

TP 2.2 – Chaînes de caractères

Objectifs –

- ◊ Savoir parcourir une chaîne de caractères.
- ◊ Savoir construire une nouvelle chaîne de caractères, caractère par caractère.
- ◊ Savoir utiliser les fonctions `ord` et `chr`.
- ◊ Savoir utiliser quelques méthodes usuelles sur les chaînes, comme `lower()`.
- ◊ Choisir un parcours adapté selon le problème posé.

Dans ce TP, on n'utilise pas les méthodes qui résolvent directement les questions posées, notamment `count`, `find`, `index` et `startswith`. On n'utilise pas non plus le slicing `texte[::-1]` pour renverser une chaîne.

Parcours principal

Exercice 1 – Recherche dans un texte

Le code ASCII, prononcé « ASKI », est un standard historique de codage informatique des caractères. Il s'agit d'une table de correspondance entre des caractères et des entiers. Les lettres majuscules usuelles sont codées par des entiers successifs. En Python, la commande `chr(i)` renvoie le caractère associé à l'entier `i`, tandis que la commande `ord(car)` renvoie l'entier associé au caractère `car`. Pour les lettres majuscules usuelles, ces entiers correspondent aux codes ASCII.

Question 1.

- 1.1. Déterminer les entiers correspondant aux caractères "a", "z", "A" et "Z".
- 1.2. En déduire une fonction `est_majuscule(car: str) -> bool` qui renvoie `True` si le caractère `car` est une lettre majuscule non accentuée, et `False` sinon. On supposera que `car` contient exactement un caractère.

Question 2. Écrire une fonction `compte_occurrences(texte: str, car: str) -> int` qui renvoie le nombre d'occurrences du caractère `car` dans le texte `texte`. On supposera que `car` contient exactement un caractère.

Par exemple :

- ◊ `compte_occurrences("aBcDGe", "B")` renvoie 1;
- ◊ `compte_occurrences("aBcDGe", "A")` renvoie 0;
- ◊ `compte_occurrences("ABRACADABRA", "A")` renvoie 5.

Exercice 2 – Autour des palindromes

Un *palindrome* est un texte qui peut être lu indifféremment de gauche à droite ou de droite à gauche. Par exemple, le mot *radar* est un palindrome.

Question 1. Écrire une fonction `renverse(texte: str) -> str` qui renvoie le texte obtenu en lisant `texte` de droite à gauche.

Par exemple :

- ◊ `renverse("texte a lire")` renvoie "eril a etxet";
- ◊ `renverse("radar")` renvoie "radar";
- ◊ `renverse("")` renvoie "".

Question 2. En utilisant la fonction `renverse`, écrire une fonction `est_palindrome_1(texte: str) -> bool` qui renvoie `True` si le texte `texte` est un palindrome, et `False` sinon.

Par exemple :

- ◊ `est_palindrome_1("radar")` renvoie `True`;
- ◊ `est_palindrome_1("Radar")` renvoie `False`;
- ◊ `est_palindrome_1("texte")` renvoie `False`.

Question 3. La méthode `lower` transforme en minuscules tous les caractères d'une chaîne. Par exemple, si la variable `texte` contient "textE", alors `texte.lower()` renvoie "texte".

Écrire une fonction `est_palindrome_2(texte: str) -> bool` qui renvoie `True` si le texte `texte` est un palindrome, et `False` sinon, après avoir transformé les majuscules en minuscules.

Par exemple :

- ◊ `est_palindrome_2("Radar")` renvoie `True`;
- ◊ `est_palindrome_2("RADAR")` renvoie `True`;
- ◊ `est_palindrome_2("texte")` renvoie `False`.

Si le temps le permet

Exercice 3 – Occurrences des lettres majuscules

Question 1.

- 1.1. Écrire une fonction `liste_maj(texte: str) -> list` qui utilise la fonction `compte_occurrences` et qui renvoie la liste des nombres d'occurrences de chaque

lettre majuscule de l'alphabet dans le texte `texte`.

La case d'indice 0 correspond à la lettre "A", la case d'indice 1 à la lettre "B", et ainsi de suite jusqu'à la case d'indice 25, qui correspond à la lettre "Z".

Par exemple, pour le texte `texte = "aBcDGe"`, l'appel `liste_maj(texte)` renvoie une liste de taille 26 qui commence par :

```
| [0, 1, 0, 1, 0, 0, 1, 0, ...]
```

■ **1.2.** Combien de fois le texte `texte` est-il parcouru lors de l'appel `liste_maj(texte)` ?

Question 2. Écrire une fonction `liste_maj_une_passe(texte: str) -> list` qui renvoie la même liste, mais en ne parcourant qu'une seule fois le texte.

On pourra commencer par créer une liste de 26 zéros :

```
| occurrences = [0] * 26
```

Lorsqu'un caractère majuscule `car` est rencontré, son indice dans la liste est donné par `ord(car) - ord("A")`.

Exercice 4 – Palindromes avec caractères ignorés

On désigne par `caracteres_exclus` une liste de caractères que l'on souhaite ignorer, comme les espaces, les apostrophes ou certains signes de ponctuation.

Question 1. Écrire une fonction `est_palindrome_3(texte: str, caracteres_exclus: list) -> bool` qui renvoie `True` si le texte `texte` est un palindrome, et `False` sinon, après avoir transformé les majuscules en minuscules et supprimé tous les caractères appartenant à la liste `caracteres_exclus`.

Par exemple :

- ◊ `est_palindrome_3("Esopé reste ici et se repose", [" "])` renvoie `True`;
- ◊ `est_palindrome_3("Engage le jeu que je le gagne", [" "])` renvoie `True`;
- ◊ `est_palindrome_3("texte a lire", [" "])` renvoie `False`.

Travail à la maison

Exercice 5 – Recherche d'un motif dans un texte

On appelle *motif* une chaîne de caractères que l'on cherche à retrouver dans un texte. Par exemple, le motif "abra" apparaît dans le texte "abracadabra" aux positions 0 et 7. Dans cet exercice, on n'utilise pas les méthodes `find` ou `index`.

Question 1. Écrire une fonction `commence_par(texte: str, motif: str) -> bool` qui renvoie `True` si le texte `texte` commence par le motif `motif`, et `False` sinon. Par exemple :

- ◊ `commence_par("abracadabra", "abra")` renvoie `True`;
- ◊ `commence_par("abracadabra", "cada")` renvoie `False`;
- ◊ `commence_par("abc", "")` renvoie `True`.

Question 2. Dans la suite de l'exercice, on suppose que le motif cherché est non vide. Écrire une fonction `motif_en_position(texte: str, motif: str, i: int) -> bool` qui renvoie `True` si le motif `motif` apparaît dans `texte` à partir de la position entière naturelle `i`, et `False` sinon. Par exemple :

- ◊ `motif_en_position("abracadabra", "abra", 0)` renvoie `True`;
- ◊ `motif_en_position("abracadabra", "abra", 7)` renvoie `True`;
- ◊ `motif_en_position("abracadabra", "abra", 3)` renvoie `False`.

Question 3. Écrire une fonction `positions_motif(texte: str, motif: str) -> list` qui renvoie la liste des positions auxquelles le motif `motif` apparaît dans `texte`. On compte aussi les occurrences qui se recouvrent. Par exemple :

- ◊ `positions_motif("abracadabra", "abra")` renvoie `[0, 7]`;
- ◊ `positions_motif("aaaa", "aa")` renvoie `[0, 1, 2]`;
- ◊ `positions_motif("abc", "d")` renvoie `[]`.

Question 4. En déduire une fonction `nombre_occurrences_motif(texte: str, motif: str) -> int` qui renvoie le nombre d'occurrences du motif `motif` dans `texte`.

Approfondissement facultatif

Exercice 6 – Chiffrement de César

Le *chiffrement de César* consiste à décaler les lettres de l'alphabet d'un nombre fixé de positions. Dans cet exercice, on ne chiffre que les lettres majuscules non accentuées, c'est-à-dire les caractères compris entre "A" et "Z". Les autres caractères sont laissés inchangés. Par exemple, avec un décalage de 3, la lettre "A" devient "D", la lettre "B" devient "E", et la lettre "Z" devient "C".

Question 1. Écrire une fonction `decale_maj(car: str, k: int) -> str` qui renvoie la lettre obtenue en décalant la lettre majuscule `car` de `k` positions. On utilisera les fonctions `ord` et `chr`. Le décalage doit être pris modulo 26.

Par exemple :

- ◇ `decale_maj("A", 3)` renvoie "D";
- ◇ `decale_maj("Z", 1)` renvoie "A";
- ◇ `decale_maj("C", -2)` renvoie "A".

Question 2. Écrire une fonction `chiffre_cesar(texte: str, k: int) -> str` qui renvoie le texte obtenu en appliquant le décalage `k` à toutes les lettres majuscules de `texte`. Les caractères qui ne sont pas des lettres majuscules sont conservés sans modification. Par exemple :

- ◇ `chiffre_cesar("VIVE ITC", 3)` renvoie "YLYH LWF";
- ◇ `chiffre_cesar("ABC XYZ", 1)` renvoie "BCD YZA".

Question 3. Comment déchiffrer un texte qui a été chiffré avec un décalage `k`? Tester cette idée sur plusieurs exemples.

Question 4. On suppose maintenant qu'un texte a été chiffré avec un décalage inconnu. Expliquer pourquoi il suffit de tester les 26 décalages possibles pour tenter de le déchiffrer.

Écrire une fonction `dechiffre_cesar_inconnu(texte: str) -> None` qui affiche les textes obtenus avec les décalages 0, -1, -2, ..., -25. On repère ensuite visuellement le texte lisible parmi les résultats affichés.