Séance 3 : Types récursifs

L1 – Université Côte d'Azur

Dans ce premier exercice, on travaille avec des listes chaînées, comme définis dans le cours. On ne manipulera les listes chaînées qu'au travers des fonctions suivantes :

- La notation couple `(tête,queue)` qui renvoie une liste dont le premier élément est `tête` et dont la suite est `queue`. La liste chaînée vide sera représentée par `None`.
- `is_empty(lc)` qui renvoie `True` si `lc` est une liste vide, et `False` sinon ;
- `head(lc)` qui renvoie le premier élément de la liste.
- `tail(lc)` qui renvoie la suite de la liste.
- exceptionnellement, pour écrire des tests, on pourra aussi utiliser la fonction `new_lc(tab)` qui renvoie une liste chaînée contenant les mêmes élements que le tableau tab.

Exercice 1 — Listes chaînées

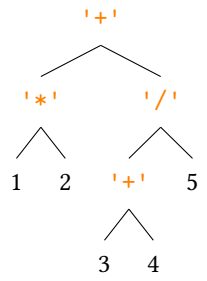
1. Téléchargez le fichier `lc.py` à l'adresse https://upinfo.univ-cotedazur.fr/~obaldellon/L1/bi2/tp3/lc.py Placez-le dans le répertoire où vous écrivez vos TP. Ce fichier contient les fonctions associées aux listes chaînées. Créer un fichier `exo1.py` dont la première ligne est `from lc import *`. Tester la fonction `new_lc` en affichant le résultat de `new_lc([100, 20, 3])`.
2. Écrire une fonction récursive `longueur(lc)` qui renvoie la longueur d'une liste chaîne.
3. Écrire une fonction `double(lc)` qui renvoie une nouvelle liste chaînée dont les éléments sont les doubles de ceux de `lc`. Par exemple, si `lc` contient 3, 4 et 7, `double(lc)` contiendra les élements 6, 8 et 14.
4. Écrire une fonction `majeur(lc)` qui renvoie la liste de tous les élements supérieurs à 18.

Dans les exercices suivants, on manipule des arbres binaires d'expression arithmétiques (non vides), comme définis dans le cours. Les expressions seront des nombres entiers (sur les feuilles de l'arbre) ou des chaînes de caractères contenant les opérations (sur les nœuds de l'arbre). On ne s'intéresse pas à la façon dont les arbres sont représentés, on va utiliser uniquement les fonctions suivantes :

- `arbre(r, Ag, Ad)` qui renvoie un arbre de racine `r` et de fils `Ag` (gauche) et `Ad` (droit) ;
- `est_feuille(obj)` qui renvoie `True` si `obj` est une feuille, et `False` sinon ;
- `racine(A)` qui renvoie la racine de l'arbre `A`
- `fg(A)` qui renvoie le fils gauche de l'arbre `A`
- `fd(A)` qui renvoie le fils droit de l'arbre `A`.

Exercice 2 — Mise en route

1. Téléchargez le fichier `abe.py` à l'adresse https://upinfo.univ-cotedazur.fr/~obaldellon/L1/bi2/tp3/abe.py Placez-le dans le répertoire où vous écrivez vos TP. Créez un fichier `tp3-arbres.py`.
   Quelle ligne faut-il écrire dans ce fichier pour pouvoir utiliser les fonctions définies dans `abe.py` ?
2. Sans utiliser de liste, mais uniquement la fonction `arbre(r, Ag, Ad)`, créez l'arbre `A1` défini ci-contre.



Exercice 3 — Manipulation d'arbres

1. Écrire une fonction `contient42(A)` qui renvoie `True` si une des feuilles de l'arbre `A` est 42, et `False` sinon.
2. Écrire une fonction `compte_pairs(A)` qui renvoie le nombre de feuilles de `A` qui sont des nombres pairs.
3. Écrire une fonction `feuilles_paires(A)` qui renvoie la liste des feuilles de `A` qui sont des nombres pairs.

On appelle *profondeur* d'un nœud la distance entre ce nœud et la racine.

4. Écrire une fonction `liste_profondeur(A,n)` qui renvoie la liste des nœuds de `A` de profondeur `n`
5. Écrire une fonction `profondeur_max_pairs(A)` qui renvoie la profondeur maximale des feuilles paires de `A` (et `None` si `A` n'a pas de feuille paire).

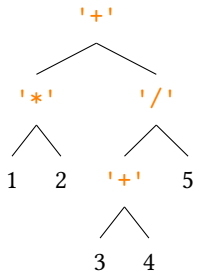
Exercice 4 — Comptage et transformation

1. Programmez une fonction `compter(A, op)` renvoyant le nombre d'apparitions de l'opérateur `op` dans l'arbre `A`.

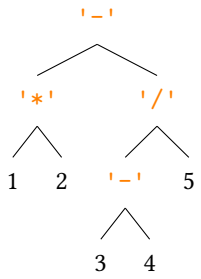
```
>>> compter(A1, '+')
2
```

2. Programmez une fonction `remplacer(A, op1, op2)` qui renvoie une copie de l'arbre `A` où chaque apparition de l'opérateur `op1` aura été remplacée par l'opérateur `op2`.

Arbre A1



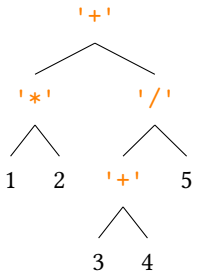
Arbre A2 en posant `A2=remplacer(A1,'+','-')`



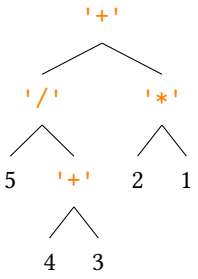
Exercice 5 — Miroir

Programmez la fonction `miroir(A)` retournant l'image inversée (à tous les niveaux) de l'arbre `A`.

Arbre A1



Miroir de A1



Dans la suite, nous travaillerons avec des piles. Nous rappelons ci-dessous les principales fonctions :

- `nouvelle_pile()` qui renvoie une nouvelle pile vide ;
- `est_vide(p)` qui renvoie `True` si `p` est une pile vide, et `False` sinon ;
- `empile(p)` ajoute `e` au somment de la pile `p` ;
- `sommet(p)` qui renvoie le sommet de la pile `p` ;
- `dépile(p)` qui supprime et renvoie le sommet de la pile `p`.

Exercice 6 — Parcours suffixe et évaluation avec une pile

Le but de cette exercice est d'évaluer des expressions arithmétiques automatiquements. Certes, Python sait déjà le faire, mais il peut intéressant de savoir le programmer soi-même.

Il est assez difficile de transformer une expression représentée par une chaîne (par exemple `"1+(4+5*3)"`) en arbre. Il faut gérer les priorités et les parenthèses. Il existe cependant une écriture, la fameuse *notation polonaise inversée* (NPI), très simple à transformer en arbre. C'était la notation utilisée par les calculatrices HP [1] lors de la jeunesse de mon père (ce qui ne nous rajeunit pas…). La NPI s'obtient par un parcours en profondeur suffixe. Par exemple, le parcours profondeur suffixe de l'expression `(1-2)*(3+4)` est

```
[1, 2, '-', 3, 4, '+', '*']
```

1. Écrivez un programme `arboriser` qui prend une liste en NPI et renvoie l'arbre correspondant. On utilisera un pile pour stocker les résultats intermédiaires. Pour information vous pouvez télécharger le module `pile.py` (celui du cours) à l'adresse : https://upinfo.univ-cotedazur.fr/~obaldellon/bi2/py/tp3/pile.py
2. En utilisant la fonction `arboriser` et la fonction `valeur` (cours 7 page 43) qui calcule la valeur associée à un arbre binaire arithmétique, écrire une fonction `calcul` qui donne le résultat du calcul donnée en argument en notation NPI.

```
>>> calcul([1, 2, '-', 3, 4, '+', '*'])
-7
```

3. Écrire une fonction `calcul_direct` qui fait la même chose que la fonction `calcul` mais sans passer par un arbre et en faisant le calcul directement avec une pile.
4. Écrire une fonction `calcul_chaîne` qui calcule la valeur d'une chaîne en NPI.

```
>>> calcul_chaîne('1 2 - 3 4 + *')
-7
```

1. https://fr.wikipedia.org/wiki/Calculatrices_HP