Interrogation 4 - Programme de révision

I/ Suites

1. Savoir-faire : étude d'une suite définie explicitement

- Calcul d'un terme
- Calcul de la somme des termes
- Liste des termes

Exemple
(u_n) définie par : $\forall n \in \mathbb{N},\ u_n = n^2 + 3$

```
'''Calcul de u_n'''
def suite_u(n):
    return n**2+3

'''Calcul de la somme u_0 + ... + u_n'''
def somme_u(n):
    S = 0
    for k in range(0,n+1):
        S = S + k**2+3
    return S

'''Liste de u_0 , ... , u_n'''
def liste_u(n):
    return [k**2+3 for k in range(0,n+1)]
```

2. Savoir-faire : étude d'une suite récurrente d'ordre 1 sans second membre

- Calcul d'un terme
- Calcul de la somme des termes
- Liste des termes
- Seuil (si on *sait* que $\lim_{n\to+\infty} v_n = +\infty$)

Exemple
(v_n) définie par : $v_0 = 2$ et pour tout $n \in \mathbb{N}$, $v_{n+1} = v_n^2$

```
'''Calcul de v_n'''
def suite_v(n) :
    if n == 0 :
        return 2
    else :
        return suite_v(n-1)**2

'''Calcul de la somme v_0 + ... + v_n'''
def somme_v(n) :
    v = 2
    S = 2
    for k in range(1,n+1) :
        v = v**2
        S = S + v
    return S

'''Liste de v_0 , ... , v_n'''
def liste_v(n) :
    v = 2
    L = [2]
    for k in range(1,n+1) :
        v = v**2
        L.append(v)
    return L

'''Premier terme v_n tel que M< v_n'''
def seuil_strict_terme_v(M) :
    v = 2
    while v <= M :
        v = v**2
    return v

'''Premier terme v_n tel que M<= v_n'''
def seuil_large_terme_v(M) :
    v = 2
    while v < M :
        v = v**2
    return v

'''Premier rang n tel que M< v_n'''
def seuil_strict_rang_v(M) :
    v = 2
    n = 0
    while v <= M :
        v = v**2
        n = n + 1
    return n

'''Premier rang n tel que M<= v_n'''
def seuil_large_rang_v(M) :
    v = 2
    n = 0
    while v < M :
        v = v**2
        n = n + 1
    return n
```

3. Savoir-faire : étude d'une suite récurrente d'ordre 1 avec second membre

- Calcul d'un terme
- Calcul de la somme des termes
- Liste des termes
- Seuil (si on *sait* que $\lim\limits_{n\to+\infty} v_n = +\infty$)

La structure des fonctions est la même que pour les suites sans second membre.
La seule différence concerne le calcul d'un terme, puisqu'on ne peut plus utiliser de fonction récursive.

Exemple
(w_n) définie par : $w_0 = 1$ et pour tout $n \in \mathbb{N}$, $w_{n+1} = \sqrt{w_n} + (n+1)$

```
from math import *
def suite_w(n):
    w = 1
    for k in range(0,n) :
        w = sqrt(w)+k+1
    return w
```

4. Savoir-faire : étude d'une suite récurrence d'ordre 2 sans second membre

- Calcul d'un terme

Exemple
(x_n) définie par : $x_0 = 1$, $x_1 = 2$ et pour tout $n \in \mathbb{N}$, $x_{n+2} = x_{n+1} + x_n$

```
def suite_x(n):
    if n==0:
        return 1
    elif n==1:
        return 2
    else:
        return suite_x(n-1)+suite_x(n-2)
```

II/ Listes

1. Savoir : recherche d'un élément par parcours complet

```
def recherche (L,x) :
    for e in L :
        if e == x :
            return True
    return False
```

2. Savoir : recherche du maximum

```
def maximum (L) :
    M = L[0]
    for e in L :
        if e > M :
            M = e
    return M
```

3. Savoir : calcul de la moyenne

```
def moyenne (L) :
    S = 0
    for e in L :
        S += e
    return S / len(L)
```

4. Savoir : recherche d'un élément par dichotomie

```
def recherche_dichotomie(L,x):
    a = 0
    b = len(L)-1
    while a <= b :
        m = (a+b) // 2
        if L[m] == x :
            return True
        elif L[m] < x :
            a = m + 1
        else :
            b = m - 1
    return False
```

III/ Fonctions

Savoir : recherche dichotomique

- Pour une fonction pour laquelle les conditions sont remplies :

```
def dichotomie_fonction(f,a,b,e) :
    while (b-a) >= e :
        c = (a+b)/2
        if f(a)*f(c) <= 0 :
            b = c
        else :
            a = c
    return (a,b)
```

- Utilisation de la commande **lambda** dans le Shell. Par exemple :
dichotomie_fonction(lambda x : x**2 - 2 , 0 , 2 , 10**(-5))