

Quizz_2023_3

February 22, 2023

Ecrire une fonction `avant()` qui prend deux paramètres de type “chaîne de caractère” (`str`) et renvoie `True` si le premier argument est placé strictement avant le second argument dans l’ordre lexicographique. On suppose que les opérateurs de comparaison (`<`, `>`, etc.) ne sont pas définis pour les `str`.

Fonctions utilisables : `len()` et `ord()` (cette dernière fonction prend un caractère unique en entrée, et renvoie son numéro d’ordre dans `unicode` : elle permet donc de comparer deux caractères).

En cas de difficulté pour répondre directement, vous pouvez commencer par proposer une fonction `egal()` prenant deux paramètres de type `str` et renvoyant `True` si les deux paramètres sont identiques.

Fonction `egal()`

La boucle principale peut se terminer soit parce que les caractères `s[i]` et `t[i]` sont différents (auquel cas il faut renvoyer `False`), soit parce que `i` a atteint la fin d’une des chaînes. Dans ce cas on ne renvoie vrai que si `i` a atteint la fin des deux chaînes (ie elles sont de même longueur).

```
[5]: # Egalité de deux chaînes : on boucle tant que les caractères sont identiques
def egal(s,t):
    i = 0
    while i < len(s) and i < len(t) and s[i] == t[i]:
        i += 1
    if i < len(s) or i < len(t):
        return False
    return i == len(s) and i == len(t)
```

Fonction `avant()`

```
[6]: # ordre lexicographique strict. Par hypothèse avant(s,s) --> False
def avant(s,t):
    i = 0
    while i < len(s) and i < len(t) and s[i] == t[i]:
        i += 1
    if i < len(s) and i < len(t):
        return ord(s[i]) < ord(t[i])
    return len(s) < len(t)
```

Variante plus ou moins inspirée des propositions des étudiants: On commence par trouver la longueur du mot le plus court Cette longueur sert de borne à une boucle sur les indices Dans la boucle: dès qu'on trouve une différence entre $s[i]$ et $t[i]$, il faut renvoyer soit True si $s[i]$ est avant $t[i]$ soit False si à la fin de la boucle on n'a rien renvoyé, c'est que le mot le plus court est un préfixe de l'autre, et dans ce cas le mot le plus court est avant l'autre (sauf en cas d'égalité de longueur, puisque j'ai posé comme hypothèse que $\text{avant}(s,s)$ donne False)

```
[10]: def avant_m(s,t):
        max = len(s) if len(s) < len(t) else len(t)
        for i in range(max):
            if ord(s[i]) > ord(t[i]):
                return False
            elif ord(s[i]) < ord(t[i]):
                return True
        return True if len(s) != len(t) else False
```

```
[11]: # Pour générer sans effort un jeu de test, j'utilise split()
        # pour créer une liste de mots
        t = "il faut dire il faut faire des fautes".split()
        # puis je teste tous les couples possibles parmi cette liste
        for x in t:
            for y in t:
                print("avant(%s,%s) --> %s" % (x,y,avant_m(x,y)))
```

```
avant(il,il) --> False
avant(il,faut) --> False
avant(il,dire) --> False
avant(il,il) --> False
avant(il,faut) --> False
avant(il,faire) --> False
avant(il,des) --> False
avant(il,fautes) --> False
avant(faut,il) --> True
avant(faut,faut) --> False
avant(faut,dire) --> False
avant(faut,il) --> True
avant(faut,faut) --> False
avant(faut,faire) --> False
avant(faut,des) --> False
avant(faut,fautes) --> True
avant(dire,il) --> True
avant(dire,faut) --> True
avant(dire,dire) --> False
avant(dire,il) --> True
avant(dire,faut) --> True
avant(dire,faire) --> True
avant(dire,des) --> False
avant(dire,fautes) --> True
```