# Séance 5 : Structures de données

L1 – Université Côte d'Azur

**Exercice 1** — Les arbres

1. finir le TP précédant (au moins jusqu'à l'exercice 7)
2. Écrire une fonction `approx_pi(A)` qui renvoie un nouvel arbre construit à partir de `A` mais dans lequel toutes les feuilles égales à `"Pi"` ont été remplacé par 3.14.
3. Écrire une fonction `négation(A)` qui applique la négation à un arbre. On rappellera les formules :
   (a) $-(a+b) = (-a) + (-b)$
   (b) $-(a-b) = b - a$
   (c) $-(a \times b) = (-a) \times b$
   (d) $-(a/b) = (-a)/b$

**Exercice 2** — Les piles (implémentation avec tableau)

Une pile est une structure de donnée n'ayant que deux opérations `push` et `pop`. Les listes en Python permettent déjà d'utiliser ses deux méthodes, mais nous souhaitons ici voir comment une pile peut-être implémentée avec uniquement des structures de bases (tableau de taille fixe)

Dans cette exercice une pile est un tableau associé à un entier `sommet`. Cette entier correspond à la première des cases vides du tableau. Attention : si `sommet` est égale à `taille` il faudra renvoyer une erreur (pile pleine) en cas de `push`, et si sommet est nul, il faudra renvoyer une erreur en cas de `pop` (pile vide).

On donne la structure du code ci-dessous. Ces sera à vous de la compléter.

```python
class Pile:
    def __init__(self,taille):
        self.taille= taille
        self.tableau = taille * [None]
        self.sommet = 0

    def __repr__(self):
        rés = "["
        for i in range(self.sommet):
            rés += f"{self.tableau[i]} ["
        return rés

    def push(self,x):
        ...

    def pop(self):
        ...
```

**Exercice 3** — Les piles (implémentation avec liste chaînée)

Même exercice que le précédant. Cette fois, les valeurs de la pile sont stockées dans une liste chaînée. Écrire les méthodes `pop`, `push` et `__repr__`

```
class Pile:
    def __init__(self):
        self.lc = None

    def push(self,x):
        ...

    def pop(self):
        ...

    def __repr__(self):
        ...
```

**Exercice 4** — Le type abstrait file

Une File est une structure de donnée avec deux méthodes `ajouter` et `servir`. La premier ajoute des éléments à la file, le second les supprime et le renvoie. À la différence de la pile qui renvoie le dernier élément ajouté, la file renvoie le premier (pensez à la boulangerie, le premier qui s'insère dans la file d'attente sera le premier servie par la boulangère).

- on ajoute 5 : `5`
- on ajoute 2 : `5:2`
- on ajoute 7 : `5:2:7`
- on sert le premier élément de la file (renvoie 5) : `2:7`
- on sert le premier élément de la file (renvoie 2) : `7`

Écrire une classe File à partir d'un objet contenant trois attributs

1. un tableau contenant les éléments
2. la capacité maximale du tableau (sa taille)
3. l'indice du premier élément
4. le nombre d'élément du tableau (sa longueur effective)

```
class File:
    def __init__(self,taille):
        self.taille= taille
        self.tableau = taille * [None]
        self.début = 0
        self.longueur = 0

    def ajoute(self,x):
       ...

    def servir(self):
       ...

    def __repr__(self):
       ...
```