Algorithmique et programmation avancée
« Récursivité »

Exercice N° 1 :

Concevoir version itérative et récursive d'une fonction « **def somme(n)** », qui calcule la somme suivante :

$$\sum_{i=1}^{n} i = 1 + \ldots + n$$

Exercice N° 2 :

Concevoir la version récursive d'une fonction « **def puissance(x, n)** », qui renvoie x^n, pour deux entiers positifs x et n passés en paramètre.

Exercice N° 3 :

Le calcul de PGCD par l'algorithme d'Euclide est un algorithme fondamentalement récursif. En effet, il utilise la propriété suivante :

pgcd(a; b) = pgcd(b; a%b)

Toute fonction récursive doit contenir une condition d'arrêt. Pour le PGCD, on a :

pgcd(a; 0) = |a|

Programmer la fonction PGCD de manière récursive.

Exercice N° 4 :

Concevoir la version récursive d'une fonction de calcul de la suite de Fibonacci :

$U_0 = 0$;

$U_1 = 1$;

$U_{n+2} = U_n + U_{n+1}$

Exercice N° 5 :

Ecrire une fonction « **def combinaisons(n, p)** »qui calcule le nombre des combinaisons en se servant de la relation de Pascal :

$$C_n^p \begin{cases} 1 & \text{si p=0 ou p=n} \\ C_{n-1}^p + C_{n-1}^{p-1} & \text{sinon} \end{cases}$$

Exercice N° 6 :

Ecrire la fonction récursive « def bin(n) » qui retourne en binaire la conversion d'un entier positif n.

Exercice N° 7 :

Ecrire la fonction récursive « def taille(L) » qui renvoie le nombre d'éléments d'une liste L.

Exercice N° 8 :

L'objet de cet exercice est de chercher un élément dans un tableau trié.

Lorsque le tableau est trié, La recherche est plus facile. La méthode dichotomique consiste à chaque étape à découper le tableau en deux parties, l'élément du milieu permettant de savoir dans quelle partie du tableau se trouve l'élément recherché.

a) Ecrivez la fonction de recherche dichotomique (itérative) qui prend en entrée une liste **L** d'entier et un entier m et retourne son indice dans le tableau s'il y est présent, −1 sinon.

   def rechercheDichoIt(L, m)

b) Ecrivez la fonction de recherche dichotomique (version récursive) qui prend en entrée une liste d'entiers, un entier m, **i** l'indice du premier élément de la liste (initialement 0) et **j** indice du dernier élément de la liste (initialement len(L)-1) et retourne son indice dans le tableau s'il y est présent, −1 sinon

   def rechercheDichoRec(L, m , i, j)