fiche 3

Fonction récursive

Version avec preuves.

Toutes les fonctions seront écrites de façon récursive.

Exercice 1 :

1. Écrire la fonction *recherche*, qui prend en argument une liste et un élément et renvoie le booleen correspondant à l'appartenance de l'élément dans la liste.
2. Écrire la fonction *recherche_dicho* qui prend en argument une liste ordonnée et un élément, et utilise la méthode dichotomique pour rechercher l'élément dans la liste.

Correction

```
#Exercice 1

def recherche (liste , element ) :
    if len(liste) == 0 :
        return False
    else :
        if liste[0] == element :
            return True
        else :
            return recherche(liste[1:] , element )


def recherche_dicho (liste,element ):
    """ recherche dans une liste ordonnee croissante"""
    if (len(liste) == 0) :
        return False
    else :
        milieu = len(liste)//2
        if liste[milieu] == element :
            return True
        else :
            if element < liste[milieu] :
                return recherche_dicho(liste[0:milieu] ,element )
            else :
                return recherche_dicho(liste[milieu +1 :] ,element )
```

Exercice 2 : **Tri par sélection :**

Le tri par sélection par la méthode récursive est relativement simple :

- Si la liste contient un élément, elle est triée.
- Sinon, on cherche le minimum de la liste. Il alors placé en tête de la reste de la liste que l'on trie...

1. Écrire la fonction `min_list` qui renvoie pour une liste donnée l le minimum de la liste ainsi que le reste de la liste :

- Si la liste n'a qu'un élément, le couple renvoyer est alors cet élément, et la liste vide.
- Sinon, on procède à la recherche du couple de la liste privé de sa tête, et on compare les résultats à la tête de liste.

2. Écrire la fonction `tri_selec` qui renvoie la liste triée par sélection.

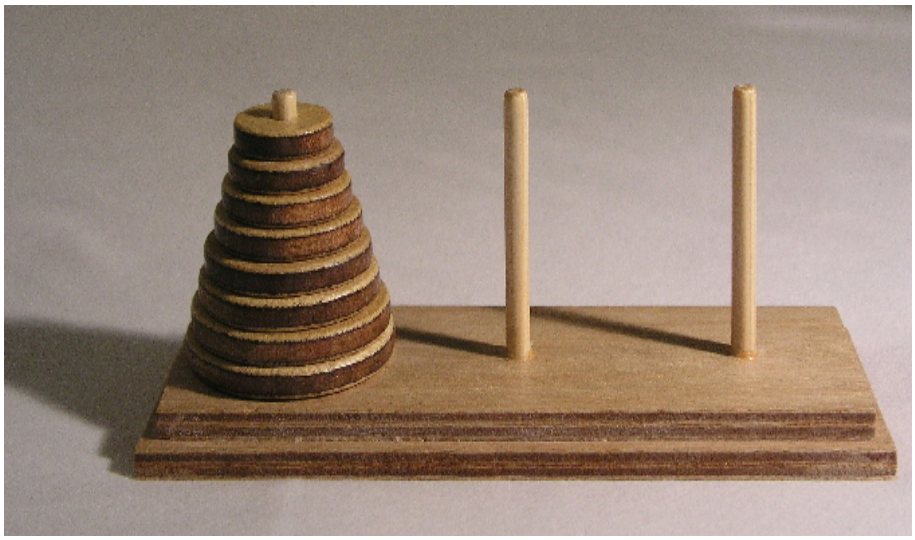
Correction

```
#exercice 2

def min_list(liste):
    """ la fonction renvoie le minimum de la liste et le reste """
    if len(liste) == 0 :
        print("erreur de liste")
    elif len(liste) == 1 :
         return (liste[0] , [] )
    else :
        tete = liste[0]
        reste = liste[1:]
        (mini_reste , reste_reste ) = min_list(reste)
        if tete < mini_reste :
            return (tete , [mini_reste] + reste_reste )
        else :
            return ( mini_reste , [tete] + reste_reste )


def tri_select(liste ) :
    if len(liste) == 0 :
        return []
    else :
        (mini , reste ) = min_list(liste)
        return [mini] + tri_select(reste)
```

Exercice 3 : Les tours de Hanoï (originellement, la tour d'Hanoïa) sont un jeu de réflexion imaginé par le mathématicien français Édouard Lucas, et consistant à déplacer des disques de diamètres différents d'une tour de « départ » à une tour d' « arrivée » en passant par une tour « intermédiaire » , et ceci en un minimum de coups, tout en respectant les règles suivantes :

- on ne peut déplacer plus d'un disque à la fois ;
- on ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide.

On suppose que cette dernière règle est également respectée dans la configuration de départ.



1. Télécharger le fichier `exo_3.py` sur le site. Résoudre le problème en un minium de coups...
2. Ajouter la méthode `auto` qui prend en argument un entier n, représentant le nombre de disques, et trois entiers représentant les tours 0, 1 et 2. La fonction aura comme but de résoudre le problème par la méthode récursive.

Correction

```
class Galette :
    def __init__(self,etoile):
        self.etoile = etoile # nb d etoiles

    def dessine(self):
        if self.etoile !=0:
            return ["␣" for i in range(4-self.etoile//2)] +  ["*" for i in
range(self.etoile)]+\
                        ["␣" for i in range(4-self.etoile//2)]
        else :
            return [ "␣" for i in range(4)]+["|"]+[ "␣" for i in range(4)]

class Dessin :
    def __init__(self) :
        self.nb_coup = 0
        self.tour = [0,5,5] # position libre
        self.tous_les_galets = [ [Galette(0)] +[Galette(i) for i in range
(1,10,2) ] , \
                                    [Galette(0) for i in range(6)] , [Galette(0)
for i in range(6)] ]

    def affichage_galets(self):
        for i in range(6) :
            for j in range(3) :
                for k in self.tous_les_galets[j][i].dessine():
                    print(k , end = "␣")
            print("␣")

    def echange_position(self,t_depart,t_arrivee) :
        g = self.tous_les_galets[t_depart][self.tour[t_depart]+1]
        self.tous_les_galets[t_depart][self.tour[t_depart]+1] = Galette(0)
        self.tous_les_galets[t_arrivee][self.tour[t_arrivee]]= g
        self.tour[t_depart] = self.tour[t_depart]+1
        self.tour[t_arrivee]=self.tour[t_arrivee] -1

    def possibilite_deplacement(self,t_d,t_a) :
        try :
            g_d = self.tous_les_galets[t_d][self.tour[t_d]+1]
            if self.tour[t_a] == 5 :
                return True
            else :
                g_a = self.tous_les_galets[t_a][self.tour[t_a]+1]
                return g_a.etoile > g_d.etoile
        except :
```

```
            return False

    def verification_fin_de_jeu(self):
        g = self.tous_les_galets[2][1]
        return g.etoile == 1

    def jeu(self):
        self.affichage_galets()
        while not(self.verification_fin_de_jeu()) :
            t_d = int(input("tour de depart : "))
            t_a = int(input("tout d'arrivee : "))
            if self.possibilite_deplacement(t_d,t_a) :
                self.echange_position(t_d,t_a)
                self.affichage_galets()
                self.nb_coup +=1
            else :
                print("deplacement impossible")
        print("gagne en " + str(self.nb_coup))



if __name__ == "__main__" :
    j = Dessin()
    j.jeu()
```