ANNÉE UNIVERSITAIRE : 2025 – 2026
UNITÉ DE RECHERCHE : Larema
ÉQUIPE : Analyse & Probabilités
SUJET : Programmation sous Python

SOLUTIONS DES EXERCICES DU TP2 [1]

Mohamed Alfaki ABOUBACRINE ASSADEK

SOLUTION DE L'EXERCICE 1 DU TP2

```python
import math # Pour les maths
from math import sqrt # Racine carree

# Resolution dans R des equations de degre 2
def solEquationD2R(a, b, c):
    # Interruption du programme si l'equation n'est pas de degre 2
    if a==0:
        raise ValueError("Ce n'est pas une equation du 2nd degre !")

    # Determinant
    d = b**2 - 4*a*c

    # Si b=0, les solutions sont immediates
    if b==0:
        if c==0:
            Lst = [0]
        elif c/a < 0:
            Lst = [ -sqrt(-c/a), sqrt(-c/a) ]
        else:
            Lst = []

    # Sinon, on ecrase le determinant a zero s'il en est "trop proche"
    # pour eviter des tests d'egalite errones
    # Ensuite on calcule les solutions selon la methode usuelle
    else:
        eps = 10**(-15)
        if abs(d)/b**2 <= eps:
            d = 0

        if d > 0:
            Lst = [ (-b-sqrt(d))/(2*a), (-b+sqrt(d))/(2*a) ]
        elif d < 0:
            Lst = []
        else:
            Lst = [ -b/(2*a) ]

    # On renvoie les solutions sous forme de liste (eventuellement vide)
    return Lst


# Resolution dans C des equations de degre 2
def solEquationD2C(a, b, c):
    # Interruption du programme si l'equation n'est pas de degre 2
    if a==0:
        raise ValueError("Ce n'est pas une equation du 2nd degre !")

    # Determinant
    d = b**2 - 4*a*c
```

1. Voir TP2 pour les énoncés

```
49
50      # Si b=0, les solutions sont immediates
51      if b==0:
52          if c==0:
53              Lst = [0]
54          elif c/a < 0:
55              Lst = [ -sqrt(-c/a), sqrt(-c/a) ]
56          else:
57              Lst = [ -1j*sqrt(c/a), 1j*sqrt(c/a) ]
58
59      # Sinon, on ecrase le determinant a zero s'il en est "trop proche"
60      # pour eviter des tests d'egalite errones
61      # Ensuite on calcule les solutions selon la methode usuelle
62      else:
63          eps = 10**(-15)
64          if abs(d)/b**2 <= eps:
65              d = 0
66
67          if d > 0:
68              Lst = [ (-b-sqrt(d))/(2*a), (-b+sqrt(d))/(2*a) ]
69          elif d < 0:
70              Lst = [ (-b-1j*sqrt(-d))/(2*a), (-b+1j*sqrt(-d))/(2*a) ]
71          else:
72              Lst = [ -b/(2*a) ]
73
74      # On renvoie les solutions sous forme de liste
75      return Lst
76
77
78  # Quelques tests
79  solEquationD2R(1, -2, 1)
80  solEquationD2R(-1, 5, -6)
81  solEquationD2R(1, 0, -4)
82  solEquationD2R(1, 0, 4)
83  solEquationD2C(1, 0, 4)
84  solEquationD2C(0, 1, 1)
85  for k in range(1, 10):
86      Lst = solEquationD2R(1, 1/10**k, 1/(4*10**(2*k)))
87      print(Lst)
```

SOLUTION DE L'EXERCICE 2 DU TP2

```
1  # Premiers termes de la somme des inverses des carres
2  def euler2(n):
3      s = 0
4      for j in range(1, n+1):
5          s += 1/j**2
6      return s
7
8  #Test
9  euler2(20)
10
11
12 # Premiers termes d'une double somme
13 def euler3_jk(n):
14     s = 0
15     for j in range(1, n+1):
16         t = 0
17         for k in range(1, n+1):
18             t += 1/(j**2 + k**3)
19         s += t
20     return s
21
22
23 # Premiers termes de la meme double somme en permutant l'ordre de sommation
24 def euler3_kj(n):
25     s = 0
```

```
    for k in range(1, n+1):
        t = 0
        for j in range(1, n+1):
            t += 1/(j**2 + k**3)
        s += t
    return s

#Test
euler3_jk(20)
euler3_kj(20)
# Sommes identiques a un arrondi pres (approximations numeriques successives)
```

SOLUTION DE L'EXERCICE 3 DU TP2

```
# Trois manieres de creer la meme liste [0, 1, 4, 9, 16, ..., n**2]
n = 10**7
L = []
for j in range(n+1):
    L.append(j**2)

L = [ j**2 for j in range(n+1) ]

def carre(j):
    return j**2

L = map(carre, range(n+1))
list(L)

%timeit [ j**2 for j in range(n+1) ]
%timeit map(carre, range(n+1))
# On voit qu'en termes de temps d'execution, la troisieme est de loin la meilleure
# Elle evite les couteuses boucles "for" par un mapping direct (vectoriel)
```

SOLUTION DE L'EXERCICE 4 DU TP2

```
# Algorithme d'Euclide pour calculer PGCD(a, b)
def pgcd(a, b):
    cpt = 0
    print(f"Boucle {cpt} : a = {a} et b = {b}")
    while b!=0:
        cpt += 1
        a, b = b, a%b
        print(f"Boucle {cpt} : a = {a} et b = {b}")
    return a

#Tests
pgcd(50, 10)
pgcd(10001, 76)
pgcd(1444323442333, 133345676)
```

SOLUTION DE L'EXERCICE 5 DU TP2

```
# Calcul de la somme de Euler 1/1 + 1/4 + 1/9 + ... + 1/N**2
# On s'arrete des que la limite pi**2/6 est approximee avec d decimales exactes
from math import pi

def eulerDecimalesExactes(d):
    lim = pi**2/6
    limd = int(lim*10**d)
    S = 0
    N = 0
    while not int(S*10**d) == limd:
        N += 1
```

```
        S += 1/N**2

    return S, N

# Calcul de la somme 1/1 + 1/8 + 1/27 + ... + 1/N**3
# On s'arrete des que la limite est approximee avec d decimales exactes
def eulerDecimalesExactes2(d):
    S = 0
    N = 0
    while (N+1)**2<10**d:
        N += 1
        S += 1/N**3

    return S, N

#Test
N=200
# Premiers termes de la somme des inverses des cubes
def euler3(n):
    s = 0
    for j in range(1, n+1):
        s += 1/j**3
    return s

# Quatres decimales exactes
Euler3(N)
eulerDecimalesExactes2(4)
```

SOLUTION DE L'EXERCICE 6 DU TP2

```
# Liste et somme des diviseurs stricts de n
def sommeDiviseurs(n):
    L = [1]
    s = 1
    for i in range(2, int((n+1)/2)+1):
        if n%i == 0:
            L.append(i)
            s += i
    return L, s


# Liste des nombres parfaits inferieurs ou egaux a n
def nbparfaits(n):
    L = []
    for i in range(2, n+1):
        if sommeDiviseurs(i)[1] == i:
            L.append(i)
    return L

# Liste des couples de nombres amicaux inferieurs ou egaux a n
# On utilise une algorithmie soignee pour booster les temps de calcul
def nbamicaux(n):
    L = []
    LstSomDiv = [ sommeDiviseurs(i)[1] for i in range(2, n+1) ]
    for a in range(2, n+1):
        if a in LstSomDiv:
            b = LstSomDiv.index(a)+2
            if b!=a and LstSomDiv[a-2] == b:
                if (b, a) not in L:
                    L.append((a, b))
    return L

#Test
nbamicaux(1000)
```

SOLUTION DE L'EXERCICE 7 DU TP2

```python
# Fonction qui supprime les doublons d'une liste
def suppr_doublons(L):
    # U est initialement vide
    U = []

    # On parcourt chaque element de L
    for e in L:
        # Si l'element n'est pas encore dans U, on l'insere a la fin
        if e not in U:
            U.append(e)
    return U

#Test
suppr_doublons([2,4,1,1,4,5])
```

SOLUTION DE L'EXERCICE 8 DU TP2

```python
# Fonction qui teste si n est premier ou non
from math import sqrt

def estPremier(n):
    #if sommeDiviseurs(n)[0] == [1]:
        #return True
    #return False
    for i in range(2, int(sqrt(n)+1)+1):
        if i!=n and n%i == 0:
            return False
    return True


# Liste des nombres premiers inferieurs a n par recherche individuelle
def nbpremiers(n):
    L = [ i for i in range(2, n+1) if estPremier(i) ]
    return L


# Liste des nombres premiers inferieurs a n par le crible d'Eratosthene
def eratosthene(n):

    # Initialisation du masque : 0 et 1 sont exclus d'emblee
    Test = [False for i in range(0, 2)] + [True for i in range(2, n+1)]
    i = 2

    # On cherche jusqu'a n/2 (on pourrait faire mieux...)
    while i <= int(n/2):

        # Recherche du prochain element True dans le masque (avant n/2)
        while not Test[i] and i <= n/2:
            i += 1

        # S'il n'y en a plus, on s'arrete
        if i > n/2:
            break

        # Ensuite, on met tous ses multiples a False
        for k in range(2*i, n+1, i):
            Test[k] = False

        # Et on passe au suivant
        i += 1

    # Selon le principe du crible, les nombres premiers sont donc ceux dont le masque est
    reste a True
    L = [i for i in range(n+1) if Test[i]]

```

```
    return L


%timeit nbpremiers(100)
%timeit eratosthene(100)

%timeit nbpremiers(1000000)
%timeit eratosthene(1000000)
# Malgre nos precautions algorithmiques dans la recherche individuelle
# Le crible reste plus efficace


# Liste des nombres jumeaux inferieurs a n
def nbjumeaux(n):
    LstP = nbpremiers(n) # La liste est dans le bon ordre (ce qui n'est pas garanti avec le "set" de eratosthene(n))
    l = len(LstP)
    L = []
    for i in range(l-1):
        # Rem : si LstP[i] est impair, ce sera aussi le cas de LstP[i]+2
        if LstP[i]%2==1 and LstP[i+1]-LstP[i]==2:
            L.append( (LstP[i], LstP[i+1]) )
    return L
```

SOLUTION DE L'EXERCICE 9 DU TP2

```
# Decide si Part est une partition de n
def estPartition(n, Part):
    # Programme interrompu si la somme n'est pas n
    if sum(Part) != n:
        return False

    # Programme interrompu des le premier ordonnancement plante
    for i in range(len(Part)-1):
        if Part[i+1] == 0 or Part[i+1] > Part[i]:
            return False

    # Si le programme arrive ici, on sait que la somme vaut n et que l'ordonnancement est correct
    return True


# Renvoie la liste des partitions p'>Part de n+1
def part_n1(n, Part):
    if not estPartition(n, Part):
        raise ValueError(f"{Part} n'est pas une partition de {n} !")

    # Test de tous les cas ou on ajoute 1 a l'un des coefficients de Part
    L = []
    k = len(Part)
    for i in range(k):
        NPart = Part.copy() # Attention !!
        NPart[i] += 1
        if estPartition(n+1, NPart):
            L.append(NPart)

    # Dernier cas a ne pas oublier
    NPart = Part.copy()
    NPart.append(1)
    L.append(NPart)

    return L

part_n1(20, [5, 6, 5, 4])
part_n1(20, [10, 5, 2, 2, 1])
part_n1(20, [5, 5, 5, 5])
part_n1(300, [100, 95, 32, 30, 14, 12, 8, 6, 2, 1])
```