Correction du TP 4

Exercice 1 — Mary Poppins

- Le programme suivant affiche cent fois le mot «Supercalifragilisticexpialidocious».

```
for k in range(100):
# k va de 0 à 99, donc il y a 100 tours de boucle.
    print("Supercalifragilisticexpialidocious")
```

- Le programme suivant affiche le mot formé par la concaténation de cent «Supercalifragilisticexpialidocious».

```
print(100 * "Supercalifragilisticexpialidocious")
```

- Le programme suivant va trop loin.

```
mot = "Supercalifragilisticexpialidocious"
envers = mot[ : : -1] # Le mot à l'envers.
for k in range(100):
# k va de 0 à 99, donc il y a 100 tours de boucle.
    print(envers)
```

Exercice 2 — Zozotement

1. Pour trouver le nombre de lettres z dans une chaine de caractères, on crée un compteur de lettres `"z"` (initialisé à 0) puis on parcourt les caractères de la chaine de caractères : dès qu'on l'on rencontre le caractère `"z"`, on incrémente le compteur.
   À la fin du parcours, on renvoie la valeur du compteur.

   - Avec un parcours sur les indices des caractères de la chaine :

```
def nombre_de_z(chaine):
    """
    Entrée : une chaine de caractères chaine.
    Sortie : le nombre de 'z' (minuscule) dans la chaine chaine.
    """
    compteur = 0
    for k in range(len(chaine)):
    # Pour k allant de 0 à len(chaine) - 1,
    # c'est-à-dire pour k parcourant les indices
    # des caractères de chaine :
        if chaine[k]  == 'z':
        # si le caractère d'indice k est un "z",
            # alors on ajoute 1 au compteur de "z".
            compteur += 1
    return compteur
```

   - Avec un parcours sur les caractères directement :

```
def nombre_de_z_bis(chaine):
    """
    Entrée : une chaine de caractères chaine.
    Sortie : le nombre de 'z' (minuscule) dans la chaine chaine.
    """
    compteur = 0
    for lettre in chaine:
    # Pour lettre parcourant un à un
    # tous les caractères de la chaine :
        if lettre  == 'z':
        # si le caractère lettre est un "z",
            # alors on ajoute 1 au compteur de "z".
            compteur += 1
    return compteur
```

- On procède de la même manière pour compter le nombre d'occurrences d'un caractère donné quelconque dans une chaine de caractères.

```
def nombre_occurrences(chaine, caractere_recherche):
    """
    Entrées : une chaine de caractères chaine,
              un caractère caractere.
    Sortie : le nombre d'occurrences du caractère caractere
             dans la chaine de caractères chaine.
    """
    compteur = 0
    for lettre in chaine:
    # Pour lettre parcourant un à un
    # les caractères de chaine :
        if lettre == caractere_recherche:
            compteur += 1
    return compteur
```

Exercice 3 — Distance de Hamming

La *distance de Hamming* entre deux chaines de caractères de même longueur est le nombre de positions où elles diffèrent.

1. La fonction suivante prend en arguments deux chaines de caractères, renvoie un message d'erreur si les chaines de caractères n'ont pas la même longueur et renvoie la distance de Hamming entre les deux chaines sinon.

```
def Hamming(chaine1, chaine2):
    """
    Entrées : deux chaines de caractères chaine1 et chaine2.
    Sortie : la distance de Hamming entre chaine1 et chaine2
             si chaine1 et chaine2 ont la même longueur,
             un message d'erreur sinon.
    """
    if len(chaine1) != len(chaine2):
        return "Ces deux chaines de caractères n'ont pas la même
        longueur ! Leur distance de Hamming n'est pas définie."
```

```
    else:
        distance = 0
        for k in range(len(chaine1)):
        # Pour k allant de 0 à len(chaine1) - 1,
        # c'est-à-dire pour k parcourant les indices
        # des caractères des deux chaines.
            if chaine1[k] != chaine2[k]:
            # On compare les caractères de même indice k
            # dans les deux chaines.
                distance = distance + 1
                # On augmente la distance de Hamming de 1
                # si les chaines diffèrent à l'indice k.
        return distance
```

2. Deux chaines de caractères sont égales si et seulement si leur distance de Hamming est nulle.

```
def egalite(chaine1, chaine2):
    """
    Entrées : deux chaines de caractères chaine1 et chaine 2.
    Sortie : True si chaine1 et chaine2 sont égales,
             False sinon.
    """
    return Hamming(chaine1, chaine2) == 0 # C'est un booléen.
```

Exercice 4 — Palindromes

Un *palindrome* est un mot ou une phrase dont l'ordre des lettres reste le même si on le lit de gauche à droite ou de droite à gauche. C'est le cas par exemple des phrases «Engage le jeu que je le gagne» ou «Un roc si biscornu», si l'on ne tient compte ni des majuscules ni des espaces.
De plus, pour simplifier, on supposera que les phrases ne contiennent aucun signe de ponctuation ni aucune lettre accentuée.

1. La fonction suivante prend en entrée une chaine de caractères et renvoie la chaine de caractères dans laquelle les espaces ont été supprimées.

   Pour cela, on crée une chaine de caractères, initialement vide, dans laquelle on réécrit tous les caractères de la chaine de départ qui ne sont pas des espaces.

```
def sans_espace(chaine):
    """
    Entrée : une chaine de caractères chaine.
    Sortie : la chaine chaine sans les espaces.
    """
    resultat = "" # Au départ, la chaine-résultat est vide.
    for caractere in chaine:
    # La variable caractere parcourt un à un
    # tous les caractères de la chaine.
        if caractere != " ":
            resultat = resultat + caractere
            # Si le caractère n'est pas une espace,
            # on le concatène au résultat.
    return resultat
```

2. Pour inverser l'ordre des caractères d'une chaine de caractères, on peut procéder de deux manières différentes.
   - En utilisant le slicing :

```
def miroir(chaine):
    """
    Entrée : une chaine de caractères chaine.
    Sortie : la chaine chaine lue de droite à gauche.
    """
    return ch[ : : -1]
```

   - En partant d'une chaine vide et en concaténant un à un les caractères de la chaine de départ en début de chaine :

```
def miroir(chaine):
    """
    Entrée : une chaine de caractères chaine.
    Sortie : la chaine chaine lue de droite à gauche.
    """
    chaine_miroir = ""
    for caractere in chaine:
    # La variable caractere parcourt un à un
    # les caractères de la chaine.
        chaine_miroir = caractere + chaine_miroir
    return chaine_miroir
```

3. Afin de remplacer toutes les majuscules d'une chaine de caractères par les minuscules correspondantes, on crée deux chaines de caractères contenant toutes les lettres non accentuées de l'alphabet, en majuscules et en minuscules, dans le même ordre.

```
ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
alphabet = "abcdefghijklmnopqrstuvwxyz"
```

   On initialise la chaine-résultat à la chaine vide, puis on parcourt alors les caractères de la chaine de départ.
   À chaque étape, on détermine si le caractère considéré est une majuscule. Pour cela, on parcourt la chaine `ALPHABET` jusqu'à ce qu'on trouve le caractère considéré :
   - si on trouve un indice k tel que le caractère considéré soit le caractère `ALPHABET[k]`, alors on concatène à la chaine-résultat la minuscule correspondante, qui est `alphabet[k]` ;
   - sinon, on arrive au bout de la chaine `ALPHABET` sans trouver le caractère, c'est-à-dire que le caractère n'est pas une majuscule. On le concatène alors tel quel à la fin de la chaine-résultat.

```
def minuscule(chaine):
    """
    Entrée : une chaine de caractères chaine.
    Sortie : la chaine chaine écrite en minuscule.
    """
```

```
    resultat = ""
    for caractere in chaine:
    # La variable caractere parcourt un à un
    # tous les caractères de la chaine.
        k = 0
        while (k < 26) and (ALPHABET[k] != caractere):
        # Tant que l'entier k est bien strictement inférieur à 26
        # (c'est-à-dire tant que k est un indice de la chaine ALPHABET)
        # et tant que le caractère d'indice k dans ALPHABET
        # n'est pas la caractère considéré :
            k = k + 1
            # on incrémente k de 1
            # (pour passer à l'indice suivant dans ALPHABET).
        if k == 26:
        # Si lorsque la boucle while s'arrête,
        # on a parcouru toute la chaine ALPHABET,
        # le caractère considéré n'était pas une majuscule.
            resultat = resultat + caractere
        else:
        # Sinon, le caractère est la majuscule ALPHABET[k].
            resultat = resultat + alphabet[k]
    return resultat
```

4. La fonction suivante prend en entrée une chaine de caractères, supprime ses espaces et l'écrit en minuscule, puis détermine si la chaine est un palindrome en la comparant avec sa chaine-miroir.

```
def palindrome(chaine):
    """
    Entrée : une chaine de caractères chaine.
    Sortie : True si la chaine chaine est un palindrome,
             False sinon.
    """
    chaine = minuscule(sans_espace(chaine))
    # On supprime les espaces de la chaine
    # et on remplace les majuscules par des miniscules.
    return chaine == miroir(chaine)
```

Exercice 5 — Recherche d'un mot dans une phrase

Pour déterminer si une chaine de caractères `mot` apparait dans une chaine de caractères `phrase`, on commence par rechercher le premier caractère du mot `mot` parmi les premiers caractères de la phrase `phrase`, de sorte que la suite du mot ait la place d'apparaitre dans la suite de la phrase, c'est-à-dire parmi les `len(phrase) - len(mot) + 1` premiers caractères de la phrase.

Si l'on trouve la première lettre du mot, on compare alors les caractères suivants de la phrase avec les caractères suivants du mot. Si l'on trouve le mot tout entier, on retourne `True`, ce qui arrête la recherche. Sinon, on essaye à partir de la prochaine occurrence de la première lettre du mot (s'il y en a une).

Si l'on ne trouve le mot après aucune des occurrences de la première lettre du mot, on renvoie `False`.

```
def recherche(mot, phrase):
    """
    Entrées : deux chaines de caractères mot et phrase.
    Sortie : True si la chaine mot apparait dans la chaine phrase,
             False sinon.
    """
    n = len(mot)
    # On cherche la première lettre du mot dans la phrase
    # en laissant de la place pour le reste du mot.
    for k in range(len(phrase) - n + 1):
    # L'entier k va de 0 à len(phrase) - n.
        if phrase[k] == mot[0]:
            # On compare alors les n - 1 lettres suivantes
            # de la phrase avec les n - 1 lettres suivantes du mot.
            j = 1
            while (j < n) and (phrase[k + j] == mot[j]):
                j = j + 1
            if j == n:
            # On a trouvé le mot dans la phrase à partir de l'indice k.
                return True
    return False
    # Si la fonction ne s'est pas arrêtée avant,
    # c'est que le mot n'apparait pas dans la phrase.
```