Bonne année !

1. Retour sur les bases (7 pts)

1) Soit T=[1,2,3,4]

a) Cela renvoie le booléen False. En effet cela teste si T[0], le premier élément de T, vaut 2.
b) Cela modifie le premier élément de la liste T en le remplaçant par 2. La variable T est désormais la liste [2,2,3,4].
c) Cela renvoie 3
d) Cela renvoie tous les éléments de T à partir de celui d'indice 1, donc la liste [2,3,4]
e) Cela renvoie le type de T, à savoir le type *list*.

2) Exprimer les nombres suivants en base 10.

$(11010)_2 = 26$
$(D0)_{16} = 13 * 16 = 208$

3) Ecrire une fonction qui prend en entrée une liste, et remplace son dernier élément par l'entier naturel 0.

```
def modifie(L) :
    L[len[L]]=0
    Return L
```

4) Écrire un code permettant d'afficher à l'écran tous les triplets pythagoriciens (triplets d'entiers (a ; b ; c) tels que $a^2 + b^2 = c^2$), tels que $1 \leq a \leq b \leq c \leq 1000$.
On choisit de retourner le résultat comme une liste de triplet (chaque tripler étant une liste).

a) On choisit de retourner le résultat comme une liste de triplet (chaque triplet étant une liste).

```
def tri_pyt():
L=[]
for i in range(1,n+1):
     for j in range(i,n+1):
          for k in range(j,n+1):
               if k**2==i**2+j**2:
                    L.append([i,j,k])
return L
```

b) On a un triple parcours imbriqué des entiers entre 1 et n, on peut s'attendre à une complexité en $O(n^3)$.

2. Algorithmique (10 pts)

5) Factorielle (13 pts)

```
def factorielle(n) :
    if type(n)!=int :
        return False
    elif n==0 :
        return 1
    else :
        a=1
        for i in range(1,n+1) :
            a=a*i
    return a
```

6) Suite récurrente linéaire 1

Soit la suite U_{n+1} telle que : $U_{n+1} = 3U_n + 2$ avec $U_0 = 4$

a) Écrire la fonction `suite(n)` qui prend en argument un entier n et qui retourne la liste U des valeurs de la suite de 0 à n inclus.
Parmi les nombreuses solutions possibles, on fabrique une liste à laquelle on concatène les valeurs de la suite calculée successivement :

```
def suite(n):
    u=[4]
    for i in range(1,n+1):
        u.append(3*u[i-1]-2)
    return u
```

b) On peut montrer que la suite converge. Proposer un autre code, impliquant un critère d'arrêt : on calcule les valeurs de la suite jusqu'à ce que la différence U_{n+1}-U_n, en valeur absolue, soit inférieure ou égale à 10^{-10}. Alors, le code s'arrête, affiche ''la suite n'évolue plus'' et renvoie la valeur U_n ainsi que le nombre d'itération.

On ne demande plus de stocker les valeurs de la suite, seules les deux dernières suffisent.

```
U0=4
U1=10
I=0
while abs(U1-U0)>eps:
    i=i+1 #Le compteur qui va donner le nombre d'itérations
    U0=U1
    U1=3*U1-2
print("la suite ne bouge plus")
print(U1,i)
```

7) Équation non linéaires

On recherche une solution de l'équation $f(x) = 0$ sur l'intervalle [a,b] avec f une fonction connue.

a) Préciser les conditions nécessaires sur f pour qu'il existe une unique solution sur [a,b].
D'après le théorème de la bijection, la fonction f est bijective. L'équation $f(x) = 0$ admet donc une unique solution.

b)

```
def dichotomie(f,a,b,eps) :
    while abs(f((b-a)/2))>epsilon:  # Critère de convergence (erreur)
        m=(a+b)/2          # m, le milieu de a,b
        if f(m)>=0:        # Si solution dans intervalle de gauche
            b=m            # On prend partie gauche
        else:              # Sinon l'inverse.
            a=m
    return m
```

c) Il n'y a qu'une seule ligne à modifier !! par lisibilité, on remet l'ensemble du code, mais dans votre copie, essayez d'être efficace et de bien expliquer.

```
def dichotomie2(f,a,b,epsilon):
    assert f(a)*f(b)<=0        # Vérification existence racine
    c=0                        # Initialisation compteur
    while abs(f((b-a)/2))>epsilon:  # Critère de convergence (erreur)
        m=(a+b)/2.             # m, le milieu de a,b
        if f(a)*f(m)<=0:# Si solution dans intervalle de gauche
            b=m            # On prend partie gauche
        else:              # Sinon l'inverse.
            a=m
    return m
```

8) Variant

On considère le programme suivant :

```
while p > 0 :
    if c == 0 :
        p = p-2
        c = 1
    else :
        p+ = 1
        c = 0
```

a) Donner les états successifs (c'est à dire les valeurs des variables globales) obtenus pendant l'exécution.

itération	0	1	2	3	4	5	6	7
P	5	3	4	2	3	1	2	0
C	0	1	0	1	0	1	0	1
2p+3c	10	9	8	7	6	5	4	3

b) Voir cours

c) Avec une suite finie et strictement décroissante d'entiers naturels, alors la simple existence de v permet de prouver que l'algorithme se termine effectivement.