Corrigé exercice n°2 du quizz n°3 : codage de `endswidth()`

Rappel de l'énoncé :

Écrire un programme qui demande deux mots à l'utilisateur, et indique si le 2e mot est un suffixe du premier.

Schéma général

Si on suppose qu'on dispose d'une fonction à deux arguments, appelée `suffixe()`, la structure générale du programme demandé est donnée à la cellule suivante (le code suivant ne marche pas parce que la fonction `suffixe()` n'est pas encore définie).

[ ]:
```
mot = input("Donnez un mot ")
suf = input("Donnez un candidat suffixe ")
if suffixe(suf,mot):
    print("oui")
else:
    print("non")
```

On peut maintenant se concentrer sur la fonction `suffixe()`. C'est une fonction booléenne (elle doit renvoyer `True` ou `False`), qui prend deux arguments, le suffixe candidat et le mot.

N.B. : la plupart des exemples qui suivent produisent une erreur si le mot est plus court que le suffixe.

On laisse en exercice la version où un test supplémentaire garantirait le bon fonctionnement du programme dans tous les cas.

Première version : méthode existante La première version peut directement se baser sur la méthode existante `endswith()`.

[1]:
```
def suffixe(s,m):
    return m.endswith(s)

mot = input("Donnez un mot ")
suf = input("Donnez un candidat suffixe ")
if suffixe(suf,mot):
    print("oui")
else:
    print("non")
```

```
Donnez un mot permenganate
Donnez un candidat suffixe ganate
oui
```

Deuxième version : utilisation des slices En python il est possible de faire référence à des sous-chaînes de caractères (*slices*) avec le symbole `:`, et par ailleurs le signe `==` est utilisable entre deux chaînes de caractères (cela implique bien une comparaison caractère par caractère, mais c'est directement géré par python). La "tranche" à extraire du mot est la suite de caractères de la fin du mot, dont la longueur est `len(s)`.

[2]:
```
def suffixe(s,m):
    return (m[-len(s):] == s)
```

Les versions qui suivent, qui visent à implémenter de différentes manières la fonction `endswith()`, sont toutes basées sur une adaptation du programme préfixe vu en séance précédemment, et rappelé ici. Le cœur de l'algorithme repose sur la boucle while qui boucle autant de fois que les lettres du préfixe candidat et celles du mot correspondent. Si toutes les lettres du préfixe ont leur correspondante dans le mot, alors la boucle va jusqu'à la fin, c'est-à-dire jusqu'à ce que `i` dépasse la longueur du préfixe.

[4]:
```
# Exercice 4
mot = input("Donnez un mot : ")
pref = input("Donnez un préfixe potentiel : ")
i = 0
while i < len(mot) and i < len(pref) and mot[i] == pref[i]:
    i += 1
if i == len(pref):
    print("le mot %s est un préfixe de %s." % (pref, mot))
```

```
Donnez un mot : mimographe
Donnez un préfixe potentiel : mimo
le mot mimo est un préfixe de mimographe.
```

Troisième version : indices négatifs La troisième version utilise la possibilité en python d'utiliser des indices négatifs. Attention, les indices négatifs vont de -1 à $-(\texttt{len(s)}+1)$ (ou $-\texttt{len(s)}-1$), il faut décrémenter l'indice au lieu de l'incrémenter et adapter les tests.

[5]:
```
def suffixe(s,m):
    i  = -1
    while i > -len(m)-1 and i > -len(s)-1 and m[i] == s[i]:
        i -= 1
    return (i == -len(s) - 1)

mot = input("Donnez un mot ")
suf = input("Donnez un candidat suffixe ")
if suffixe(suf,mot):
    print("oui")
else:
    print("non")
```

```
Donnez un mot pantomime
Donnez un candidat suffixe mime
oui
```

Quatrième version : indices décalés On peut remarquer qu'alors que pour le préfixe, les lettres à comparer dans le mot et dans le préfixe ont le même indice, ce n'est pas le cas pour le suffixe : il faut comparer la première lettre du suffixe avec la lettre du mot en position $(\texttt{len(m)} - \texttt{len(s)})$. La version suivante met en œuvre ce cas.

[6]:
```
def suffixe(s,m):
    i  = len(m) - len(s)
    while i < len(m) and m[i] == s[i-len(m)]:
        i += 1
    return (i == len(m))

mot = input("Donnez un mot ")
suf = input("Donnez un candidat suffixe ")
if suffixe(suf,mot):
    print("oui")
else:
    print("non")
```

```
Donnez un mot pantomime
Donnez un candidat suffixe rime
non
```

Cinquième version : basé sur une inversion des chaînes de caractères Pour réutiliser le code qui calcule le préfixe, on peut inverser les chaînes de caractères, et ramener le problème du suffixe à un problème du préfixe. C'est une stratégie inutilement coûteuse, mais qui permet de simplifier la conception.

[7]:
```
def suffixe(s,m):
    # inversion des deux mots
    pref = s[::-1]
    mot = m[::-1]
    # algo standard du préfixe
    i = 0
    while i < len(mot) and i < len(pref) and mot[i] == pref[i]:
        i += 1
    return (i == len(pref))

mot = input("Donnez un mot ")
suf = input("Donnez un candidat suffixe ")
if suffixe(suf,mot):
    print("oui")
else:
    print("non")
```

```
Donnez un mot pantomime
Donnez un candidat suffixe mime
oui
```

Remarque sur le mot-clé `in` Le mot-clé `in` peut être utilisé entre deux chaînes de caractères, pour déterminer si une chaîne est incluse dans une autre, mais il ne permet pas simplement de déterminer si la sous-chaîne est en position de suffixe.