

EXAMEN 2nde SESSION JUIN 2025*Documents autorisés : une feuille A4 manuscrite recto-verso***Ex 1 (3 pts) : Fonction mystère**

Considérons la fonction Python suivante :

1) Dérouler "à la main" l'appel `mystere(4)`. Que calcule cette fonction ?

2) Donner une version récursive terminale (avec accumulateur) de cette fonction.

```
def mystere(n):
    if n <= 0 :
        return 0
    else :
        return n + mystere(n-1)
```

Ex 2 (3 pts) : Tri ternaire

On souhaite trier une liste contenant les nombres 0, 1 et 2 dans l'ordre croissant, i.e. tous les 0 se retrouvent à gauche, tous les 1 au milieu, et tous les 2 à droite.

Ecrire une fonction `tri_ternaire` effectuant ce tri en vous inspirant du tri par comptage (**aucune comparaison entre les nombres ne sera faite**). Cette fonction renvoie une nouvelle liste triée.

Ex : l'appel `tri_ternaire([0,1,0,2,0,1,1,1,2,0])` renvoie `[0,0,0,0,1,1,1,1,2,2]`.

Ex 3 (3 pts) : Pair ou impair ?

Pour déterminer si un nombre entier positif est pair ou impair, on peut partir d'un constat simple : si son prédécesseur existe et est impair, alors il est pair ; et si son prédécesseur existe et est pair, alors il est impair (tout en sachant que 0 est pair ...).

En utilisant cette stratégie, écrire deux fonctions `est_pair(n)` et `est_impair(n)` prenant en entrée un nombre entier positif `n`, et retournant un booléen indiquant respectivement si `n` est pair ou impair. Ces fonctions s'appelleront l'une l'autre.

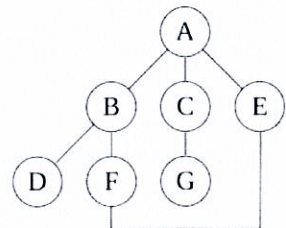
Ex 4 (4 pts) : Parcours

On souhaite effectuer un parcours en profondeur (DFS) du graphe ci-contre, à partir du sommet A.

1) Quelle structure de données permet d'implémenter (représenter) ce graphe ? Donner l'implémentation de ce graphe.

2) De quelle structure de données a-t-on besoin pour effectuer ce type de parcours ? Pourquoi ?

3) Effectuer "à la main" un tel parcours, en faisant apparaître l'état de chaque structure de données lors de chaque itération, comme en TD.

**Ex 5 (4 pts) : Fusion de piles**

Nous rappelons ci-contre le contenu de la classe Pile (définie à partir d'une liste `_contenu`), comportant un constructeur, les primitives `pile_vide`, `empiler`, `depiler`, ainsi qu'une méthode `inverser_pile` qui retourne la "version inversée" de l'instance courante.

Écrire une fonction `fusion(self,P)` où `self` et `P` sont deux piles contenant des nombres empilés dans l'ordre croissant. Cette fonction renvoie la pile obtenue par réunion de ces deux piles, où les nombres sont également empilés dans l'ordre croissant.

Attention, les piles ne contiennent pas forcément le même nombre d'éléments.

```
class Pile:
    def __init__(self):
        self._contenu=[]
    def pile_vide(self):
        return self._contenu==[]
    def empiler(self,x):
        self._contenu.append(x)
    def sommet(self):
        return self._contenu[-1]
    def depiler(self):
        return self._contenu.pop()
    def inverser_pile(self):
        Q = Pile()
        while not self.pile_vide():
            Q.empiler(self.depiler())
        return Q
```

Ex 6 (3 pts) : Graphe orienté pondéré

Effectuer "à la main" le déroulement de l'algorithme de Dijkstra sur le graphe ci-contre, en donnant les distances de poids minimal de chaque sommet au sommet A, ainsi que le chemin associé.

On utilisera un tableau, comme fait en CM.

