Programmation en Python 2022-2023

## Feuille d'exercices

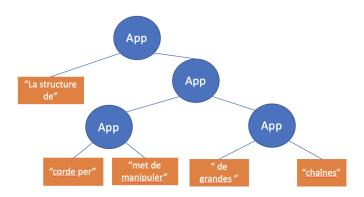
## Partie 4: Programmation objets

## La structure de corde

Les chaînes de caractères de Python ont comme principal désavantage que les opérations de concaténation s1 + s2 et d'extraction s[i:j] ont une complexité (en temps ou espace) linéaire en la taille des chaînes manipulées. Ainsi, cette structure est inutilisable quand il faut manipuler de grandes chaînes de caractères (de plusieurs centaines voire milliers de Ko).

Pour pallier ces problèmes, on utilise une structure de données appelée corde (rope en anglais), où les chaînes de caractères sont représentées par des arbres binaires dont les nœuds App représentent des concaténations et les feuilles Str sont des « petites chaînes de caractères » Python. Par exemple, la chaîne "La structure de corde permet de manipuler de grandes chaînes", vue comme la concaténation de chaînes

peut être représentée par la corde suivante :



Dans cet exercice, on souhaite réaliser l'implémentation des cordes en Python. L'interface des cordes est donnée sous la forme d'une classe Rope définie ci-dessous.

```
class Rope(ABC):
    def __init__(self,d):
        self.length = 0

    @abstractmethod
    def __getitem__(self,i):
        pass

    @abstractmethod
    def __add__(self,r):
        pass

    @abstractmethod
    def __str__(self):
        pass
```

Les méthodes de cette interface permettent de manipuler des cordes avec des opérateurs similaires à ceux des String.

La méthode \_\_getitem\_\_(self,i) permet d'implémenter à la fois l'opération d'accès r[i] au ième caractère de r, mais également l'extraction r[a:b] qui renvoie la sous-corde entre la position a (inclus) et b (exclu) quand i est une valeur slice(a,b,\_).

La méthode \_\_add\_\_(self,s) permet d'implémenter l'opération de concaténation r + s entre deux cordes r et s (r est la corde sur laquelle l'opération s'applique, i.e. l'argument self vaut r).

La méthode \_\_str\_\_(self) permet de convertir une corde en une chaîne de caractères (String) (cette méthode est par exemple appelée implicitement par la fonction print).

À ces méthodes s'ajoute la variable d'instance length qui contient la longueur de la corde.

L'implémentation des arbres binaires des cordes est réalisé à l'aide de deux classes App et Str qui héritent de Rope.

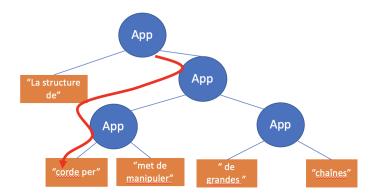
**Question 1** Définir les en-têtes des deux classes App et Str avec leur constructeur \_\_init\_\_ respectif. Vérifier votre solution en déclarant une corde r comme ci-dessous :

**Question 2** Écrire les méthodes \_\_str\_\_ pour ces deux classes afin de renvoyer la chaîne de caractères (de type String) correspondant à une corde. Pour tester votre code, vous pouvez par exemple vérifier que vous obtenez bien le résultat suivant :

## >>> print(r)

La structure decorde permet de manipuler de grandes chaînes

Pour renvoyer le *i*-ème caractère d'une corde, il faut descendre dans l'arbre à travers les nœuds App jusqu'à la bonne feuille Str, puis extraire le caractère se trouvant à la bonne position dans la chaîne stockée dans cette feuille. Par exemple, le chemin pour renvoyer le caractère 'o' à la position 16 dans la corde définie ci-dessus est le suivant :



Question 3 En supposant que l'argument i de \_\_getitem\_\_(self,i) est un simple entier, écrire le code des méthodes \_\_getitem\_\_ des deux classes App et Str afin de renvoyer le caractère se trouvant à la position i dans une corde. Pour tester votre code, vous pouvez par exemple vérifier que vous obtenez bien les résultats suivants :

Question 4 Étendre ces fonctions pour renvoyer un *slice* d'une corde. Pour distinguer ce cas du précédent, il faut tester si l'argument i représente un *slice* en utilisant isinstance(i,slice). Si c'est le cas, a et b sont représentés respectivement par i.start et i.stop. Pour tester votre code, vous pouvez par exemple vérifier que vous obtenez bien les résultats suivants:

```
>>> print(r[21:54])
permet de manipuler de grandes ch
```

La concaténation de deux cordes peut être facilement réalisée en temps constant à l'aide de la fonction suivante :

```
def __add__(self, r):
    return App(self, r)
```

Cependant, avec cette fonction, plus on fait de concaténations et plus la hauteur de la corde augmente, ce qui a pour conséquence de dégrader l'accès aux caractères de la corde. Afin de limiter ce problème, on évite l'apparition de trop nombreuses petites chaînes de caractères (de taille plus petite que 10 caractères par exemple) aux feuilles en les concaténant explicitement.

Question 5 Modifier les méthodes \_\_add\_\_(self,r) des deux classes App et Str afin de concaténer les petites chaînes de caractères. Par exemple, la corde définie par (Str('debut de la ') + Str('chai')) + Str('ne') devrait correspondre à la valeur App(App(Str('debut de la '),Str('chai')), Str('ne')). Mais en concaténant les petites chaînes, cette corde doit correspondre à App(Str('debut de la '),Str('chaine')).