N. S. I. Term. - DS04 (jeudi 16 décembre 2021) – 1 heures– v0

Ne sont pas à faire par les 1/3 temps : les exercices marqués [sauf 1/3 temps]

1 Tableaux [Ne pas faire pendant le DS]

En informatique, la **structure de données abstraite de tableau** est une structure de données de **taille fixe** (le nombre d'éléments d'un tableau est fixé à la création du tableau), linéaire (les valeurs dans un tableau sont organisées comme une suite ordonnée) et homogène (toutes les valeurs d'un tableau sont de même type).
On appelle « taille » du tableau son nombre d'éléments. Les éléments d'un tableau de taille n , $n \geq 1$, sont numérotés, soit de 0 à $n-1$, soit de 1 à n, selon le modèle choisi.
Le numéro associé à chaque valeur dans le tableau est appelé « indice » (ou index) et permet d'accéder à la valeur associée, pour la lire ou la modifier.

On représente un tableau de taille n de la façon suivante (si la numérotation commence à 1) :

- exemple d'un tableau d'entiers de taille $n = 6$, contenant les valeurs 13, 9, 8, 15, 10, 15, dans cet ordre

Indice	1	2	3	4	5	6
Valeur	13	9	8	15	10	15

On cherche à définir en Python, une classe `Tableau` dans laquelle un tableau est représenté par une liste Python.

Exercice 1.

1. Pour instancier un objet de la classe `Tableau`, on donnera la taille du tableau, ainsi qu'une valeur qui sera celle de chaque cellule du tableau.
Ainsi l'instruction `Tableau(6, 0)` construira le tableau représenté par

Indice	1	2	3	4	5	6
Valeur	0	0	0	0	0	0

sous la forme d'une liste Python de longueur 6. La liste Python représentant le tableau sera enregistrée sous la forme d'un attribut nommé `liste`.
Ainsi, par exemple, `Tableau(6, 0).liste` aura pour valeur `[0, 0, 0, 0, 0, 0]`.

2. Définir un attribut `taille`, dont la valeur donne le nombre d'éléments du tableau ;
3. Définir un attribut `typetab`, dont la valeur donne le type des valeurs dans le tableau.
Ainsi, par exemple, `Tableau(6, "").type` aura pour valeur `'<class str>'`.
4. Définir une méthode `lire`, telle que, si `tab` est un objet de la classe `Tableau`, `tab.lire(i)` renvoie la valeur située à la position i dans le tableau `tab` (on rappelle qu'on a choisi une numérotation commençant à 1).
Si la valeur i n'est pas comprise entre 1 et `tab.taille` (au sens large), on lèvera une exception `IndexError` avec le message « indice invalide ».
5. Définir une méthode `remplace`, telle que, si `tab` est un objet de la classe `Tableau`, `tab.remplace(i, x)` remplace la valeur en position i par la valeur x.
Si la valeur i n'est pas comprise entre 1 et `tab.taille` (au sens large), on lèvera une exception `IndexError` avec le message « indice invalide ».
Si la valeur x n'est pas du type `tab.type`, on lèvera une exception `TypeError` avec le message « type de valeur invalide » (rappel : on vérifie, par exemple, qu'une variable `x` est de type `str`, par le test « `isinstance(x, str)` ».

2 Listes chaînées

Pour les exercices de cette partie 1., on suppose qu'une liste chaînée vide est représentée par la valeur `None`, tandis que les listes chaînées non vides sont implémentées sous la forme d'un objet de la classe `Cellule` dont la définition est donnée ci-dessous.

```
class Cellule:
    """une cellule d'une liste chaînée"""
    def __init__(self, v, s):
        self.valeur   = v
        self.suivante = s
```

Les exercices **1**, **2**, **3** et **4**., n'utilisent que ce qui est dans cette introduction.

Exercice 2.

On suppose que l'on dispose d'une liste chaînée `lst0` = ['a' ; 'b' ; 'c' ; 'd'].

Note : l'écriture n'est qu'une représentation de la liste ; cette écriture ne signifie pas `lst0` est de type `list` ; `lst0` est implémentée en machine sous la forme d'une liste chaînée.

1. Comment accède-t-on à la valeur 'a' dans cette liste ?

2. Construire la liste `lst1` dont les éléments sont ['e' ; 'a' ; 'b' ; 'c' ; 'd'], à partir de la liste `lst0`.

3. Écrire une instruction, ou une séquence d'instructions, pour construire une liste chaînée `lst2` = [3 ; 2 ; 1 ; 0].

~~**4.** Définir une fonction `renverser`, prenant en argument une liste chaînée et renvoyant une liste chaînée dont les éléments sont les mêmes que ceux de la liste en argument, mais dans l'ordre inverse. L'appel `renverser(lst0)` renverra une liste ['d' ; 'c' ; 'b' ; 'a'].~~

Exercice 3.

Recopier et compléter la définition de la fonction récursive `longueur`, prenant en argument une liste chaînée et renvoyant son nombre d'éléments :

```
def longueur(lst):
    ## traitement du cas de base
    if

    ## traitement du cas général
```

Exercice 4. [sauf 1/3 temps]

Écrire une fonction `affiche_liste(lst)`, qui affiche, en utilisant la fonction `print`, tous les éléments de la liste `lst`, séparés par des espaces, en terminant l'affichage par un retour chariot (retour à la ligne).

L'écrire comme une fonction récursive, puis la réécrire, en utilisant cette fois une boucle `while`.

Exercice 5.

Écrire une fonction récursive `k_eme_element(lst, k)`, qui renvoie, l'élément en position i dans la liste `lst`.

Les éléments seront numérotés à partir de zéro.

Si la liste `lst` est de longueur n et que k n'est pas compris entre 0 et $n-1$, la fonction renverra `None`.

De même, si la liste est vide, la fonction renverra `None`.

Exercice 6. [sauf 1/3 temps]

Écrire une fonction `trouve(x, lst)`, qui renvoie le rang de la première occurrence de `x` dans `lst`, le cas échéant, et `None` sinon.

L'écrire comme une fonction récursive, puis avec une boucle `while`.

3 Choisir une structure de données adaptée :

arbre, dictionnaire, tableau, liste chaînée, pile ou file.

Exercice 7. [sauf 1/3 temps]

Quelle structure de données choisir pour chacune de ces tâches ?

1. Représenter un répertoire téléphonique.
2. Stocker l'historique des actions effectuées dans un logiciel et disposer d'une commande Annuler (ou Undo).
3. Envoyer des fichiers au serveur d'impression.

4 .Calculer une expression arithmétique (exemple : $4 + (3 \times 2 + 8) \times 5$

On justifiera brièvement.

4 Arbres binaires quelconques

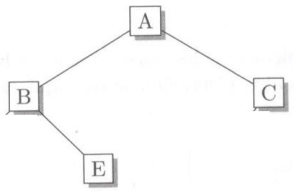
Exercice 8.

1. Compléter la définition de la classe `Noeud` ci-dessous, pour qu'elle corresponde à la définition adoptée en classe

```
class Noeud:
    def ... (  ):
        self.valeur = ...
        ...
```

2. Définir dans ce modèle :

- **a.** un arbre vide nommé `ab0` ;
- **b.** un arbre nommé `ab1` dont la représentation graphique est ci-dessous



3. **a.** Quelle est la taille de l'arbre `ab0` ? de l'arbre `ab1` ?

b. Implémenter une fonction `taille(a)`, renvoyant la taille de l'arbre a.

4. **a.** Quelle est la hauteur de l'arbre `ab0` ? de l'arbre `ab1` ?

b. Implémenter une fonction `hauteur(a)`, renvoyant la hauteur de l'arbre a.

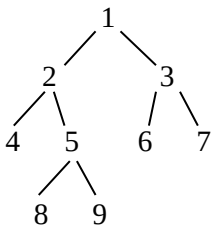
5. Quelle est la profondeur du nœud étiqueté 'B' dans l'arbre `ab1` ?

Exercice 9.

1. Rappeler combien il existe de squelettes d'arbres binaire de taille 0, de taille 1, de taille 2.
2. Dessiner tous les squelettes d'arbres binaires de taille 3. Combien y en a-t-il ?
3. Dénombrer les arbres binaires de taille 5, sachant qu'il y a 14 arbres binaires de taille 4.

Exercice 10.

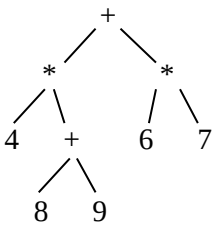
On considère l'arbre binaire suivant :



1. Dans un parcours préfixe de cet arbre, dans quel ordre s'affichent les étiquettes des nœuds ?
2. [sauf tiers-temps] dans un parcours suffixe de cet arbre, dans quel ordre s'affichent les étiquettes des nœuds ?

Ne pas faire les questions 3-4-5 pendant le temps du DS

On considère l'arbre binaire suivant :



3. Quelle est l'expression mathématique représentée par cet arbre ?
4. Faut-il utiliser un parcours préfixe, infixe ou suffixe pour calculer la valeur de cette expression ?
5. Définir une fonction `evalue(a)` qui renvoie la valeur d'une expression arithmétique représentée par un arbre binaire (on suppose que les opérations étiquetant les nœuds sont, soit des additions, soit des multiplications, soit des passages à l'opposé, les opérations seront représentées par les symboles '+', '*' et '-', de type `str`).