

Séance 7 : MODULES ET TYPES ABSTRAITS

L1 – Université Côte d’Azur

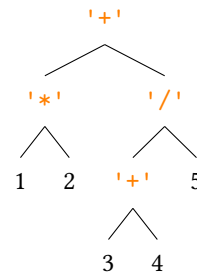
Dans les exercices suivants, on manipule des arbres binaires d’expression arithmétiques (non vides), comme définis dans le cours. Les expressions seront des nombres entiers (sur les feuilles de l’arbre) ou des chaînes de caractères contenant les opérations (sur les nœuds de l’arbre).

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 1 – Mise en route (★)

1. Téléchargez le fichier `abe.py` à l’adresse <https://upinfo.univ-cotedazur.fr/~obaldellon/L1/py/tp7/abe.py>. Placez-le dans le répertoire où vous écrivez vos TP. Créez un fichier `tp7-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 2 – 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_paires`(`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_paires`(`A`) qui renvoie la profondeur maximale des feuilles paires de `A` (et `None` si `A` n’a pas de feuille paire).

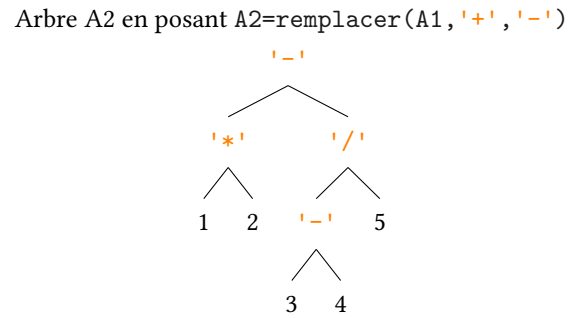
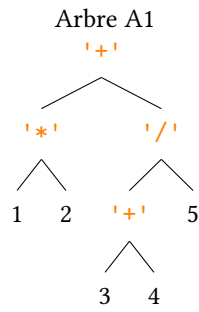
Exercice 3 – Comptage et transformation (★★)

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

```

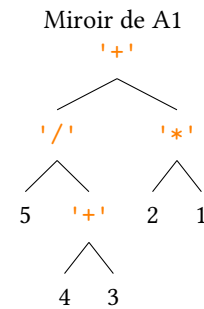
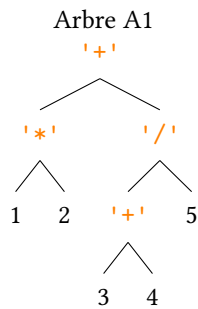
1 >>> compter(A1, '+')
2 2
    
```

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`.



Exercice 4 – Miroir ()**

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



Exercice 5 – Parcours suffixe et évaluation avec une pile (*)**

Le but de cette exercice est d’évaluer des expressions arithmétiques automatiquement. 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 HP¹ 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 profond 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/L1/py/tp7/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.

```

1 >>> calcul([1, 2, '-', 3, 4, '+', '*'])
2 -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_chaine` qui calcule la valeur d’une chaîne en NPI.

```

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

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