
exemples :

- pour `ch = 'programme'`
 - `ch.count('m')` retourne 2
 - `ch.count('e')` retourne 1
 - `ch.count('t')` retourne 0
- pour `l = (3, 1, 7, 2, 5, 4, 0, 9, 3)`
 - `l.count(3)` retourne 2
 - `l.count(7)` retourne 1
 - `l.count(8)` retourne 0
- pour `t = ['a', 1, 6, 'a', 2, 0]`
 - `t.count('a')` retourne 2
 - `t.count(2)` retourne 1
 - `t.count(8)` retourne 0

Exercice 1. Implémentation manuelle d'une fonction de comptage

Écrivez une fonction *myCount(iterable, element)* qui prend en paramètres un itérable (str, list ou tuple) et un élément (une chaîne de caractères ou un nombre par exemple) et qui retourne le nombre d'occurrences de l'élément dans l'itérable. Cette fonction n'utilisera pas la méthode *count*.

Exercice 2. Élément majoritaire

Écrivez une fonction *element_majoritaire(it)* qui retourne l'élément qui apparaît le plus souvent dans l'itérable *it*. Si plusieurs éléments sont majoritaires à égalité, la fonction renvoie l'un d'eux.

Par exemple :

- `element_majoritaire("programmer")` renverra "r",
- `element_majoritaire((3, 1, 7, 2, 5, 4, 0, 9, 3))` renverra 3,
- `element_majoritaire(['a', 1, 6, 'a', 2, 0])` renverra "a".

Exercice 3. Anagrammes (permutations)

Écrivez une fonction *anagramme(ch1, ch2)* qui teste si deux chaînes *ch1* et *ch2* sont anagrammes l'une de l'autre (autrement dit, si les chaînes contiennent exactement les mêmes lettres, mais dans un ordre différent).

Par exemple :

- `anagramme("bourges", "bruges")` renverra False,
- `anagramme("orleans", "orelsan")` renverra True,
- `anagramme("aa", "aab")` renverra False.

On ne comparera que des chaînes de caractères minuscules et sans accent.

On pourra réutiliser les fonctions précédentes, notamment *myCount* (ou encore la méthode *count*).

Votre fonction est-elle adaptable aux permutations de listes et de tuples ?

3 Expressions formatées

Exercice 1. Nom de variable correct

Dans la plupart des langages de programmation, les noms de variables doivent respecter quelques consignes. On dit qu'une chaîne de caractères est un "identifiant correct" lorsque :

- elle est constituée uniquement de chiffres, de lettres ou de tirets ("-", "_") (pas d'espaces ni de caractères spéciaux),
- elle ne commence pas par un chiffre,
- elle n'appartient pas à la **liste** des mots réservés du langage. On suppose que cette liste est nommée "reserved" et est donnée dans le programme. Elle contient notamment les mots "if", "then", "while", "return", etc.

Écrivez une fonction *identifiant_correct(ch)* qui retourne un booléen qui dit si oui ou non la chaîne *ch* est un identifiant correct.

Exercice 2. Adresses mail

On suppose qu'une adresse mail est correcte si elle est de la forme **ch1@ch2.ch3** avec les conditions suivantes :

- la chaîne *ch1* peut contenir des chiffres, des minuscules, des tirets (`_` et `-`) mais pas de caractère spécial (`!`, `?`, `%`, etc) ni de ponctuation.
- *ch1* doit commencer par une minuscule,
- *ch1* doit se terminer par une minuscule ou un chiffre,
- *ch2* ne contient que des minuscules,
- *ch3* ne contient que des minuscules (au moins 2, au plus 4).

Écrivez une fonction *mail_correct(ch)* qui teste si la chaîne *ch* est une adresse mail correcte au sens des contraintes données ci-dessus.

Vous pourrez organiser votre travail en écrivant éventuellement d'autres fonctions.

4 Acide Ribonucléique (ARN)

Un brin d'ARN, représenté dans une chaîne de caractères, est bien formé si :

- il est uniquement composé de 'A' (Adénine), 'C' (Cytosine), 'G' (Guanine) et 'U' (Uracile) ;
- le nombre de lettres est un multiple de 3, avec au moins 6 lettres.

Par exemple, "ACAGUG" et "AAACGUACG" sont des brins corrects, mais pas "AAACGUCU" (8 lettres) ni "ABCGUCAA" (lettre B).

Pour chaque question, vous pourrez donner :

- une version où vous n'utiliserez ni boucle ni instruction conditionnelle, mais où vous pourrez utiliser la méthode *count* des chaînes de caractères ;
- une version avec boucle et test.

Exercice 1. Longueur correcte

Écrivez une fonction *longueur_correcte(brin)* qui prend en paramètre une chaîne de caractères *brin* et qui retourne un booléen qui dit si oui ou non la chaîne a la longueur requise pour représenter un brin d'ARN correct.

Exercice 2. Composition correcte

Écrivez une fonction *composition_correcte(brin)* qui prend en paramètre une chaîne de caractères *brin* et qui retourne un booléen qui dit si oui ou non la chaîne n'est constituée que des composants autorisés.

Exercice 3. Brin correct

Écrivez une fonction *est_correct(brin)* qui prend en paramètre une chaîne de caractères *brin* et qui retourne un booléen qui dit si oui ou non la chaîne représente un brin d'ARN correct.

L'ARN est organisé en séquences de 3 caractères appelées codons.

Par exemple, dans le brin "ACAGUGACU", il y a les 3 codons "ACA", "GUG" et "ACU".

Exercice 4. Les codons d'un brin

Écrivez une *codons(brin)* qui, si la chaîne *brin* représente un brin d'ARN valide, affiche le nombre total de codons puis la liste des codons. Si la chaîne *brin* ne représente pas un brin valide, alors la fonction affiche le message "chaîne non valide".

Par exemple :

- `codons("UUUCCAACUUA")` affiche "4 codons" puis les codons UUU, CCA, ACU et UAA,
- `codons("UUUCCAACUUA")` affiche "chaîne non valide",
- `codons("AGAUUU")` affiche "2 codons" puis les codons AGA et UUU,
- `codons("UGU")` affiche "chaîne non valide".

Pour l'affichage des codons, vous pourrez revoir les sous-chaînes de caractères dans le cours.