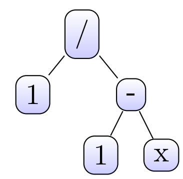
Exercices 35 (devoir)

Exercice 35 (manipulation d'expressions algébriques).

On considère une expression algébrique représentée par un arbre, dont chaque nœud est étiqueté, avec les conventions suivantes :

- une feuille peut être étiquetée avec un nombre (de type `int` ou `float`), ou avec la chaîne de caractères 'x' (qui désigne alors la variable de l'expression algébrique);
- un nœud interne (c'est-à-dire un nœud qui n'est pas une feuille) a nécessairement deux enfants, et est étiqueté avec une chaîne de caractères de longueur 1 qui code l'opération algébrique portant sur les enfants ('+' ou '-' ou '*' ou '/').

Par exemple, l'expression algébrique $\dfrac{1}{1-x}$ sera représentée par l'arbre 

(1) Représenter l'arbre construit par le pseudo-code ci-dessous, et écrire l'expression algébrique associée.

```
A1 ← creer_arbre('-',[creer_arbre(1),creer_arbre('x')])
A2 ← creer_arbre('*',[creer_arbre('x'),creer_arbre('x')])
A3 ← creer_arbre('+',[creer_arbre(1),A2])
A ← creer_arbre('/',[A1,A3])
```

Dans ce pseudo-code, la fonction creer_arbre(e,L) crée et retourne un arbre dont la racine a pour étiquette e (e peut être un nombre ou une chaîne de caractères), et dont les enfants sont les éléments de la liste L, qui sont eux-mêmes des arbres. Si la liste L est vide, ou si seul l'argument e est spécifié, l'arbre retourné n'a pas d'enfants (c'est une feuille d'étiquette e).

(2) Réciproquement, représenter l'arbre associé à l'expression algébrique définie par la fonction

$$f(x) = \frac{3x+2}{1-x(1+x)},$$

et donner un pseudo-code permettant de construire cet arbre selon le modèle de la question précédente.

(3) Construire effectivement l'arbre A de la question 2, en utilisant l'implémentation ci-dessous (qui représente les arbres par des listes selon le modèle vu en cours) pour la fonction creer_arbre() :

```python
def creer_arbre(e,L=[]):
    """
    Retourne un arbre dont la racine a pour étiquette e,
    et pour enfants les éléments de la liste L (éventuellement vide)
    signature: étiquette x liste d'arbres -> arbre
    """
    return [e]+L
```

Visualiser la liste `A` ainsi produite sous Python.

(4) On souhaite écrire une fonction **evaluer()** qui prend en entrée une expression algébrique (représentée par un arbre A) et un nombre x, et évalue l'expression algébrique pour cette valeur de x. Par exemple, si A représente l'arbre considéré dans l'introduction (A représente l'expression algébrique $\frac{1}{1-x}$), alors evaluer(A,-1) retournera $\frac{1}{1-(-1)} = 0.5$.

Compléter, dans le pseudo-code ci-dessous, les passages marqués sous la forme de points de suspensions.

```
fonction evaluer(A, x)
|   // retourne la valeur de l'expression algébrique A pour la valeur de x spécifiée
|   e ← etiquette(A)
|   E ← enfants(A)
|   si E est vide // cas d'une feuille
|   |   si e est un nombre
|   |   |   retourner e
|   |   vérifier que e ='x'
|   |   retourner ........
|   // cas d'un noeud interne (2 enfants)
|   vérifier que E est de taille 2
|   a ← evaluer(E(0),x)
|   b ← evaluer(E(1),x)
|   si e='+' // addition
|   |   retourner .........
|   si e='-' // soustraction
|   |   .........
|   ................
```

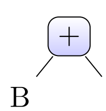
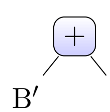
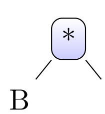
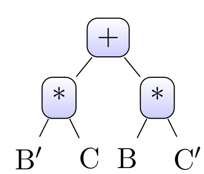
(5) Implémenter la fonction **evaluer()** en Python. On pourra utiliser les implémentations suivantes des fonctions etiquette() et enfants() :

```python
def etiquette(A):
    "retourne l'étiquette de la racine de l'arbre A"
    return A[0]

def enfants(A):
    "retourne la liste des enfants de la racine de l'arbre A"
    return A[1:]
```

Tester la fonction obtenue avec `evaluer(A,0)`, où `A` est l'arbre construit à la question 3.

(**6**) On souhaite calculer la dérivée d'une expression algébrique représentée par un arbre A. Pour cela, on adopte une démarche récursive, en remarquant que :

- si A est la feuille $\boxed{x}$, alors derivee(A) est la feuille $\boxed{1}$
- si A est la feuille $\boxed{17}$ (ou tout autre nombre), alors derivee(A) est la feuille $\boxed{0}$
- si A est l'arbre  , alors derivee(A) est l'arbre  , où B′ et C′ sont obtenus en appliquant la fonction derivee() aux arbres B et C (les enfants de A)
- si A est l'arbre  , alors derivee(A) est l'arbre 

Quels sont les autres cas à considérer, et quel est l'arbre résultat à chaque fois ?

En déduire l'écriture d'une fonction récursive Python **derivee(A)** qui prend en entrée un arbre A représentant une expression algébrique, et renvoie un arbre représentant la dérivée de cette expression algébrique (on pourra s'inspirer du modèle de la fonction **evaluer()** vue précédemment).

Vérifier le résultat en calculant de deux manières différentes $f'(0)$, où f est la fonction définie à la question 2 : 1) en calculant à la main la dérivée de f; 2) avec Python à l'aide des fonctions **evaluer()** et **derivee()**.

Calculer ensuite avec Python la valeur de $f^{(5)}(0)$ (dérivée cinquième de f en 0).

(**7***) Écrire une fonction Python **DL0(A,n)**, qui prend en entrée un arbre A représentant une expression algébrique, et renvoie une chaîne de caractères représentant le développement limité en 0 à l'ordre n de la fonction représentée par A. Cette fonction pourra utiliser la formule de Taylor, et faire appel aux fonctions **derivee()** et **evaluer()** définies précédemment. Par exemple, si A représente l'arbre considéré dans l'introduction (qui représente l'expression algébrique $\frac{1}{1-x}$), alors DL0(A,5) devra retourner la chaîne de caractères

```
'1.0 + (1.0).x^1 + (1.0).x^2 + (1.0).x^3 + (1.0).x^4 + (1.0).x^5 + o(x^5)'
```

En déduire le développement limité à l'ordre 6 en 0 de la fonction f définie à la question 2.
