# MPSI - ITC - TD 9
# Récursivité 2

## Site pour exécuter du python pas à pas

L'adresse du site est https://pythontutor.com/python-compiler.html

## Point de cours sans lien avec le TD

Si une variable `u` contient une paire, on peut déconstruire cette paire avec une affectation de la forme :

```
(u0, u1) = u
```

Mais **attention**, si vous écrivez une fonction qui prend en paramètre des paires, vous ne pouvez pas déconstruire ces paires dès l'en-tête de la fonction. Ainsi, si vous voulez écrire une fonction produit scalaire, c'est incorrect d'écrire

```
def scalaire((u0, u1), (v0, v1)):
    return u0*v0  + u1*v1
```

Par contre une façon correcte de l'écrire est

```
def scalaire(u, v):
    (u0, u1) = u
    (v0, v1) = v
    return u0*v0  + u1*v1
```

Une autre possibilité est bien sûr d'utiliser des crochets pour accéder aux éléments des paires, comme pour une liste :

```
def scalaire(u, v):
    return u[0]*v[0]  + u[1]*v[1]
```

*(Cette erreur a été tellement fréquente au DS 1 que je n'ai pas enlevé de points quand je l'ai vue. Mais ne la refaites pas.)*

## Point de cours : fonctions auxiliaires

On peut définir une fonction à l'intérieur d'une autre fonction.
Par exemple, si on veut une fonction qui affiche la table de multiplication d'un entier `n` donné, on peut définir

```
def table(n):
    def ligne(i): ############################## <- Début de la fonction ligne
```

```
        print(n, " fois ", i, " vaut ", n*i) #
        ##################################### <- fin de la fonction ligne
    for i in range(1, 11):
        ligne(i)
    # Fin de la fonction table
```

Ensuite, si on effectue l'appel `table(7)` on obtient l'affichage

```
7  fois  1  vaut  7
7  fois  2  vaut  14
7  fois  3  vaut  21
7  fois  4  vaut  28
7  fois  5  vaut  35
7  fois  6  vaut  42
7  fois  7  vaut  49
7  fois  8  vaut  56
7  fois  9  vaut  63
7  fois  10  vaut  70
```

**Exercice 1** (Recherche dichotomique récursive)

Coder la recherche dichotomique en utilisant une fonction auxiliaire récursive.
Votre code aura la forme suivante

```
def recherche(liste, x):
    """
    Entrées
    ---------
      x : un nombre
      liste :  une liste de nombres, triée par ordre croissant

    Sortie
    -------
      True si x apparaît dans liste, False sinon
    """
    def aux(i, j):
        """
        Indique si x apparaît dans liste
        entre les indices i (inclus) et j (exclu).
        """
        # À compléter...
        # ...
    # Fin de la fonction aux
    return aux(0, len(liste))
```

*(Pour des explications sur la recherche dichotomique, revoir le cours de la semaine 7, juste avant les vacances. En résumé l'idée du cas récursif est « On regarde au milieu de la plage en cours d'exploration. Si l'élément cherché est plus petit que celui au milieu, on cherche dans la partie gauche de la plage. Si l'élément cherché est plus grand que celui au milieu, on cherche dans la partie droite de la plage. »)*

**Exercice 2**

Finir le TD de la semaine dernière.

**Exercice 3** (Bonus : tri par insertion dichotomique)

Écrire un tri par insertion (voir DS2) où l'étape d'insertion est faite par une fonction auxiliaire récursive.