TD3.1bio2.py

```
001| ## Exo 1 : Nombre de surjections
002| print("Exo 1 : Nombre de surjections")
003| def binome(n,k):
004|     if k > n/2:
005|         k = n-k #symétrie des coefficients binomiaux
006|     p1 = p2 = 1
007|     for i in range(k):
008|         p1 *= n-i
009|         p2 *= k-i
010|     return p1//p2
011|
012| # Alternative utilisant la formule du pion itéré
013| def binomeBis(n,k):
014|     if k > n/2:
015|         k = n-k #symétrie des coefficients binomiaux
016|     p = 1
017|     for i in range(k):
018|         p *= (n-i)/(k-i)
019|     return round(p)
020|
021| # Alternative récursive utilisant la formule du pion itéré
022| def binomeTer(n,k):
023|     if k > n/2:
024|         k = n-k #symétrie des coefficients binomiaux
025|     if k == 0:
026|         return 1 # cas de base
027|     return round(n/k*binomeTer(n-1,k-1))
028|
029| def nb_surj(p,n):
030|     if p<n:
031|         return 0
032|     s = 0
033|     for k in range(1,n+1):
034|         s += (-1)**(n-k)*binome(n,k)*k**p
035|     return s
036|
037| print("Vérifications :")
038| s1=nb_surj(3,2)
039| print("Il y a {} surjections de [[1,{}]] dans [[1,{}]]".format(s1,3,2))
040| s2=nb_surj(3,3)
041| print("Il y a {} surjections de [[1,{}]] dans [[1,{}]]".format(s2,3,3))
042| s3=nb_surj(3,4)
043| print("Il y a {} surjection de [[1,{}]] dans [[1,{}]]".format(s3,3,4))
044| s4=nb_surj(4,2)
045| print("Il y a {} surjections de [[1,{}]] dans [[1,{}]]".format(s4,4,2))
046|
047| ## Exo 2 : Factorielle, le retour
048| print()
049| print("Exo 2 : Factorielle, le retour")
050| def factorielle(n):
051|     if n<0 or n != int(n):
052|         return "n doit être un entier naturel"
053|     n = int(n) # si n est de la forme 5.0 on le transforme en 5
054|     f = 1
055|     for k in range(2,n+1):
056|         f *= k
057|     return f
058|
059| print("Vérifications :")
060| a=factorielle(-1)
061| print("factorielle(-1) renvoie : {}".format(a))
062| b=factorielle(3.5)
063| print("factorielle(3.5) renvoie : {}".format(b))
064| c=factorielle(4)
065| print("factorielle(4) renvoie : {}".format(c))
066| d=factorielle(5.0)
```

```
print("factorielle(5.0) renvoie : {}".format(d))

## Exo 3 : Manipulation d'une liste d'entiers
print()
print("Exo 4 : Manipulation d'une liste d'entiers")
import random as rd
def manip(lst,n):
    p=len(lst)
    for k in range(p-1,-1,-1):
        if lst[k]==n:
            lst.pop(k)
    lst.reverse()
    p=len(lst)
    i=rd.randint(0,p)
    q=rd.randint(1,10)
    lst.insert(i,q)
    j=rd.randint(0,p)
    lst.pop(j)
    lst.append(sum(lst))

## Exo 4 : Saisie de notes
print()
print("Exo 4 : Saisie de notes")
n = eval(input("Entrez le nombre de notes : "))
lst = []
note_elim = eval(input("Entrer la valeur de la note  éliminatoire \
(entrer -1 s'il n'y en a pas) : "))
elimine = False
for k in range(n):
    a = eval(input("Entrer une note : "))
    if a <= note_elim:
        elimine = True
        break
    lst.append(a)
if elimine:
    print("Ce candidat est éliminé")
else:
    print("La moyenne du candidat est :", sum(lst)/len(lst))

def saisie(n):
    lst = []
    note_elim = eval(input("Entrer la valeur de la note  éliminatoire \
(entrer -1 s'il n'y en a pas) : "))
    for k in range(n):
        a = eval(input("Entrer une note : "))
        if a <= note_elim:
            return "Ce candidat est éliminé"
        lst.append(a)
    return sum(lst)/len(lst)

## Exo 5: Concaténation des listes des termes d'indices pairs et impairs
print()
print("Exo 5 : Concaténation des listes des termes d'indices pairs et impairs")

print("Solution avec une seule boucle")
# k%2 renvoie le reste de la division euclidienne de k par 2
def scission_parite1(lst):
    n = len(lst)
    lst0, lst1 = [], [] # et surtout pas lst0 = lst1 = []
    for k in range(n):
        if k%2: # k est impair
            lst1.append(lst[k])
        else: # k est pair
            lst0.append(lst[k])
    return lst0+lst1, max(lst0)-min(lst0), max(lst1)-min(lst1)

print("Vérification :")
lst=[3,5,2,1,4]
```

```
a=scission_parite1(lst)
print("scission_parite1([3,5,2,1,4]) renvoie :", a)

print("Solution avec deux boucles")
def scission_parite2(lst):
    n = len(lst)
    lst0, lst1 = [], [] # et surtout pas lst0 = lst1 = []
    for k in range(0,n,2):
        lst0.append(lst[k])
    for k in range(1,n,2):
        lst1.append(lst[k])
    return lst0+lst1, max(lst0)-min(lst0), max(lst1)-min(lst1)

print("Vérification :")
lst=[3,5,2,1,4]
b=scission_parite2(lst)
print("scission_parite2([3,5,2,1,4]) renvoie :", b)

#print("Solution par slicing (pour les plus aguerris)")
#lst[n:p:q] est une extraction de lst de n (inclus) à p (exclu) avec un pas de q
def scission_parite_sclice(lst):
    n = len(lst)
    lst0, lst1 = lst[::2], lst[1::2]
    return lst0+lst1, max(lst0)-min(lst0), max(lst1)-min(lst1)

## Exo 6 : Jeu de devinette humain contre ordinateur
print()
print("Exo 6 : Jeu de devinette humain contre ordinateur")
import random as rd
secret, cptE = rd.randint(1,50), 1
essai = eval(input("Entrer un nombre entre 1 et 50 : "))
while essai != secret:
    if essai < secret:
        print("Votre nombre est trop petit")
    else:
        print("Votre nombre est trop grand")
    essai = eval(input("Entrer un nouveau nombre entre 1 et 50 : "))
    cptE += 1
print("Vous avez trouvé le nombre",secret,"en",cptE,"essais")

## Exo 7 : L’ordinateur joue contre lui-même
print()
print("Exo 7 : L’ordinateur joue contre lui-même")

def strategie1():
    """
    renvoie le nombre d'essais pour trouver secret dans une stratégie sans
    mémoire. Simulation de la loi géométrique (prgm bcpst2) de paramètre 1/50
    """
    secret = rd.randint(1,50)
    cptE = essai = 0
    while essai != secret:
        essai = rd.randint(1,50)
        cptE += 1
    return cptE

def strategie2():
    """
    renvoie le nombre d'essais en rejetant les essais déjà effectués, suit U(50)
    """
    secret = rd.randint(1,50)
    cptE = essai = 0
    nonchoisis = [k for k in range(1,51)]
    while essai != secret:
        n = len(nonchoisis)
        i = rd.randint(0,n-1)
        essai = nonchoisis.pop(i)
        cptE += 1
```

```
    return cptE # ou 51 - n

def strategie3():
    """
    renvoie le nombre d'essais dans une strategie d'essais aléatoires prenant
    en compte la position relative des essais par rapport à secret
    """
    secret = rd.randint(1,50)
    cptE = essai = 0
    a, b = 1, 50
    while essai != secret:
        essai = rd.randint(a,b)
        cptE += 1
        if essai > secret:
            b = essai - 1
        elif essai < secret:
            a = essai + 1
    return cptE

def strategie4():
    """
    renvoie le nombre d'essais dans une strategie de recherche dichotomique
    prenant en compte la position relative des essais par rapport à secret
    """
    secret = rd.randint(1,50)
    cptE = essai = 0
    a, b = 1, 50
    while essai != secret:
        essai = (a+b)//2
        cptE += 1
        if essai > secret:
            b = essai - 1
        elif essai < secret:
            a = essai + 1
    return cptE

def moy(n=10000):
    lst = [0]*4
    for _ in range(n):
        lst[0] += strategie1()
        lst[1] += strategie2()
        lst[2] += strategie3()
        lst[3] += strategie4()
    return [k/n for k in lst]

print("Exo 7 question 7 : Comparaison des 4 stratégies sur 10000 simulations")
print("moy() renvoie {}".format(moy()))
print("Cette liste est décroissante par conséquent :")
print("ces stratégies sont classées par ordre croissant d'efficacité.")
print("La stratégie sans mémoire trouve, en moyenne, le nombre")
print("en une cinquantaine de tentatives.")
print("À l'opposé, la stratégie dichotomique trouve, en moyenne, le nombre")
print("en moins de 5 tentatives.")

# Les espérances des 4 stratégies sont dans l'ordre : 50, 25.5, 6.18, 4.86.

## Exo 8: Algorithme et suite de Kaprekar

print()
print("Exo 8 : Algorithme et suite de Kaprekar")
# question 1
def digits(n):
    lst = list(str(n))
    lst.sort()
    return [eval(k) for k in lst]

# question 2
def conversion(lst):
```

```
    n = len(lst)
    nbre = 0
    for k in range(1,n+1):
        nbre += lst[-k]*10**(k-1)
    return nbre

# question 3
def kaprekar(n):
    lst = digits(n)
    n1 = conversion(lst)
    lst.reverse()
    n2 = conversion(lst)
    return n2 - n1

# question 4
def suiteK(n):
    suite, u = [], n
    while not u in suite and u != 0:
        suite.append(u)
        u = kaprekar(u)
    suite.append(u)
    return suite

# question 5
print("Exo 8 question 5 : Vérifications de la fonction suiteK")
print("suiteK(24) renvoie", suiteK(24))
print("suiteK(53955) renvoie", suiteK(53955))
```