1. Suites arithmétiques – Suites géométriques

Activité 1

Activité 1 suites_1.py

```
##############################
# Suites
##############################

##############################
# Activité 1 - Suites arithmétiques
##############################

##############################
## Question 1 ##

def arithmetique_1(n,u0,r):
    """ Renvoie le terme de rang n d'une suite arithmétique
    de terme initial u0 et de raison r
    par formule de récurrence """

    u = u0
    for i in range(n):
        u = u + r
    return u

##############################
## Question 2 ##

def arithmetique_2(n,u0,r):
    """ Renvoie le terme de rang n d'une suite arithmétique
    de terme initial u0 et de raison r
    par formule directe """

    return u0 + n*r

##############################
## Question 3 ##

def liste_arithmetique(n,u0,r):
    """ Renvoie la liste des termes de rang 0 à n d'une suite arithmétique
    de terme initial u0 et de raison r
    par formule de récurrence """

    liste = [u0]
    u = u0
    for i in range(n):
        u = u + r
        liste = liste + [u]
    return liste

##############################
## Question 4 ##

def est_arithmetique(liste):
    """ Teste si la liste correspond aux premiers termes d'une suite arithmétique """

    n = len(liste)-1

    u0 = liste[0]
    u1 = liste[1]
```

```
    r = u1 - u0

    liste_arith = liste_arithmetique(n,u0,r)

    if liste == liste_arith:
        return True
    else:
        return False

# Tests
print("--- Suites arithmétiques ---")
u0 = 2
r = 3
print(arithmetique_1(1,u0,r))
print(arithmetique_2(1,u0,r))
print(liste_arithmetique(1,u0,r))

liste = [2,5,8,11,12]
print(est_arithmetique(liste))

##############################
## Question 5 ##

def somme_arithmetique_1(n,u0,r):
    """ Renvoie la somme des termes de rang 0 à n d'une suite arithmétique
    de terme initial u0 et de raison r
    par formule de récurrence """

    u = u0
    s = u0
    for i in range(n):
        u = u + r
        s = s + u

    return s

def somme_arithmetique_2(n,u0,r):
    """ Renvoie la somme des termes de rang 0 à n d'une suite arithmétique
    de terme initial u0 et de raison r
    par formule directe """

    s = (n+1)*u0+(n*(n+1)//2)*r

    return s

# Tests
print("--- Sommes suites arithmétiques ---")
u0 = 2
r = 3
print(somme_arithmetique_1(10,u0,r))
print(somme_arithmetique_2(10,u0,r))
```

Activité 2

Activité 2 suites_2.py

```
##############################
# Suites
##############################
```

```
##############################
# Activité 2 - Suites arithmétiques
##############################

# But : rechercher dans une liste s'il existe trois termes
# qui forment un partie d'une suite arithmétiques.
# Il s'agit donc de trouver trois termes u[i],u[j],u[k] tels que
# u[i] = u[j] - r, u[k] = u[j] + r (pour un certain r>0).
# On suppose la liste ordonnée.

# Ref : "Finding longest arithmetic progressions" by Jeff Erickson

def chercher_arithmetique(u):
    """ Renvoie trois termes de la liste qui forme une
    progression arithmétique (ou None) """

    n = len(u)-1
    for j in range(1,n-1):
        i = j-1
        k = i + 1
        while (i>=0) and (k<n):
            if u[j]-u[i] == u[k]-u[j]:
                return u[i],u[j],u[k]
            if u[j]-u[i] < u[k]-u[j]:
                i = i - 1
            if u[j]-u[i] > u[k]-u[j]:
                k = k + 1

    return None

# Test
print("--- Recherche d'une progression arithmétique ---")

u = [10,11,13,17,19,23,29,31]
print(chercher_arithmetique(u))
```

Activité 3

Activité 3 suites_3.py

```
##############################
# Suites
##############################


##############################
# Activité 3 - Suites géométrique
##############################

##############################
## Question 1 ##

def geometrique_1(n,u0,q):
    """ Renvoie le terme de rang n d'une suite géométrique
    de terme initial u0 et de raison q
    par formule de récurrence """

    u = u0
    for i in range(n):
        u = u * q
```

```
    return u

##############################
## Question 2 ##
def geometrique_2(n,u0,q):
    """ Renvoie le terme de rang n d'une suite géométrique
    de terme initial u0 et de raison q
    par formule directe """
    return u0 * (q ** n)

##############################
## Question 3 ##
def liste_geometrique(n,u0,q):
    """ Renvoie la liste des termes de rang 0 à n d'une suite géométrique
    de terme initial u0 et de raison q
    par formule de récurrence """
    liste = [u0]
    u = u0
    for i in range(n):
        u = u * q
        liste = liste + [u]
    return liste

##############################
## Question 4 ##
def est_geometrique(liste):
    """ Teste si la liste correspond aux premiers
    termes d'une suite géométrique """
    n = len(liste)-1
    u0 = liste[0]
    u1 = liste[1]
    q = u1 / u0
    liste_arith = liste_geometrique(n,u0,q)
    if liste == liste_arith:
        return True
    else:
        return False

# Tests
print("--- Suites géométriques ---")
u0 = 2
q = 3
print(geometrique_1(2,u0,q))
print(geometrique_2(2,u0,q))
print(liste_geometrique(5,u0,q))
liste = [2,6,18,54,162]
print(liste)
print(est_geometrique(liste))

##############################
## Question 5 ##
def somme_geometrique_1(n,u0,q):
    """ Renvoie la somme des termes de rang 0 à n d'une suite géométrique
```

```
    de terme initial u0 et de raison q
    par formule de récurrence """

    u = u0
    s = u0
    for i in range(n):
        u = u * q
        s = s + u

    return s

def somme_geometrique_2(n,u0,q):
    """ Renvoie la somme des termes de rang 0 à n d'une suite géométrique
    de terme initial u0 et de raison q
    par formule directe """

    s = u0 * (1 - q**(n+1))/(1-q)

    return s

# Tests
print("--- Sommes suites géométriques ---")
u0 = 1
q = 1/2
print(somme_geometrique_1(10,u0,q))
print(somme_geometrique_2(10,u0,q))
```

Activité 4

Activité 4 suites_4.py

```
################################
# Suites
################################

################################
# Activité 4 - Tracer la somme d'une suite géométrique
################################

from turtle import *

################################
## Question 1 ##

def affiche_un_carre(longueur):
    """ Affiche un carré """

    for i in range(4):
        forward(longueur)
        left(90)
    return

################################
## Question 2 ##

def affiche_un_rectangle(longueur):
    """ Affiche un rectangle
    correspondant à un demi-carré """

    for i in range(2):
        forward(longueur)
```

```
        left(90)
        forward(longueur/2)
        left(90)
    return

##############################
## Question 3 ##

def affiche_les_carres(n):
    """  Itère la construction en alternant carré/rectangle """

    cote  = 256
    up()
    goto(-cote//2,-cote//2)
    down()
    width(2)
    color('blue')

    k = 0
    while 2*k <= n:
        affiche_un_carre(cote / 2**k)
        if 2*k < n:
            affiche_un_rectangle(cote / 2**k)
        k = k + 1

    exitonclick()
    return

# Lancement !
affiche_les_carres(8)
```

Activité 5

Activité 5 suites_5.py

```
##############################
# Suites
##############################


##############################
# Rappels - Activité 1
##############################

def liste_arithmetique(n,u0,r):
    """ Renvoie la liste des termes de rang 0 à n d'une suite arithmétique
    de terme initial u0 et de raison r
    par formule de récurrence """

    liste = [u0]
    u = u0
    for i in range(n):
        u = u + r
        liste = liste + [u]
    return liste

##############################
# Activité 5 - Meilleure suite arithmétique
##############################
```

```
# On a une liste
# On cherche les termes d'une suite arithmétique
# qui approximent le mieux cette liste

# v la liste à approcher
# u la meileure suite de premier terme u0 et de raison r
# c-à-d somme(|v_i-u_i|) minimale

##############################
## Question 1 ##

def distance(u,v):
    """ Calcule la distance entre deux listes de même longueur """

    n = len(u)-1
    somme = 0
    for i in range(n+1):
        somme = somme + abs(v[i]-u[i])

    return somme


##############################
## Question 2 ##

# Tiré de de Python (Livre 1) "Analyse de données - Statistique"

def calcule_mediane(liste):
    """ Calcule la médiane des éléments
    Entrée : une liste de nombre
    Sortie : leur médiane """

    liste_triee = sorted(liste)

    n = len(liste_triee)

    if n%2 == 0:   # n est pair
        indice_milieu = n//2
        mediane = (liste_triee[indice_milieu-1]+liste_triee[indice_milieu]) / 2
    else:
        indice_milieu = (n-1)//2
        mediane = liste_triee[indice_milieu]

    return mediane

# Test
print("--- Calcul de la médiane ---")
liste = [3,6,9,11]
print(liste)

m = calcule_mediane(liste)
print("Médiane :",m)

liste_m = [m]*4
d = distance(liste,liste_m)
print("Distance à la médiane :",d)


##############################
## Question 3 ##

def balayage(v,N):
    """ Recherche d'une progression arithmétique u
    qui approche au mieux la liste v.
    Le paramètre N correspond à la précision du balayge.
    La fonction renvoie un terme intitial u0
    et une raison r. """

    n = len(v)-1
```

```
    pas = 2*(v[1]-v[0])/N
    dmin = 10000  # l'infini
    r = 0
    for k in range(N+1):
        w = [v[i]-i*r for i in range(n+1)]
        u0 = calcule_mediane(w)
        u = liste_arithmetique(n,u0,r)
        d = distance(u,v)
        if d < dmin:
            dmin = d
            rmin = r
            u0min = u0
        r = r + pas
    print(u0min,rmin,dmin)
    return u0min,rmin
# Test
print("--- Balayage r ---")
v = [3,6,9,11]
n = len(v)-1
u0,r = balayage(v,10)
print("---")
print("Suite à approcher :",v)
print("r =",r)
print("u0 =",u0)
u = liste_arithmetique(n,u0,r)
print("Suite arithmétique trouvée",u)
print("Erreur =",distance(u,v))
# Test
print("--- Balayage r ---")
v = [6,11,14,20,24,29,37]
n = len(v)-1
u0,r = balayage(v,10000)
print("---")
print("Suite à approcher :",v)
print("r =",r)
print("u0 =",u0)
u = liste_arithmetique(n,u0,r)
print("Suite arithmétique trouvée",u)
print("Erreur =",distance(u,v))
```

2. Nombres complexes I

Activité 1

Activité 1 complexes_I_1.py

```
##############################
# Complexes I
##############################

##############################
# Activité 1 - Complexes avec Python, Ecriture a + ib
##############################

##############################
## Question 1 ##

z1 = 1+2j
z2 = 3-1j

print(z1+z2)

print(z1*z2)

print(z1 ** 2)

print(abs(z1))

print(1/z1)

##############################
## Question 2 ##

z = (3-4j)**2 * (2-1j)
print(z.real)
print(z.imag)
print(z.conjugate())

##############################
## Question 3 ##

# addition
def addition(a,b,aa,bb):
    return (a+aa,b+bb)

# multiplication
def multiplication(a,b,aa,bb):
    return (a*aa-b*bb,a*bb+b*aa)

# conjugué
def conjugue(a,b):
    return (a,-b)

# module
def module(a,b):
    return sqrt(a**2 + b**2)

# inverse
def inverse(a,b):
    if a != 0 and b !=0:
        r2 = a**2 + b**2
        return (a/r2,-b/r2)
    else:
        return None
```

```
# puissance
def puissance(a,b,n):
    if n == 0:
        return (1,0)

    aa = a
    bb = b
    for __ in range(n):
        aa,bb = multiplication(a,b,aa,bb)

    return (aa,bb)
```

Activité 2

Activité 2 complexes_I_2.py

```
##############################
# Complexes I
##############################

##############################
# Rappels activité 1
##############################

def addition(a,b,aa,bb):
    return (a+aa,b+bb)

def multiplication(a,b,aa,bb):
    return (a*aa-b*bb,a*bb+b*aa)

def conjugue(a,b):
    return (a,-b)

def module(a,b):
    return sqrt(a**2 + b**2)

def inverse(a,b):
    if a != 0 and b !=0:
        r2 = a**2 + b**2
        return (a/r2,-b/r2)
    else:
        return None

def puissance(a,b,n):
    if n == 0:
        return (1,0)
    aa = a
    bb = b
    for __ in range(n):
        aa,bb = multiplication(a,b,aa,bb)
    return (aa,bb)

##############################
##############################
# Cours : visualisation

import matplotlib.pyplot as plt

plt.clf()  # Efface tout
```

```
plt.axes().set_aspect('equal')  # Repère orthonormé
plt.axhline(y=0, color='r', linestyle='-')  # Axe x
plt.axvline(x=0, color='r', linestyle='-')  # Axe y

x = 2
y = 1

plt.scatter(x,y,color='blue',s=80)  # Un point
# plt.show()  # Lancement de la fenêtre

##############################
# Activité 2 - Visualisation
##############################

from math import *

import matplotlib.pyplot as plt

##############################
## Question 1 ##

# fig,ax = plt.subplots()
z = -2+3j
x = z.real
y = z.imag

plt.clf() # Efface tout
plt.axes().set_aspect('equal') # Repère orthonormé
plt.axhline(y=0, color='r', linestyle='-')  # Axe x
plt.axvline(x=0, color='r', linestyle='-')  # Axe y

plt.scatter(x,y,color='red',s=20)
plt.scatter(1,0,color='green',s=20)
plt.scatter(0,1,color='blue',s=20)

# plt.show()

##############################
## Question 2 ##

z0 = 3-2j
x0 = z0.real
y0 = z0.imag

x1,y1 = multiplication(x0,y0,2,0)
x2,y2 = multiplication(x0,y0,0,-1)

xx3,yy3 = puissance(x0,y0,2)
r = module(x0,y0)
x3,y3 = xx3/r, yy3/r

x4,y4 = conjugue(x0,y0)
x5,y5 = inverse(x0,y0)

plt.clf()
plt.axes().set_aspect('equal')
plt.axhline(y=0, color='r', linestyle='-')
plt.axvline(x=0, color='r', linestyle='-')

plt.scatter(x0,y0,color='red',s=20)
plt.scatter(x1,y1,color='green',s=20)
plt.scatter(x2,y2,color='blue',s=20)
plt.scatter(x3,y3,color='brown',s=20)
plt.scatter(x4,y4,color='orange',s=20)
plt.scatter(x5,y5,color='purple',s=20)

# plt.show()
```

```
##############################
## Question 3 ##

def affiche_triangle(z1,z2,z3):
    """ Trace un triangle à partir des affixes donnés """
    x1,y1 = z1.real,z1.imag
    x2,y2 = z2.real,z2.imag
    x3,y3 = z3.real,z3.imag

    plt.scatter(x1,y1,color='red',s=50)
    plt.scatter(x2,y2,color='green',s=50)
    plt.scatter(x3,y3,color='blue',s=50)

    plt.plot([x1,x2,x3,x1],[y1,y2,y3,y1],color='black')

    return

plt.clf()
plt.axes().set_aspect('equal')
plt.axhline(y=0, color='r', linestyle='-')
plt.axvline(x=0, color='r', linestyle='-')

# Test
# affiche_triangle(2+1j,1-1j,-2+1.5j)

z = 1+2j
# affiche_triangle(z,2*z,(1+2j)*z)  # Triangle rectangle

z = 1+2j
omega = -1/2+sqrt(3)/2*1j
affiche_triangle(z,omega*z,omega*omega*z)  # Triangle isocèle

plt.show()
```

Activité 3

Activité 3 — complexes_I_3.py

```
##############################
# Complexes I
##############################

##############################
# Activité 3 - Hack - Equation de degré 1
##############################

def solution_equation_lineaire(equation):
    """
    Résoud une équation linéaire réelle de degré 1
    Entrée : une chaîne de caractère sous la forme "3*(x+1) + x = 2*x+1"
    Sortie : la valeur de la solution x (par exemple ici renvoie -1)
    Remarque : utilise astucieusement les nombres complexes !
    """

    # Parties gauche et droite de l'équation
    # Ex "3*(x+1) + x = 2*x+1" -> "3*(x+1) + x" et "2*x+1"
    eq_gd = equation.split("=")

    # On bascule tout à gauche
    # Ex : on obtient "3*(x+1) + x - ( 2*x+1 )" (sous-entendu = 0)
    new_eq = eq_gd[0] + "- (" + eq_gd[1] + " )"  #
```

```
    # On remplace les x par le nb complexe 1j
    # Ex :on obtient "3*(1j+1) + 1j - ( 2*1j+1 )"
    z_str = new_eq.replace("x","1j")

    # On évalue la chaîne
    # Ex : la chaîne devient le nb complexe z = 3*(1j+1) + 1j - ( 2*1j+1 )
    z = eval(z_str)

    # On récupère les parties réelle et imaginaire
    # Ex : a = 2, b = 2
    a = z.real
    b = z.imag

    # Solution de l'équation qui correspond en fait à "a + bx = 0"
    # Ex : sol = -1
    sol = -a/b

    return sol

# Test
print("--- Solution d'une équation linéaire réelle ---")
eq = "7*x+3 = 0"
# eq = "3*(x+1) + x = 2*x+1"
x = solution_equation_lineaire(eq)
print("Equation :",eq)
print("Solution :", x)
```

Activité 4

Activité 4 — complexes_I_4.py

```
##############################
# Complexes I
##############################

##############################
# Activité 4 - Equation du second degré
##############################

from math import *

##############################
## Question 1 ##

def solution_trinome(a,b,c):
    """ Solutions de ax^2 + bx + c = 0 avec a,b,c réels """

    Delta = b**2 - 4*a*c

    if Delta == 0:
        sol = [-b/(2*a),-b/(2*a)]

    if Delta > 0:
        d = sqrt(Delta)
        sol = [(-b-d)/(2*a), (-b+d)/(2*a)]

    if Delta < 0:
        d = 1j*sqrt(-Delta)
        sol = [(-b-d)/(2*a), (-b+d)/(2*a)]

    return sol
```

```
# Test
print("--- Solution d'une équation de degré 2 ---")
sol = solution_trinome(1,-2,1)  # x^2 - 2x + 1, Delta = 0
print("Solution :", sol)
sol = solution_trinome(1,1,-1)  # x^2 + x - 1, Delta > 0
print("Solution :", sol)
sol = solution_trinome(1,1,1)  # x^2 + x + 1, Delta < 0
print("Solution :", sol)

##############################
## Question 2 ##

def solution_somme_produit(S,P):
    return solution_trinome(1,-S,P)

# Test
print("--- Solution somme/produit ---")
x,y = solution_somme_produit(10,20)  # S = 10, P = 20
print("Solution :", x,y)
print("Vérification :", x+y, x*y)

##############################
## Question 3 ##

def solution_bicarre(a,b,c):
    """ Solutions de ax^4 + bx^2 + c = 0 avec a,b,c réels et Delta >= 0 """

    Delta = b**2 - 4*a*c

    if Delta < 0:
        print("Discriminant négatif. Je ne sais pas faire.")
        return None

    # On récupère les solution de aX^2 + bX + c
    X1, X2 = solution_trinome(a,b,c)

    sol = []
    for X in [X1,X2]:

        if X >= 0:
            # Si Xi >= 0 on prend les racines carrées
            x1 = +sqrt(X)
            x2 = -sqrt(X)
            sol = sol + [x1,x2]

        if X < 0:
            # Si Xi < 0 on prend les racines carrées  +/- i*racine(-Xi)
            x1 = +1j*sqrt(-X)
            x2 = -1j*sqrt(-X)
            sol = sol + [x1,x2]

    return sol

# Test
print("--- Solution bicarré ---")
sol = solution_bicarre(1,-2,-3)
print(sol)
```

Activité 5

Activité 5 complexes_I_5.py

```
################################
# Complexes I
################################

################################
# Rappels
################################

from math import *

def solution_trinome(a,b,c):
    """ Solutions de ax^2 + bx + c = 0 avec a,b,c réels """

    Delta = b**2 - 4*a*c

    if Delta == 0:
        sol = [-b/(2*a),-b/(2*a)]

    if Delta > 0:
        d = sqrt(Delta)
        sol = [(-b-d)/(2*a), (-b+d)/(2*a)]

    if Delta < 0:
        d = 1j*sqrt(-Delta)
        sol = [(-b-d)/(2*a), (-b+d)/(2*a)]

    return sol

################################
# Activité 5 - Famille de racines
################################

import matplotlib.pyplot as plt

################################
## Question 1 ##

def affiche_racines(a,b,c,couleur='red'):
    """ Trace les solutions d'une équation du second degré """
    z1,z2 = solution_trinome(a,b,c)
    x,y = z1.real, z1.imag
    plt.scatter(x,y,color=couleur,s=40)
    x,y = z2.real, z2.imag
    plt.scatter(x,y,color=couleur,s=40)
    return

# Test
print("--- Test affiche racines ---")
plt.clf()
plt.axes().set_aspect('equal')
plt.axhline(y=0, color='r', linestyle='-')
plt.axvline(x=0, color='r', linestyle='-')

sol = solution_trinome(1,-2,1)
affiche_racines(1,-2,1,couleur='red') # x^2 - 2x + 1, Delta = 0
affiche_racines(1,1,-1,couleur='blue') # x^2 + x - 1, Delta > 0
affiche_racines(1,1,1,couleur='green')   # x^2 + x + 1, Delta < 0
# plt.show()

################################
## Question 2 ##
```

```
def affiche_famille(b0,c0,b1,c1,n=100):
    """ Trace les solutions d'une famille
    d'équations u second degré """

    for k in range(n):
        t = k/n
        b = (1-t)*b0 + t*b1
        c = (1-t)*c0 + t*c1
        affiche_racines(1,b,c,couleur='blue')

    affiche_racines(1,b0,c0,couleur='red')
    affiche_racines(1,b1,c1,couleur='green')

    return

# Test
print("--- Test famille de racines ---")
plt.clf()
plt.axes().set_aspect('equal')
plt.axhline(y=0, color='r', linestyle='-')
plt.axvline(x=0, color='r', linestyle='-')

affiche_famille(-2,2,3,12/5,n=4)
plt.show()
```

3. Nombres complexes II

Activité 1

Activité 1 complexes_II_1.py

```
################################
# Complexes II
################################

################################
# Activité 1 - Module/argument
################################

################################
## Question 1 ##

import cmath  # ne pas faire "from cmath import *" car conflit sqrt
from math import *

z = 1+3j

module = abs(z)
argument = cmath.phase(z)

print("Module :", module)
print("Argument :", argument)

# z = complex(1+1j)
z = 1+1j
print("Module, Argument :", cmath.polar(z))

################################
## Question 2 ##

z = cmath.rect(2,pi/3)
```

```
print("z =",z)

z = cmath.rect(3,5*pi/6)
print("z = ",z)

##############################
## Question 3 ##

import matplotlib.pyplot as plt

plt.clf()  # Efface tout
plt.axes().set_aspect('equal')  # Repère orthonormé
plt.axhline(y=0, color='r', linestyle='-')  # Axe x
plt.axvline(x=0, color='r', linestyle='-')  # Axe y

z = cmath.rect(sqrt(2),pi/6)
x = z.real
y = z.imag

plt.scatter(x,y,color='red',s=100)

plt.show()

##############################
## Question 4 ##

def dessine_polygone(n):
    """ Trace un polygone régulier à n côtés
    en utilisant les nombres complexes """

    omega = cmath.rect(1,2*pi/n)
    listex = []
    listey = []
    for k in range(n):
        z = omega ** k
        x = z.real
        y = z.imag
        plt.scatter(x,y,color='blue',s=100)
        listex.append(x)
        listey.append(y)
    listex.append(listex[0])
    listey.append(listey[0])
    plt.plot(listex,listey,color='black')
    return

plt.clf()  # Efface tout
plt.axes().set_aspect('equal')  # Repère orthonormé
plt.axhline(y=0, color='r', linestyle='-')  # Axe x
plt.axvline(x=0, color='r', linestyle='-')  # Axe y

dessine_polygone(5)

plt.show()
```

Activité 2

Activité 2 complexes_II_2.py

```
################################
# Complexes II
################################

################################
# Activité 2 - Passage polaire/cartésien
################################

from math import *
import cmath

################################
## Question 1 ##

def polaire_vers_cartesien(module,argument):
    """ Passage de (r,theta) -> z = a+ib """
    x = module*cos(argument)
    y = module*sin(argument)
    z = x + 1j*y
    return z

# Test
print("--- Polaire vers cartésien ---")
z1 = polaire_vers_cartesien(3,pi/6)
z2 = cmath.rect(3,pi/6)
print(z1)
print(z2)

################################
## Question 2 ##

def cartesien_vers_polaire(z):
    """ Passage de z = a+ib -> (r, theta) """
    x = z.real
    y = z.imag
    print(x)
    print(y)
    module = sqrt(x**2 + y**2)
    argument = atan2(y,x)
    return (module,argument)

# Test
print("--- Cartésien vers polaire ---")
mod_arg_1 = cartesien_vers_polaire(-2+5j)
mod_arg_2 = cmath.polar(-2+5j)
print(mod_arg_1)
print(mod_arg_2)

################################
## Question 3 ##

def argument_dans_intervalle(angle):
    """ Ramène un angle dans l'intervalle ]-pi,+pi]
    par réduction modulo 2pi """
    k = floor(angle/(2*pi))
    print(k)
    new_angle = angle - 2*k*pi
```

```
    if new_angle > pi:
        new_angle += -2*pi
    return new_angle

# Test
print("--- Argument dans intervalle ---")

theta =  -pi/2 + 12*pi
print(theta)
print(argument_dans_intervalle(theta))
print(theta % 2*pi)
```

Activité 3

Activité 3 complexes_II_3.py

```
################################
# Complexes II
################################

from math import *

################################
# Rappels

def polaire_vers_cartesien(module,argument):
    x = module*cos(argument)
    y = module*sin(argument)
    z = x + 1j*y
    return z

def cartesien_vers_polaire(z):
    x = z.real
    y = z.imag
    print(x)
    print(y)
    module = sqrt(x**2 + y**2)
    argument = atan2(y,x)
    return (module,argument)

################################
# Activité 3 - Formule d'Euler / de Moivre / Gauss
################################

################################
## Question 1 ##

def cosinus(t):
    """ Calcule le cosinus d'un angle par la formule d'Euler """
    eplus = polaire_vers_cartesien(1,t)
    emoins = polaire_vers_cartesien(1,-t)
    cos_complexe = (eplus+emoins)/2
    cos_reel = cos_complexe.real
    return cos_reel

def sinus(t):
    """ Calcule le sinus d'un angle par la formule d'Euler """
    eplus = polaire_vers_cartesien(1,t)
    emoins = polaire_vers_cartesien(1,-t)
```

```
    sin_complexe = (eplus-emoins)/(2*1j)
    sin_reel = sin_complexe.real
    return sin_reel

# Test
print("--- Formules d'Euler ---")

t = pi/6
print(cosinus(t))
print(cos(t))
print(sinus(t))
print(sin(t))

##############################
## Question 2 ##

def puissance_bis(z,n):
    """ Calcule z à la puissance n par la formule de Moivre """
    r,theta = cartesien_vers_polaire(z)
    r_n = r ** n
    theta_n = n*theta
    z_n = polaire_vers_cartesien(r_n,theta_n)
    return z_n

# Test
print("--- Formule de De Moivre ---")
z = 2-3j
n = 10
print(puissance_bis(z,n))
print(z**n)

##############################
## Question 3 ##

def multiplication(a,b,c,d):
    """ Multiplication classique """
    return (a*c-b*d,a*d+b*c)

def multiplication_bis(a,b,c,d):
    """ Multiplication Gauss 1 """
    r = a*c
    s = b*d
    t = (a+b)*(c+d)
    return r-s, t-r-s

def multiplication_ter(a,b,c,d):
    """ Multiplication Gauss 2 """
    r = c*(a+b)
    s = a*(d-c)
    t = b*(c+d)
    return r-t, r+s

# Test
print("--- Formules de Gauss ---")
print(multiplication(2,5,3,-2))
print(multiplication_bis(2,5,3,-2))
print(multiplication_ter(2,5,3,-2))
```

Activité 4

Activité 4 complexes_II_4.py

```
##############################
# Complexes II
##############################

from math import *

##############################
# Rappels

def polaire_vers_cartesien(module,argument):
    x = module*cos(argument)
    y = module*sin(argument)
    z = x + 1j*y
    return z

def cartesien_vers_polaire(z):
    x = z.real
    y = z.imag
    print(x)
    print(y)
    module = sqrt(x**2 + y**2)
    argument = atan2(y,x)
    return (module,argument)

##############################
# Activité 4 - Cercles et droites
##############################

import matplotlib.pyplot as plt

##############################
## Question 1 ##

def affiche_liste(zliste,couleur='blue',taille=10):
    """ Trace des points à partir d'une liste d'affixes """
    for z in zliste:
        x = z.real
        y = z.imag
        plt.scatter(x,y,color=couleur,s=taille)
    return

##############################
## Question 2 ##

def trace_cercle(z0,r,numpoints=100):
    """ Trace (les points d') un cercle à partir d'un centre et un rayon """
    zliste = []
    for k in range(numpoints):
        theta = 2*pi*k/numpoints
        z = z0 + polaire_vers_cartesien(r,theta)
        zliste.append(z)
    return zliste

##############################
## Question 3 ##

def trace_segment(z0,z1,numpoints=100):
```

```
    """ Trace (les points d') un segemnt entre deux points donnés """
    zliste = []
    for k in range(numpoints):
        t = k/numpoints
        z = (1-t)*z0+t*z1
        zliste.append(z)
    return zliste

# Test
plt.clf()
plt.axes().set_aspect('equal')
plt.axhline(y=0, color='r', linestyle='-')  # Axe x
plt.axvline(x=0, color='r', linestyle='-')  # Axe y

zliste = trace_cercle(2+3j,sqrt(2))
affiche_liste(zliste)

zliste = trace_segment(-2-1j,-1+3j)
affiche_liste(zliste)

plt.show()
```

Activité 5

Activité 5 — complexes_II_5.py

```
################################
# Complexes II
################################

from math import *

################################
# Rappels

def polaire_vers_cartesien(module,argument):
    x = module*cos(argument)
    y = module*sin(argument)
    z = x + 1j*y
    return z

def cartesien_vers_polaire(z):
    x = z.real
    y = z.imag
    print(x)
    print(y)
    module = sqrt(x**2 + y**2)
    argument = atan2(y,x)
    return (module,argument)

################################
# Rappels - Activité 4
################################

import matplotlib.pyplot as plt

################################
## Question 1 ##

def affiche_liste(zliste,couleur='blue',taille=10):
```

```
    for z in zliste:
        x = z.real
        y = z.imag
        plt.scatter(x,y,color=couleur,s=taille)
    return

##############################
## Question 2 ##

def trace_cercle(z0,r,numpoints=100):
    zliste = []
    for k in range(numpoints):
        theta = 2*pi*k/numpoints
        z = z0 + polaire_vers_cartesien(r,theta)
        zliste.append(z)
    return zliste

##############################
## Question 3 ##

def trace_segment(z0,z1,numpoints=100):
    zliste = []
    for k in range(numpoints):
        t = k/numpoints
        z = (1-t)*z0+t*z1
        zliste.append(z)
    return zliste


##############################
# Activité 5 - Transformations complexes
##############################

##############################
## Question 1 ##

def translation(zliste,v):
    """ Ajoute v à chaque complexe z de la liste """
    new_zliste = []
    for z in zliste:
        new_zliste.append(z+v)
    return new_zliste

def homothetie(zliste,k):
    """ Multiplie chaque complexe z de la liste par k """
    new_zliste = []
    for z in zliste:
        new_zliste.append(k*z)
    return new_zliste

def rotation(zliste,theta):
    """ Multiplie chaque complexe z de la liste par exp(i theta) """
    new_zliste = []
    for z in zliste:
        w = polaire_vers_cartesien(1,theta)
        new_zliste.append(z*w)
    return new_zliste

def symetrie(zliste):
    """ Chaque z est conjugué """
    new_zliste = []
    for z in zliste:
```

```
        new_zliste.append(z.conjugate())
    return new_zliste


##############################
## Question 2 ##

cercle = trace_cercle(2+2j,1)
carre = trace_segment(0,1) + trace_segment(1,1+1j) + trace_segment(1+1j,1j) + trace_segment
    ↪ (1j,0)
ensemble = cercle + carre
ensemble_transforme = translation(ensemble,2-1j)
ensemble_transforme = homothetie(ensemble,1.5)
ensemble_transforme = symetrie(ensemble)
ensemble_transforme = rotation(ensemble,pi/3)

plt.clf()
plt.axes().set_aspect('equal')
plt.axhline(y=0, color='r', linestyle='-')
plt.axvline(x=0, color='r', linestyle='-')

affiche_liste(ensemble,couleur="blue")
affiche_liste(ensemble_transforme,couleur="red")

# plt.show()

##############################
## Question 3 ##

def inversion(zliste):
    """ Chaque z devient 1/z """
    new_zliste = []
    for z in zliste:
        if z != 0:
            new_zliste.append(1/z.conjugate())
    return new_zliste

# Cercles et droites
cercle = trace_cercle(-2+1.5j,1)
carre = trace_segment(1+0.5j,2+0.5j) + trace_segment(2+0.5j,2+1.5j) + trace_segment(2+1.5j
    ↪ ,1+1.5j) + trace_segment(1+1.5j,1+0.5j)
ensemble = cercle + carre
ensemble_transforme = inversion(ensemble)

plt.clf()
plt.axes().set_aspect('equal')
plt.axhline(y=0, color='r', linestyle='-')
plt.axvline(x=0, color='r', linestyle='-')

affiche_liste(ensemble,couleur="blue")
affiche_liste(ensemble_transforme,couleur="red")

# plt.show()

# Grille
# n = 5
# lignes_verticales = []
# lignes_horizontales = []
# for k in range(1,n):
#     lignes_verticales += trace_segment(k/2-n*1j,k/2+n*1j,numpoints=50)
#     lignes_horizontales += trace_segment(-n+k/2*1j,n+k/2*1j,numpoints=50)

# ensemble = lignes_verticales + lignes_horizontales
# ensemble_transforme = inversion(ensemble)
```

```
# plt.clf()
# plt.axes().set_aspect('equal')
# plt.axhline(y=0, color='r', linestyle='-')
# plt.axvline(x=0, color='r', linestyle='-')

# affiche_liste(ensemble,couleur="blue",taille=5)
# affiche_liste(ensemble_transforme,couleur="red",taille=1)

# plt.show()

##############################
## Question 4 ##

def au_carre(zliste):
    """ Chaque z devient z^2 """
    new_zliste = []
    for z in zliste:
        new_zliste.append(z**2)
    return new_zliste

# Cercles et droites
cercle = trace_cercle(2+1j,1.5)
carre = trace_segment(1+0.5j,2+0.5j) + trace_segment(2+0.5j,2+1.5j) + trace_segment(2+1.5j
    ↪ ,1+1.5j) + trace_segment(1+1.5j,1+0.5j)
ensemble = cercle + carre
ensemble_transforme = au_carre(ensemble)

plt.clf()
plt.axes().set_aspect('equal')
plt.axhline(y=0, color='r', linestyle='-')
plt.axvline(x=0, color='r', linestyle='-')

affiche_liste(ensemble,couleur="blue")
affiche_liste(ensemble_transforme,couleur="red")

plt.show()
```

4. Dérivée – Zéros de fonctions

Activité 1 à 4

Activité 1 à 4 derivee.py

```
##############################
# Zéros de fonction - Dérivées
##############################

from math import *

##############################
# Cours
##############################

f = lambda x: x**2

print(f(2))

def f(x):
    return x**2
```

```
print(f(2))

def est_plus_grand(f,a,b):
    if f(a) > f(b):
        return True
    else:
        return False

print(est_plus_grand(f,1,2))
print(est_plus_grand(lambda x:1/x,1,2))

################################
# Activité 1 - Calcul de la dérivée en un point
################################

################################
## Question 1  ##

# Fonction définie comme lambda-fonction
f = lambda x: x*(1-sqrt(x))

# Evaluation
# for a in range(6):
#     print("a =",a)
#     print("f(a) =",f(a))

################################
## Question 2  ##

def derivee(f,a,h=0.00000001):
    """ Calcul approché de la dérivée de f en a par le taux d'accroissement """
    taux = (f(a+h)-f(a))/h
    return taux

# Test
f = lambda x: x**3
for a in range(6):
    print("a =",a)
    print("f(a) =",f(a))
    print("f'(a) =",derivee(f,a))
    print("f'(a) =",3*a**2)

# f = lambda x: sqrt(x)
# for a in range(1,6):
#     print("a =",a)
#     print("f(a) =",f(a))
#     print("f'(a) =",derivee(f,a))
#     print("f'(a) =",1/(2*sqrt(a)))

################################
# Activité 2 - Graphe d'une fonction
################################

################################
## Question 1  ##

def graphe(f,a,b,n):
    """ n points du graphe de f sur [a,b] """
    liste_points = []
    h = (b-a)/n
    x = a
    for i in range(n+1):
        y = f(x)
        liste_points.append( (x,y) )
        x = x + h
```

```
    return liste_points

# Test
f = lambda x: x*x
print(graphe(f,0,2,4))

##############################
## Question 2  ##

def afficher_un_point(i,j,couleur="red",taille=5):
    canvas.create_rectangle(i-taille,j-taille,i+taille,j+taille,fill=couleur,width=1)
    return

def afficher_points(points,echelle=50):

    # Axes
    canvas.create_line(50,300,750,300,fill="blue",width=5)
    canvas.create_line(400,550,400,50,fill="blue",width=5)

    for p in points:
        x,y = p
        i = round(x*echelle)
        j = round(y*echelle)
        afficher_un_point(400+i,300-j,couleur="black")


    return

##############################
## Question 3  ##

def relier_points(points,echelle=50):
    # Axes
    canvas.create_line(50,300,750,300,fill="blue",width=5)
    canvas.create_line(400,550,400,50,fill="blue",width=5)

    n = len(points) - 1

    for k in range(n):
        x,y = points[k]
        xx,yy = points[k+1]
        i = round(x*echelle)
        j = round(y*echelle)
        ii = round(xx*echelle)
        jj = round(yy*echelle)
        canvas.create_line(400+i,300-j,400+ii,300-jj,fill="red",width=5)

    return

def tracer_graphe(f,a,b,n=20,echelle=50):
    points = graphe(f,a,b,n)
    relier_points(points,echelle=echelle)

    # Axes
    canvas.create_line(50,300,750,300,fill="blue",width=5)
    canvas.create_line(400,550,400,50,fill="blue",width=5)

    # To do : marques
    return

##############################
## Question 5  ##

def tracer_tangente(f,a,echelle=50):
    """ Tracer de la tangente de f en a
    en utilisant la dérivée """
```

```
    x = a
    y = f(a)
    dx = 1
    dy = derivee(f,a)
    i = round(x*echelle)
    j = round(y*echelle)
    ii = round((x+dx)*echelle)
    jj = round((y+dy)*echelle)
    iii = round((x-dx)*echelle)
    jjj = round((y-dy)*echelle)

    # Point
    canvas.create_rectangle(400+i-4,300-j-4,400+i+4,300-j+4,fill="gray",width=1)

    # Demi-tangentes
    canvas.create_line(400+i,300-j,400+ii,300-jj,fill="black",width=3)
    canvas.create_line(400+i,300-j,400+iii,300-jjj,fill="black",width=3)

    return

# Fenêtre tkinter
from tkinter import *
root = Tk()
canvas = Canvas(root, width=800, height=600, background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

f = lambda x: sqrt(x)
points = graphe(f,0,4,4)
# relier_points(points,echelle=80)
# afficher_points(points,echelle=80)
# tracer_graphe(f,0,4,echelle=80)
# tracer_tangente(f,1,echelle=80)

root.mainloop()

##############################
# Activité 3 - Dichotomie
##############################

def dichotomie(f,a,b,epsilon):
    """ Résolution approchée de f(x)=0 sur [a,b]
    Renvoie un encadrement de longueur plus petit que epsilon """

    # Vérification de l'hypothèse
    assert f(a)*f(b) <= 0

    # Boucle
    while b-a > epsilon:
        c = (a+b)/2
        if f(a)*f(c) <= 0:
            b = c
        else:
            a = c

    return a,b   # intervalle

# Test
f = lambda x: x*x-2
print(dichotomie(f,0,2,1e-5))


##############################
# Activité 4 - Méthode de Newton
##############################
```

```
def newton(f,a,n):
    """ Résolution approchée de l'équation f(x)=0 par la méthode de Newton.
    a est la valeur de départ, n le nombre d'itérations """
    x = a
    for i in range(n):
        x = x - f(x)/derivee(f,x)

    return x

# Test
f = lambda x: x*x-2
print(newton(f,2,5))
```

5. Exponentielle

Activité 1 à 4

Activité 1 à 4 exponentielle_1.py

```
################################
# Exponentielle
################################

################################
# Activité 1 - L'exponentielle
################################

from math import *

################################
## Question 1 ##

# --- Grains de riz ---

################################
## Question 1.a ##
N = 64 # Nombre de cases
n = 1
somme = 0
for k in range(N):
    somme = somme + n
    n = 2*n

print("--- Grains de riz ---")
print("Nombre de grain de riz :",somme)
# print(2**N-1)

################################
## Question 1.b ##

# 50 000 grains de riz pèse un kilogramme
masse = somme / (50000*1000)  # en tonne
print("Masse (en tonnes) :",masse)

################################
## Question 2 ##

# Enoncé : nénuphar,
```

```
# surface mutliplié par 1.5 chaque jour
# jour 10, surface = 100
# surface au jour 15 ?
# surface au jour 0 ?
# équation S(x) = c * (1.5)**x = c * exp(1.5 *ln(x))
# quand surface vaut 10000 ?

###############################
## Question 2.a ##

surface_15 = 100 * 1.5**5

###############################
## Question 2.b ##

surface_0 = 100 / 1.5**10

###############################
## Question 2.c ##

def surface_nenuphar(x):
    """ Surface au bout de x jours """
    S0 = 100 / 1.5**10   # Surface initiale
    S = S0 * exp(x*log(1.5)) # ou alors
    return S

print("--- Nénuphar ---")

print("Surface jour 15 :",surface_15)
print("Surface jour 0  :",surface_0)

###############################
## Question 2.d ##

# Trouver jour par tatonnement ou balayage
x = 21.3578
print("Jour :",x)
print("Surface :",surface_nenuphar(x))

###############################
## Question 2.e  ##

# Trouver jour par équation
def jour_nenuphar(S):
    """ Nb de jours pour atteindre la surface S """
    S0 = 100 / 1.5**10   # Surface initiale
    x = (log(S)-log(S0))/log(1.5)
    return x

print("Jour :",jour_nenuphar(200))
print("Jour :",jour_nenuphar(10000))
print("Jour :",surface_nenuphar(jour_nenuphar(10000)))

###############################
## Question 3 ##

# Loi de refroidissement de Newton
# T(t) - Tinfini = (T(0)-Tinfini)*exp(-k*t)

# Temperature ambiante (au bout d'un temps infini) = 25
# Au départ 100 °C, au bout de 10 minutes 65 degrés
Tinfini = 25
T0 = 100 # t0 = 0
T1 = 65
t1 = 10

###############################
```

```
## Question 3.a ##

# Calculer k
k = -1/t1 * log((T1-Tinfini)/(T0-Tinfini))

##############################
## Question 3.b ##

# Fonction + temperature au bout de 20 minutes
def temperature(t):
    """ Temperature au bout du temps t """
    T = (T0-Tinfini)*exp(-k*t) + Tinfini
    return T

print("--- Loi de refroidissement de Newton ---")
print("k =",k)
print("Température initiale ",temperature(0))
print("Température à 10 mn ",temperature(10))
print("Température à 20 mn ",temperature(20))

##############################
## Question 3.c ##

# Quand température atteint 30 °C (tatonnement, balayage, équation)
print("Temperature de 30 °C",temperature(43))

##############################
## Question 4 ##

# Demi-vie et datation au carbone 14

##############################
## Question 4.a ##

def carbone14(t,N0=1000,T=5730):
    """ Nb d'atomes de carbone 14 au bout d'un temps t
    N0 est le nb d'atomes initial, T est la période de demi-vie """
    return N0*exp(-t*log(2)/T)

# Test
print("--- Carbone 14 ---")
t = 100
print("t =",t)
print(carbone14(t))

##############################
## Question 4.b ##
def carbone14_bis(t,N0=1000,T=5730):
     """ Variante avec exposant au lieu d'exponentielle """
    return N0*2**(-t/T)

# Test
print(carbone14_bis(t))
t = 5730
print(carbone14_bis(t))

##############################
## Question 4.c ##
def datation14(N,N0=1000,T=5730):
    """ Temps écoulé afin qu'il reste N atomes de carbones
     N0 est le nb d'atomes initial, T est la période de demi-vie """
    return -T/log(2)*log(N/N0)

print(datation14(500))
print(datation14(200))
```

Activité 5

Activité 5 exponentielle_2.py

```
##############################
# Exponentielle
##############################

from math import *

# La fonction exponentielle est importée mais
# juste pour vérification des formules

##############################
# Activité 2 - Définition de l'exponentielle
##############################

##############################
## Question 1  ##

def exponentielle_limite(x,n):
    """ Valeur approchée de exp(x) par limite """
    expo = (1+x/n)**n
    return expo

##############################
## Question 2  ##

def factorielle(n):
    """ Calcul de n! par une boucle """
    fact = 1
    for k in range(1,n+1):
        fact = fact* k
    return fact

##############################
## Question 3  ##

def exponentielle_somme(x,n):
    """ Valeur approchée de exp(x) par somme """
    expo = 0
    for k in range(n+1):
        expo = expo + (x**k)/factorielle(k)
    return expo

##############################
## Question 4  ##

# e^x = 1 + (x/1) (1 + (x/2) (1 + (x/3) (........) ) )
def exponentielle_horner(x,n):
    """ Valeur approchée de exp(x) par Hörner pour limiter les mutliplications """
    expo = 1
    for k in reversed(range(1,n+1)):
        expo = x/k * expo + 1
    return expo


##############################
## Question 5  ##

# Exponentielle via les fractions continues
# https://en.wikipedia.org/wiki/Euler%27s_continued_fraction_formula
def exponentielle_euler(x,n):
```

```
    """ Valeur approchée de exp(x) par les fractions continues """
    expo = 0
    for k in reversed(range(1,n+1)):
        expo = x/(k+x-k*expo)
    expo = 1/(1-expo)
    return expo

##############################
## Question 6  ##

# e = exponentielle_somme(1,n=25)

def exponentielle_astuce(x,n):
    """ Valeur approchée de exp(x) en réduisant d'abord la valeur de x """
    e = 2.718281828459045
    k = floor(x)  # partie entière
    f = x-k       # partie fractionnaire
    expo_entier = e**k
    expo_frac = exponentielle_somme(f,n=n)
    expo = expo_entier * expo_frac
    return expo

# Test
print("--- Définition(s) de l'exponentielle ---")
print("- Exemple 1 -")
x = 2.8
print("x =",x)
print("Valeur python :",exp(x))
print("Valeur limite :",exponentielle_limite(x,n=100))
print("Valeur somme :",exponentielle_somme(x,n=10))
# print("Valeur somme inverse :",exponentielle_somme_inverse(x,n=20))
print("Valeur somme Hörner :",exponentielle_horner(x,n=10))
print("Valeur Euler :",exponentielle_euler(x,n=10))
print("Valeur astuce :",exponentielle_astuce(x,n=10))

##
print("- Exemple 2 -")
x = 0.1
print("x =",x)
print("Valeur python :",exp(x))
print("Valeur limite :",exponentielle_limite(x,n=100))
print("Valeur somme :",exponentielle_somme(x,n=10))
# print("Valeur somme inverse :",exponentielle_somme_inverse(x,n=20))
print("Valeur somme Hörner :",exponentielle_horner(x,n=10))
print("Valeur Euler :",exponentielle_euler(x,n=10))
print("Valeur astuce :",exponentielle_astuce(x,n=10))

##
print("- Exemple 3 -")
x = 100.5
print("x =",x)
print("Valeur python :",exp(x))
print("Valeur limite :",exponentielle_limite(x,n=100))
print("Valeur somme :",exponentielle_somme(x,n=10))
# print("Valeur somme inverse :",exponentielle_somme_inverse(x,n=20))
print("Valeur somme Hörner :",exponentielle_horner(x,n=10))
# print("Valeur Euler :",exponentielle_euler(x,n=35))
print("Valeur astuce :",exponentielle_astuce(x,n=10))
```

6. Logarithme

Activité 1

Activité 1 logarithme_1.py

```
##############################
# Logarithme
##############################

##############################
# Activité 1 - Le logarithme décimal - Échelle de Richter
##############################

from math import *

##############################
## Question 1 ##

def magnitude(E):
    """ Calcul la magnitude d'un seisme en fonction de l'energie """
    E0 = 1.6*10**-5
    M = 2/3 * log(E/E0,10) - 3.2
    return M

# Test
print("--- Échelle de Richter ---")
# Vérification -> M ~ 4
E1 = 10**6
print("Energie E =",E1,"magnitude M = ",magnitude(E1))

##############################
## Question 2 ##

# Energie = puissance de 10, afficher magnitude jusqu'à avoir M>9
print("--- Quelques calculs de magnitude ---")
for i in range(6,15):
    E = 10**i
    print("E = 10^",i,"  Energie E =",E,"  magnitude M = ",magnitude(E))


##############################
## Question 3 ##

# Trouver E tel que M = 7 (tatonnment, balyage, calcul):
# Tâtonnement + balayage
print("--- Balayage des énergie pour obtenir un magnitude 7 ---")

for E in range(10**10,10**11,10**9):
    print("Energie E =",E,"  magnitude M = ",magnitude(E))

print("Vérification")
E = 3.2 * 10*10
print("Energie E =",E,"  magnitude M = ",magnitude(E))

##############################
## Question 4 ##

print("--- Magnitude augmenté de 1 ----")
# Montrer que si E2 = 1000 E1 alors M2 = M1 + 2
E1 = 10**7  # n'importe quelle valeur
E2 = 1000 * E1
```

```
print("Energie E1 =",E1,"  magnitude M1 = ",magnitude(E1))
print("Energie E2 =",E2,"  magnitude M2 = ",magnitude(E2))

# Montrer que si E2 = sqrt(1000) E1 alors M2 = M1 + 1
# Réponse k = sqrt(1000) ~ 32
E1 = 10*7  # n'importe quelle valeur
E2 = sqrt(1000) * E1   # sqrt(1000) ~ 32
print("Energie E1 =",E1,"  magnitude M1 = ",magnitude(E1))
print("Energie E2 =",E2,"  magnitude M2 = ",magnitude(E2))
```

Activité 2

Activité 2 logarithme_2.py

```
################################
# Logarithme
################################

from math import *

################################
# Activité 2 - Le logarithme décimal - Décibels
################################

################################
## Question 1 ##

def decibel(P):
    """ Calcule l nombre de décibels d'un son en fonction d'une puissance """
    P0 = 2 * 10**-5
    D = 20 * log(P/P0,10)
    return D

# Test
print("--- Décibels ---")
# Vérification -> D ~ 94
P = 1
print("Pression P =",P,"décibels D = ",decibel(P))

################################
## Question 2 ##

# compléter tableau

def inverse_decibel(D):
    """ Calcule la puissance en fonction du nombre de décibels """
    P0 = 2 * 10**-5
    P = P0 * 10**(D/20)
    return P

# Test
print("--- Inverse décibels ---")
D = 94
print("Pression P =",inverse_decibel(D),"décibels D = ",D)

# Moteur d'avion à réaction (à 1 mètre) P = 632, D = 150
# Marteau-piqueur (à 1 mètre) P = 2,
# Niveau de dommage à l'oreille P > 0.355, D > 85
# Niveau de gêne D > 70
# Conversation (à 1 mètre) P = 0.002 à 0.02 (D = 40 à 60)
```

```
# Chambre calme (son environnant)  D = 10 à 20
# Seuil de l'audition à 1kHz (à l'oreille)   P = 2*10**-5
# Chambre anéchoïque   D = -10 dB

print("--- Tableau ----")
# P -> D
for P in [632,2,0.355,0.02,0.002,2*10**-5]:
    print("Pression P =",P," - Décibels D = ",decibel(P))

# D -> P
for D in [150,85,60,40,-10]:
    print("Pression P =",inverse_decibel(D)," - Décibels D = ",D)
```

Activité 3

Activité 3 logarithme_3.py

```
##############################
# Logarithme
##############################

from math import *

##############################
# Activité 3 - Le logarithme décimal - Échelle logarithmique
##############################

import matplotlib.pyplot as plt

def afficher_points_xy(points):
    for (x,y) in points:
        plt.scatter(x,y,color="red")

def afficher_points_xlogy(points):
    for (x,y) in points:
        plt.scatter(x,log(y,10),color="green")

def afficher_points_logxlogy(points):
    for (x,y) in points:
        plt.scatter(log(x,10),log(y,10),color="blue")

# Test
points1 = [ (x,1.5*x+2) for x in [2,3,5,7,11] ]      # y = ax+b a=1.5, b=-2
points2 = [ (x,exp(0.25*x+1)) for x in [2,3,5,7,11] ]  # y = exp(a x + b) log(y) = a x + b
points3 = [ (x,2*x**1.5) for x in [2,3,5,7,11] ]     # y = beta x^alpha, log(y) = alpha*log(
    ↪ x) + log(beta)
points = points3
print(points)
afficher_points_xy(points)
# afficher_points_xlogy(points)
# afficher_points_logxlogy(points)


plt.axes().set_aspect('equal')
plt.xlim(xmin=0)
plt.ylim(ymin=0)
plt.grid()
plt.show()
```

Activité 4

Activité 4 logarithme_4.py

```
##############################
# Logarithme
##############################

##############################
# Activité 4 - Le logarithme népérien
##############################

from math import *

##############################
## Question 1 ##

a = 2
b = 3
e = exp(1)
n = 7

print("--- Propriétés du logarithme ---")
print(log(a*b),log(a)+log(b))
print(log(a/b),log(a)-log(b))
print(log(1/a),-log(a))
print(log(a**n),n*log(a))
print(log(sqrt(a)),0.5*log(a))
print(a**b,exp(b*log(a)))

##############################
## Question 2 ##

def table_ln(x,N):
    """ Simule une table x -> ln(x) """
    l = floor(log(x)*10**N)/10**N
    return l

def table_exp(x,N):
    """ Simule une table x -> exp(x) autrement dit ln(x) -> x """
    e = floor(exp(x)*10**N)/10**N
    return e

# Test
print("--- Table des logarithme ---")
N = 4
print("Nombre de chiffre après la virgule N=",N)
x = 54
print("x =",x)
print("ln(x) =",table_ln(x,N))
y = 1.23
print("y =",y)
print("exp(y) =",table_exp(y,N))


##############################
## Question 3 ##

def multiplication(a,b,N):
    """ Multiplication par table des logarithmes """
    la = table_ln(a,N)
    lb = table_ln(b,N)
    lc = la + lb
```

```
    c = table_exp(lc,N)
    return c
# Test
print("--- Produit par tables des logarithme ---")
N = 7
print("Nombre de chiffres après la virgule de la table N=",N)
a = 98.765
b = 43.201
print("a =",a)
print("b =",b)
print("par table   : c =",multiplication(a,b,N))
print("vérfication : c =",a*b)
```

Activité 5

Activité 5 logarithme_5.py

```
################################
# Logarithme
################################

################################
# Activité 5 - Le logarithme en base 2 (ou pas)
################################

from math import *

################################
## Question 1 ##

def logarithme_entier(x):  # x >= 1
    """ Logarithme décimal entier de x """
    l = 0
    while 10**l <= x:
        l = l + 1
    return l-1

# Test
print("--- Partie entière du logarithme : base 10 ---")
for x in range(1,11):
    print("x =",x)
    print("mon calcul",logarithme_entier(x))
    print("Python",log(x,10))

################################
## Question 2 ##

def logarithme_entier_2(n):
    """ Logarithme entier en base 2 """
    l = 0
    while 2**l <= n:
        l = l + 1
    return l-1

 # Test
print("--- Partie entière du logarithme : base 2 ---")
for x in range(1,20):
    print("x =",x)
```

```
    print("mon calcul",logarithme_entier_2(x))
    print("Python",log(x,2))


##############################
## Question 3 ##

def dichotomie(n):
    """ Simule la dichotomie en compte le nombre d'étape maximal """
    compteur = 0
    while n>1:
        n = max(n//2,n-n//2)
        compteur = compteur + 1
    return compteur

 # Test
print("--- Dichotomie et logarithme en base 2 ---")
for n in range(4,10):
    print("n =",n)
    print("dichotomie",dichotomie(n))
    print("Python",ceil(log(n,2)))
    print("log2 entier",logarithme_entier_2(n))  # valable uniquement si n puissance de 2 (
    ↪ sinon faire +1)

##############################
## Question 4 ##

def logarithme_base(x,b):
    """ Logarithme en base b qcq à partir de ln """
    return log(x)/log(b)

# Test
print("--- Logarithme en base quelconque ---")
x = 555
b = 10
print("x =",x)
print("Moi log(x,b)",logarithme_base(x,b))
print("Python log(x,b)",log(x,b))


##############################
## Question 5 ##

def nombre_de_chiffres(n,b):
    """ Nombre de chiffres d'un entier n dans son écriture en base b """
    return floor(logarithme_base(n,b))+1
# Test
print("--- Nombre de chiffres ---")
n = 123
print("base 10, n =",n)
print("nb chiffres",nombre_de_chiffres(n,10))
print("base 2, n =",bin(n))
print("nb chiffres",nombre_de_chiffres(n,2))
print("base 16, n =",hex(n))
print("nb chiffres",nombre_de_chiffres(n,16))
```

Activité 6

Activité 6 logarithme_6.py

```
################################
# Logarithme
################################

from math import *

################################
# Activité 6 - Calculs des logarithmes I
################################

################################
## Question 1 ##

# ln(x) = ln(1+u) = u-u^2/2+u^3/3 + ...
# u = x - 1
def logarithme_serie_1(x,N):
    """ Calcul approché de ln(x) par une somme """
    u = x-1
    somme = 0
    for i in range(1,N):
        if i%2 == 0:
            somme = somme - (u**i)/i
        else:
            somme = somme + (u**i)/i
    return somme

################################
## Question 2 ##

# ln(x) = ln(1+u/1-u) = 2 u + 2u^3/3 + 2 u^5/5 + ...
# u = (x-1)/(x+1)
def logarithme_serie_2(x,N):
    """ Calcul approché de ln(x) par une somme """
    u = (x-1)/(x+1)
    somme = 0
    for i in range(1,N,2):
        somme = somme + 2*(u**i)/i
    return somme

################################
## Question 3 ##

def reduction_intervalle_e(x):
    """ Ecriture x = y * e^k avec 0.5 < y < 1.5, k exposant entier
    On ramène x par division par e à l'intervalle [0.5,1.5] """
    y = x
    k = 0
    # Cas x trop grand
    while y > 1.5:
        y =  y/e  # e = exp(1) définie dans le module math
        k += 1
    # Cas x trop petit
    while y < 0.5:
        y = y*e
        k += -1
    return y, k

# Test
```

```
print("--- Calcul du logarithme ---")
x = 1.543
# x = 0.1234
N = 10
print("x =",x)
print("Précision N =",N)
print("Valeur python",log(x))
print("Valeur série 1",logarithme_serie_1(x,N))
print("Valeur série 2",logarithme_serie_2(x,N))

print("--- Réduction intervalle ---")
x = 12.34
y,k = reduction_intervalle_e(x)
print("x =",x,"y = ",y,"k =",k,"vérif = ",y*e**k)

##############################
## Question 4 ##

def logarithme_serie_3(x,N):
    """ Calcul approché de ln(x) par une somme """
    y,k = reduction_intervalle_e(x)
    logy = logarithme_serie_2(y,N)    # Méthode valide car y ~ 1
    logx = logy + k                   # Décalage
    return logx

# Test
print("--- Logarithme après réduction d'intervalle ---")
x = 154.3
N = 10
print("x =",x)
print("Précision N =",N)
print("Valeur python",log(x))
print("Valeur série 3 (après réduction)",logarithme_serie_3(x,N))
```

Activité 7

Activité 7 — logarithme_7.py

```
##############################
# Logarithme
##############################

from math import *

##############################
# Activité 7 - Calculs des logarithmes II
##############################

##############################
## Question 1 ##

def logarithme_inverse(x,N):
    """ Calcul approché de ln(x) comme solution de exp(y)=x
    Calcul par méthode de Newton  avec
    f(u) = exp(u) - x u <- u - f(u)/f'(u) """
    u = 1

    for i in range(N):
        # print(i,u)
```

```
        u = u - (exp(u)-x)/exp(u)

    return u

# Test
print("--- Logarithme comme réciproque de l'exponentielle ---")
x = 1.234
N = 10
print("x =",x)
print("Précision N =",N)
print("Valeur python",log(x))
print("Valeur par réciproque",logarithme_inverse(x,N))

##############################
## Question 2 ##

# Ecriture x = y * 10^k avec 1 <= y < 10, k exposant entier
def reduction_intervalle_10(x):
    """ Ecriture x = y * 10^k avec 1 <= y < 10, k exposant entier
    On ramène x par division par e à l'intervalle [0.5,1.5] """
    y = x
    k = 0
    # Cas x trop grand
    while y >= 10:
        y =  y/10
        k += 1
    # Cas x trop petit
    while y < 1:
        y = y*10
        k += -1
    return y, k

##############################
## Question 3 ##

def logarithme_cordic(x,N):
    """ Calcul approché de ln(x) par l'algorithme CORDIC
    D'après Nicole Robb - APMEP """

    # Réduction d'intervalle
    y, k = reduction_intervalle_10(x)

    # y dans [1,10[
    p = log(10)
    for i in range(N):
        q = 1 + 10**-i
        print(q,y)
        while q*y <= 10:
            print(q,y)
            y = q*y
            p = p - log(q)

    return p + k*log(10)  # Résultat après décalage de k*ln(10)

# Test
print("--- Logarithme par algorithme CORDIC ---")
x = 37
N = 4
print("x =",x)
print("Précision N =",N)
print("Valeur python",log(x))
print("Valeur CORDIC",logarithme_cordic(x,N))
```

```
##############################
## Question 4 ##

def logarithme_briggs(x,epsilon):
    """ Calcul approché de ln(x) par l'algorithme de Briggs
    2^n ln(x) = ln(x^(1/2^n))
    puis ln(u) ~ u-1 pour u proche de 0 """
    n = 0
    while abs(x-1) > epsilon:
        x = sqrt(x)
        n = n+1

    l = x-1
    for i in range(n):
        l = 2*l

    return l

# Test
print("--- Logarithme par algorithme de Briggs ---")
x = 1.543
epsilon = 10**-10
print("x =",x)
print("Précision epsilon =",epsilon)
print("Valeur python",log(x))
print("Valeur Briggs",logarithme_briggs(x,epsilon))
```

7. Intégrale

Activité 1 à 3

Activité 1 à 3 integrale.py

```
##############################
# Analyse - Intégration
##############################

##############################
# Activité 1 - Primitive
##############################

##############################
## Question 1  ##

# Rappel de "Dérivées"

def derivee(f,a,h=0.00001):
    """ Calcul approché de f'(a) """
    taux = (f(a+h)-f(a))/h
    return taux

def verification_primitive(f,F,a,b,n=10,epsilon=0.01):
    """ Vérification expérimentale que F'(x) = f(x) """
    h = (b-a)/n
    x = a

    valide = True
```

```
    for i in range(n+1):
        ecart = f(a)-derivee(F,a)
        if abs(ecart) > epsilon:
            valide = False
            print(a,f(a),derivee(F,a))
        a = a + h

    return valide

# Test
f = lambda x: x**3
F = lambda x: 1/4*x**4

print("Primitive ok ?",verification_primitive(f,F,0,2))

##############################
## Question 2 ##

def integrale_primitive(F,a,b):
    """ Calcule int_a^b f(t) dt par la formule F(b)-F(a) """
    return F(b)-F(a)

f = lambda x: x**3
F = lambda x: 1/4*x**4

print("Intégrale connaissant la primitive :",integrale_primitive(F,0,2))

##############################
# Activité 2.1 - Méthode des rectangles
##############################

def integrale_rectangles(f,a,b,n):
    """ Calcule int_a^b f(t) dt par la méthode des rectangles """
    h = (b-a)/n
    x = a

    integrale = 0

    for i in range(n):
        integrale = integrale + f(a)*h
        a = a + h

    return integrale

f = lambda x: x**3

print("Intégrale par rectangle :",integrale_rectangles(f,0,2,n=100))


##############################
# Activité 2.2 - Méthode des trapèzes
##############################

def integrale_trapezes(f,a,b,n):
    """ Calcule int_a^b f(t) dt par la méthode des trapèzes """

    h = (b-a)/n
    x = a

    integrale = 0

    for i in range(n):
        integrale = integrale + (f(a)+f(a+h))/2*h
        a = a + h

    return integrale
```

```
f = lambda x: x**3

print("Intégrale par trapèzes :",integrale_trapezes(f,0,2,n=100))

##############################
# Activité 2.3 - Méthode de Simpson
##############################

def integrale_simpson(f,a,b,n):
    """ Calcule int_a^b f(t) dt par la méthode de Simpson """

    h = (b-a)/n
    x = a

    integrale = 0

    for i in range(n):
        integrale = integrale + (f(a)+4*f(a+h/2)+f(a+h))/6*h
        a = a + h

    return integrale

f = lambda x: x**3

print("Intégrale par Simpson :",integrale_simpson(f,0,2,n=100))

# Fonction utile pour la suite
def integrale(f,a,b,n):
    return integrale_simpson(f,a,b,n)

##############################
# Activité 3 - Intégrale de Gauss
##############################

from math import *

def integrale_gauss(x,mu,sigma2):
    """ Calcul de l'intégrale de Gauss """

    P = lambda x: 1/sqrt(2*pi*sigma2)*exp( -1/2 * (x-mu)**2 / sigma2 )
    infini = 20*sqrt(sigma2)  # une grande valeur
    p = integrale(P,-infini,x,100)
    return p

# Test
print(integrale_gauss(1,0,1))
print(integrale_gauss(0,0,1))

print('QI',integrale_gauss(115,100,225))
```

8. Programmation objet

Activité 1

Activité 1 objet_1.py

```
##############################
# Programmation objet
##############################

##############################
```

```
# Activité 1 - Matrice 2x2
##############################

class Matrice:

    def __init__(self,a,b,c,d):
        self.a = a
        self.b = b
        self.c = c
        self.d = d

    def __str__(self):
        ligne1 = str(self.a) + " " + str(self.b) + "\n"
        ligne2 = str(self.c) + " " + str(self.d)
        return ligne1 + ligne2

    def trace(self):
        tr = self.a + self.d
        return tr

    def determinant(self):
        det = self.a*self.d - self.b*self.c
        return det

    def produit_par_scalaire(self,k):
        M = Matrice(k*self.a,k*self.b,k*self.c,k*self.d)
        return M

    def inverse(self):
        det = self.determinant()
        if det == 0:
            return None
        else:
            M = Matrice(self.d,-self.b,-self.c,self.a)
            MM = M.produit_par_scalaire(1/det)
        return MM

    def __add__(self,other):
        M = Matrice(self.a+other.a,self.b+other.b,self.c+other.c,self.d+other.d)
        return M

    def __mul__(self,other):
        M = Matrice(self.a*other.a + self.b*other.c,
                    self.a*other.b + self.b*other.d,
                    self.c*other.a + self.d*other.c,
                    self.c*other.b + self.d*other.d
            )
        return M

# Test
print("--- Objet : Matrice() ---")

print("- Affichage -")
M = Matrice(1,2,3,4)
print(M)
M.__str__
print("Trace :",M.trace())
print("Déterminant : ",M.determinant())

print("- Produit par scalaire -")

MM = M.produit_par_scalaire(5)
print(MM)

print("- Inverse -")
```

```
print(M.inverse())

print("- Addition -")

M1 = Matrice(4,3,2,1)
M2 = Matrice(1,0,-1,1)
M3 = M1+M2
print(M3)

print("- Multiplication -")

M4 = M1*M2
print(M4)
print(M2*M1)

print("- Verif. inverse -")
M1 = Matrice(4,3,2,1)
M5 = M1.inverse()
# print(M5)
print(M1*M5)

# Application à Fibonnacci
print("- Fibonacci -")

M = Matrice(0,1,1,1)
n = 100
Mn = M
for k in range(n-1):
    Mn = Mn * M
print("n =",n)
print("Matrice puissance n :",Mn)
print("Print coeff. Fn (donné par d) :",Mn.d)
```

Activité 2

Activité 2 objet_2.py

```
################################
# Programmation objet
################################

################################
# Activité 2 - Tortue
################################

################################
## Question 1  ##

class TortueBasique:

    def __init__(self):
        self.x = 0
        self.y = 0
        self.trace = True
        self.couleur = 'red'

    # Q1
    def renvoyer_xy(self):
        return self.x, self.y

    # Q2
    def aller_a_xy(self,x,y):
```

```
        x0 = self.x  # Position actuelle
        y0 = self.y

        self.x = x   # Nouvelle position
        self.y = y

        if self.trace:
            canvas.create_line(x0,y0,x,y,fill=self.couleur,width=5)
        return

    # Q3
    def abaisser_stylo(self):
        self.trace = True

    def relever_stylo(self):
        self.trace = False

    def changer_couleur(self,couleur):
        self.couleur = couleur

# Fenêtre tkinter
from tkinter import *
root = Tk()
canvas = Canvas(root, width=800, height=600, background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

tortue = TortueBasique()

tortue.aller_a_xy(100,100)
tortue.relever_stylo()
tortue.aller_a_xy(200,100)
tortue.abaisser_stylo()
tortue.aller_a_xy(100,200)

root.mainloop()

# print("--- Objet : TortueBasique() ---")
# print(tortue.renvoyer_xy())

##############################
## Question 4 ##

from math import *

tortue1 = TortueBasique()
tortue2 = TortueBasique()
tortue2.changer_couleur('blue')
tortue3 = TortueBasique()
tortue3.changer_couleur('orange')
# tortue4 = TortueBasique()
# tortue4.changer_couleur('green')

for i in range(0,400):
    tortue1.aller_a_xy(3/2*i,i)
    tortue2.aller_a_xy(i,5*sqrt(i))
    x1,y1 = tortue1.renvoyer_xy()
    x2,y2 = tortue2.renvoyer_xy()
    x3,y3 = round((x1+x2)/2), round((y1+y2)/2)
    tortue3.aller_a_xy(x3,y3)

    # if i%20 == 0:
    #     tortue4.aller_a_xy(x1,y1)
    #     tortue4.aller_a_xy(x2,y2)

root.mainloop()
```

Activité 3

Activité 3 objet_3.py

```
##############################
# Programmation objet
##############################

##############################
# Rappel de l'activité 2 - Tortue
##############################

##############################
## Question 1  ##

class TortueBasique:

    def __init__(self):
        self.x = 0
        self.y = 0
        self.trace = True
        self.couleur = 'red'

    def renvoyer_xy(self):
        return self.x, self.y

    def aller_a_xy(self,x,y):
        x0 = self.x  # Position actuelle
        y0 = self.y

        self.x = x   # Nouvelle position
        self.y = y

        if self.trace:
            canvas.create_line(x0,y0,x,y,fill=self.couleur,width=5)
        return

    def abaisser_stylo(self):
        self.trace = True

    def relever_stylo(self):
        self.trace = False

    def changer_couleur(self,couleur):
        self.couleur = couleur


##############################
# Activité 3 - Tortue Héritage
##############################

from math import *

##############################
## Question 1 ##

class TortueTournante(TortueBasique):

    def __init__(self):
        TortueBasique.__init__(self)
        self.direction = 0

    def fixer_direction(self,direction):
        self.direction = direction

    def tourner(self,angle):
```

```
        self.direction = self.direction + angle

    def avancer(self,longueur):
        angle = self.direction
        dx = longueur*cos(2*pi/360*angle)
        dy = longueur*sin(2*pi/360*angle)
        self.aller_a_xy(self.x+dx,self.y+dy)

    def sorienter_vers(self,other):
        x1,y1 = self.x, self.y
        x2,y2 = other.x, other.y
        angle = 360/(2*pi)*atan2(y2-y1,x2-x1)
        self.direction = angle

# Fenêtre tkinter
from tkinter import *
root = Tk()
canvas = Canvas(root, width=800, height=600, background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

print("--- Objet : TortueTournante() ---")

# tortue = TortueTournante()

# tortue.aller_a_xy(100,100)
# tortue.avancer(100)
# tortue.tourner(90)
# tortue.avancer(100)
# print(tortue.direction)

tortue1 = TortueTournante()
tortue2 = TortueTournante()
tortue2.changer_couleur('blue')
tortue2.relever_stylo()
tortue2.aller_a_xy(700,1)
tortue2.abaisser_stylo()
tortue2.fixer_direction(90)

for i in range(400):
    tortue2.avancer(1)
    tortue1.sorienter_vers(tortue2)
    tortue1.avancer(2)
root.mainloop()
```

9. Mouvement de particules

Activité 1

Activité 1 particule_1.py

```
##############################
# Particules
##############################

from math import *
from time import *

##############################
```

```
# Activité 1 - Particule
##############################

class Particule():

    def __init__(self,x,y,vx,vy,m):
        self.x = x
        self.y = y
        self.vx = vx
        self.vy = vy
        self.m = m

    def __str__(self):
        ligne = "("+str(self.x)+","+str(self.y)+"),("+str(self.vx)+", "+str(self.vy)+"), "+
    ↪ str(self.m)
        return ligne

    def action_vitesse(self):
        self.x = self.x + self.vx
        self.y = self.y + self.vy

    def action_gravite(self,gravite=0.2):
        self.vy = self.vy - gravite

    def action_frottement(self,frottement=0.005,exposant=2):
        vx,vy,m = self.vx,self.vy,self.m
        vitesse = sqrt(vx**2+vy**2)
        self.vx = vx - 1/m * frottement * (vitesse**exposant) * (vx/vitesse)
        self.vy = vy - 1/m * frottement * (vitesse**exposant) * (vy/vitesse)

    def affiche(self,avec_fleche=False):
        x,y,vx,vy,m = self.x,self.y,self.vx,self.vy,self.m
        i,j = xy_vers_ij(x,y)

        echelle = 1
        if avec_fleche:
            fleche = canvas.create_line(i,j,i+vx*echelle,j-vy*echelle,fill="blue",arrow="
    ↪ last",width=4)

        rayon = min(max(1,2*sqrt(m)),15)
        disque = canvas.create_oval(i-rayon,j-rayon,i+rayon,j+rayon,fill="red")

    def rebondir_si_bord_atteint(self):
        x,y = self.x,self.y
        i,j = xy_vers_ij(x,y)
        if i <= 0 or i >= Largeur:
            self.vx = -self.vx
        if j <= 0 or j >= Hauteur:
            self.vy = -self.vy

    def mouvement(self):
        self.action_vitesse()
        self.action_gravite()
        self.action_frottement()
        self.rebondir_si_bord_atteint()
        # self.affiche()


# Exemple
print("--- Objet : Particule() ---")
p1 = Particule(-100,100,10,10,1)
print(p1)


# Constantes pour l'affichage
```

```
Largeur = 800
Hauteur = 600

# Conversion de coordonnées
def xy_vers_ij(x,y):
    i = Largeur//2 + x
    j = Hauteur//2 - y
    return (i,j)

# Fenêtre tkinter
from tkinter import *

root = Tk()
canvas = Canvas(root, width=Largeur, height=Hauteur, background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

# Exemple 1 - Une particule fixe
p1 = Particule(-100,200,70,30,5)
print(p1)
p1.affiche(avec_fleche=True)

# Exemple 2 - Une particule fixe (sans flèche)
p2 = Particule(100,200,70,30,15)
print(p2)
p2.affiche()

# Exemple 3 - Une particule en mouvement linéaire
p = Particule(-250,50,10,5,1)
p.affiche()
for k in range(40):
    p.action_vitesse()
    p.affiche()

# Exemple 4 - Une particule sous l'action de la gravité,
# avec frottements et avec rebonds
p = Particule(-360,-107,10,15,5)
p.affiche()
for k in range(198):
    p.action_vitesse()
    p.action_gravite(gravite=1)
    p.action_frottement(frottement=0.01,exposant=1.8)
    p.affiche()
    p.rebondir_si_bord_atteint()

root.mainloop()
```

Activité 2

Activité 2 particule_2.py

```
##############################
# Particules
##############################

from math import *
from random import *
from time import *

##############################
# Rappels - Activité 1 - Particule
```

```
##############################

class Particule():

    def __init__(self,x,y,vx,vy,m):
        self.x = x
        self.y = y
        self.vx = vx
        self.vy = vy
        self.m = m

    def __str__(self):
        ligne = "("+str(self.x)+","+str(self.y)+"),("+str(self.vx)+", "+str(self.vy)+"), "+
    ↪ str(self.m)
        return ligne

    def action_vitesse(self):
        self.x = self.x + self.vx
        self.y = self.y + self.vy

    def action_gravite(self,gravite=0.2):
        self.vy = self.vy - gravite

    def action_frottement(self,frottement=0.005,exposant=2):
        vx,vy,m = self.vx,self.vy,self.m
        vitesse = sqrt(vx**2+vy**2)
        self.vx = vx - 1/m * frottement * (vitesse**exposant) * (vx/vitesse)
        self.vy = vy - 1/m * frottement * (vitesse**exposant) * (vy/vitesse)

    def affiche(self,avec_fleche=False):
        x,y,vx,vy,m = self.x,self.y,self.vx,self.vy,self.m
        i,j = xy_vers_ij(x,y)

        echelle = 1
        if avec_fleche:
            fleche = canvas.create_line(i,j,i+vx*echelle,j-vy*echelle,fill="blue",arrow="
    ↪ last",width=4)

        rayon = min(max(1,2*sqrt(m)),15)
        disque = canvas.create_oval(i-rayon,j-rayon,i+rayon,j+rayon,fill="red")

    def rebondir_si_bord_atteint(self):
        x,y = self.x,self.y
        i,j = xy_vers_ij(x,y)
        if i <= 0 or i >= Largeur:
            self.vx = -self.vx
        if j <= 0 or j >= Hauteur:
            self.vy = -self.vy

    def mouvement(self):
        self.action_gravite()
        self.action_frottement()
        self.rebondir_si_bord_atteint()
        self.action_vitesse()
        # self.affiche()

##############################
# Activité 2 - Particules en mouvement

class TkParticule(Particule):

    def __init__(self,x,y,vx,vy,m,couleur="red"):
        Particule.__init__(self,x,y,vx,vy,m)
        self.couleur = couleur
        # Creation de l'objet tkinter
```

```
        i,j = xy_vers_ij(x,y)
        rayon = min(max(1,m),10)
        disque = canvas.create_oval(i-rayon,j-rayon,i+rayon,j+rayon,fill=self.couleur)
        self.id = disque

    def affiche(self,avec_trace=False):
        # Trace (option)
        if avec_trace:
            x,y,m = self.x, self.y, self.m
            i,j = xy_vers_ij(x,y)
            rayon = min(max(2,m),10)//2
            canvas.create_oval(i-rayon,j-rayon,i+rayon,j+rayon,fill='gray')
        # Mouvement
        canvas.move(self.id,self.vx,-self.vy)


##############################

# Constantes pour l'affichage
Largeur = 800
Hauteur = 600

# Conversion de coordonnées
def xy_vers_ij(x,y):
    i = Largeur//2 + x
    j = Hauteur//2 - y
    return (i,j)

# Fenêtre tkinter
from tkinter import *

root = Tk()
canvas = Canvas(root, width=Largeur, height=Hauteur, background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

##############################
# Choix d'une couleur

def hasard_couleur():
    R,V,B = randint(0,255),randint(0,255),randint(0,255)
    couleur = '#%02x%02x%02x' % (R%256, V%256, B%256)
    return couleur


# Exemple 1 - Une particule en mouvement
# p = TkParticule(-300,10,5,2,10)
# for k in range(500):
#     p.mouvement()
#     p.affiche()
#     canvas.update()
#     sleep(0.05)

# root.mainloop()

# Exemple 2 - Plusieurs particules en mouvement
# Choix des couleurs
# mes_couleurs = [hasard_couleur() for j in range(10)]
# print(mes_couleurs)
mes_couleurs = ['#8963bc', '#56988b', '#3ef562', '#120cad', '#3e4b21', '#f8e328', '#c8ee51',
    ↪ '#c55cdd', '#1926c5', '#1f6cea']
# Plusieurs particules en mouvement
liste_particules = [TkParticule(-350,0,10,j,5,couleur=mes_couleurs[j]) for j in range(10)]
for k in range(110):
    for p in liste_particules:
```

```
        p.mouvement()
        p.affiche(avec_trace=True)

    canvas.update()
    sleep(0.05)

root.mainloop()
```

Activité 3

Activité 3 particule_3.py

```
##############################
# Particules
##############################

from math import *
from time import *

##############################
# Rappel - Activité 1 - Particule
##############################

class Particule():

    def __init__(self,x,y,vx,vy,m):
        self.x = x
        self.y = y
        self.vx = vx
        self.vy = vy
        self.m = m

    def __str__(self):
        ligne = "("+str(self.x)+","+str(self.y)+"),("+str(self.vx)+", "+str(self.vy)+"), "+
    ↪ str(self.m)
        return ligne

    def action_vitesse(self):
        self.x = self.x + self.vx
        self.y = self.y + self.vy

    def action_gravite(self,gravite=0.2):
        self.vy = self.vy - gravite

    def action_frottement(self,frottement=0.005,exposant=2):
        vx,vy,m = self.vx,self.vy,self.m
        vitesse = sqrt(vx**2+vy**2)
        self.vx = vx - 1/m * frottement * (vitesse**exposant) * (vx/vitesse)
        self.vy = vy - 1/m * frottement * (vitesse**exposant) * (vy/vitesse)

    def affiche(self,avec_fleche=False):
        x,y,vx,vy,m = self.x,self.y,self.vx,self.vy,self.m
        i,j = xy_vers_ij(x,y)

        echelle = 1
        if avec_fleche:
            fleche = canvas.create_line(i,j,i+vx*echelle,j-vy*echelle,fill="blue",arrow="
    ↪ last",width=4)

        rayon = min(max(1,2*sqrt(m)),15)
        disque = canvas.create_oval(i-rayon,j-rayon,i+rayon,j+rayon,fill="red")
```

```
    def rebondir_si_bord_atteint(self):
        x,y = self.x,self.y
        i,j = xy_vers_ij(x,y)
        if i <= 0 or i >= Largeur:
            self.vx = -self.vx
        if j <= 0 or j >= Hauteur:
            self.vy = -self.vy

    def mouvement(self):
        self.action_gravite()
        self.action_frottement()
        self.rebondir_si_bord_atteint()
        self.action_vitesse()
        # self.affiche()


##############################
# Activité 3 - Planètes
##############################

class Planete(Particule):

    def action_attraction(self,other,G=100):
        x1, y1, vx1, vy1, m1 = self.x,self.y,self.vx,self.vy,self.m
        x2, y2, vx2, vy2, m2 = other.x,other.y,other.vx,other.vy,other.m

        x = x2-x1
        y = y2-y1
        r = sqrt(x**2+y**2)  # Distance entre les corps

        gx = G*m1*m2/(r**2) * x/r
        gy = G*m1*m2/(r**2) * y/r

        self.vx = self.vx + gx/m1
        self.vy = self.vy + gy/m1

    def mouvement(self):
        self.action_vitesse()

class TkPlanete(Planete):

    def __init__(self,x,y,vx,vy,m,couleur="red"):
        Particule.__init__(self,x,y,vx,vy,m)
        self.couleur = couleur
        # Creation de l'objet tkinter
        i,j = xy_vers_ij(x,y)
        rayon = 2*min(max(1,m),10)
        disque = canvas.create_oval(i-rayon,j-rayon,i+rayon,j+rayon,fill=self.couleur)
        self.id = disque

    def affiche(self,avec_trace=False):
        # Trace (option)
        if avec_trace:
            x,y,m = self.x, self.y, self.m
            i,j = xy_vers_ij(x,y)
            rayon = 2*min(max(2,m),10)//2
            canvas.create_oval(i-rayon,j-rayon,i+rayon,j+rayon,fill='gray')
        # Mouvement
        canvas.move(self.id,self.vx,-self.vy)


##############################

# Constantes pour l'affichage
```

```
Largeur = 800
Hauteur = 600

# Conversion de coordonnées
def xy_vers_ij(x,y):
    i = Largeur//2 + x
    j = Hauteur//2 - y
    return (i,j)

# Fenêtre tkinter
from tkinter import *

root = Tk()
canvas = Canvas(root, width=Largeur, height=Hauteur, background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

# # Deux astres : Terre et Soleil
# soleil = TkPlanete(0,0,0,0,100,"yellow")
# terre = TkPlanete(-200,0,0,-5,3,"blue")

# for k in range(140):
#     terre.action_attraction(soleil)
#     terre.mouvement()
#     terre.affiche(avec_trace=True)
#     canvas.update()
#     sleep(0.01)

# root.mainloop()

# On peut remplacer la Terre par une comète
# soleil = TkPlanete(0,0,0,0,100,"yellow")
# comete = TkPlanete(-300,-200,6,0,5,"green")

# for k in range(80):
#     comete.action_attraction(soleil)
#     comete.mouvement()
#     comete.affiche(avec_trace=True)
#     canvas.update()
#     sleep(0.01)

# root.mainloop()

# Trois planètes : Terre et Soleil et Mars

soleil = TkPlanete(0,0,0,0,100,"yellow")
terre = TkPlanete(-200,0,0,-5,3,"blue")
mars = TkPlanete(-300,0,0,-5,2,"red")

for k in range(200):
    terre.action_attraction(soleil)
    terre.action_attraction(mars)
    terre.mouvement()
    terre.affiche(avec_trace=True)
    mars.action_attraction(soleil)
    mars.action_attraction(terre)
    mars.mouvement()
    mars.affiche(avec_trace=True)

    canvas.update()
    sleep(0.02)

root.mainloop()
```

10. Algorithmes récursifs

Activité 1

Activité 1 recursivite_1.py

```
##############################
# Récursivité
##############################

##############################
# Activité 1 - Pour bien commencer...
##############################

##############################
## Question 1 ##
def somme_carres_classique(n):
    """ Somme des carrés de 1 à n """
    S = 1
    for k in range(1,n+1):
        S = S + k**2
    return S

def somme_carres(n):
    """ Somme des carrés de 1 à n (fonction récursive) """
    if n==1:
        S = 1
    else:
        S = somme_carres(n-1) + n**2
    return S
# Test
print("--- Somme des carrés ---")
n = 100
print("n =",n)
print("S = ",somme_carres_classique(n))
print("S = ",somme_carres(n))

##############################
## Question 2 ##
def inverser_classique(liste):
    pass

def inverser(liste):
    """ Inverser l'ordre des éléments d'une liste (fonction récursive) """
    if len(liste) <= 1:
        return liste
    else:
        fin_liste = liste[1:]
        fin_liste_inverse = inverser(fin_liste)
        nouv_liste =  fin_liste_inverse + [liste[0]]
        return nouv_liste
# Test
print("--- Inverser une liste ---")
liste = [1,2,3,4,5]
print("liste =",liste)
print("inverse = ",inverser(liste))
```

```
# print("inverse = ",inverser_classique(liste))

##############################
## Question 3 ##

def maximum_classique(liste):
    """ Maximum des éléments d'une liste """
    M = liste[0]
    for x in liste:
        if x>= M:
            M = x

    return M

def maximum(liste):
    """ Maximum des éléments d'une liste (fonction récursive) """
    if len(liste) == 1:
        M = liste[0]
        return M
    else:
        M1 = liste[0]
        sous_liste = liste[1:]
        M2 = maximum(sous_liste)
        if M1 > M2:
            return M1
        else:
            return M2

# Test
print("--- Maximum ---")
liste = [7,5,3,9,1]
print("liste =",liste)
print("max = ",maximum_classique(liste))
print("max = ",maximum(liste))

##############################
## Question 4 ##

def binaire(n):
    """ Ecriture binaire de l'netier n (fonction récursive) """

    if n == 0:
        return '0'
    if n == 1:
        return '1'

    # Cas général
    if n%2 == 0:
        ecriture = binaire(n//2) + '0'
    else:
        ecriture = binaire(n//2) + '1'

    return ecriture

# Test
print("--- Binaire ---")
n = 23
print("n =",n)
print("binaire = ",binaire(n))
print("Python = ",bin(n))


##############################
```

```
## Question 5 ##

def est_palindrome_classique(mot):
    """ Tese si un mot est un palindrome
    (c-à-d se lit dans les deux sens) """
    ok_palind = True  # drapeau
    n = len(mot)
    for i in range(n//2):
        if mot[i] != mot[n-i-1]:
            ok_palind = False

    return ok_palind

def est_palindrome(mot):
    """ Tese si un mot est un palindrome
    (c-à-d se lit dans les deux sens) (fonction récursive) """

    if len(mot) <= 1:
        return True
    else:
        if mot[0] == mot[-1]:
            ok_debut_fin = True
        else:
            ok_debut_fin = False

        mot_milieu = mot[1:-1]
        ok_mileu = est_palindrome(mot_milieu)

        ok_palind = ok_debut_fin and ok_mileu

        return ok_palind

# Test
print("--- Palindrome ---")
mot = "RADAR"
print("mot =",mot)
print("Est palindrome ?",est_palindrome_classique(mot))
print("Est palindrome ?",est_palindrome(mot))
mot = "ABCXYCBA"
print("mot =",mot)
print("Est palindrome ?",est_palindrome_classique(mot))
print("Est palindrome ?",est_palindrome(mot))
```

Activité 2

Activité 2 recursivite_2.py

```
################################
# Récursivité
################################

################################
# Activité 2 - Fibonacci et Pascal
################################

################################
## Question 1 ##

 # Fibonacci
def fibonacci(n):
```

```
    """ Terme de rang n de la suite de Fibonacci (fonction doublement récursive) """
    if n==0:
        return 0
    elif n==1:
        return 1

    # if n == 2:
    #     print("Tiens je calcule encore F_2 !")

    # Cas n >= 2
    F_n_1 = fibonacci(n-1)
    F_n_2 = fibonacci(n-2)
    return F_n_1 + F_n_2

# Test
print("--- Fibonacci ---")
n = 15
print("n =", n)
print("Fibonacci =",fibonacci(n))


##############################
## Question 2 ##

# Triangle de Pascal

def binome(k,n):
    """ Coefficient 'k parmi n' du binôme de Newton (fonction doublement récursive) """
    if (k == 0) or (k == n):
        return 1

    else:
        return binome(k-1,n-1) + binome(k,n-1)

# Test
print("--- Binôme de Newton ---")
k, n = 3, 10
print("k, n =", k, n)
print("k parmi n =",binome(k,n))


##############################
## Question 3 ##

def afficher_pascal(N):
    """ Affichage dans la console des coeff du binôme de Newton """
    for n in range(N+1):
        for k in range(n+1):
            C = binome(k,n)
            print(C,end=" ")
        print()

    return

# Test
print("--- Triangle de Pascal ---")
afficher_pascal(10)

##############################
## Question 4 ##

def afficher_pascal_impair(N):
    """ Marquage des binômes impairs dans le triangle de Pascal """

    for n in range(N+1):
```

```
        for k in range(n+1):
            C = binome(k,n)
            if C%2 == 0:
                print(" ",end="")
            else:
                print("X",end="")
        print()

    return

# Test
print("--- Triangle de Pascal impair ---")
afficher_pascal_impair(10)

##############################
## Question 5 ##

def somme_chiffres(n):
    """ Calcul la somme des chiffres de n (fonction récursive) """
    if n < 10:
        return n

    # Cas général
    unite = n%10
    nn = n//10
    S = unite + somme_chiffres(nn)

    return S

# Test
print("--- Somme des chiffres ---")
n = 1357869
print("n =", n)
print("Somme =",somme_chiffres(n))

##############################
## Question 6 ##

def residu_chiffres(n):
    """ Calcul le résidu obtenu en itérant la somme des chiffres
    jusqu'à plus soif (fonction récursive) """

    if n < 10:
        return n

    # Cas général
    S = somme_chiffres(n)
    r = residu_chiffres(S)

    return r

# Test
print("--- Résidu des chiffres ---")
n = 1357869
print("n =", n)
print("Residu =",residu_chiffres(n))
```

Activité 3

Activité 3 recursivite_3.py

```
##############################
# Récursivité
##############################

##############################
# Activité 3 - Parcours d'arbre
##############################

##############################
## Question 1 ##

def pile_ou_face(n):
    """ Liste de tous les tirages de longueur n de piles ou faces """
    if n == 1:
        return ['P','F']  # ou bien n == 0 il faut ['']

    # Cas général
    sous_arbre = pile_ou_face(n-1)
    arbre_deb_P = ['P' + chaine for chaine in sous_arbre]
    arbre_dep_F = ['F' + chaine for chaine in sous_arbre]
    arbre = arbre_deb_P + arbre_dep_F

    return arbre

# Test
print("--- Pile ou face ---")
print(pile_ou_face(3))

##############################
## Question 2 ##

def une_seule_liste(liste):
    """ Transforme une liste qui contient des élément
    et des sous-listes (et même des sous-sous-liste...)
    et une liste d'éléments """
    reduc_liste = []
    for l in liste:
        if isinstance(l, int):
            reduc_liste = reduc_liste + [l]     # Cas terminal
        else:
            reduc_liste = reduc_liste + une_seule_liste(l)

    return reduc_liste

# Test
print("--- Une seule liste ---")
liste = [[1,2], 3, [9,8,7,[6,5],[4,3]], 2, [[1],[1]], 0]
print(liste)
print(une_seule_liste(liste))

##############################
## Question 3 ##

def atteindre_somme(S,liste):
    """ Comment atteindre la somme S en additionnant
    les élements de la liste donnée (les répétitions sont possibles) """

    # Cas terminaux
    if S == 0:
```

```
        return []
    if S < 0:
        return None

    # Cas général
    for x in liste:
        parcours = atteindre_somme(S-x,liste)
        if parcours != None:
            parcours = [x] + parcours
            return parcours

    return None

# Test
print("--- Atteindre une somme ---")
liste = [5, 7, 11]
somme = 19
print(liste)
print(somme)
parcours = atteindre_somme(somme,liste)
print("Parcours :",parcours)
if parcours:
    print("Vérification :",sum(parcours))
```

Activité 4

Activité 4 recursivite_4.py

```
################################
# Récursivité
################################

################################
# Activité 4 - Diviser pour régner
################################

################################
## Question 1 ##
def minimum(liste):
    """ Renvoie le minimum d'une liste par fonction récursive """
    if len(liste) == 0:
        return None
    if len(liste) == 1:
        M = liste[0]
        return M
    else:
        n = len(liste)
        liste_gauche = liste[:n//2]
        liste_droite = liste[n//2:]

        Mgauche = minimum(liste_gauche)
        Mdroite = minimum(liste_droite)
        if Mgauche < Mdroite:
            return Mgauche
        else:
            return Mdroite
```

```
# Test
print("--- Minimum ---")
liste = [7,5,3,9,1,12,13]
print("liste =",liste)
print("min = ",minimum(liste))


##############################
## Question 2 ##
def distance_hamming(liste1,liste2):
    """ Calcule le nb d'endroit où les deux listes diffèrent """
    if len(liste1) == 0:
        return 0
    if len(liste1) == 1:
        if liste1[0] == liste2[0]:
            return 0
        else:
            return 1
    # Cas général

    n = len(liste1)
    liste_gauche1 = liste1[:n//2]
    liste_droite1 = liste1[n//2:]

    liste_gauche2 = liste2[:n//2]
    liste_droite2 = liste2[n//2:]

    Hgauche = distance_hamming(liste_gauche1,liste_gauche2)
    Hdroite = distance_hamming(liste_droite1,liste_droite2)
    H = Hgauche + Hdroite
    return H

# Test
print("--- Distance de Hamming ---")
liste1 = [1,2,3,4,5,6,7]
liste2 = [1,2,0,4,5,0,7]
print("liste1 =",liste1)
print("liste2 =",liste2)

print("Distance =",distance_hamming(liste1,liste2))


##############################
## Question 3 ##
def produit(a,b):
    """ Produit des éléments : a(a+1)(a+2)...(b-2)(b-1) """
    if b==a:
        return 1
    if b==a+1:
        return a

    # Cas général
    k = b-a
    Pgauche =  produit(a,a+k//2)
    Pdroite = produit(a+k//2,b)
    P = Pgauche*Pdroite
    return P

# Test
print("--- Produit ---")
a, b = 5,10
print("a, b =",a, b)
```

```
print("produit = ",produit(a,b))

## Rappels ##

def factorielle(n):
    """ Factorielle par formule récursive classique """
    if n == 0:      # Cas terminal
        f = 1
    else:           # Cas général
        f = factorielle(n-1)*n
    return f

print("--- Factorielle ---")
n = 10
print("n =",n)
print("n! = ",produit(1,n+1))
print("n! = ",factorielle(n))
```

Activité 5

Activité 5 recursivite_5.py

```
################################
# Récursivité
################################

from math import *

################################
# Activité 5 - Dérangements
################################

################################
## Question 1 ##

def derangement_classique(n):
    """ Formule dérangement d_n par formule classique """
    d = 1
    for k in range(1,n+1):
        if k%2 == 0:
            epsilon = 1
        else:
            epsilon = -1
        d = d*k + epsilon
    return d

################################
## Question 2 ##

def derangement(n):
    """ Formule dérangement d_n par formule récursive """
    if n==0:
        d = 1
    else:
        if n%2 == 0:
            epsilon = 1
        else:
            epsilon = -1
```

```
        d = derangement(n-1)*n + epsilon
    return d

# Test
print("--- Dérangement ---")
n = 100
print("n =",n)
print("n! = ",derangement_classique(n))
print("n! = ",derangement(n))

##############################
## Rappels ##

def factorielle(n):
    """ Factorielle par formule récursive classique """
    if n == 0:      # Cas terminal
        f = 1
    else:           # Cas général
        f = factorielle(n-1)*n
    return f

##############################
## Question 3 ##

def quotient(n):
    """ Proportion des dérangements parmi les permutations """
    return derangement(n)/factorielle(n)

# Test
print("--- Limite ---")
n = 10
print("n =",n)
print("d(n)/n! = ",quotient(n))
print("1/e = ",1/exp(1))

##############################
## Question 4 ##

def est_derangement(permutation):
    """ Est-ce que la permutation donnée est un dérangement ? """
    n = len(permutation)
    der = True
    for i in range(n):
        if permutation[i] == i:
            der = False
    return der

# Test
print("--- Etre ou ne pas être un dérangement ---")
permutation = [2,1,0,3]
print("permutation =",permutation)
print("dérangement ? ",est_derangement(permutation))

permutation = [3,2,0,1]
print("permutation =",permutation)
print("dérangement ? ",est_derangement(permutation))

##############################
## Question 5 ##

# Idée 1 : liste des permutations par ajout de l'élément n à toutes les permutations de
    ↪ {0,...,n-1}
# avantage : un seule paramètre n
```

```
def toutes_permutations(n):
    """ Renvoie la liste de toutes les permutations de n éléments """
    if n == 1:
        return [[0]]    # ou bien n == 0 il faut [[]]

    old_liste = toutes_permutations(n-1)
    new_liste = []
    for old_permut in old_liste:
        for i in range(n):
            new_permut = old_permut[:i] + [n-1] + old_permut[i:]
            new_liste += [new_permut]
    return new_liste

# Test
print("--- Toutes les permutations  ---")
n = 3
permutations = toutes_permutations(n)
print("n =",n)
print(permutations)
print("Total =",len(permutations))
print("n! =",factorielle(n))

# Idées 2 : permutation de n objets, puis recursivité avec n-1 (autres) objets

def toutes_permutations_bis(liste):
    """ Renvoie la liste de toutes les permutations de n éléments """

    n = len(liste)
    if n == 1:
        permutations = [liste]
    else:
        permutations = []
        for i in range(n):
            element = liste[i]
            sous_liste =  liste[:i] + liste[i+1:]

            for permut in toutes_permutations_bis(sous_liste):
                nouv_perm = [element] + permut
                permutations += [nouv_perm]
    return permutations

# Test
print("--- Toutes les permutations (bis) ---")
liste = [0,1,2,3]
permutations = toutes_permutations_bis(liste)
n = len(liste)
print(liste)
print(permutations)
print("Total =",len(permutations))
print("n! =",factorielle(n))

##############################
## Question 6 ##

def tous_derangements(n):
    """ Renvoie la liste de toutes les dérangements de n éléments """
    derangements = []
    for permutation in toutes_permutations(n):
        if est_derangement(permutation):
            derangements += [permutation]
    return derangements

# Test
```

```
print("--- Tous les dérangements ---")
n = 3
print("n =",n)
liste = tous_derangements(n)
print(liste)
print("Total =",len(liste))
print("d(n) =",derangement(n))
```

Activité 6

Activité 6 recursivite_6.py

```
##############################
# Récursivité
##############################

##############################
# Activité 6 - Tortue récursive
##############################

##############################
## Question 1 ##
from turtle import *

def koch(l,n):
    """ Trace le flocon de Koch d'ordre n ('l' est une longueur) """
    if n==0:
        forward(l)
        return

    koch(l/3,n-1)
    left(60)
    koch(l/3,n-1)
    right(120)
    koch(l/3,n-1)
    left(60)
    koch(l/3,n-1)

    return

# Test
# speed('fastest')
# width(2)
# up()
# goto(-400,-200)
# down()
# koch(800,4)
# exitonclick()

##############################
## Question 2 ##

def arbre(l,n):
    """ Dessine un arbre d'ordre n ('l' est une longueur) """
    if n==0:
        return

    P = position()
```

```
    setheading(-120)
    forward(l)
    arbre(l/2,n-1)
    goto(P)
    setheading(-60)
    forward(l)
    arbre(l/2,n-1)
    goto(P)

    return

# Test
# speed('fastest')
# width(3)
# up()
# goto(0,300)
# down()
# arbre(300,6)
# exitonclick()


##############################
## Question 3 ##

def triangle(l,n):
    """ Dessine le triangle de Sierpinski d'ordre n ('l' est une longueur) """
    if n==0:
        return

    for __ in range(3):
        triangle(l/2,n-1)
        forward(l)
        left(120)

    return

# Test
speed('fastest')
hideturtle()
width(3)
up()
goto(-300,-250)
down()
triangle(600,6)
exitonclick()

##############################
## Question 4 ##

# Depuis la doc du module 'turtle'
def hilbert(angle,n):
    """ Dessine la courbe de Hilbert d'ordre n """
    if n == 0:
        return

    left(-angle)
    hilbert(-angle,n-1)
    forward(longueur)
    left(angle)
    hilbert(angle,n-1)
    forward(longueur)
    hilbert(angle,n-1)
    left(angle)
```

```
    forward(longueur)
    hilbert(-angle,n-1)
    left(-angle)

    return

from math import *

# Test
# speed('fastest')
# width(3)
# up()
# goto(-300,300)
# down()
# n = 4
# longueur = 40
# hilbert(90,n)
# exitonclick()

##############################
## Question 5 ##

def quart_cercle(pas):
    """ Trace un quart de cercle """
    for i in range(45):
        left(2)
        forward(pas)

# Test
# quart_cercle(1)
# exitonclick()

from random import *

def fractale_cercle(l,n):
    """ Dessine une fractale aléatoire à base de (quart de) cercles
    n est l'ordre de la fracatale, 'l' est un paramètre de longueur """
    if n == 0:
        return

    for __ in range(4):
        quart_cercle(l)

        hasard = randint(0,3)
        if hasard <= 2:
            fractale_cercle(l/3,n-1)

    return

# Test
up()
goto(0,-300)
down()
speed('fastest')
width(3)
l = 11
fractale_cercle(l,4)
exitonclick()
```

11. Tri – Complexité

Activité 1

Activité 1 complexite.py

```
##############################
# Tri - Compléxité
##############################

##############################
# Activité 1 - Notation grand O
##############################

from math import *

import matplotlib.pyplot as plt

def afficher_suite(u,couleur="red",N=10):
    liste_xy = [(n,u(n)) for n in range(N)]
    liste_un = [u(n) for n in range(N)]
    plt.plot(liste_un,color=couleur)
    for (x,y) in liste_xy:
        plt.scatter(x,y,color=couleur,s=15)

afficher_suite(lambda u:exp(u),couleur="brown",N=5)
afficher_suite(lambda u:u**2,couleur="red",N=6)
afficher_suite(lambda u:u,couleur="orange",N=26)
afficher_suite(lambda u:sqrt(u),couleur="green",N=41)
afficher_suite(lambda u:log(u+0.1),couleur="blue",N=41)

plt.axes().set_aspect('equal')
plt.xlim(xmin=0,xmax=40)
plt.ylim(ymin=0,ymax=25)
plt.grid()
# plt.show()

def est_grand_O(u,v,Nmin=10,Nmax=1000,k=10):
    ok = True
    for n in range(Nmin,Nmax):
        if u(n) > k*v(n):
            ok = False
    return ok

print("--- Est grand O ---")
u = lambda n:n**2
# def u(n): return n**2
v = lambda n:2**n
print("Est grand O :",est_grand_O(u,v))

u = lambda n:n
v = lambda n:sqrt(n)
print("Est grand O :",est_grand_O(u,v))

u = lambda n:1000*n**2
v = lambda n:0.001*exp(n)
print("Est grand O :",est_grand_O(u,v,Nmin=20,Nmax=40))
```

Activité 2 à 5

Activité 2 à 5 tri.py

```
##############################
# Tri
##############################

##############################
# Activité 1 - Tri par sélection
##############################

def tri_selection(liste):
    """ Ordonne la liste du plus petit au plus grande.
    Méthode : tri par sélection """
    cliste = list(liste)
    n = len(cliste)

    for i in range(n):
        rg_min = i
        for j in range(i+1,n):
            if cliste[j] < cliste[rg_min]:
                rg_min = j

        if rg_min != i:  # alors échange
            cliste[i], cliste[rg_min] = cliste[rg_min], cliste[i]

    return cliste

# Test
liste = [4,3,2,1,6,5]
print(liste)
print(tri_selection(liste))

##############################
# Activité 2 - Tri par insertion
##############################

def tri_insertion(liste):
    """ Ordonne la liste par la méthode du tri par insertion """
    cliste = list(liste)
    n = len(cliste)

    for i in range(1,n):
        el = cliste[i]
        j = i
        while (j>0) and (cliste[j-1]>el):
            cliste[j] = cliste[j-1]
            j = j-1

        cliste[j] = el

    return cliste

# Test
liste = [4,3,2,1]
print(liste)
print(tri_insertion(liste))

##############################
# Activité 3 - Tri à bulles
##############################

def tri_a_bulles(liste):
```

```
    """ Ordonne la liste par le tri à bulles """
    cliste = list(liste)
    n = len(cliste)

    for i in range(n-1,-1,-1):
        for j in range(i):
            if cliste[j+1] < cliste[j]:
                cliste[j], cliste[j+1] = cliste[j+1], cliste[j]

    return cliste

# Test
liste = [4,3,2,1]
print(liste)
print(tri_a_bulles(liste))


##############################
# Activité 4 - Tri fusion
##############################

# def fusion_recursive(liste_g,liste_d):
#     if len(liste_g)==0: return liste_d
#     if len(liste_d)==0: return liste_g
#     if liste_g[0] <= liste_d[0]:
#         liste_fus = [liste_g[0]] + fusion_recursive(liste_g[1:],liste_d)
#     else:
#         liste_fus = [liste_d[0]] + fusion_recursive(liste_g,liste_d[1:])
#     return liste_fus

# # Test
# print("--- Fusion récursive ---")
# liste_g = [1,4,7]
# liste_d = [2,5,6,9]
# print(liste_g,liste_d)
# print(fusion_recursive(liste_g,liste_d))

def fusion(liste_g,liste_d):
    """ Fusionne et ordonne deux listes déjà ordonnées """
    n, m = len(liste_g), len(liste_d)
    i, j = 0, 0
    liste_fus = []
    while (i<n) and (j<m):
        if liste_g[i] < liste_d[j]:
            liste_fus.append(liste_g[i])
            i = i +1
        else:
            liste_fus.append(liste_d[j])
            j = j +1
    while (i<n):
        liste_fus.append(liste_g[i])
        i = i +1
    while (j<m):
        liste_fus.append(liste_d[j])
        j = j +1

    return liste_fus

# Test
print("--- Fusion boucle ---")
liste_g = [1,4,7]
liste_d = [2,5,6,9]
```

```
print(liste_g,liste_d)
print(fusion(liste_g,liste_d))


def tri_fusion(liste):
    """ Ordonne une liste par la méthode du tri fusion.
    La fonction est récursive et utilse la fonction fusion() """
    cliste = list(liste)
    n = len(cliste)
    if n <= 1:
        return cliste
    else:
        liste_g = tri_fusion(cliste[:n//2])
        liste_d = tri_fusion(cliste[n//2:])
        liste_tri = fusion(liste_g,liste_d)
    return liste_tri

# Test
print("--- Tri fusion ---")
liste = [4,3,2,1]
print(liste)
print(tri_fusion(liste))
```

Activité 6

Activité 6 tri_avec_complexite.py

```
##############################
# Tri
##############################

##############################
# Activité 1 - Tri par sélection
##############################

def tri_selection_complexite(liste):
    """ Trie par sélection et renvoie le nb de comparaisons effectuées """
    cliste = list(liste)
    n = len(cliste)
    comp = 0

    for i in range(n):
        rg_min = i
        for j in range(i+1,n):
            if cliste[j] < cliste[rg_min]:
                rg_min = j
            comp = comp + 1

        if rg_min != i:  # alors échange
            cliste[i], cliste[rg_min] = cliste[rg_min], cliste[i]

    return cliste, comp

##############################
# Activité 2 - Tri par insertion
##############################

def tri_insertion_complexite(liste):
    """ Trie par insertion et renvoie le nb de comparaisons effectuées """
```

```
    cliste = list(liste)
    n = len(cliste)
    comp = 0

    for i in range(1,n):
        el = cliste[i]
        j = i
        while (j>0) and (cliste[j-1]>el):
            cliste[j] = cliste[j-1]
            j = j-1
            comp = comp + 1
        if j>0:
            comp = comp + 1 # +1 pour la comparaison qui sort du while

        cliste[j] = el

    return cliste, comp

##############################
# Activité 3 - Tri à bulles
##############################

def tri_a_bulles_complexite(liste):
    """ Tri à bulles et nb de comparaisons effectuées """

    cliste = list(liste)
    n = len(cliste)
    comp = 0

    for i in range(n-1,-1,-1):
        for j in range(i):
            if cliste[j+1] < cliste[j]:
                cliste[j], cliste[j+1] = cliste[j+1], cliste[j]
            comp = comp + 1

    return cliste, comp



##############################
# Activité 4 - Tri fusion
##############################

# def fusion_recursive(liste_g,liste_d):
#     if len(liste_g)==0: return liste_d
#     if len(liste_d)==0: return liste_g
#     if liste_g[0] <= liste_d[0]:
#         liste_fus = [liste_g[0]] + fusion_recursive(liste_g[1:],liste_d)
#     else:
#         liste_fus = [liste_d[0]] + fusion_recursive(liste_g,liste_d[1:])
#     return liste_fus

# # Test
# print("--- Fusion récursive ---")
# liste_g = [1,4,7]
# liste_d = [2,5,6,9]
# print(liste_g,liste_d)
# print(fusion_recursive(liste_g,liste_d))

def fusion_complexite(liste_g,liste_d):
    """ Fusion et compléxité de deux listes ordonnées """
    n, m = len(liste_g), len(liste_d)
    i, j = 0, 0
    liste_fus = []
```

```
    comp = 0
    while (i<n) and (j<m):
        if liste_g[i] < liste_d[j]:
            liste_fus.append(liste_g[i])
            i = i +1
        else:
            liste_fus.append(liste_d[j])
            j = j +1
        comp = comp + 1
    while (i<n):
        liste_fus.append(liste_g[i])
        i = i +1
    while (j<m):
        liste_fus.append(liste_d[j])
        j = j +1

    return liste_fus, comp

def tri_fusion_complexite(liste):
    """ Tri fusion et compléxité d'une liste """
    cliste = list(liste)
    n = len(cliste)
    comp = 0
    if n <= 1:
        return cliste, 0
    else:
        liste_g, comp_g = tri_fusion_complexite(cliste[:n//2])
        liste_d, comp_d = tri_fusion_complexite(cliste[n//2:])
        liste_tri, comp_f = fusion_complexite(liste_g,liste_d)
        comp = comp_g + comp_d + comp_f
    return liste_tri, comp

# Test
from random import *
from math import *

print("--- Complexité des algorithmes de tri ---")

# Liste aléatoire
liste = [randint(0,10000) for i in range(1000)]

# Liste déjà ordonnée
# liste = list(range(1000))

# Liste ordonnée dans le mauvais sens
# liste = list(reversed(range(1000)))

liste1,comp1 = tri_selection_complexite(liste)
liste2,comp2 = tri_insertion_complexite(liste)
liste3,comp3 = tri_a_bulles_complexite(liste)
liste4,comp4 = tri_fusion_complexite(liste)

print("  -- Complexités --")
if liste1 == liste2 and liste2==liste3 and liste3==liste4:
    print("Tri ok !")
else:
    print("Problème de tri :",liste1,liste2,liste3,liste4)

n = len(liste)
print("n^2 =",n**2/2,"   n*ln_2(n) =",n*log(n,2))
print(comp1,comp2,comp3,comp4)
```

12. Calculs en parallèle

Activité 1

Activité 1 parallele_1.py

```
##############################
# Calculs en parallèle
##############################

##############################
# Activité 1 - Somme, produit
##############################

from math import *

##############################
## Question 1 ##

def calcule_en_parallele(liste_instructions,N):
    """ Effectue une liste d'instructions données sous la forme de chaîne
    en simulant un calcul en parallèle avec N processeurs.
    Renvoie la liste des résultats, le nb de calculs et le temps des calculs """

    liste_resultats = []
    calculs = 0 # work
    temps = 0   # depth

    for instruction in liste_instructions:
        liste_resultats += [eval(instruction)]

    calculs += len(liste_instructions)
    temps += ceil(len(liste_instructions)/N)

    return liste_resultats, calculs, temps

# Test
print("--- Machine à calculs parallèles ---")
instructions = ['2+3','6*7','8-2','5+4','4*3','12//3']
print(instructions)
for N in range(1,7):
    print('Nombre de processeurs :',N)
    print(calcule_en_parallele(instructions,N))

##############################
## Question 2 ##

# Addition de deux vecteurs

def addition_vecteurs(v1,v2):
    """ Addition de deux vecteurs en utilisant le calcul en parallèle """
    n = len(v1)
    liste_instructions = []
    for i in range(n):
        instruction = str(v1[i]) + '+' + str(v2[i])
        liste_instructions += [instruction]
    liste_resultats, calculs, temps = calcule_en_parallele(liste_instructions,N)
    v = liste_resultats
    return v, calculs, temps

# Test
print("--- Addition de deux vecteurs ---")
```

```
N = 2  # Nombre de processeurs
v1 = [1,2,3,4]
v2 = [10,11,12,13]
v, calculs, temps = addition_vecteurs(v1,v2)
print('v1 =',v1)
print('v2 =',v2)
print('addition =',v)
print('Nombre de processeurs :',N)
print('calculs =',calculs,', temps =',temps)


##############################
## Question 3 ##

# Somme avec 2 processeurs

def somme(v):
    """ Somme des termes de la liste v en utilisant le calcul en parallèle """
    s = 0
    calculs_total = 0
    temps_total = 0

    while len(v)>1:
        k = len(v)//2
        liste_instructions = []
        for i in range(k):
            instruction = str(v[2*i]) + '+' + str(v[2*i+1])
            liste_instructions += [instruction]

        w, calculs, temps = calcule_en_parallele(liste_instructions,N)

        if len(v) % 2 != 0:   # si nombre impair de termes
            w += [v[-1]]

        v = w

        calculs_total += calculs
        temps_total += temps

    resultat = v[0]
    return resultat, calculs_total, temps_total

# Test
print('--- Somme ---')
N = 2  # Nombre de processeurs
v = [1,2,3,4,5,6,7,8]
resultat, calculs, temps = somme(v)
print("Liste :",v)
print("Somme :",resultat)
print('Nombre de processeurs :',N)
print('calculs =',calculs,', temps =',temps)

##############################
## Question 4 ##

def somme_recursive(v):
    """ Somme des termes de la liste v en utilisant une formule récursive
    et le calcul en parallèle """
    n = len(v)
    if n==0: return 0,0,0
    if n==1: return v[0],0,0
    v_gauche = v[:n//2]
    v_droite = v[n//2:]
    somme_gauche,calculs_gauche,temps_gauche = somme_recursive(v_gauche)
```

```
    somme_droite,calculs_droite,temps_droite = somme_recursive(v_droite)
    somme =  somme_gauche + somme_droite
    calculs = calculs_droite + calculs_droite + 1
    temps = max(temps_droite,temps_droite) + 1

    return somme,calculs,temps

# Test
print('--- Somme récursive---')
# On suppose de suffisement de processeurs : N est grand (N>=n//2)
v = [1,2,3,4,5,6,7,8]
resultat, calculs, temps = somme_recursive(v)
print("Liste :",v)
print("Somme :",resultat)
print('calculs =',calculs,', temps =',temps)

##############################
## Question 5 ##

def multiplication_vecteurs(v1,v2):
    """ Multiplication terme à terme de deux vecteurs """

    n = len(v1)
    liste_instructions = []
    for i in range(n):
        instruction = str(v1[i]) + '*' + str(v2[i])
        liste_instructions += [instruction]
    liste_resultats, calculs, temps = calcule_en_parallele(liste_instructions,N)
    v = liste_resultats
    return v, calculs, temps

def produit_scalaire(v1,v2):
    """ Produit scalaire de deux vecteurs """
    v, calculs1, temps1 = multiplication_vecteurs(v1,v2)
    S, calculs2, temps2 = somme(v)
    calculs = calculs1 + calculs2
    temps = temps1 + temps2
    return S, calculs, temps

# Test
print("--- Multiplication de deux vecteurs ---")
N = 8  # Nombre de processeurs
v1 = [1,2,3,4]*100
v2 = [2,3,4,5]*100
v, calculs, temps = multiplication_vecteurs(v1,v2)
print('v1 =',v1)
print('v2 =',v2)
print('multiplication =',v)
print('Nombre de processeurs :',N)
print('calculs =',calculs,', temps =',temps)

print("--- Produit scalaire ---")
produit, calculs, temps = produit_scalaire(v1,v2)
print('produit scalaire =',produit)
print('Nombre de processeurs :',N)
print('calculs =',calculs,', temps =',temps)


# Projet : multiplication de matrices
```

Activité 2

Activité 2 parallele_2.py

```
##############################
# Calculs en parallèle
##############################

##############################
# Activité 2 - Doublons
##############################

##############################
## Indéxation ##

# Première méthode par indexation

# Liste d'entiers de 0 à 99
liste = [59, 72, 8, 37, 37, 8, 21, 22, 37, 59]

def enlever_tous_doublons(liste):
    """ Ne garde qu'un exemplaire des éléments d'une liste """
    nouv_liste = []  # future liste sans doublons
    table = [0]*100  # liste de 100 zéros
    for i in liste:
        if table[i] == 0:  # si élément x pas déjà retenu
            table[i] = 1   # on le note
            nouv_liste += [i]  # et on le rajoute

    return nouv_liste

# Test
print("--- Doublons : indexation ---")
print(enlever_tous_doublons(liste))

##############################
## Hachage ##

# Seconde méthode : table de hachage

print("--- Doublons : hachage ---")

##############################
## Question 1 ##

ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
print(ALPHABET)

def hachage(mot,p):
    """ Une fonction de hachage d'un mot modulo p """
    hach = 0
    i = 0
    for c in mot:
        k = ALPHABET.index(c)
        hach = (hach + k*26**i) % p  # mieux avec pow(26,i,p) du module math
        i = i + 1
    return hach

# Test
print("--- Hachage ---")
p = 11
print('p =',p)
print(hachage('LAPIN',p))
print(hachage('CHAT',p))
```

```
print(hachage('CHIEN',p))
print(hachage('TORTUE',p))
print(hachage('SINGE',p))
print(hachage('',p))

##############################
## Question 2 ##

def enlever_des_doublons(liste,p):
    """ Retire certains doublons d'une liste en utilisant le hachage des mots modulo p """

    nouv_liste = []  # future liste avec moins de doublons
    table = ['']*p    # liste de p zéros
    for mot in liste:
        hach = hachage(mot,p)
        if table[hach] == '':  # si élément mot pas déjà retenu
            table[hach] = mot   # on le note au rang du hachage
            nouv_liste += [mot] # et on le rajoute
        else:
            if mot != table[hach]:   # dans le cas mot différent avec même hash
                nouv_liste += [mot]  # on le rajoute

    return nouv_liste

print("--- Doublons : une passe ---")
liste = ['AA','BB','CC','AA','LAPIN','CHAT','CHIEN','AA','BB','CC','AA','LAPIN','CHAT','CHIEN'
    ↪ ]
liste = ['LAPIN','CHAT','ZEBRE','CHAT','CHIEN','TORTUE','CHIEN','SINGE','SINGE','CHAT','CHAT',
    ↪ 'CHIEN']
liste = ['CHIEN','LAPIN','CHIEN','SINGE','SINGE']

print(liste)
print(enlever_des_doublons(liste,10))

##############################
## Question 3 ##

def iterer_enlever_des_doublons(liste,nb_iter=3):
    """ Itération pour enlever plus de doublons """
    p = 2*len(liste)
    for i in range(nb_iter):
        liste = enlever_des_doublons(liste,p)
        p = p+1
    return liste

print("--- Doublons : itération ---")
# liste = ['CHIEN','LAPIN','CHIEN','SINGE','SINGE']
liste = ['LAPIN','CHAT','ZEBRE','CHAT','CHIEN','TORTUE','CHIEN','SINGE','SINGE','CHAT','CHAT',
    ↪ 'CHIEN']
print(liste)
print(iterer_enlever_des_doublons(liste,2))
```

Activité 3

Activité 3 parallele_3.py

```
##############################
# Calculs en parallèle
##############################


##############################
# Activité 3 - Listes
##############################

##############################
## Question 1 ##

# Avec 2 processeurs

def maximum(liste):
    """ Maximum d'une liste par formule récursive en séparant la liste en 2 """
    infini = 1000
    if len(liste)==0: return -infini
    if len(liste)==1: return liste[0]
    k = len(liste)//2
    liste_gauche = liste[:k]
    liste_droite = liste[k:]
    max_gauche = maximum(liste_gauche)
    max_droite = maximum(liste_droite)
    maxi = max(max_gauche,max_droite)
    return maxi

# Test
print("--- Maximum ---")
liste = [6,3,8,10,4]
print(liste)
print(maximum(liste))

##############################
## Question 1bis ##

# Renvoie l'élément juste avant le maximum
# A faire

def sous_maximum(liste):
    pass

##############################
## Question 2 ##

def extraire_pairs(liste):
    """ Renvoie les éléments pairs d'une liste (fonction récursive) """
    if len(liste)==0: return []
    if len(liste)==1:
        if liste[0]%2 == 0:
            return [liste[0]]
        else:
            return []

    # On coupe la liste en 2
    k = len(liste)//2
    liste_gauche = liste[:k]
    liste_droite = liste[k:]
```

```
    liste_paire_gauche = extraire_pairs(liste_gauche)
    liste_paire_droite = extraire_pairs(liste_droite)
    liste_paire = liste_paire_gauche + liste_paire_droite
    return liste_paire

print("--- Extraire les termes pairs ---")
liste = [6,8,7,3,9,10,11,5,8,2]
print(liste)
print(extraire_pairs(liste))

##############################
## Question 3 ##

# Avec 2 processeurs

def premier_rang(liste):
    """ Renvoie le rang du premier élément non nul (fonction récursive) """
    if len(liste)==0: return None

    if len(liste)==1:
        if liste[0]==0:
            return None
        else:
            return 0

    k = len(liste)//2
    liste_gauche = liste[:k]
    liste_droite = liste[k:]
    premier_rang_gauche = premier_rang(liste_gauche)
    premier_rang_droite = premier_rang(liste_droite)
    if premier_rang_gauche != None:
        return premier_rang_gauche
    elif premier_rang_droite != None:
        return k + premier_rang_droite
    else:
        return None

# Test
print("--- Premier rang non nul ---")
liste = [0,0,0,0,0,1,0,1,1,0]
print(liste)
print(premier_rang(liste))
```

Activité 4

Activité 4 parallele_4.py

```
##############################
# Calculs en parallèle
##############################

##############################
# Activité 4 - Sommes partielles (prefix-sum)
##############################

##############################
## Question 1 ##

def sommes_partielles(liste):
```

```
    """ Liste des sommes partielles d'une liste """
    liste_sommes = []
    s = 0
    for x in liste:
        s = s + x
        liste_sommes += [s]
    return liste_sommes

# Test
print("--- Sommes partielles (prefix-sum) séquentiel ---")
liste = [1,2,3,4,5,6,7,8]
liste = [10,4,0,2,1,0,3,21]
print(liste)
print(sommes_partielles(liste))

##############################
## Question 2 ##

def sommes_partielles_recursif(liste):
    """ Liste des sommes partielles d'une liste (fonction récursive)
    C'esl l'algorithme 'prefix-sum'
    La liste doit être de longueur une puissance de 2 """

    n = len(liste)
    if n==1: return [liste[0]]

    sous_liste = [liste[2*i] + liste[2*i+1] for i in range(n//2)]
    # print(sous_liste)

    liste_remontee = sommes_partielles_recursif(sous_liste)

    print(liste_remontee)

    liste_descente = [liste[0]]
    for i in range(1,n):
        # print(liste,sous_liste,liste_remontee,i,i//2)
        if i%2 == 0:
            liste_descente += [liste_remontee[i//2-1] + liste[i]]
        else:
            liste_descente += [liste_remontee[(i-1)//2]]

    return liste_descente

# Test
print("--- Sommes partielles (prefix-sum) récursif ---")
liste = [1,2,3,4,5,6,7,8]
print(sommes_partielles_recursif(liste))
print(sommes_partielles(liste))


##############################
## Question 3 ##

def selection(liste,filtre):
    """ Application. Filtrage d'une liste en conservant l'ordre :
    on ne conserve que les éléments de la liste associé à 1 dans le filtre
    L'algorithme utilise le calcul des sommes partielles précédent et est donc
    ↪ parallèlisable """

    sommes = sommes_partielles_recursif(filtre)

    n =sommes[-1]  # Nombre d'éléments à garder
    selec = [0]*n  # Liste à remplir

    for i in range(len(liste)):  # peut se faire en parallèle
```

```
        if filtre[i] != 0:
            selec[sommes[i]-1] = liste[i]
    return selec

# Test
print("--- Rang non nul ---")
liste = [5,8,6,1,5,9,3,2]
filtre = [0,0,1,0,1,0,0,1]
# liste = [0,0,0,1,0,0,0,0,0,1,0,0,0,1,0,0]
print(liste)
print(filtre)
print(sommes_partielles_recursif(filtre))
print(selection(liste,filtre))
```

13. Automates

Activité 1

Activité 1 automate_1.py

```
##############################
# Automates
##############################

##############################
# Activité 1 - Une suite logique
##############################

# Programmation de la suite logique :
# 1
# 11
# 21
# 1211
# 111221
# 312211
# 13112221

# Une suite logique

def lecture(mot):
    """ Calcule la lecture du mot """

    carac_prec = ""
    nb = 0

    nouv_mot = ""

    for carac in mot:
        if (carac_prec == "") or (carac == carac_prec):
            nb = nb + 1
        else:
            nouv_mot = nouv_mot + str(nb) + carac_prec
            nb = 1

        carac_prec = carac

    # Fin
    nouv_mot = nouv_mot + str(nb) + carac_prec

    return nouv_mot
```

```
# Pour la répétition
def itere(n,mot):
    """ Répète n lectures """
    for __ in range(n):
        print(mot)
        mot = lecture(mot)
    return mot

# Test
print("--- Suite logique ---")
mot = "112"
print("Le mot suivant de",mot,"est", lecture(mot))
print("En partant de 1 on obtient :")
itere(12,"1")
```

Activité 2

Activité 2 automate_2.py

```
##############################
# Automates
##############################
##############################
# Activité 2 - Automate linéaire
##############################
##############################
## Question 1 ##

def cellule_suivante(a,b,c,regle):
    """ Calcule la couleur 0/1 de la cellule suivante selon
    les trois cellules a,b,c du dessus selon la règle donnée """
    rang = 4*a+2*b+c
    new_cellule = regle[rang]
    return new_cellule

# Test
print("--- Test affichage règle ---")
regle = [0, 0, 1, 0, 1, 1, 0, 1]
print(cellule_suivante(0,0,1,regle))

##############################
## Question 2 ##
def affiche_regle(regle):
    """ Affichage de la règle """
    for a in range(2):
        for b in range(2):
            for c in range(2):
                rang = 4*a+2*b+c
                cellule = regle[rang]
                print(a,b,c," -> ",cellule)
```

```
    return

# Test
print("--- Test affichage règle ---")
regle = [0, 0, 1, 0, 1, 1, 0, 1]
affiche_regle(regle)

##############################
## Question 3 ##

def ligne_suivante(ligne,regle):
    """ Calcule la ligne de cellules suivante """

    p = len(ligne)
    nouvelle_ligne = []

    for j in range(p):
        if j == 0:
            a,b,c = 0, ligne[j], ligne[j+1]
        elif j == p-1:
            a,b,c = ligne[j-1], ligne[j], 0
        else:
            a,b,c = ligne[j-1], ligne[j], ligne[j+1]

        cellule = cellule_suivante(a,b,c,regle)
        nouvelle_ligne = nouvelle_ligne + [cellule]

    return nouvelle_ligne

# Test
print("--- Test ligne suivante ---")

regle = [0, 0, 1, 0, 1, 1, 0, 1]
ligne = [0,0,1,0,1]
next_ligne = ligne_suivante(ligne,regle)
print(ligne)
print(next_ligne)

##############################
## Question 4 ##

def plusieurs_lignes(n,ligne,regle):
    """ Calcule n lignes suivantes """

    print(ligne)
    for __ in range(n):
        ligne = ligne_suivante(ligne,regle)
        print(ligne)
    return

# Test
print("--- Test plusieurs lignes suivantes ---")
regle = [0, 0, 1, 0, 1, 1, 0, 1]

print("Règle : ",regle)

ligne = [0]*5 + [1] + [0]*5
plusieurs_lignes(4,ligne,regle)

##############################
## Question 5 ##

def afficher_lignes(n,ligne,regle):
    """ Affichage de n lignes à partir
    d'une ligne initiale et d'une règle """
```

```
    p = len(ligne)
    for i in range(n):
        for j in range(p):
            if ligne[j] == 1:
                couleur = "black"
            else:
                couleur = "white"
            canvas.create_rectangle(10+j*echelle,10+i*echelle,10+(j+1)*echelle,10+(i+1)*
    ↪ echelle,fill=couleur,outline="red")
        ligne = ligne_suivante(ligne,regle)
    return

from tkinter import *

root = Tk()

canvas = Canvas(root, width=1000, height=800, background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

# Echelle
echelle = 30

# Automate
regle = [0, 0, 1, 0, 1, 1, 0, 1]  # Règle 45
print("Règle : ",regle)
affiche_regle(regle)
ligne = [0]*30 + [1] + [0]*30

# afficher_lignes(30,ligne,regle)

# root.mainloop()


##############################
## Question 7 ##

def definir_regle(numero):
    """ Définir une règle à partir d'un numéro entre 0 et 255 """

    regle = []
    for __ in range(8):
        r = numero % 2
        regle = [r] + regle
        numero = numero // 2
    return regle

# Test
print("--- Numérotation des règles ---")
numero = 45
regle = definir_regle(numero)
print("Numéro :",numero)
print("Règle : ",regle)
affiche_regle(regle)

##############################
def conversion_standard_vers_perso(numero):

    liste = definir_regle(numero)
    print(liste)
    perso = sum([2**i * liste[i] for i in range(8)])
    return perso

# Test
print("--- Conversion numérotation standard ---")
```

```
std = 106
numero = conversion_standard_vers_perso(std)
print(std,"->", numero)
print(definir_regle(numero))

##############################
## Question 7 ##

# Chaotique
numero = 120  # std = 30
numero = 106  # std = 86

# Sierpinski
# numero = 90  # std = 90

# Deux voyageurs
# numero = 42  # std = 84

# Pli
# numero = 136  # std = 17

# Monstre qui fait peur
# numero = 150 # std = 105

# Sierpinski étiré ?
# numero = 135 # std = 225

# Pyramide
numero = 92  # std = ?

# Coulée
numero = 98  # std = ?

# Exemple du cours
numero = 45

numero = 105

regle = definir_regle(numero)

ligne = [0]*15 + [1] + [0]*15

afficher_lignes(40,ligne,regle)

root.mainloop()
```

14. Cryptographie

Activité 1

Activité 1 cryptographie_1.py

```
##############################
# Cryptographie
##############################

##############################
# Activité 1 - Le chiffre de César
##############################

Alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

##############################
```

```
## Question 1 ##

def chiffre_cesar_caractere(car,k):
    """ Calcule le décalage de k lettres """

    i = Alphabet.index(car)
    j = (i+k) % 26
    car_code = Alphabet[j]
    return car_code

# Test
print("--- Codage d'un caractère ---")
print(chiffre_cesar_caractere('A',3))

##############################
## Question 2 ##

def chiffre_cesar_phrase(phrase,k):
    """" Chiffre une phrase par un décalage de k lettres """

    phrase_codee = ""
    for car in phrase:
        if car not in Alphabet:
            phrase_codee += car
        else:
            car_code = chiffre_cesar_caractere(car,k)
            phrase_codee += car_code
    return phrase_codee

# Test
print("--- Chiffre de César ---")
phrase = "CAPTUREZ IDEFIX !"
phrase_codee = chiffre_cesar_phrase(phrase,3)
print(phrase)
print(phrase_codee)

# Pour le cours
phrase = "BLOQUEZ ASTERIX"
phrase_codee = chiffre_cesar_phrase(phrase,3)
print(phrase)
print(phrase_codee)

phrase = "OU EST PANORAMIX"
phrase_codee = chiffre_cesar_phrase(phrase,10)
print(phrase)
print(phrase_codee)


##############################
## Question 3 ##

def dechiffre_cesar_phrase(phrase,k):
    """" Déchiffre une phrase par un décalage de k lettres à rebours """
    return chiffre_cesar_phrase(phrase,-k)

# Test
print("--- Déchiffrement de César ---")

phrase_decodee = dechiffre_cesar_phrase(phrase_codee,3)
print(phrase_codee)
print(phrase_decodee)

##############################
## Question 4 ##
```

```
def attaque_cesar(phrase):
    """ Affichage de toutes les possibilités """

    for k in range(26):
        print(k,chiffre_cesar_phrase(phrase,-k))

    return

print("--- Attaque de César ---")
phrase = "PANORAMIX N A PLUS DE POTION"
phrase_codee = chiffre_cesar_phrase(phrase,18)
print(phrase_codee)
print("Attaque :")
attaque_cesar(phrase_codee)
```

Activité 2

Activité 2 cryptographie_2.py

```
################################
# Cryptographie
################################

################################
# Activité 2 - Substitution mono-alphabétique
################################

Alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

# Génération d'un mélange aléatoire
# from random import *
# liste_lettre = list(Alphabet.lower())
# shuffle(liste_lettre)
# Melange = "".join(liste_lettre)

print("--- Alphabet mélangé ---")
Melange = "ykcodmfjgzaxrnbutqiphwesvl"
print(Melange)

################################
## Question 1 ##

def chiffre_substitution_caractere(car):
    """ Calcule la substitution d'une lettre """

    i = Alphabet.index(car)
    car_code = Melange[i]
    return car_code

# Test
print("--- Substitution d'un caractère ---")
print(chiffre_substitution_caractere('A'))

################################
## Question 2 ##

def chiffre_substitution_phrase(phrase):
    """" Chiffre une phrase par substitution """

    phrase_codee = ""
```

```
    for car in phrase:
        if car not in Alphabet:
            phrase_codee += car
        else:
            car_code = chiffre_substitution_caractere(car)
            phrase_codee += car_code
    return phrase_codee

# Test
print("--- Chiffrement par substitution ---")
phrase = "PAS DE POTION POUR OBELIX !"
phrase_codee = chiffre_substitution_phrase(phrase)
print(phrase)
print(phrase_codee)

# Cours
phrase = "BONJOUR"
phrase_codee = chiffre_substitution_phrase(phrase)
print(phrase)
print(phrase_codee)

phrase = "AVE CESAR"
phrase_codee = chiffre_substitution_phrase(phrase)
print(phrase)
print(phrase_codee)

##############################
## Question 3 ##

def dechiffre_substitution_caractere(car):
    """ Calcule la substitution inverse d'une lettre """

    i = Melange.index(car)
    car_decode = Alphabet[i]
    return car_decode

def dechiffre_substitution_phrase(phrase):
    """" Déchiffre une phrase par substitution """

    phrase_decodee = ""
    for car in phrase:
        if car not in Melange:
            phrase_decodee += car
        else:
            car_decode = dechiffre_substitution_caractere(car)
            phrase_decodee += car_decode
    return phrase_decodee

# Test
print("--- Déchiffrement d'une phrase codée par substitution ---")

phrase_decodee = dechiffre_substitution_phrase(phrase_codee)
print(phrase_codee)
print(phrase_decodee)

exit()

##############################
## Question 4 ##

def attaque_cesar(phrase):
    """ Affichage de toutes les possibilités """

    for k in range(26):
```

```
        print(k,chiffre_cesar_phrase(phrase,-k))

    return

print("--- Attaque de César ---")
phrase = "PANORAMIX N A PLUS DE POTION"
phrase_codee = chiffre_cesar_phrase(phrase,18)
print(phrase_codee)
print("Attaque :")
attaque_cesar(phrase_codee)
```

Activité 3

Activité 3 cryptographie_3.py

```
################################
# Cryptographie
################################

################################
# Activité 2 - Attaque du chiffrement par substitution
################################

Alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

# Génération d'un mélange aléatoire
from random import *
liste_lettre = list(Alphabet.lower())
shuffle(liste_lettre)
Melange = "".join(liste_lettre)

print("--- Alphabet mélangé ---")
Alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
Melange  = "bzryjxscwqiveguopldmtanfhk"
print(Melange)

################################
## Question 1 ##

def substitution(phrase,Alphabet_depart,Alphabet_arrivee):
    """" Transforme une phrase par substitution """

    phrase_subs = ""
    for car in phrase:
        if car not in Alphabet_depart:
            phrase_subs += car
        else:
            i = Alphabet_depart.index(car)
            car_subs = Alphabet_arrivee[i]
            phrase_subs += car_subs
    return phrase_subs

# Test
print("--- Substitution avec alphabet partiel ---")

Alphabet_depart  = "abcdefghijklmnopqrstuvwxyz"
Alphabet_arrivee = "...S.....E..T............."

phrase = "ESPRIT ES TU LA ?"
```

```
phrase_codee = substitution(phrase,Alphabet,Melange)
phrase_decodee = substitution(phrase_codee,Alphabet_depart,Alphabet_arrivee)
print(phrase)
print(phrase_codee)
print(phrase_decodee)

##############################
## Question 2 ##

Minuscules = "abcdefghijklmnopqrstuvwxyz"

def statistique(phrase,mon_alphabet=Minuscules):
    """ Nb d'apparition pour chaque lettre """

    stat = [0]*26
    for car in phrase:
        if car in mon_alphabet:
            i = mon_alphabet.index(car)
            stat[i] += 1

    return stat

def affiche_statistiques(phrase,mon_alphabet=Minuscules):
    print("-- Statistique --")
    print(phrase)
    stat = statistique(phrase,mon_alphabet)
    for i in range(26):
        if stat[i] != 0:
            print(mon_alphabet[i],stat[i])
    return

# Test
print("--- Apparition des lettres ---")
phrase = "jdolwm jd mt vb ?"
stat = statistique(phrase)
print(phrase)
affiche_statistiques(phrase)


##############################
## Question 3 ##

def frequence(phrase,mon_alphabet=Minuscules):
    """ Fréquence de chaque lettre """

    stat = [0]*26
    nb = 0
    for car in phrase:
        if car in mon_alphabet:
            i = mon_alphabet.index(car)
            stat[i] += 1
            nb += 1

    freq = [s/nb for s in stat]

    return freq

def affiche_frequences(phrase,mon_alphabet=Minuscules):
    print("-- Fréquence --")
    print(phrase)
    freq = frequence(phrase,mon_alphabet)
    for i in range(26):
        if freq[i] != 0:
            print(mon_alphabet[i],'{0:.2f}'.format(freq[i]*100),"%")
```

```
    return

# Test
print("--- Apparition des lettres ---")
phrase = "jdolwm jd mt vb ?"
freq = frequence(phrase)
print(phrase)
affiche_frequences(phrase)

# Cours
print("--- Apparition des lettres ---")
phrase = "ESPRIT ES TU LA ?"
freq = frequence(phrase,Alphabet)
print(phrase)
affiche_frequences(phrase,Alphabet)


# Cours
print("--- Exemple cours ---")
phrase = "ET SI ON ESSAYAIT D ETRE HEUREUSES"

affiche_statistiques(phrase,Alphabet)
freq = frequence(phrase,Alphabet)
affiche_frequences(phrase,Alphabet)

Melange = "bzryjxscwqiveguopldmtanfhk"
phrase_codee = substitution(phrase,Alphabet,Melange)
print(phrase_codee)

affiche_statistiques(phrase_codee)
freq = frequence(phrase_codee)
affiche_frequences(phrase_codee)

##############################
## Question 4 ##

from random import *
liste_lettre = list(Alphabet.lower())
shuffle(liste_lettre)
Melange = "".join(liste_lettre)
print(Melange)

##############################

Melange1 = "ybzfvotenupajgisxhdmlwqckr"
# # Les frères de Goncourt
print("--- Les frères Goncourt :")
phrase1 = "LA STATISTIQUE EST LA PREMIERE DES SCIENCES INEXACTES"
# affiche_frequences(phrase1,Alphabet)
enigme1 = substitution(phrase1,Alphabet,Melange1)
print(enigme1)
# # affiche_frequences(enigme1,Minuscules)

##############################

Melange2 = "oufkplzivxqamnbwtjygdsechr"
# Charles Darwin
print("--- Charles Darwin :")
phrase2 = "LES ESPECES QUI SURVIVENT NE SONT PAS LES ESPECES LES PLUS FORTES NI LES PLUS
    ↪ INTELLIGENTES MAIS CELLES QUI S ADAPTENT LE MIEUX AUX CHANGEMENTS"
# affiche_frequences(phrase2,Alphabet)
enigme2 = substitution(phrase2,Alphabet,Melange2)
print(enigme2)
```

```
# # affiche_frequences(enigme2,Minuscules)

##############################

Melange3 = "whixnabjctdkepzolfmyqsrvug"
# Albert Einstein
print("--- Albert Einstein :")
phrase3 = "LA THEORIE, C EST QUAND ON SAIT TOUT ET QUE RIEN NE FONCTIONNE. LA PRATIQUE, C
    ↪ EST QUAND TOUT FONCTIONNE ET QUE PERSONNE NE SAIT POURQUOI. ICI, NOUS AVONS REUNI
    ↪ THEORIE ET PRATIQUE : RIEN NE FONCTIONNE ET PERSONNE NE SAIT POURQUOI !"
# affiche_frequences(phrase3,Alphabet)
enigme3 = substitution(phrase3,Alphabet,Melange3)
print(enigme3)
# affiche_frequences(enigme3,Minuscules)
```

Activité 4

Activité 4 cryptographie_4.py

```
##############################
# Cryptographie
##############################

##############################
# Activité 4 - Le chiffrement de Vigenère
##############################

Alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

##############################
## Question 1 ##

def chiffre_vigenere(phrase,cle):
    """" Chiffre une phrase par un décalage de Vigenère """

    phrase_codee = ""
    rang = 0
    n = len(cle)
    for car in phrase:
        if car not in Alphabet:
            phrase_codee += car
        else:
            i = Alphabet.index(car)
            j = (i+cle[rang]) % 26
            car_code = Alphabet[j]
            phrase_codee += car_code
            rang = (rang + 1) % n
    return phrase_codee

# Test
print("--- Chiffrement de Vigenère ---")
phrase = "AAA ABC"
phrase_codee = chiffre_vigenere(phrase,[1,2,3])
print(phrase)
print(phrase_codee)

# Cours
phrase = "IL ETAIT UNE FOIS"
phrase = "ILE TAI TUN EFO IS"
```

```
phrase_codee = chiffre_vigenere(phrase,[4,2,3])
print(phrase)
print(phrase_codee)

# Cours
phrase = "MILOU ET TINTIN"
phrase_codee = chiffre_vigenere(phrase,[4,2,3])
print(phrase)
print(phrase_codee)

##############################
## Question 2 ##

phrase = "LE PROBLEME EN CE BAS MONDE EST QUE LES IMBECILES SONT SURS ET FIERS D EUX ALORS
    ↪ QUE LES GENS INTELLIGENTS SONT EMPLIS DE DOUTES"
cle = [18,7,10,16]
phrase_codee = chiffre_vigenere(phrase,cle)
print(phrase)
print(phrase_codee)
```

15. Le compte est bon

Activité 1 à 4

Activité 1 à 4 chiffres.py

```
##############################
# Des chiffres et des lettres
##############################

##############################
# Activité 1 - Tirages au hasard
##############################

from random import *

##############################
## Question 1 ##

def tirage_total(Nmin=100,Nmax=999):
    return randint(Nmin,Nmax)

# Test
print("--- Nombre à obtenir ---")
total = tirage_total()
print(total)

##############################
## Question 2 ##

def tirage_chiffres(nb_chiffres=6):

    CHIFFRES = [1,2,3,4,5,6,7,8,9,10,1,2,3,4,5,6,7,8,9,10,25,50,75,100]

    # Sans remise
    liste_choix = [c for c in CHIFFRES]
    liste = []
    for i in range(nb_chiffres):
        n = len(liste_choix)
        k = randint(0,n-1)
```

```
        liste += [liste_choix[k]]
        del liste_choix[k]
    return liste.sort()

# Test
print("--- Chiffres à jouer ---")
chiffres = tirage_chiffres()
print(chiffres)

##############################
## Question 3 ##
def operation(a,b,op):
    if op == '+':
        calcul = a + b
    elif op == '-':
        calcul = a - b
    elif op == '*':
        calcul = a * b
    elif op == '/':
        calcul = a / b
    return calcul

##############################
# Activité 2 -  Recherche basique
##############################

##############################
## Pour le cours ##

print("--- Pour le cours ---")

chiffres = [2,3,5,6,8,10]           # Plaques disponibles
total = 18                          # Objectif

chiffres1 = list(chiffres)          # Copie
for c1 in chiffres1:                # Premier chiffre
    chiffres2 = list(chiffres1)     # Copie
    chiffres2.remove(c1)            # Retirer la plaque déjà tirée

    for op in ['+','*']:           # Opération

        for c2 in chiffres2:        # Second chiffre

            calcul = operation(c1,c2,op)   # Résultat du calcul

            if calcul == total:                 # Total atteint ?
                print("Trouvé !",c1,op,c2,'=',calcul)

print("--- Fin pour le cours ---")

##############################
# Pour le cours
## Activité ##

def recherche_basique(total,chiffres):

  OPERATIONS = ['+','-','*','/']

  chiffres1 = list(chiffres)

  for c1 in chiffres1:
    if c1 == total:
      print("Trouvé !",c1)

    chiffres2 = list(chiffres1)
    chiffres2.remove(c1)
```

```
    for op1 in OPERATIONS:

      for c2 in chiffres2:
        calcul1 = operation(c1,c2,op1)

        if calcul1 == total:
          print("Trouvé !",c1,op1,c2,'=',total)

        chiffres3 = list(chiffres2)
        chiffres3.remove(c2)

        for op2 in OPERATIONS:

          for c3 in chiffres3:
            calcul2 = operation(calcul1,c3,op2)

            if calcul2 == total:
              print("Trouvé !",
                    c1,op1,c2,'=',calcul1,",",
                    calcul1,op2,c3,'=',total)

            chiffres4 = list(chiffres3)
            chiffres4.remove(c3)

            for op3 in OPERATIONS:

              for c4 in chiffres4:
                calcul3 = operation(calcul2,c4,op3)

                if calcul3 == total:
                  print("Trouvé !",
                        c1,op1,c2,'=',calcul1,",",
                        calcul1,op2,c3,'=',calcul2,",",
                        calcul2,op3,c4,'=',total)

                chiffres5 = list(chiffres4)
                chiffres5.remove(c4)

                for op4 in OPERATIONS:

                  for c5 in chiffres5:
                    calcul4 = operation(calcul3,c5,op4)

                    if calcul4 == total:
                      print("Trouvé",
                            c1,op1,c2,'=',calcul1,",",
                            calcul1,op2,c3,'=',calcul2,",",
                            calcul2,op3,c4,'=',calcul3,",",
                            calcul3,op4,c5,'=',total),",",

                    chiffres6 = list(chiffres5)
                    chiffres6.remove(c5)

                    for op5 in OPERATIONS:

                      for c6 in chiffres6:
                        calcul5 = operation(calcul4,c6,op5)

                        if calcul5 == total:
                          print("Trouvé !",
                                c1,op1,c2,'=',calcul1,",",
                                calcul1,op2,c3,'=',calcul2,",",
                                calcul2,op3,c4,'=',calcul3,",",
                                calcul3,op4,c5,'=',calcul4,",",
                                calcul4,op5,c6,'=',total)

  return
# Test
```

```
print("--- Recherche basique ---")
print(" -- Test 809 ---")
# recherche_basique(809,[2, 3, 6, 8, 75, 100])
print(" -- Test 779 ---")
# recherche_basique(779,[5, 6, 7, 8, 25, 50])

print(" -- Test 773 ---")
# recherche_basique(773,[2, 4, 6, 8, 10, 50])
print(" -- Test 769 ---")
# recherche_basique(769,[2, 4, 6, 8, 10, 50])
print(" -- Test 752 ---")
recherche_basique(756,[2, 4, 6, 8, 10, 50])

##############################
# Activité 3 - Recherche d'une somme
##############################

def atteindre_somme(S,chiffres):
    # Cas terminaux
    if S == 0:
        return []
    if S < 0:
        return None

    # Cas général
    for x in chiffres:
        parcours = atteindre_somme(S-x,chiffres)
        if parcours != None:
            parcours = [x] + parcours
            return parcours

    return None

# Test
print("--- Atteindre une somme ---")
chiffres = [5, 7, 11]
somme = 19
print(chiffres)
print(somme)
parcours = atteindre_somme(somme,chiffres)
print("Parcours :",parcours)
if parcours:
    print("Vérification :",sum(parcours))


##############################
# Activité 4 - Le compte est bon (recherche récursive).
##############################

def compte_est_bon(total,chiffres):
    # Cas terminaux
    if total in chiffres:
        return [str(total)]
    if len(chiffres) < 2:
        return None

    # Cas général
    chiffres.sort()    # Tri de la liste...
    chiffres.reverse() # ...du plus grand au plus petit

    n = len(chiffres)
    for i in range(n-1):
        for j in range(i+1,n):
            c1 = chiffres[i]
```

```
            c2 = chiffres[j]
            # print(c1,c2)

            for op in ['+','-','*','/']:
                nouv_chiffres = [cc for cc in chiffres]
                nouv_chiffres.remove(c1)
                nouv_chiffres.remove(c2)

                calcul_str =  str(c1) + op + str(c2)
                calcul = operation(c1,c2,op)

                if (calcul>0) and isinstance(calcul,int):  # Evite la division par zéro ou
    ↪ div. non entière
                    nouv_chiffres += [calcul]

                    parcours = compte_est_bon(total,nouv_chiffres)
                    if parcours != None:
                        operation_str = calcul_str + '=' + str(eval(calcul_str))
                        nouv_parcours = [operation_str] + parcours
                        return nouv_parcours

    return None

# Test
print("--- Le compte est bon ---")
chiffres = [3,5,7,2]
total = 72
print("Chiffres :",chiffres)
print("Total :",total)
parcours = compte_est_bon(total,chiffres)
print("Parcours :",parcours)

# Test
print("--- Tests réel ---")
# Exemple facile
print("  -- Facile --")
chiffres = [2, 3, 7, 9, 9, 25]
total = 457
print("Chiffres :",chiffres)
print("Total :",total)
parcours = compte_est_bon(total,chiffres)
print("Parcours :",parcours)

# Exemple long
print("  -- Impossible --")
# chiffres = [2, 4, 5, 6, 8, 10]
# total = 851
print("Chiffres :",chiffres)
print("Total :",total)
parcours = compte_est_bon(total,chiffres)
print("Parcours :",parcours)

# Exemple très long (pas possible)
chiffres = [1, 6, 4, 1, 8, 4]
total = 970

# Recrhche de totaux réalisable à partir d'une liste de chiffres fixés
# chiffres = [2, 4, 5, 6, 8, 10]  # 114 totaux pas trouvés
# chiffres = [1, 2, 5, 7, 75, 100]  # 0 totaux pas atteints
# chiffres = [10, 10, 25, 50, 75, 100] # beaucoup pas atteints
chiffres = [2, 4, 6, 8, 10, 50]
print("Chiffres :",chiffres)
```

```
nb_echec = 0
for total in range(750,760):
    parcours = compte_est_bon(total,chiffres)
    if parcours == None:
        print("Total pas atteint :",total)
        nb_echec +=1
print("Nombre de totaux pas atteints",nb_echec)
```

16. Le mot le plus long

Activité 1 à 4

Activité 1 à 4 lettres.py

```
##############################
# Des chiffres et des lettres
##############################

from random import *

# Nomenclature :
# * repertoire : liste de mots
# * indice : les lettres ordonnées du mot. ex : indice("BAC") = "ABC"
# * dictionnaire et clé : seulement pour notion Python
# * éviter à tout prix la confusion dictionnaire (python) et dictionnaire (liste de mots)

# On peut travailler avec le répertoire simple (20239 mots) ou le répertoire complet (131896
    ↪ mots)

##############################
# Activité 1 - Chercher un mot dans un répertoire
##############################

##############################
## Question 1 ##

def lire_repertoire(fichier):
    """ Lire un des deux fichiers qui contient tous les mots français
    et le transformer en une liste """

    # Fichier à lire
    fic_in = open(fichier,"r")

    repertoire = []
    for ligne in fic_in:
        mot = ligne.strip()
        repertoire.append(mot)

    fic_in.close()

    return repertoire

# Test
print("--- Liste obtenue par lecture d'un répertoire depuis un fichier ---")
repertoire = lire_repertoire("repertoire_francais_simple.txt")
# repertoire = lire_repertoire("repertoire_francais_tout.txt")

print("Nombre de mots dans le répertoire :",len(repertoire))

##############################
```

```
## Question 2 ##

def recherche_basique(mot,liste):
    """ Recherche séquentielle d'un mot dans une liste """
    i = 0
    while i<len(liste):
        if mot == liste[i]:
            return i
        i = i + 1

    return None

# Test
print("--- Recherche d'un mot dans le répertoire ---")
mot = "SERPENT"
trouve = recherche_basique(mot,repertoire)
print(mot,"dans répertoire ?",trouve)
mot = "PYTHON"
trouve = recherche_basique(mot,repertoire)
print(mot,"dans répertoire ?",trouve)

##############################
## Question 3 ##

# On profite du fait que la liste soit ordonnée
# Python connaît l'ordre alphabétique

def recherche_dichotomie(mot,liste):
    """ Recherche d'un mot dans une liste ordonnée par dichotomie """

    a = 0
    b = len(liste)-1

    while b>=a:
        k = (a+b)//2
        if mot == liste[k]:
            return k
        elif mot > liste[k]:
            a = k+1
        else:
            b = k-1

    return None

# Test
print("--- Recherche dichotomique dans le répertoire ---")
mot = "SERPENT"
trouve = recherche_dichotomie(mot,repertoire)
print(mot,"dans répertoire ?",trouve)
mot = "PYTHON"
trouve = recherche_dichotomie(mot,repertoire)
print(mot,"dans répertoire ?",trouve)

# Test
from math import log,floor
print("--- Comparaison séquentielle/dichotomie ---")
print("  -- Répertoire simple --")
repertoire = lire_repertoire("repertoire_francais_simple.txt")
n = len(repertoire)
print("n =",n,"log_2 = ",floor(log(n,2)+1))

print("  -- Répertoire simple --")
repertoire = lire_repertoire("repertoire_francais_tout.txt")
n = len(repertoire)
```

```
print("n =",n,"log_2 = ",floor(log(n,2)+1))

##############################
# Activité 2 - Dictionnaire en Python
##############################

# à faire ....


##############################
# Activité 3 - Dictionnaire d'anagrammes
##############################

##############################
## Question 1 ##

def calculer_indice(mot):
    """ Réordonne les lettres d'un mot par ordre alphabétique : c'est l'indice du mot """
    liste = list(mot)
    liste.sort()
    indice = "".join(liste)
    return indice

# Test
print("--- Indice d'un mot ---")
mot = "KAYAK"
indice = calculer_indice(mot)

print("mot =",mot)
print("indice =",indice)

##############################
## Question 2 ##

def sont_anagrammes(mot1,mot2):
    """ Test si deux mots sont annagrammes
    C'est exactemment tester si ils ont le même indice """
    indice1 = calculer_indice(mot1)
    indice2 = calculer_indice(mot2)
    if indice1 == indice2:
        return True
    else:
        return False

# Test
print("--- Test anagramme ---")
mot1,mot2 = "PRIERES", "RESPIRE"
print("Les mots ",mot1," et ",mot2)
print("Sont anagrammes ?",sont_anagrammes(mot1,mot2))

##############################
## Question 3 ##

def dictionnaire_indices_mots(liste):
    """ Cosntruit le dictionnaire clé -> forme de la forme
    indice -> valeur=[mot1,mot2,...] """
    dico = {}
    for mot in liste:
        indice = calculer_indice(mot)
        if indice in dico:
            dico[indice].append(mot)
        else:
            dico[indice] = [mot]
```

```
    return dico

# Test
print("--- Dico des indice d'une liste de mots ---")
liste = ["CRIME","COUCOU","PRIERES","MERCI","RESPIRE","REPRISE"]
dico = dictionnaire_indices_mots(liste)
print("liste =",liste)
print("dico =",dico)

##############################
## Question 4 ##

def liste_anagrammes(liste):
    """ Trouve tous les annagrammes d'une liste """
    dico = dictionnaire_indices_mots(liste)
    liste_anag = []
    for cle in dico:
        if len(dico[cle])>1:
            liste_anag.append(dico[cle])

    return liste_anag

def afficher_anagrammes(liste):
    """ Affiche tous les annagrammes d'une liste """
    liste_anag = liste_anagrammes(liste)
    print("Nombre d'anagrammes :",len(liste_anag))
    for anag in liste_anag:
        for mot in anag:
            print(mot,end=" ")
        print()
    return

# Test
print("--- Anagrammes français ---")
repertoire = lire_repertoire("repertoire_francais_simple.txt")
# repertoire = lire_repertoire("repertoire_francais_tout.txt")
print("Nombre d'anagrammes :",len(liste_anagrammes(repertoire)))
# afficher_anagrammes(repertoire)

##############################
## Question 5 ##

def fichier_indice_mots(fichier_in,fichier_out):
    """ Ecris le fichier associant à chaque indice la liste de mots correspondant.
    Chaque ligne est de la forme : "indice : mot1 mot2 mot3" """
    liste = lire_repertoire(fichier_in)
    dico = dictionnaire_indices_mots(liste)

    fic_out = open(fichier_out,"w")

    for cle,valeur in sorted(dico.items()):
            fic_out.write(cle+" : ")
            for mot in valeur:
                fic_out.write(mot+" ")
            fic_out.write("\n")

    fic_out.close()

    return

# Test
print("--- Conversion d'un répertoire vers liste de 'indice:mot1 mot2 mot3'  ---")
fichier_indice_mots("repertoire_francais_simple.txt","dico_indice_mots.txt")
# fichier_indice_mots("repertoire_francais_tout.txt","dico_indice_mots.txt")
```

```
##############################
# Activité 3 - Le mot le plus long
##############################

##############################
## Question 1 ##
def tirage_lettres(n):
    """ Tirage aléatoire de n lettres majuscules
    (certaines lettres ont plus de chances d'tres tirées que d'autres) """
    Alphabet = "AEIOU"*5 + "BCDFGHLMNPRST"*2 + "KJQVWXYZ"
    print(Alphabet)
    print(len(Alphabet))
    tirage = []
    for __ in range(n):
        k = randint(0,len(Alphabet)-1)
        tirage.append(Alphabet[k])
    return tirage

# Test
print("--- Tirage de lettres ---")
n = 4
tirage = tirage_lettres(n)
print("Tirage :",tirage)

##############################
## Question 2 ##
def liste_binaire(n):
    """ Liste des écritures binaires de tous les entiers sur n bits """
    liste = []
    for k in range(2**n):
        b_str = list(format(k, '0'+str(n)+'b'))
        b_str.reverse()
        b = [int(bs) for bs in b_str]
        liste.append(b)
    return liste

# Test
print("--- Listes binaires possibles ---")
n = 4
liste = liste_binaire(n)
print("n =",n)
print(liste)

##############################
## Question 3 ##
def indices_depuis_tirage(tirage):
    """ A partir d'un tirage de lettres, construit tous les sous-tirages possibles """
    tirage.sort() # Tri
    n = len(tirage)
    liste = []
    for k in range(1,2**n):
        b_str = list(format(k, '0'+str(n)+'b'))
        b_str.reverse()
        b = [int(bs) for bs in b_str]
```

```
        mot = ""
        for i in range(n):
            if b[i] == 1:
                mot = mot + tirage[i]

        liste.append(mot)

    return liste

# Test
print("--- Mots possibles depuis un tirage ---")
n = 4
tirage = tirage_lettres(n)
tirage = ['A','B','C','L']
print("Tirage :",tirage.sort())
liste = indices_depuis_tirage(tirage)

# Ordre de la plus longue au plus petit
liste.sort(key=lambda item:len(item),reverse=True)

print("Liste indices :",liste)


##############################
## Question 4 ##

def mot_le_plus_long(tirage,dico):
    """ Trouve tous les mots français à partir d'un tirage de lettres
    et d'un dictionnaire indice->liste de mots français """
    liste = indices_depuis_tirage(tirage)              # Tous les sous-tirages possibles
    liste.sort(key=lambda item:len(item),reverse=True) # Du plus long au plus court

    # Recherche des clés (= indices) qui existent
    liste_cles = []
    for cle in liste:
        if cle in dico:
            liste_cles.append(cle)

    # Conversion en mots
    liste_mots = [dico[cle] for cle in liste_cles]
    return liste_mots

# Test
print("--- Génération des indices (à faire une seule fois) ---")
repertoire = lire_repertoire("repertoire_francais_simple.txt")
repertoire = lire_repertoire("repertoire_francais_tout.txt")

dico = dictionnaire_indices_mots(repertoire)
print("Nombre de mots dans le répertoire :",len(repertoire))
print("Nombre d'indices :",len(dico))

print("--- Le mot le plus long ---")
# tirage = ["A","B","C","E","F","H"]
# tirage = ['G', 'E', 'A', 'T', 'G', 'A', 'N']
# tirage = ['A','B','C','L']
n = 10
tirage = tirage_lettres(n)
print("Tirage :",tirage)
solutions = mot_le_plus_long(tirage,dico)
print("Liste :",*solutions)

print("--- Le mot le plus long - Exemples ---")

repertoire = lire_repertoire("repertoire_francais_simple.txt")
# repertoire = lire_repertoire("repertoire_francais_tout.txt")
```

```
dico = dictionnaire_indices_mots(repertoire)

# tirage = ['Z', 'M', 'O', 'N', 'U', 'E', 'G']
# tirage = ['H', 'O', 'I', 'P', 'E', 'U', 'C', 'R']
# tirage = ['H', 'A', 'S', 'T', 'I', 'D', 'O', 'I', 'T']
tirage = ['E', 'T', 'N', 'V', 'E', 'U', 'Z', 'O', 'V', 'N']
print("Tirage :",tirage)
solutions = mot_le_plus_long(tirage,dico)
print("Liste :",*solutions)


# Idées :
# * recherche des palindromes (RADAR)
# * recherche des anacycliques (LEON <-> NOEL)
```

17. Images et matrices

Activité 1 à 4

Activité 1 à 4 matrice.py

```
################################
# Matrice
################################


################################
# Activité 1 - Convolution de matrices
################################

################################
## Question 1 ##

def afficher_matrice(M):
    """ Affichage propre d'une matrice dans la console """
    n = len(M)     # Nb de lignes
    p = len(M[0])  # Nb de colonnes
    for ligne in M:
        for x in ligne:
            if isinstance(x,int):
                print('{:3d}'.format(x),end=" ")
            else:
                print('{0:.3f}'.format(x),end=" ")
        print()
    return

# Test
print("--- Afficher une matrice ---")
M = [[1,2,3], [4,5,6], [7,8,9]]
afficher_matrice(M)

################################
## Question 2 ##

def element_convolution(C,M):
    """ Calcul de l'élément de convolution d'une matrice 3x3
    sur une matrice 3x3. Le résultat est un nombre """
```

```
    n = len(C)     # Nb de lignes
    p = len(C[0])  # Nb de colonnes
    conv = 0
    for i  in range(n):
        for j in range(p):
            conv += C[i][j]*M[i][j]
    return conv

# Test
print("--- Convolution ---")
C = [[1,1,1], [1,5,1], [1,1,1]]
M = [[1,2,3], [4,5,6], [7,8,9]]
print(element_convolution(C,M))

##############################
## Question 3 ##

def convolution(C,M):
    """ Calcul de la matrice de convolution.
    C est une matrice 3x3 qui agit sur M matrice nxp.
    Le résultat est une matrice nxp """

    n = len(M)     # Nb de lignes
    p = len(M[0])  # Nb de colonnes
    N = [[0 for j in range(p)] for i in range(n)]
    for i in range(n):
        for j in range(p):
            MM = [  [ M[(i-1)%n][(j-1)%p], M[(i-1)%n][j], M[(i-1)%n][(j+1)%p] ],
                    [ M[i][(j-1)%p],  M[i][j],  M[i][(j+1)%p] ],
                    [ M[(i+1)%n][(j-1)%p], M[(i+1)%n][j], M[(i+1)%n][(j+1)%p] ] ]

            N[i][j] = element_convolution(C,MM)
    return N

# Test
print("--- Convolution ---")
C = [[0,1,0], [0,0,0], [0,0,0]]
# C = [[0,1,0], [1,2,1], [0,1,0]]
M = [[1,2,3,4], [5,6,7,8], [9,10,11,12]]

print(" C = ")
afficher_matrice(C)
print(" M = ")
afficher_matrice(M)
N = convolution(C,M)
print(" N = ")
afficher_matrice(N)

##############################
## Question 3 ##

def convolution_entiere(C,M):
    """ Comme convolution()
    C matrice 3x3, M matrice nxp
    mais matrice résultat à coeff entiers, entre 0 et 255 """

    n = len(M)     # Nb de lignes
    p = len(M[0])  # Nb de colonnes
    N = convolution(C,M)
    NN = [[0 for j in range(p)] for i in range(n)]
    for i  in range(n):
        for j in range(p):
```

```
            conv = N[i][j]
            conv = round(conv)
            conv = max(0,conv)
            conv = min(255,conv)
            NN[i][j] = conv
    return NN

# Test
print("--- Convolution entière ---")
C = [[0,-1/2,0],[-1/2,3,-1/2],[0,-1/2,0]]

M = [[101,102,103,104],[201,151,101,51],[50,100,150,200]]
print(" C = ")
afficher_matrice(C)
print(" M = ")
afficher_matrice(M)
N = convolution_entiere(C,M)
print(" N = ")
afficher_matrice(N)
NN = convolution(C,M)
print(" NN = ")
afficher_matrice(NN)

##############################
# Activité 2 - Lire et écrire des images
##############################

##############################
## Question 1 ##

def pgm_vers_matrice_simple(fichier):
    """ Lit un fichier image au format pgm (format simple) et renvoie une matrice """

    # Fichier à lire
    fic_in = open(fichier,"r")

    i = 0  # Numéro de ligne du fichier
    matrice = []
    for ligne in fic_in:
        if i >= 3:
            liste_str = ligne.split()
            liste = [int(x) for x in liste_str]
            matrice += [liste]
        i = i + 1

    # Fermeture des fichiers
    fic_in.close()
    return matrice

# Test
print("--- Fichier pgm vers matrice (basique) ---")
M = pgm_vers_matrice_simple("exemple_image.pgm")
afficher_matrice(M)

##############################
## Question 1 ##

def pgm_vers_matrice(fichier):
    """ Lit un fichier image au format pgm (format qcq) et renvoie une matrice """

    # Fichier à lire
    fic_in = open(fichier,"r")

    i = 0  # Numéro de ligne du fichier (sans compter les lignes commentées)
    matrice = []  # Pour la matrice finale
```

```
    ligne_matrice = []    # Pour une ligne de la matrice
    for ligne in fic_in:
        if '#' in ligne:
            continue              # On zappe les lignes commentées (i n'augmente pas)

        if i == 1:            # Taille de l'image
            liste_ligne = ligne.split()
            p = int(liste_ligne[0])
            n = int(liste_ligne[1])

        if i >= 3:
            liste_str = ligne.split()
            liste = [int(x) for x in liste_str]

            while len(liste)>0 :
                while len(liste)>0 and len(ligne_matrice) < p:
                    ligne_matrice += [liste.pop(0)]
                if len(ligne_matrice) == p:
                    matrice += [ligne_matrice]
                    ligne_matrice = []

        i = i + 1

    # Fermeture des fichiers
    fic_in.close()
    return matrice

# Test
print("--- Fichier pgm vers matrice (new) ---")
# M = pgm_vers_matrice("exemple_image.pgm")
M1 = pgm_vers_matrice("cours-matrice-pgm-1.pgm")
M2 = pgm_vers_matrice("cours-matrice-pgm-2.pgm")
M3 = pgm_vers_matrice("cours-matrice-pgm-3.pgm")
print("M1")
afficher_matrice(M1)
print("M2")
afficher_matrice(M2)
print("M3")
afficher_matrice(M3)

##############################
## Question 2 ##

def matrice_vers_pgm(M,fichier):
    """ Ecrit une matrice dans un fichier au format pgm """
    fic_out = open(fichier,"w")

    # Entête du fichier pgm
    n = len(M)     # Nb de lignes
    p = len(M[0])  # Nb de colonnes
    fic_out.write("P2\n")  # Image en niveaux de gris
    fic_out.write(str(p) + " " + str(n) + "\n")  # Nb de colonnes et de lignes
    fic_out.write("255\n")  # Niveaux de gris
    for ligne in M:
        for x in ligne:
            fic_out.write('{:4d}'.format(x))
        fic_out.write('\n')

    fic_out.close()
    return

# Test
print("--- Matrice vers fichier pgm ---")
M = [[101,102,103],[104,105,106],[107,108,109]]
```

```
matrice_vers_pgm(M,"export_image.pgm")
MM = pgm_vers_matrice("export_image.pgm")
print('Avant, M = ')
afficher_matrice(M)
print('Après, M = ')
afficher_matrice(MM)

# Test
print("--- Matrice vers fichier pgm (exemple réel) ---")
M = pgm_vers_matrice("input/monde.pgm")
# afficher_matrice(M)
matrice_vers_pgm(M,"output/monde_export.pgm")

##############################
# Activité 3 - Convolution et image
##############################

##############################
## Question 1 ##

def convolution_image(C,fichier_in,fichier_out):
    """ Fait la convolution de la matrice C sur l'image du fichier d'entrée.
    Le résultat est une image transformée écrite dans le fichier de sortie """
    M = pgm_vers_matrice(fichier_in)
    N = convolution_entiere(C,M)
    matrice_vers_pgm(N,fichier_out)
    return

# Test
print("--- Convolution sur image ---")

## Flou ##
C = [[1/9,1/9,1/9],[1/9,1/9,1/9],[1/9,1/9,1/9]]           # flou classique
# C = [[1/16,2/16,1/16],[1/16,4/16,2/16],[1/16,2/16,1/16]]  # flou gaussien

# convolution_image(C,'exemple_image.pgm','exemple_image_conv.pgm')

# convolution_image(C,'input/chat.pgm','output/chat_conv_flou.pgm')
# convolution_image(C,'input/colonnes.pgm','output/colonnes_conv_flou.pgm')
# convolution_image(C,'input/totem.pgm','output/totem_conv_flou.pgm')
# convolution_image(C,'input/renne.pgm','output/renne_conv_flou.pgm')
# convolution_image(C,'input/monde.pgm','output/monde_conv_flou.pgm')

## Bord ##
C = [[-1,-1,-1],[-1,8,-1],[-1,-1,-1]]
# C = [[0,1,0],[1,-4,1],[0,1,0]]
# C = [[1,0,-1],[0,0,0],[-1,0,1]]

# convolution_image(C,'input/chat.pgm','output/chat_conv_bord.pgm')
# convolution_image(C,'input/colonnes.pgm','output/colonnes_conv_bord.pgm')
# convolution_image(C,'input/totem.pgm','output/totem_conv_bord.pgm')
# convolution_image(C,'input/renne.pgm','output/renne_conv_bord.pgm')
# convolution_image(C,'input/monde.pgm','output/monde_conv_bord.pgm')

## Piqué ##
C = [[0,-1,0],[-1,5,-1],[0,-1,0]]
# convolution_image(C,'input/chat.pgm','output/chat_conv_pique.pgm')
# convolution_image(C,'input/colonnes.pgm','output/colonnes_conv_pique.pgm')
# convolution_image(C,'input/totem.pgm','output/totem_conv_pique.pgm')
# convolution_image(C,'input/renne.pgm','output/renne_conv_pique.pgm')
# convolution_image(C,'input/monde.pgm','output/monde_conv_pique.pgm')

## Estampé ##
```

```
C = [[-2,-1,0],[-1,1,1],[0,1,2]]
# convolution_image(C,'input/chat.pgm','output/chat_conv_estampe.pgm')
# convolution_image(C,'input/colonnes.pgm','output/colonnes_conv_estampe.pgm')
# convolution_image(C,'input/totem.pgm','output/totem_conv_estampe.pgm')
# convolution_image(C,'input/renne.pgm','output/renne_conv_estampe.pgm')
# convolution_image(C,'input/monde.pgm','output/monde_conv_estampe.pgm')


##############################
## Question 2 ##

def decalage_haut_image(k,fichier_in,fichier_out):
    """ Translate une image vers le haut de k pixels """

    M = pgm_vers_matrice(fichier_in)
    C = [[0,0,0],[0,0,0],[0,1,0]]
    for __ in range(k):
        M = convolution_entiere(C,M)
    matrice_vers_pgm(M,fichier_out)
    return

# Test
print("--- Convolution sur image ---")
# decalage_haut_image(15,'input/chat.pgm','output/chat_conv_decal.pgm')
# decalage_haut_image(75,'input/monde.pgm','output/monde_conv_decal.pgm')

## Commande bash pour conversion simple pgm vers png :
## convert toto.pgm toto.png
## Conversion multiple :
## for fic in $(ls *.pgm); do convert "$fic" $(basename $fic .pgm).png; done

##############################
# Activité 4 - Transformation
##############################

from math import *

##############################
## Question 1 ##

def dilatation_matrice(kx,ky,M):
    """ Agrandit une matrice selon la direction horizontale (facteur kx)
    et verticale (facteur ky) """

    # kx,ky entiers
    n = len(M)     # Nb de lignes
    p = len(M[0])  # Nb de colonnes
    N = [[0 for j in range(kx*p)] for i in range(ky*n)]
    for i  in range(ky*n):
        for j in range(kx*p):
            N[i][j] = M[i//ky][j//kx]
    return N

# Test
print("--- Dilatation ---")
M = [[1,2,3],[4,5,6],[7,8,9]]
print(" M = ")
afficher_matrice(M)
N = dilatation_matrice(3,2,M)
print(" N = ")
afficher_matrice(N)

# Test
print("--- Dilatation sur image ---")
```

```
# fichier_in = 'input/chat.pgm'
# fichier_out = 'output/chat_biho.pgm'
# M = pgm_vers_matrice(fichier_in)
# N = dilatation_matrice(2,3,M)
# matrice_vers_pgm(N,fichier_out)

##############################
## Question 2 ##

def vecteur_image(T,x,y):
    """ Image d'un vecteur (x,y) par une matrice T de taille 2x2 """
    a,b,c,d = T[0][0],T[0][1],T[1][0],T[1][1]
    xx = a*x + b*y
    yy = c*x + d*y
    return xx,yy


def inverse_matrice(T):
    """ Inverse de la matrice T de taille 2x2 """
    a,b,c,d = T[0][0],T[0][1],T[1][0],T[1][1]
    det = a*d-b*c
    aa,bb,cc,dd = d/det,-b/det,-c/det,a/det
    Tinv = [[aa,bb],[cc,dd]]
    return Tinv

# Test
print("--- Matrice de transformation ---")
theta = pi/3
Ttheta = [[cos(theta),-sin(theta)],[sin(theta),cos(theta)]]
x,y = 4,5
xx,yy = vecteur_image(Ttheta,x,y)
print("Matrice T(theta) :",Ttheta,"  x,y :",x,y,"  x',y' :",xx,yy)

theta = -pi/3
Tmoinstheta = [[cos(theta),-sin(theta)],[sin(theta),cos(theta)]]
print("Matrice T(-theta) :          ", Tmoinstheta)
print("Matrice inverse de T(theta) :", inverse_matrice(Ttheta))

##############################
## Question 3 ##

def transformation(T,M):
    """ Transformation de la matrice M en faisant agir la matrice T """

    n = len(M)     # Nb de lignes
    p = len(M[0])  # Nb de colonnes

    x1, y1 = vecteur_image(T,0,n)
    x2, y2 = vecteur_image(T,p,n)
    x3, y3 = vecteur_image(T,p,0)

    xmin = round(min(0,x1,x2,x3))
    xmax = round(max(0,x1,x2,x3))
    ymin = round(min(0,y1,y2,y3))
    ymax = round(max(0,y1,y2,y3))

    pp = xmax-xmin
    nn = ymax-ymin

    Tinv = inverse_matrice(T)

    N = [[0 for jj in range(pp)] for ii in range(nn)]
    for ii  in range(nn):
        for jj in range(pp):
```

```
            j, i = vecteur_image(Tinv,jj+xmin,ii+ymin)
            j, i = floor(j), floor(i)
            if (0 <= i < n) and (0 <= j < p):
                N[ii][jj] = M[i][j]
            else:
                N[ii][jj] = 0

    return N

# Test
print("--- Transformation de matrice ---")
T = [[4/3,0],[0,2]]
print(" T = ")
afficher_matrice(T)
M = [[1,2,3],[4,5,6],[7,8,9]]
print(" M = ")
afficher_matrice(M)
N = transformation(T,M)
print(" N = ")
afficher_matrice(N)

# Test
print("--- Transformation sur image : dilatation ---")
# T = [[pi,0],[0,1]]
# fichier_in = 'input/chat.pgm'
# fichier_out = 'output/chat_dilatation.pgm'
# M = pgm_vers_matrice(fichier_in)
# N = transformation(T,M)
# matrice_vers_pgm(N,fichier_out)

# Test
print("--- Transformation sur image : rotation ---")
# theta = pi/3
# T = [[cos(theta),-sin(theta)],[sin(theta),cos(theta)]]
# fichier_in = 'input/chat.pgm'
# fichier_out = 'output/chat_rotation.pgm'
# M = pgm_vers_matrice(fichier_in)
# N = transformation(T,M)
# matrice_vers_pgm(N,fichier_out)

# Test
print("--- Transformation sur image : symétrie ---")
# T = [[1,0],[0,-1]]
# fichier_in = 'input/chat.pgm'
# fichier_out = 'output/chat_symetrie.pgm'
# M = pgm_vers_matrice(fichier_in)
# N = transformation(T,M)
# matrice_vers_pgm(N,fichier_out)

# Test
print("--- Transformation sur image : quelconque ---")
# T = [[3,1],[1,5]]
# fichier_in = 'input/chat.pgm'
# fichier_out = 'output/chat_transfo.pgm'
# M = pgm_vers_matrice(fichier_in)
# N = transformation(T,M)
# matrice_vers_pgm(N,fichier_out)


##############################
## Question 4 (old) ##
```

```
def ajout_bord(M,k,l):
    """ Rajoute des bords """
    n = len(M)     # Nb de lignes
    p = len(M[0])  # Nb de colonnes
    nn = n + 2*k
    pp = n + 2*l

    N = [[0 for j in range(pp)] for ii in range(nn)]
    for ii  in range(nn):
        for jj in range(pp):
            i = ii - k
            j = jj - l
            if (0 <= i < n) and (0 <= j < p):
                N[ii][jj] = M[i][j]
            else:
                N[ii][jj] = 0

    return N

# Test
# print("--- Test ---")
# afficher_matrice(T)
# M = [[1,2,3],[4,5,6],[7,8,9]]
# print(" M = ")
# afficher_matrice(M)
# N = ajout_bord(M,1,2)
# print(" N = ")
# afficher_matrice(N)

# Test
# print("--- Test sur image ---")
# T = [[sqrt(3)/2,-1/2],[1/2,sqrt(3)/2]]
# fichier_in = 'input/chat.pgm'
# fichier_out = 'output/chat_transfo.pgm'
# M = pgm_vers_matrice(fichier_in)
# MM = ajout_bord(M,100,200)
# N = transformation(T,MM)
# matrice_vers_pgm(N,fichier_out)

##############################
## Question 5 (old) ##

def transformation_centre(T,M):
    """ Action à partir du centre de l'image """
    n = len(M)     # Nb de lignes
    p = len(M[0])  # Nb de colonnes

    Tinv = inverse_matrice(T)

    N = [[0 for jj in range(p)] for ii in range(n)]
    for ii  in range(n):
        for jj in range(p):
            j, i  = vecteur_image(Tinv,jj-p/2,ii-n/2)
            j, i = floor(j+p/2), floor(i+n/2)
            if (0<= i <n) and (0<= j <p):
                N[ii][jj] = M[i][j]
            else:

    return N

# Test
# print("--- Test sur image ---")
```

```
# # T = [[sqrt(3)/2,-1/2],[1/2,sqrt(3)/2]]
# T = [[1,0],[0,-1]]
# fichier_in = 'input/chat.pgm'
# fichier_out = 'output/chat_transfo.pgm'
# M = pgm_vers_matrice(fichier_in)
# MM = ajout_bord(M,100,200)
# N = transformation_centre(T,MM)
# matrice_vers_pgm(N,fichier_out)
```

18. Ensemble de Mandelbrot

Activité 1 à 4

Activité 1 à 4 — mandelbrot_complexe.py

```
################################
# Mandelbrot(version réelle)
################################

################################
# Activité 1 - Mandelbrot (version complexe)
################################

################################
## Question 1 ##

def f(z,c):
    """ z <- z^2 + c """
    return z*z + c

################################
## Question 2 ##

Max_iter = 100

def iterer(c):
    """ Calcul du nb d'itérations avant que la suite
    ne s'échappe à l'infini.
    C'est la répétition de cette opération qui prend beaucoup de temps """

    z = 0     # Point initial
    i = 1
    while (i < Max_iter) and (abs(z) <= 2):
        z = f(z,c)
        i = i + 1

    if i >= Max_iter: # La suite converge
        return 0
    else:             # La suite s'échappe à partir du rang i
        return i

# Test
# print(iterer(1.5+1.5j))

################################
## Question 3 ##

################################
```

```
# Allumer un pixel

def afficher_pixel(i,j,couleur):
    canvas.create_line(i,j,(i+1),(j+1),fill=couleur,width=1)
    return

##############################
# Choix d'une couleur

def choix_couleur(i):
    """ Choix d'une couleur en fonction de la vitesse i d'échapemment """
    if i == 0:
        R,V,B = 0,0,0
    else:
        R,V,B = 50 + 2*i,0,0

    couleur = '#%02x%02x%02x' % (R, V, B)

    return couleur

##############################
# Fenêtre

xmin, xmax = -2.2, 1
ymin, ymax = -1.2, 1.2

Nx = 400   # Nb de pixels pour les x
Ny = round( (ymin-ymax)/(xmin-xmax) * Nx )

##############################
## Question 4 ##

def mandelbrot():
    """ Fonction principale : affichage pixel par pixel
    de l'ensemble de Mandelbrot dans la fenêtre demandée """

    pasx = (xmax-xmin)/Nx
    pasy = (ymax-ymin)/Ny

    a = xmin
    b = ymin

    for i in range(Nx):
        for j in range(Ny):
            c = a + b*1j
            vitesse = iterer(c)
            couleur = choix_couleur(vitesse)
            afficher_pixel(i,j,couleur)
            b = b + pasy

        b  = ymin
        a = a + pasx

    return

##############################


# Fenêtre tkinter
from tkinter import *
root = Tk()
canvas = Canvas(root, width=Nx, height=Ny, background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

mandelbrot()

root.mainloop()
```

Activité 1 à 4 mandelbrot_reel.py

```
##############################
# Mandelbrot
##############################

##############################
# Activité 2 - Mandelbrot (version réelle)
##############################

##############################
## Question 1 ##

def f(x,y,a,b):

    xx = x*x - y*y + a
    yy = 2*x*y + b

    return xx, yy

##############################
## Question 2 ##

Max_iter = 100

def iterer(a,b):
    """ Calcul du nb d'itérations avant que la suite
    ne s'échappe à l'infini.
    C'est la répétition de cette opération qui prend beaucoup de temps """

    x, y = 0, 0  # Point initial
    i = 1
    while (i < Max_iter) and (x*x + y*y <= 4):
        x, y = f(x,y,a,b)
        i = i + 1

    if i >= Max_iter: # La suite converge
        return 0
    else:             # La suite s'échappe à partir du rang i
        return i

# Test
# print(iterer(1.5,1.5))


##############################
## Question 5  ##

def iterer_opt(a,b):
    """ Version un peu optimisée de iterer() """

    x, y = 0, 0   # Point initial
    x2, y2 = x*x, y*y    # Carrés

    i = 1
    while (i < Max_iter) and (x2 + y2 <= 4):
        x, y = x2 - y2 + a, (x+y)*(x+y) - x2-y2 + b
        x2, y2 = x*x, y*y
        i = i + 1

    if i >= Max_iter: # La suite converge
        return 0
    else:             # La suite s'échappe à partir du rang i
        return i
```

```
# Test
# print(iterer_opt(1.5,1.5))

##############################
## Question 3 ##

##############################
# Allumer un pixel

def afficher_pixel(i,j,couleur):
    canvas.create_line(i,j,(i+1),j,fill=couleur,width=1)
    return

##############################
# Choix d'une couleur

def choix_couleur(i):
    """ Choix d'une couleur en fonction de la vitesse i d'échapemment """
    if i == 0:
        R,V,B = 0,0,0
    else:
        R,V,B = 50+2*i,0,00            # Nuances de rouge
        R,V,B = 255-3*i,255,255            # Pour impression n&b
        # R,V,B = 50+2*i,150+i,100-i    # Couleurs
        # R,V,B = 50+2*i,50+2*i,50+2*i  # Gris

    couleur = '#%02x%02x%02x' % (R%256, V%256, B%256)

    return couleur


##############################
# Fenêtre

# Tout Mandelbrot
xmin, xmax = -2.2, 1
ymin, ymax = -1.2, 1.2

# Petit zoom
# xmin, xmax = 0.3, 0.5
# ymin, ymax = 0.3, 0.45

# Moyen zoom
# xmin, xmax = 0.4414, 0.4418
# ymin, ymax = 0.3765, 0.3768
# Max_iter = 200

Nx = 1200   # Nb de pixels pour les x
Ny = round( (ymin-ymax)/(xmin-xmax) * Nx )

##############################
## Question 4 ##

def mandelbrot():
    """ Fonction principale : affichage pixel par pixel
    de l'ensemble de Mandelbrot dans la fenêtre demandée """

    pasx = (xmax-xmin)/Nx
    pasy = (ymax-ymin)/Ny

    a = xmin
    b = ymin

    for i in range(Nx):
        for j in range(Ny):
```

```
            vitesse = iterer(a,b)
            couleur = choix_couleur(vitesse)
            afficher_pixel(i,j,couleur)
            b = b + pasy

        b  = ymin
        a = a + pasx

    return

##############################

# Fenêtre tkinter
from tkinter import *
root = Tk()
canvas = Canvas(root, width=Nx, height=Ny, background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

mandelbrot()

root.mainloop()
```

Activité 5 et 6

Activité 5 et 6 — mandelbrot_complexe.py

```
##############################
# Mandelbrot
##############################

# A faire :
# * avec complex z <- z^2 + c
# * Julia
# * Buddhabrot

##############################
# Activité 1 - Mandelbrot
##############################

##############################
## Question 1 ##

def f(z,c):
    """ z <- z^2 + c """
    zz = z*z + c
    return zz


##############################

def iterer(z0,c,Max_iter=100):
    z = z0
    i = 1
    while (i < Max_iter) and (abs(z) <= 2):
        z = f(z,c)
        i = i + 1

    if i >= Max_iter: # La suite converge
        return 0
```

```
    else:           # La suite s'échappe à partir du rang i
        return i

##############################
def choix_couleur(i):
    if i == 0:
        R,V,B = 0,0,0
    else:
        R,V,B = 50 + 2*i,0,0

    coul = '#%02x%02x%02x' % (R, V, B)

    return coul

##############################
def afficher_pixel(i,j,couleur):
    #canvas.create_rectangle(i*taille,j*taille,(i+1)*taille,(j+1)*taille,outline=couleur,
    ↪ fill=couleur)
    canvas.create_line(i,j,(i+1),(j+1),fill=couleur,width=1)
    return


##############################
# Fenêtre

xmin, xmax = -1.5, 1.5
ymin, ymax = -1.5, 1.5

Nx = 400   # Nb de pixels pour les x
Ny = round( (ymin-ymax)/(xmin-xmax) * Nx )

def julia(c):

    pasx = (xmax-xmin)/Nx
    pasy = (ymax-ymin)/Ny

    a = xmin
    b = ymin

    for i in range(Nx):
        for j in range(Ny):
            z0 = complex(a,b)
            vitesse = iterer(z0,c,Max_iter)
            couleur = choix_couleur(vitesse)
            afficher_pixel(i,j,couleur)
            b = b + pasy

        b  = ymin
        a = a + pasx

    return

##############################

Max_iter = 100

# Fenêtre tkinter
from tkinter import *
root = Tk()
canvas = Canvas(root, width=Nx, height=Ny, background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

# c = complex(0.3,0.55)
c = 0.3+0.55j
julia(c)
```

```
root.mainloop()
```

Activité 5 et 6 mandelbrot_reel.py

```
################################
# Mandelbrot
################################

# A faire :
# * avec complex z <- z^2 + c
# * Julia
# * Buddhabrot

################################
# Activité 1 - Mandelbrot
################################

################################
## Question 1 ##

def f(x,y,a,b):

    xx = x*x - y*y + a
    yy = 2*x*y + b

    return xx, yy


################################

def iterer(x0,y0,a,b,Max_iter=100):
    x, y = x0, y0
    i = 1
    while (i < Max_iter) and (x*x + y*y <= 4):
        x,y = f(x,y,a,b)
        i = i + 1

    if i >= Max_iter: # La suite converge
        return 0
    else:           # La suite s'échappe à partir du rang i
        return i

################################

def choix_couleur(i):
    if (i ==0):
        R,V,B = 0,0,0
    else:
        # R,V,B = 50 + 2*i,0,0
        # R,V,B = 255-2*i,255-2*i,255-2*i           # Pour impression n&b
        R,V,B = 255-2*i,255,255    # Couleurs
        #R,V,B = 50+2*i,50+2*i,50+2*i  # Gris
    coul = '#%02x%02x%02x' % (R, V, B)

    return coul

################################
def afficher_pixel(i,j,couleur):
    #canvas.create_rectangle(i*taille,j*taille,(i+1)*taille,(j+1)*taille,outline=couleur,
```

```
    ↪ fill=couleur)
    canvas.create_line(i,j,(i+1),(j+1),fill=couleur,width=1)
    return


##############################
# Fenêtre

xmin, xmax = -2, 2
ymin, ymax = -1.5, 1.5

Nx = 800   # Nb de pixels pour les x
Ny = round( (ymin-ymax)/(xmin-xmax) * Nx )

def julia(cx,cy):

    pasx = (xmax-xmin)/Nx
    pasy = (ymax-ymin)/Ny

    x0 = xmin
    y0 = ymin

    for i in range(Nx):
        for j in range(Ny):
            vitesse = iterer(x0,y0,a,b,Max_iter)
            couleur = choix_couleur(vitesse)
            afficher_pixel(i,j,couleur)
            y0 = y0 + pasy

        y0  = ymin
        x0 = x0 + pasx

    return

##############################

Max_iter = 100

# Fenêtre tkinter
from tkinter import *
root = Tk()
canvas = Canvas(root, width=Nx, height=Ny, background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

# c = complex(0.3,0.55)

# Julia 1
# xmin, xmax = -1.5, 1.5
# ymin, ymax = -1.5, 1.5
# a, b = 0.3, 0.55

# Julia 2
# xmin, xmax = -0.25, 0.25
# ymin, ymax = -0.75, -0.25
# a, b = 0.3, 0.55

# Julia 3
# xmin, xmax = -2, 2
# ymin, ymax = -1, 1
a, b = -1.31, 0

# Julia 4
# a, b = -0.101, 0.956

julia(a,b)

root.mainloop()
```

19. Images 3D

Activité 1

Activité 1 images3d_surface.py

```
##############################
# Images 3D
##############################

##############################
# Activité  - Surface z = f(x,y)
##############################

from math import *
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

##############################
## Question 1 ##

# Constantes globales
xmin, xmax = -1, 1
ymin, ymax = -1, 1

nbpoints=10

# Fonction à tracer

# Bol
def f(x,y):
    val = x**2 + y**2
    return val

# Goutte qui tombe dans l'eau
# def f(x,y):
#     r = 20*(x**2 + y**2)
#     val = sin(r)/r
#     return val

# Boîte d'oeuf
# def f(x,y):
#     val = sin(10*x)+cos(10*y)
#     return val

# Selle de cheval
# def f(x,y):
#     val = x**2-y**2
#     return val

# Un sommet et un col à franchir
# xmin, xmax = -2, 3
# ymin, ymax = -2.5, 2.5
# def f(x,y):
#     val = exp(-1/3*x**3 + x - y**2)
#     return val

##############################
## Question 2 ##

def liste_points_xcst(x):
    """ Une liste de points de la surface avec x imposé """
    y = ymin
```

```
    h = (ymax-ymin)/nbpoints

    liste_points= []
    for k in range(nbpoints+1):
        P = (x,y,f(x,y))
        liste_points.append(P)
        y = y + h

    return liste_points

# Test
print("--- Liste de points ---")
x = 0
print("Liste :",liste_points_xcst(x))

def liste_points_ycst(y):
    """ Une liste de points de la surface avec y imposé """
    x = xmin
    h = (xmax-xmin)/nbpoints

    liste_points= []
    for k in range(nbpoints+1):
        P = (x,y,f(x,y))
        liste_points.append(P)
        x = x + h

    return liste_points

##############################
## Question 3 ##

def trace_ligne(liste_points,couleur='gray'):
    """ Tracé d'une suite de segments reliant des points """
    listex = [x for x,y,z in liste_points]
    listey = [y for x,y,z in liste_points]
    listez = [z for x,y,z in liste_points]

    ax.plot(listex,listey,listez,color=couleur)
    return

def trace_surface():
    """ Tracé d'une surface en affichant des lignes à x constants,
    puis à y constants """

    # Lignes à x = cst
    x = xmin
    h = (xmax-xmin)/nbpoints

    for k in range(nbpoints+1):
        ligne = liste_points_xcst(x)
        trace_ligne(ligne,couleur='blue')
        x = x + h

    # Lignes à y = cst
    y = ymin
    h = (ymax-ymin)/nbpoints

    for k in range(nbpoints+1):
        ligne = liste_points_ycst(y)
        trace_ligne(ligne,couleur='red')
        y = y + h

    return

# Test
```

```
print("--- Trace surface ---")
fig = plt.figure()
ax = fig.gca(projection='3d',proj_type = 'ortho')
# trace_ligne([(0,0,0),(2,0,1),(1,2,3)])
# ligne = [(0, -1, 1), (0, -0.6, 0.36), (0, -0.2, 0.04), (0, 0.2, 0.04), (0, 0.6, 0.36), (0,
    ↪  1.0, 1.0)]
# trace_ligne(ligne,couleur='blue')
trace_surface()
plt.show()
```

Activité 2

Activité 2 images3d_perspective.py

```
################################
# Images 3D
################################

################################
# Activité - Perspective
################################

cube  = [(0,0,0),(1,0,0),(1,1,0),(0,1,0),(0,0,1),(1,0,1),(1,1,1),(0,1,1)]

################################
## Question 1 ##

from math import *
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def affiche_cube_3d(cube):
    """ Affichage d'un cube en 3d en reliant les sommets """

    P0,P1,P2,P3,P4,P5,P6,P7 = cube

    # Première face
    ax.plot([P0[0],P1[0],P2[0],P3[0],P0[0]],[P0[1],P1[1],P2[1],P3[1],P0[1]],[P0[2],P1[2],P2
    ↪ [2],P3[2],P0[2]],color='red')

    # Seconde face
    ax.plot([P4[0],P5[0],P6[0],P7[0],P4[0]],[P4[1],P5[1],P6[1],P7[1],P4[1]],[P4[2],P5[2],P6
    ↪ [2],P7[2],P4[2]],color='red')

    # Arète joignant les faces
    ax.plot([P0[0],P4[0]],[P0[1],P4[1]],[P0[2],P4[2]],color='red')
    ax.plot([P1[0],P5[0]],[P1[1],P5[1]],[P1[2],P5[2]],color='red')
    ax.plot([P2[0],P6[0]],[P2[1],P6[1]],[P2[2],P6[2]],color='red')
    ax.plot([P3[0],P7[0]],[P3[1],P7[1]],[P3[2],P7[2]],color='red')

    return

# Test
# print("--- Trace cube 3d ---")
# fig = plt.figure()
# ax = fig.gca(projection='3d',proj_type = 'ortho')

# affiche_cube_3d(cube)

# plt.show()
```

```
##############################
## Question 2 ##
def perspective_cavaliere(P,alpha=pi/4,k=0.5):
    """ Calcul des coordonnées la projection 2d d'un point 3d  """
    x,y,z = P
    X = x + k*cos(alpha)*y
    Y =     k*sin(alpha)*y + z
    return (X,Y)

def affiche_cube_2d(cube2d):
    """ Tracé dans le plan de la projection d'un cube en reliant les sommets projetés """
    P0,P1,P2,P3,P4,P5,P6,P7 = cube2d
    # Première face
    plt.plot([P0[0],P1[0],P2[0],P3[0],P0[0]],[P0[1],P1[1],P2[1],P3[1],P0[1]],color='blue')
    # Seconde face
    plt.plot([P4[0],P5[0],P6[0],P7[0],P4[0]],[P4[1],P5[1],P6[1],P7[1],P4[1]],color='blue')
    # Arète joignant les faces
    plt.plot([P0[0],P4[0]],[P0[1],P4[1]],color='blue')
    plt.plot([P1[0],P5[0]],[P1[1],P5[1]],color='blue')
    plt.plot([P2[0],P6[0]],[P2[1],P6[1]],color='blue')
    plt.plot([P3[0],P7[0]],[P3[1],P7[1]],color='blue')
    return

# Test
print("--- Trace cube 2d : perspective cavalière ---")
# plt.axes().set_aspect('equal')
# # cube2d = [perspective_cavaliere(P,alpha=pi/4,k=0.5) for P in cube]
# cube2d = [perspective_cavaliere(P,alpha=pi/6,k=0.7) for P in cube]
# affiche_cube_2d(cube2d)
# plt.show()


##############################
## Question 3 ##
def perspective_axonometrique(P,alpha=acos(sqrt(2/3)),omega=pi/4):
    """ Calcul des coordonnées la projection 2d d'un point 3d  """
    x,y,z = P
    X = cos(omega)*x - sin(omega)*y
    Y = -sin(omega)*sin(alpha)*x -cos(omega)*sin(alpha)*y + cos(alpha)*z
    return (X,Y)

# Test
print("--- Trace cube 2d : perspective axonométrique ---")
plt.axes().set_aspect('equal')
# Perspective isométrique
# alpha = 0.6154797086703874 # = acos(sqrt(2/3))
# cube2d = [perspective_axonometrique(P,alpha=acos(sqrt(2/3)),omega=pi/4) for P in cube]
# alpha = -10 degrés, omega = 30 degrés
# cube2d = [perspective_axonometrique(P,alpha=-10*2*pi/360,omega=40*2*pi/360) for P in cube]
# affiche_cube_2d(cube2d)
# plt.show()

##############################
## Question 4 ##
```

```
def perspective_conique(P,f=5):
    """ Calcul des coordonnées la projection 2d d'un point 3d  """
    x,y,z = P
    k = f/(y+f)
    X = k*x
    Y = k*z
    return (X,Y)

# Test
# print("--- Trace cube 2d : perspective conique ---")

plt.axes().set_aspect('equal')

cube  = [(1,1,-1),(2,1,-1),(2,2,-1),(1,2,-1),(1,1,-2),(2,1,-2),(2,2,-2),(1,2,-2)]
cube2d = [perspective_conique(P,f=10) for P in cube]

affiche_cube_2d(cube2d)
plt.show()
```

Activité 3

Activité 3 images3d_sphere.py

```
################################
# Images 3D
################################

################################
# Activité  - Coordonnées sphèriques
################################

from math import *
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

################################
## Question 1 ##

# r rayon,
# phi : latitude
# lamb : longitude

def latlong_vers_xyz(r,phi,lamb):
    """ (r,phi,lamb) -> (x,y,z) """
    x = r*cos(phi)*cos(lamb)
    y = r*cos(phi)*sin(lamb)
    z = r*sin(phi)
    return x,y,z


# Test
print("--- Coordonnées xyz d'après latitude/longitude ---")
r,phi,lamb = 1,45*2*pi/360,30*2*pi/360
P = latlong_vers_xyz(r,phi,lamb)
print("(x,y,z) =", P)

################################
## Question 2 ##

def xyz_vers_latlong(x,y,z):
```

```
    """ (x,y,z) -> (r,phi,lamb) """
    r = sqrt(x**2+y**2+z**2)
    phi = asin(z/r)
    lamb = asin(y/(r*cos(phi)))
    return r,phi,lamb

# Test
print("--- Coordonnées latitude/longitude d'après xyz ---")
x,y,z = (1,2,3)
S = xyz_vers_latlong(x,y,z)
print("(r,phi,lamb) =", S)

print("--- Vérification ---")
P = latlong_vers_xyz(*S)
print("(x,y,z) =", P)

##############################
## Question 3 ##

def trace_meridien(r,lamb,nbpoints=100,couleur='red'):
    """ Tracé d'un méridien connaissant sa longitude """
    phi = 0
    h = 2*pi/nbpoints
    liste_points = []
    for k in range(nbpoints+1):
        P = latlong_vers_xyz(r,phi,lamb)
        liste_points.append(P)
        phi = phi + h

    listex = [x for x,y,z in liste_points]
    listey = [y for x,y,z in liste_points]
    listez = [z for x,y,z in liste_points]

    ax.plot(listex,listey,listez,color=couleur)
    return

def trace_parallele(r,phi,nbpoints=100,couleur='blue'):
    """ Tracé d'un parallèle connaissant sa latitude """

    lamb = 0
    h = 2*pi/nbpoints
    liste_points = []
    for k in range(nbpoints+1):
        P = latlong_vers_xyz(r,phi,lamb)
        liste_points.append(P)
        lamb = lamb + h

    listex = [x for x,y,z in liste_points]
    listey = [y for x,y,z in liste_points]
    listez = [z for x,y,z in liste_points]

    ax.plot(listex,listey,listez,color=couleur)
    return

def trace_meridiens_parallelles(r,N=24):
    """ Tracé d'une série de méridiens et parallèles """
    for k in range(N):
        lamb = 2*k*pi/N
        # trace_meridien(r,lamb)
        trace_meridien(r,lamb,couleur='#C0C0C0')

    for k in range(N):
        phi = k*pi/(N//2)
        # trace_parallele(r,phi)
```

```
        trace_parallele(r,phi,couleur='#C0C0C0')
    return

def trace_point(r,phi,lamb,couleur='green'):
    """ Affichage d'un points connaissant (r,phi,lamb) """
    P = latlong_vers_xyz(r,phi,lamb)
    ax.scatter(*P,color=couleur,s=50)
    return

# Test
print("--- Trace méridien et parallèle ---")
fig = plt.figure()
ax = fig.gca(projection='3d',proj_type = 'ortho')
# trace_meridien(1,pi/6)
# trace_point(1,pi/4,0)
trace_meridiens_parallelles(r=1)
# trace_meridien(1,-pi/4)
# trace_parallele(1,pi/8)

# plt.show()

##############################
## Rappels - Activité "Vecteurs" ##

def norme(u):
    """ Norme (= longueur) d'un vecteur de l'espace """
    x,y,z = u
    n = sqrt(x**2 + y**2 + z**2)
    return n

##############################
## Question 4 ##

def trace_grand_cercle(P,Q,nbpoints=100,couleur="orange"):
    """ Calcul du grand cercle passant par deux points P et Q d'une sphère
    Idée : considérer les points comme de vecteurs u et v
    w = (cos(t) u + sin(t) v)/norme(w) pour t dans [0,2pi] """

    # Calcul du vecteur w
    u = latlong_vers_xyz(*P)
    v = latlong_vers_xyz(*Q)
    r = norme(u)
    # Calcul des points du grand cercle
    t = 0
    h = 2*pi/nbpoints
    liste_points = []
    for k in range(nbpoints+1):
        x1,y1,z1 = u
        x2,y2,z2 = v
        x = cos(t)*x1 + sin(t)*x2
        y = cos(t)*y1 + sin(t)*y2
        z = cos(t)*z1 + sin(t)*z2
        rr = norme((x,y,z))
        w = (x*r/rr,y*r/rr,z*r/rr)

        liste_points.append(w)
        t = t + h

    # Tracé
    listex = [x for x,y,z in liste_points]
    listey = [y for x,y,z in liste_points]
```

```
    listez = [z for x,y,z in liste_points]
    ax.plot(listex,listey,listez,color=couleur,linewidth=2)
    return


# Test
print("--- Grand cercle ---")
P = (1,pi/4,pi/6)
Q = (1,pi/3,-pi/4)

trace_grand_cercle(P,Q)
trace_point(*P)
trace_point(*Q)
plt.show()
```

Activité 4

Activité 4 images3d_vecteurs.py

```
################################
# Images 3D
################################

################################
# Activité  - Vecteurs
################################

from math import *

################################
## Question 1.a ##

def produit_scalaire(u,v):
    """ Produit scalaire de deux vecteurs de l'espace """
    x,y,z = u
    xx,yy,zz = v
    p = x*xx + y*yy + z*zz
    return p

################################
## Question 1.b ##

def norme(u):
    """ Norme d'un vecteur de l'espace """
    x,y,z = u
    n = sqrt(x**2 + y**2 + z**2)
    return n

################################
## Question 1.c ##

def angle(u,v):
    """ Angle (en radian) entre deux vecteurs de l'espace """
    p = produit_scalaire(u,v)
    cosinus = p/(norme(u)*norme(v))
    mon_angle = acos(cosinus)
    return mon_angle

################################
## Question 1.d ##
```

```
# Conversion angle radians/degrés

def degres_vers_radians(a):
    return 2*pi*a/360

def radians_vers_degres(a):
    return 360*a/(2*pi)


# Test
print("--- Produit scalaire ---")
u = (1,2,3)
v = (1,0,1)
print("u =", u, "   v =",v)
print("Norme de u :",norme(u))
print("Produit scalaire :",produit_scalaire(u,v))
print("Angle entre u et v (en radians) ",angle(u,v))
print("Angle entre u et v (en degrés) ",radians_vers_degres(angle(u,v)))

##############################
## Question 1.e ##

def est_visible(P,u,theta):
    """ Détermine si un point P est visible en regardant
    selon une direction u avec un angle de vision theta """
    alpha = angle(u,P)
    return (abs(alpha) <= theta)

# Test
print("--- Application : points visibles ---")
theta_degres = 50
theta_radians = degres_vers_radians(theta_degres)
u = (1,1,1)
P1 = (1,1,2)
P2 = (-1,-1,-2)
P3 = (80,10,0)
P4 = (85,10,0)

for P in [P1,P2,P3,P4]:
    print("Le point",P,"est-il visible ?",est_visible(P,u,theta_radians))

##############################
## Question 1.d ##

def rebond(u,n):
    """ Calcul comment une particule qui arrive selon un vecteur u
    rebondirait sur un plan dont le vecteur normal est n.
    Le résultat est un vecteur. """
    x,y,z = u
    xn,yn,zn = n

    N = norme(n)
    xxn,yyn,zzn = xn/N,yn/N,zn/N
    nn = (xxn,yyn,zzn)

    p = produit_scalaire(u,nn)
    print(p)
    xx = x - 2*p*xxn
    yy = y - 2*p*yyn
    zz = z - 2*p*zzn
    v = (xx,yy,zz)
    return v

# Test
print("--- Rebond ---")
```

```
u = (1,2,-1)
n = (1,1,1)

v = rebond(u,n)
print("u =", u, "   n =",n)
print("Rebond v :",v)


##############################
## Question 2.a ##

def produit_vectoriel(u,v):
    """ Produit vectoriel de deux vecteurs de l'espace """
    x,y,z = u
    xx,yy,zz = v
    w = ( y*zz-yy*z, z*xx-zz*x, x*yy-xx*y)
    return w

# Test
print("--- Produit vectoriel ---")
u = (1,2,3)
v = (1,0,1)
w = produit_vectoriel(u,v)
print("u =", u, "   v =",v)
print("w =", w)

print("Produit scalaire w avec u:",produit_scalaire(w,u))
print("Produit scalaire w avec v:",produit_scalaire(w,v))

##############################
## Question 2.b ##

# Test
print("--- Produit vectoriel : application au calcul d'un équation de plan ---")
u = (-1,2,5)
v = (2,0,3)
n = produit_vectoriel(u,v)
print("u =", u, "   v =",v)
print("n =", n)

##############################
## Question 2.c ##
# Application : surface d'un parallélogramme/triangle dans l'espace

# Test
print("--- Produit vectoriel : application au calcul d'une surface d'un triangle ou d'un
    ↪ parallélogramme ---")
u = (1,2,-5)
v = (1,-2,4)
w = produit_vectoriel(u,v)
print("u =", u, "   v =",v)
print("w =", w)
print("norme au carré :",norme(w)**2)
print("aire du parallèlogramme :",norme(w))
print("aire du triangle :",1/2*norme(w))

##############################
## Question 3.a ##

def produit_mixte(u,v,w):
    """ Produit mixte de trois vecteurs de l'espace """
    ww = produit_vectoriel(u,v)
    p = produit_scalaire(ww,w)
```

```
    return p

# Test
print("--- Produit mixte ---")
u = (1,2,3)
v = (1,0,1)
w = (4,1,0)
p = produit_mixte(u,v,w)
print("u =", u, "   v =",v, "w =", w)
print("Produit mixte:",produit_mixte(u,v,w))


################################
## Question 3.b ##

# Application : volume parallélépipède, tétraèdre
# Test
print("--- Application : volume ---")
u = (1,0,0)
v = (1,1,0)
w = (1,1,1)
p = produit_mixte(u,v,w)
print("u =", u, "   v =",v, "w =", w)
print("Volume parallélépipède :",abs(produit_mixte(u,v,w)))
print("Volume tétraèdre :",1/6*abs(produit_mixte(u,v,w)))
```

Activité 5

Activité 5 images3d_skyline.py

```
################################
# Images 3D
################################

################################
# Activité 1 - Skyline
################################

immeubles = [(0,1,5), (2,4,10), (3,5,7), (3,8,2), (7,9,4)]

################################
## Question 1 ##

# Hauteur maximale d'une liste d'immeubles
# et 0 si pas d'immeubles

def hauteur_max_immeubles(immeubles):
    """ Renvoie la hauteur max d'une liste d'immeubles """
    hmax = 0
    for x,y,h in immeubles:
        if h > hmax:
            hmax = h
    return hmax

# Test
print("--- Hauteur maximale d'une liste d'immeubles ---")
h = hauteur_max_immeubles(immeubles)
print(h)

################################
```

```
## Question 2 ##

# Calcul et tri des bords
# sans redondance

def calcul_bords(immeubles):
    """ Renvoie la liste des bords (droite + gauche) d'une listes d'immeubles """
    liste_bords = []
    for x,y,h in immeubles:
        if x not in liste_bords:
            liste_bords.append(x)
        if y not in liste_bords:
            liste_bords.append(y)
    liste_bords.sort()
    return liste_bords

# Test
print("--- Calcul des bords ---")
liste_bords = calcul_bords(immeubles)
print(liste_bords)

##############################
## Question 3 ##

# dico : bord -> liste des numéros des immeubles correspondants

def dictionnaire_bords_immeubles(immeubles):
    """ Renvoie un dictionnaire qui à un bord associe
    la liste des numéros des immeubles correspondants """
    dico = {}
    n = len(immeubles)
    for i in range(n):
        x,y,h = immeubles[i]
        if x not in dico:
            dico[x] = [i]
        else:
            dico[x].append(i)
        if y not in dico:
            dico[y] = [i]
        else:
            dico[y].append(i)

    return dico

# Test
print("--- Calcul du dictionnaire bords/immeubles ---")
dico = dictionnaire_bords_immeubles(immeubles)
print(dico)

##############################
## Question 4 ##

# Calcul de la skyline

def calcul_skyline(immeubles):
    """ Calcul de la skyline d'une liste d'immeubles """

    # Initialisation des variables
    liste_bords = calcul_bords(immeubles)
    dico = dictionnaire_bords_immeubles(immeubles)
    skyline = []
    num_immeubles_actifs = []

    # Boucle principale
    h_avant = 0
```

```
    for x in liste_bords:

        # Calcul des immeubles actifs
        for i in dico[x]:
            if i not in num_immeubles_actifs:
                num_immeubles_actifs.append(i)
            else:
                num_immeubles_actifs.remove(i)

        # Calcul de la nouvelle hauteur maximale
        immeubles_actifs = [immeubles[k] for k in num_immeubles_actifs]
        h_apres = hauteur_max_immeubles(immeubles_actifs)

        # Est-ce que cela change la skyline ?
        if h_avant != h_apres:
            skyline.append((x,h_avant))
            skyline.append((x,h_apres))

        # Pour passer à la suite
        h_avant = h_apres
    return skyline

# Test
print("--- Calcul de la skyline ---")
skyline = calcul_skyline(immeubles)
print("Immeubles :",immeubles)
print("Skyline :",skyline)


##############################
## Question 5 ##

from random import randint
# Couleur au hasard
def choix_couleur():
    """ Renvoie couleur au hasard """
    R,V,B = randint(0,255), randint(0,255), randint(0,255)
    coul = '#%02x%02x%02x' % (R, V, B)
    return coul

# Affichage graphique avec matplotlib

import matplotlib.pyplot as plt

def affichage_skyline(immeubles,avec_immeubles=True,avec_skyline=True):
    """ Affichage de la skyline """

    plt.axes().set_aspect('equal')

    if avec_immeubles:
        for (x,y,h) in immeubles:
            listex = [x,x,y,y]
            listey = [0,h,h,0]
            plt.plot([min(listex),max(listex)],[0,0],color='gray',linewidth=2)
            plt.plot(listex,listey,"r",color='gray',linewidth=2,alpha=0.7)
            plt.fill(listex,listey,"r",color=choix_couleur(),linewidth=2,alpha=0.7)

    if avec_skyline:
        skyline = calcul_skyline(immeubles)
        listex = [x for (x,y) in skyline]
        listey = [y for (x,y) in skyline]
        plt.plot([min(listex),max(listex)],[0,0],color='black',linewidth=3)
        plt.plot(listex,listey,color='black',linewidth=3)

    plt.show()
```

```
    return

# Test
print("--- Affichage ---")

immeubles = [(0,1,5), (2,4,10), (3,5,7), (3,8,2), (7,9,4)]

immeubles = [(0,3,2), (2,4,7), (4,8,5), (6,7,8), (9,10,10), (11,12,9)]

# affichage_skyline(immeubles,avec_immeubles=True,avec_skyline=True)

##############################
def hasard_immeubles(n=10,N=10):
    """ Renvoie une liste aléatoire d'immeubles """
    liste = []
    for __ in range(N):
        x = randint(0,n)
        y = x + 1 + randint(5,n)//10
        h = 1+randint(0,n)//4
        liste += [(x,y,h)]
    return liste

immeubles = hasard_immeubles(n=100,N=30)
affichage_skyline(immeubles,avec_immeubles=True,avec_skyline=True)
```

20. Sudoku

Activité 1

Activité 1 sudoku_piles.py

```
##############################
# Problèmes des 8 reines
##############################

n = 4  # Nb max de piles

les_piles = []   # Suites de piles

##############################
##############################

##############################
## Question 0 ##

def affiche_piles():
    """ Affichage de toutes les piles. """
    r = len(les_piles)
    if r == 0: return
    h = max([len(pile) for pile in les_piles])  # hauteur
    for j in range(h):                          # pour chaque ligne
        for i in range(r):                      # pour chaque colonne
            pile = les_piles[i]
            if 0 <= h-j-1 < len(pile):
                print(pile[h-j-1],end="")
            else:
                print(" ",end="")
        print("")
    return
```

```
# Test
print("--- Affichage des les_piles ---")
les_piles = [[1,2,3],[6],[5,7],[3]]
affiche_piles()

##############################
## Question 1 ##

def haut_des_piles():
    """ Renvoie la liste des éléments en haut de chaque pile. """
    return [pile[-1] for pile in les_piles]

##############################
## Question 2 ##

def choix_1(i):
    """ Choix pour problème 1. """
    if i == 0:
        return [1,2,3,4]
    if i == 1:
        return [5,6]
    if i == 2:
        return [7,8]
    if i == 3:
        return [9]

##############################
## Question 5a ##

def choix_2(i):
    """ Choix pour problème 2. """

    haut = haut_des_piles()  # Configuration en cours
    if i == 0:
        return [1,2,3]
    if i == 1:
        return [2*haut[0]]
    if i == 2:
        return [5,7,9]
    if i == 3:
        return [haut[2]]

##############################
## Question 5b ##

def choix_3(i):
    """ Choix pour problème 3. """

    haut = haut_des_piles()  # Configuration en cours

    if i == 0:
        return [1,3,5,7,9]
    if i == 1:
        if haut[0] <= 5:
            return [2,4]
        else:
            return [6,8]
    if i == 2:
        return [haut[1]//2]
    if i == 3:
        return [haut[0]-1,9]
```

```
##############################
## Question 5c ##

def choix_4(i):
    """ Choix pour problème 4. """
    if i == 0:
        return [0,1]
    if i == 1:
        return [0,1]
    if i == 2:
        return [0,1]
    if i == 3:
        return [0,1]

##############################

# Pour pouvoir choisir
def choix(i):
    return choix_3(i)

# Test remplissage des les_piles
print("--- Test remplissage ---")
les_piles = []
les_piles = les_piles + [choix(0)]
les_piles = les_piles + [choix(1)]
les_piles = les_piles + [choix(2)]
les_piles = les_piles + [choix(3)]
affiche_piles()
print(haut_des_piles())


##############################
## Question 3 ##

def retour():
    """ Revient à une configuration d'un cran avant. """

    global les_piles
    r = len(les_piles)-1                    # Numéro de la dernière pile
    while r >= 0 and len(les_piles[r]) == 1: # Si un seul élément sur la dernière pile
        les_piles = les_piles[0:r]          # On supprime la dernière pile
        r = r - 1
    if r >= 0:
        k = len(les_piles[r])               # Hauteur de la dernière pile
        les_piles[r] = les_piles[r][0:k-1]  # On retire l'élément en haut de la dernière
    ↪ pile
    return

# Test retour
print("--- Test retour ---")
for __ in range(10):
    print("---")
    affiche_piles()
    retour()

##############################
## Question 4 ##

def recherche():
    """ Recherche une ou toutes les solutions au problème. """

    global les_piles
```

```
    les_piles = []            # On part de rien
    les_piles = [ choix(0) ]  # Première pile

    termine = False

    while not termine:
        r = len(les_piles)

        # affiche_piles()  # Affichage pour mieux comprendre

        if r == 0:  # piles vides
            termine = True   # Plus rien à tester

        if 0 < r < n:
            nouv_pile = choix(r)

            if len(nouv_pile) != 0:  # Oui il y a des possibilités
                les_piles = les_piles + [nouv_pile]
            else:                    # Non il n'y a plus des possibilités
                retour()

        if r == n:   # On a une solution
            print("Solution :",haut_des_piles())
            retour()          # On veut toutes les solutions
            # termine = True  # Ou bien une seule solution nous suffit

    return

# Test recherche
print("--- Test recherche ---")
recherche()
```

Activité 2

Activité 1 — sudoku_reines.py

```
##############################
# Problèmes des 8 reines
##############################

n = 8  # Nb max de piles (et taille de l'échiquier)

les_piles = []   # Suites de piles

##############################
# Depuis l'activité "piles"
##############################

def affiche_piles():
    """ Affichage de toutes les piles. """
    r = len(les_piles)
    if r == 0: return
    h = max([len(pile) for pile in les_piles])  # hauteur
    for j in range(h):                      # pour chaque ligne
        for i in range(r):                  # pour chaque colonne
            pile = les_piles[i]
            if 0 <= h-j-1 < len(pile):
                print(pile[h-j-1],end="")
            else:
                print(" ",end="")
```

```
        print("")
    return

##############################

def haut_des_piles():
    """ Renvoie la liste des éléments en haut de chaque pile. """
    return [pile[-1] for pile in les_piles]

##############################
##############################

# Nouvelle fonction choix

def choix(i):
    """ Fonction choix pour le pb des reines.
    La pile de rang i correspond au palcement de la reine sur la colonne numéro i."""

    haut = haut_des_piles()

    # Eviter les directions verticales
    eviter = haut

    # Eviter les diagonales
    for j in range(len(haut)):

        droite = haut[j]+i-j   # Diagonale à droite
        if 0<= droite < n:
            eviter = eviter + [droite]

        gauche = haut[j]-i+j   # Diagonale à droite
        if 0 <= gauche < n:
            eviter = eviter + [gauche]

    liste_choix = [k for k in range(n) if k not in eviter]

    return liste_choix


# Test remplissage des piles
print("--- Test remplissage ---")
les_piles = [choix(0)]
print("0",les_piles)
les_piles = les_piles + [choix(1)]
print("1",les_piles)
print(les_piles)
print(haut_des_piles())

##############################
# Depuis l'activité "piles"
##############################

def retour():
    """ Revient à une configuration d'un cran avant. """

    global les_piles
    r = len(les_piles)-1                    # Numéro de la dernière pile
    while r >= 0 and len(les_piles[r]) == 1: # Si un seul élément sur la dernière pile
        les_piles = les_piles[0:r]          # On supprime la dernière pile
        r = r - 1
    if r >= 0:
        k = len(les_piles[r])               # Hauteur de la dernière pile
        les_piles[r] = les_piles[r][0:k-1]  # On retire l'élément en haut de la dernière
    ↪ pile
    return
```

```
##############################

def recherche():
    """ Recherche une ou toutes les solutions au problème. """

    global les_piles

    les_piles = []            # On part de rien
    les_piles = [ choix(0) ]  # Première pile

    nb_solutions = 0
    termine = False

    while not termine:
        r = len(les_piles)

        if r == 0:  # piles vides
            termine = True   # Plus rien à tester

        if 0 < r < n:
            nouv_pile = choix(r)

            if len(nouv_pile) != 0:  # Oui il y a des possibilités
                les_piles = les_piles + [nouv_pile]
            else:                    # Non il n'y a plus des possibilités
                retour()

        if r == n:   # On a une solution
            print("Solution :",haut_des_piles())
            nb_solutions += 1
            retour()          # On veut toutes les solutions
            # termine = True  # Ou bien une seule solution nous suffit

    return nb_solutions

# Test recherche
print("--- Test recherche des reines ---")
nb_sol = recherche()
print("Nombre de solutions :", nb_sol)
```

Activité 3

Activité 1 sudoku_sudoku.py

```
##############################
# Sudoku
##############################

# Grille 4x4 facile

N = 4   # Taille de la grille
Ml = 2  # Nb de lignes d'un bloc
Mc = 2  # Nb de colonnes d'un bloc

grille_depart = [[0 for j in range(N)] for i in range(N)]
# 0 signifie par encore déterminé, sinon nb de 1 à N

grille_depart[0][0] = 1
grille_depart[1][2] = 2
grille_depart[3][1] = 3
grille_depart[3][3] = 4
```

```
print(grille_depart)

# Grille 6x6 facile

# N = 6   # Taille de la grille
# Ml = 2  # Hauteur d'un bloc
# Mc = 3  # Largeur d'un bloc

# grille_depart = [[0 for j in range(N)] for i in range(N)]
# # 0 signifie par encore déterminé, sinon nb de 1 à N

# grille_depart[0][2] = 5
# grille_depart[0][4] = 6
# grille_depart[1][0] = 6
# grille_depart[2][0] = 4
# grille_depart[2][3] = 3
# grille_depart[3][1] = 3
# grille_depart[3][5] = 2
# grille_depart[4][5] = 1
# grille_depart[5][1] = 5
# grille_depart[5][3] = 2

##############################
def voir_grille(grille):
    """ Affiche d'une grile. """
    print()
    for i in range(N):
        for j in range(N):
            if grille[i][j] != 0:
                print(grille[i][j], end=" ")
            else:
                print('_', end=" ")
        print()
    print()
    return

# Test
print("--- Test affichage grille ---")
voir_grille(grille_depart)

##############################
def case_vers_numero(i,j):
    """ Renvoie le numéro d'ordre d'une case (i,j) -> k """
    return i*N+j

##############################
def numero_vers_case(k):
    """ Renvoie les coordonnées d'une case : k -> (i,j) """
    return(k//N,k%N)

# Test
print("--- Test numérotation ---")
k = case_vers_numero(1,2)
print(k)
print(numero_vers_case(k))

##############################
def liste_vers_grille(liste):

    grille = [[0 for j in range(N)] for i in range(N)] # Grille vide
    for k in range(len(liste)):
        i,j = numero_vers_case(k)
        grille[i][j] = liste[k]
```

```
    return grille

# Test
print("--- Test piles vers grille ---")
ma_grille = liste_vers_grille([1,2,3,4,4,3,2,1,2])
voir_grille(ma_grille)

##############################
def chiffres_ligne(i,grille):
    """ Renvoie la liste des chiffres présent sur la ligne i. """
    chiffres = []
    for j in range(N):
        if grille[i][j] != 0:
            chiffres = chiffres + [grille[i][j]]

    return chiffres

##############################
def chiffres_colonne(j,grille):
    """ Renvoie la liste des chiffres présent sur la colonne j. """
    chiffres = []
    for i in range(N):
        if grille[i][j] != 0:
            chiffres = chiffres + [grille[i][j]]

    return chiffres

##############################
def chiffres_bloc(i,j,grille):
    """ Renvoie la liste des chiffres présent dans le même bloc que la case (i,j). """
    chiffres = []
    a = Ml*(i//Ml)
    b = Mc*(j//Mc)

    for i in range(a,a+Ml):
        for j in range(b,b+Mc):
            if grille[i][j] != 0:
                chiffres = chiffres + [grille[i][j]]

    return chiffres

# Test
print("--- Test chiffres déjà placé ---")
voir_grille(grille_depart)
i = 1
print("Chiffres de la ligne i =",i,chiffres_ligne(i,grille_depart))
j = N-1
print("Chiffres de la colonne j =",j,chiffres_colonne(j,grille_depart))
print("Chiffres du bloc contenant (i,j) =",i,j,chiffres_bloc(i,j,grille_depart))



##############################
##############################

##############################
# Copier/coller de l'activité "retour en arrière"
##############################

n = N*N  # Nb max de piles (une par case)

les_piles = []   # Suites de piles
```

```
##############################
def haut_des_piles():
    """ Renvoie la liste des éléments en haut de chaque pile. """
    return [pile[-1] for pile in les_piles]

##############################
# Nouvelle fonction choix
##############################

##############################
def choix(k):

    i,j = numero_vers_case(k)

    # Si la case a un numéro au départ on le conserve (un seul choix) !
    if grille_depart[i][j] != 0:
        return [grille_depart[i][j]]

    # Sinon il faut calculer les chiffres possibles

    # On commence par recréer la grille
    haut = haut_des_piles()  # La configuration en cours
    grille = liste_vers_grille(haut)
    # On rajoute aussi les numéros au départ
    for ii in range(N):
        for jj in range(N):
            if grille_depart[ii][jj] != 0:
                grille[ii][jj] = grille_depart[ii][jj]

    # Eviter les chiffres sur la même ligne
    # ou sur la même colonne ou sur la case
    eviter = chiffres_ligne(i,grille) + chiffres_colonne(j,grille) + chiffres_bloc(i,j,
    ↪ grille)

    # Choix possibles : c'est donc le complément
    liste_choix = [k for k in range(1,N+1) if k not in eviter]

    return liste_choix


# Test remplissage des piles
print("--- Test remplissage ---")
les_piles = les_piles + [choix(0)]
print("0",les_piles)
les_piles = les_piles + [choix(1)]
print("1",les_piles)
print(les_piles)
les_piles = les_piles + [choix(3)]
print("2",les_piles)
print(les_piles)
print(haut_des_piles())

##############################
# Depuis l'activité "piles"
##############################

##############################
def retour():
    """ Revient à une configuration d'un cran avant. """

    global les_piles
    r = len(les_piles)-1                    # Numéro de la dernière pile
    while r >= 0 and len(les_piles[r]) == 1: # Si un seul élément sur la dernière pile
        les_piles = les_piles[0:r]           # On supprime la dernière pile
```

```
        r = r - 1
    if r >= 0:
        k = len(les_piles[r])               # Hauteur de la dernière pile
        les_piles[r] = les_piles[r][0:k-1]   # On retire l'élément en haut de la dernière
    ↪ pile
    return

##############################
def recherche():
    """ Recherche une ou toutes les solutions au problème. """

    global les_piles

    les_piles = []            # On part de rien
    les_piles = [ choix(0) ]  # Première pile

    termine = False

    while not termine:
        r = len(les_piles)

        # affiche_piles()  # Affichage pour mieux comprendre

        if r == 0:  # piles vides
            termine = True   # Plus rien à tester

        if 0 < r < n:
            nouv_pile = choix(r)

            if len(nouv_pile) != 0:  # Oui il y a des possibilités
                les_piles = les_piles + [nouv_pile]
            else:                    # Non il n'y a plus des possibilités
                retour()

        if r == n:   # On a une solution
            print("Solution :",haut_des_piles())
            # retour()          # On veut toutes les solutions
            termine = True  # Ou bien une seule solution nous suffit

    return haut_des_piles()   # renvoie la solution

# Test recherche
print("--- Test recherche - Exemple 4x4 ---")
liste_solution = recherche()
grille_solution = liste_vers_grille(liste_solution)
voir_grille(grille_solution)

# Exemples 9x9
N = 9   # Taille de la grille
Mc = 3  # Largeur d'un bloc
Ml = 3  # Hauteur d'un bloc

# Grille 9x9 facile (0 secondes)
liste_depart = [
3,0,4, 0,8,0, 0,5,0,
7,0,0, 0,1,0, 0,0,3,
8,0,0, 0,0,2, 6,0,0,
0,0,9, 1,0,0, 3,0,5,
4,0,5, 3,0,7, 9,0,2,
6,0,8, 0,0,9, 7,0,0,
0,0,7, 4,0,0, 0,0,6,
5,0,0, 0,9,0, 0,0,8,
0,4,0, 0,7,0, 5,0,9,
]

# Grille 9x9 moyenne (0 seconde)
```

```
liste_depart = [
5,0,8, 0,3,0, 4,6,0,
0,0,0, 2,0,0, 8,0,0,
1,9,0, 4,0,0, 7,3,0,
8,0,7, 9,2,0, 0,0,0,
0,0,9, 6,0,4, 2,0,0,
0,0,0, 0,8,3, 1,0,5,
0,3,1, 0,0,2, 0,7,6,
0,0,2, 0,0,9, 0,0,0,
0,7,5, 0,6,0, 9,0,8,
]

# Grille 9x9 très difficile (0.1 seconde)
liste_depart = [
0,4,0, 1,0,0, 9,0,0,
0,0,0, 0,0,0, 0,6,0,
0,8,0, 6,3,0, 0,0,0,
5,1,7, 2,9,0, 0,0,8,
0,0,4, 0,5,0, 2,0,0,
9,0,0, 0,1,4, 7,5,6,
0,0,0, 0,7,5, 0,8,0,
0,9,0, 0,0,0, 0,0,0,
0,0,1, 0,0,2, 0,4,0,
]

# # Grille 9x9 très difficile
liste_depart = [
# 8,0,0, 0,0,0, 0,0,0,  # Ligne 1 : Vraie premiere ligne (impossible car trop long)
8,1,2, 0,0,0, 0,0,0,  # Ligne 1 : Difficile (2 secondes)
# 8,1,0, 0,0,0, 0,0,0,   # Ligne 1 : Très difficile (20 secondes)
0,0,3, 6,0,0, 0,0,0,
0,7,0, 0,9,0, 2,0,0,
0,5,0, 0,0,7, 0,0,0,
0,0,0, 0,4,5, 7,0,0,
0,0,0, 1,0,0, 0,3,0,
0,0,1, 0,0,0, 0,6,8,
0,0,8, 5,0,0, 0,1,0,
0,9,0, 0,0,0, 4,0,0,
]

grille_depart = liste_vers_grille(liste_depart)
# 0 signifie par encore déterminé, sinon nb de 1 à N
voir_grille(grille_depart)
n = N*N
piles = []
piles = piles + [ choix(0) ]
# print(piles)
liste_solution = recherche()
grille_solution = liste_vers_grille(liste_solution)
voir_grille(grille_solution)
```

21. Fractale de Lyapunov

Activité 1

Activité 1 lyapunov_1.py

```
##############################
# Lyapunov
##############################

##############################
# Activité 1 - Suite logistique
##############################

from math import *

##############################
## Question 1 ##

def liste_suite(r,Nmin,Nmax):
    """ Liste des termes de la suite logistique (u_n)
    pour n entre Nmin et Nmax """
    k = 0
    u =  1/2   # terme initial
    liste = [u]
    while k < Nmax-1:
        u = r*u*(1-u)
        liste += [u]
        k += 1
    return liste[Nmin:]

# Test
print("--- Suite logistique ---")
print("r=0.5")
print(liste_suite(0.5,0,10))
print("r=1.5")
print(liste_suite(1.5,0,10))
print("r=3.2")
print(liste_suite(3.2,0,20))
print("r=3.5")
print(liste_suite(3.5,0,10))

##############################
## Question 2 ##

def bifurcation(Nmin=100,Nmax=200,epsilon=0.01):
    """ Liste de points de l'ensemble de bifurcation (r,u_n)
    avec n entre Nmin et Nmax
    et r qui varie entre 0 et 4.0 par pas de longueur epsilon """
    u0 = 0.5 # terme initial
    r = 0 # r initial
    mespoints = []

    while r <= 4.0:
        liste_u = liste_suite(r,Nmin,Nmax) # On calcule la suite
        for u in liste_u:
            mespoints = mespoints + [(r,u)]
        r = r + epsilon

    return mespoints
```

```
# Test
print("--- Points d'accumulation ---")
# fixer epsilon = 1 pour test
print(bifurcation(Nmin=0,Nmax=5,epsilon=1))

##############################
## Question 3 ##

# Constante pour fenêtre réelle et écran
xmin, xmax = 0, 4
ymin, ymax = 0, 1

# Zoom
# xmin, xmax = 3.4, 3.9
# ymin, ymax = 0.6, 1

Nx = 1000   # Nb de pixels pour les x
Ny = round( (ymin-ymax)/(xmin-xmax) * Nx )

def xy_vers_ij(x,y):
    """ Passage des coordonnées réelles (x,y)
    aux coordonnées graphiques (i,j) """
    pasx = (xmax-xmin)/Nx
    pasy = (ymax-ymin)/Ny

    i = round((x-xmin)/pasx)
    j = round(Ny-(y-ymin)/pasy)

    return i,j

##############################
# Allumer un pixel

def afficher_pixel(i,j,couleur):
    """ Affiche un pixel """
    canvas.create_rectangle(i,j,i+1,j+1,fill=couleur,outline=couleur,width=1)
    # canvas.create_line(i,j,i+1,j+1,fill=couleur,width=1)
    return

def afficher_axes():
    """ Dessine les axes """
    i,j = xy_vers_ij(xmin,0)
    ii,jj = xy_vers_ij(xmax,0)
    canvas.create_line(i,j,ii,jj,fill='black',width=2)
    i,j = xy_vers_ij(xmin,ymin)
    ii,jj = xy_vers_ij(xmin,ymax)
    canvas.create_line(i+4,j,ii+4,jj,fill='black',width=2)
    for x in range(0,xmax+1):
       i,j = xy_vers_ij(x,0)
       canvas.create_line(i,j+5,i,j-5,fill='black',width=2)
       canvas.create_text(i+4,j-15,text=str(x),fill='black')

# Fenêtre tkinter
from tkinter import *
root = Tk()
canvas = Canvas(root, width=Nx, height=Ny, background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

afficher_axes()

for x,y in bifurcation():
    i,j = xy_vers_ij(x,y)
```

```
    afficher_pixel(i,j,'blue')

root.mainloop()
```

Activité 2

Activité 2 lyapunov_2.py

```
##############################
# Lyapunov
##############################

##############################
# Rappels - Activité 1 - Suite logistique
##############################
from math import *

def liste_suite(r,Nmin,Nmax):
    """ Liste des termes de la suite logistique (u_n)
    pour n entre Nmin et Nmax """
    k = 0
    u =  1/2   # terme initial
    liste = [u]
    while k < Nmax-1:
        u = r*u*(1-u)
        liste += [u]
        k += 1
    return liste[Nmin:]

def xy_vers_ij(x,y):
    """ Passage des coordonnées réelles (x,y)
    aux coordonnées graphiques (i,j) """
    pasx = (xmax-xmin)/Nx
    pasy = (ymax-ymin)/Ny

    i = round((x-xmin)/pasx)
    j = round(Ny-(y-ymin)/pasy)

    return i,j

##############################
# Allumer un pixel

def afficher_pixel(i,j,couleur):
    canvas.create_line(i,j,(i+1),j,fill=couleur,width=1)
    return

def afficher_ligne(i,j,ii,jj,couleur):
    canvas.create_line(i,j,ii,jj,fill=couleur,width=1)
    return

def afficher_axes():
    i,j = xy_vers_ij(xmin,0)
    ii,jj = xy_vers_ij(xmax,0)
    canvas.create_line(i,j,ii,jj,fill='black',width=2)
    i,j = xy_vers_ij(xmin,ymin)
    ii,jj = xy_vers_ij(xmin,ymax)
    canvas.create_line(i+4,j,ii+4,jj,fill='black',width=2)
```

```
    for x in range(0,ceil(xmax)+1):
       i,j = xy_vers_ij(x,0)
       canvas.create_line(i,j+5,i,j-5,fill='black',width=2)
       canvas.create_text(i+4,j-15,text=str(x),fill='black')
    for y in range(floor(ymin),ceil(ymax)+1):
       i,j = xy_vers_ij(xmin,y)
       canvas.create_line(i-5,j,i+5,j,fill='black',width=2)
       canvas.create_text(i+15,j+4,text=str(y),fill='black')

##############################
# Activité 2 - Exposant de Lyapunov
##############################

##############################
## Question 1 ##

def exposant_lyapunov(liste_u,r):
    """ Calcul de l'exposant de Lyapunov d'une suite de termes """
    lyap = 0
    n = len(liste_u)
    for u in liste_u:
        if u != 0.5:
            lyap = lyap + log(abs(r - 2*r*u))
    return lyap/n

# Test
print("---Exposant de Lyapunov ---")
r = 0.5
print("r =",r)
print(exposant_lyapunov(liste_suite(r,0,10),r))
r = 0.5
print("r =",r)
print(exposant_lyapunov(liste_suite(r,100,200),r))
r = 3
print("r =",r)
print(exposant_lyapunov(liste_suite(r,100,200),r))
r = 3.2
print("r =",r)
print(exposant_lyapunov(liste_suite(r,100,200),r))
r = 3.7
print("r =",r)
print(exposant_lyapunov(liste_suite(r,100,200),r))
########################################

##############################
## Question 2 ##

# Constante pour fenêtre réelle et écran
xmin, xmax = 0.1, 4   # pour r
ymin, ymax = -1.3, 1.1  # pour u et l'exposant

Nx = 700   # Nb de pixels pour les x
Ny = round( (ymin-ymax)/(xmin-xmax) * Nx )

def bifurcation_lyapunov():
    """ Affichage de l'ensemble de bifurcation
    et du graphe des exposants de Lyapunov """
    Nmin = 600 # On oublie Nmin premiers termes
    Nmax = 1000 # On conserve les termes entre Nmin et Nmax
    u0 = 0.5 # terme initial
    r = xmin
    pasx = (xmax-xmin)/Nx
```

```
    for i in range(Nx):
        # 1. La suite pour r fixé
        liste_u = liste_suite(r,Nmin,Nmax) # On calcule la suite

        # 2. On affiche les points
        for u in liste_u:
            i,j = xy_vers_ij(r,u)
            afficher_pixel(i,j,'blue')

        # 3. Exposant de Lyapunov
        if i>0:
            old_lyap = lyap
            old_i,old_j = xy_vers_ij(r,old_lyap)

        lyap = exposant_lyapunov(liste_u,r)
        i,j = xy_vers_ij(r,lyap)

        if i>0:
            afficher_ligne(old_i,old_j,i,j,'black')

        # 4. On décale r d'un cran
        r = r + pasx

    return


# Fenêtre tkinter
from tkinter import *
root = Tk()
canvas = Canvas(root, width=Nx, height=Ny, background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

afficher_axes()
bifurcation_lyapunov()

root.mainloop()
```

Activité 3

Activité 3 — lyapunov_3.py

```
##############################
# Lyapunov
##############################

##############################
# Rappels - Activité 1 - Suite logistique
##############################

from math import *

def xy_vers_ij(x,y):
    pasx = (xmax-xmin)/Nx
    pasy = (ymax-ymin)/Ny

    i = round((x-xmin)/pasx)
    j = round(Ny-(y-ymin)/pasy)

    return i,j
```

```
##############################
# Allumer un pixel

def afficher_pixel(i,j,couleur):
    canvas.create_line(i,j,(i+1),j,fill=couleur,width=1)
    return


##############################
# Activité 3 - Fracatale de Lyapunov
##############################

##############################
## Question 1 ##

##############################
# Lettre dans motif

def A_ou_B(motif,n):
    """ Renvoie lettre A ou B selon le rang dans le motif répété """
    p = len(motif)
    lettre = motif[n % p]
    return lettre

# Test
print("--- A ou B d'après motif---")
print(A_ou_B('AB',10))
print(A_ou_B('AAABBB',123))

##############################
## Question 2 ##

# Suite d'après un motif
def liste_suite_motif(r1,r2,motif,Nmin,Nmax):
    """ Suite logistique en choisissant r1 ou r2
    d'après la lettre A ou B e motif répété """
    # Pour la suite
    u =  0.5
    liste = [u]
    n = 0

    while n < Nmax-1:

        # Choix de la lettre
        if A_ou_B(motif,n) == 'A':
            r = r1
        else:
            r = r2

        # Terme de la suite
        u = r*u*(1-u)
        liste += [u]

        n += 1
    return liste[Nmin:]

# Test
print("--- Suite logistique suivant un motif ---")
r1, r2 = 0.5, 3.5
motif = 'AB'
print(liste_suite_motif(r1,r2,motif,0,10))

##############################
## Question 3 ##
```

```
def exposant_lyapunov_motif(r1,r2,motif,Nmin,Nmax):
    """ Exposant de Lyapunov de la suite logistique suivant un motif """
    # Pour la suite
    u =  0.5
    liste = [u]
    n = 0

    # Pour l'exposant
    lyap = 0
    N = Nmax - Nmin

    while n < Nmax-1:

        # Choix de la lettre
        if A_ou_B(motif,n) == 'A':
            r = r1
        else:
            r = r2

        # Terme de la suite
        u = r*u*(1-u)
        liste += [u]

        # Terme de l'exposant
        if u != 0.5:
            lyap = lyap + log(abs(r - 2*r*u))

        n += 1
    return lyap/N

# Test
print("--- Exposant de Lyapunov d'une suite suivant un motif ---")
r1, r2 = 0.5, 3.5
motif = 'AB'
print(exposant_lyapunov_motif(r1,r2,motif,0,10))
print(exposant_lyapunov_motif(r1,r2,motif,100,200))

##############################
## Question 4 ##

##############################
# Choix d'une couleur

def choix_couleur(l):
    """ Choix d'une couleur selon un entier """
    i = round(150*l)

    R,V,B = i,0,0            # Nuances de rouge
    # R,V,B = 255-i,255-i,255-i   # Pour impression n&b

    couleur = '#%02x%02x%02x' % (R%256, V%256, B%256)

    return couleur

# Constante pour fenêtre réelle et écran
xmin, xmax = 2, 3  # pour r1
ymin, ymax = 3, 4  # pour r2

Nx = 201   # Nb de pixels pour les x
Ny = round( (ymin-ymax)/(xmin-xmax) * Nx )

def fractale_lyapunov(motif):
    """ Affichage de la fractale de Lyapunov suivant un motif
    Motif de base 'AB' """

    Nmin = 200 # On oublie Nmin premiers termes
```

```
    Nmax = 300 # On conserve les termes entre Nmin et Nmax
    pasx = (xmax-xmin)/Nx
    pasy = (ymax-ymin)/Ny

    r1 = xmin
    for i in range(Nx):
        r2 = ymin
        for j in range(Ny):

            # 1. Exposant de Lyapunov
            lyap = exposant_lyapunov_motif(r1,r2,motif,Nmin,Nmax)

            # 2. Choix d'une couleur
            couleur = choix_couleur(lyap)

            # 3. On affiche le point
            afficher_pixel(i,j,couleur)

            r2 = r2 + pasy
        r1 = r1 + pasx

    return


# Fenêtre tkinter
from tkinter import *
root = Tk()
canvas = Canvas(root, width=Nx, height=Ny, background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

motif = 'AB'
# motif = 'BBBBBBAAAAAA'
fractale_lyapunov(motif)

root.mainloop()
```

22. Big data I

Activité 1

Activité 1 bigdata_sondage.py

```
##############################
# Big data
##############################

##############################
# Activité 1 - Sondage
##############################

# Fichiers pour tests (l'entier indique le nb d'entrées) :
# sexe,nom,prenom,naissance(jj/mm/aaaa),ville,poids,taille,groupe sanguin
# "personnes_100.csv"
# "personnes_1000.csv"
# "personnes_10000.csv"
# "personnes_100000.csv"

# généré par https://fr.fakenamegenerator.com/

##############################
```

```
## Question 1 ##

def age_moyen(debut,fin,fichier):

    annee_aujourdhui = 2019

    # Fichier à lire
    fic_in = open(fichier,"r")

    num = 0  # Numéro de ligne
    somme = 0
    for ligne in fic_in:
        if debut <= num < fin:
            ligne = ligne.strip()  # Pour retirer la fin de ligne
            liste = ligne.split(",")
            date = liste[3]
            annee_naissance = int(date.split("/")[2])
            age = annee_aujourdhui - annee_naissance
            somme = somme + age
        num = num + 1

    # Fermeture des fichiers
    fic_in.close()

    moyenne = somme / (fin-debut)

    return moyenne

print("--- Age moyen à partir d'un sondage ---")
moyenne = age_moyen(0,100,"personnes_1000.csv")
print("Moyenne sur échantillon",moyenne)
moyenne = age_moyen(0,1000,"personnes_1000.csv")
print("Moyenne sur tout",moyenne)

##############################
## Question 2 ##

def probabilite_initiale(lettre,debut,fin,fichier):

    # Fichier à lire
    fic_in = open(fichier,"r")

    num = 0    # Numéro de ligne
    nb = 0     # Nb d'initiales testées
    nb_ok = 0  # Nb de bonnes intiales
    for ligne in fic_in:
        if debut <= num < fin:
            ligne = ligne.strip()  # Pour retirer la fin de ligne
            liste = ligne.split(",")
            init = liste[1][0]   # La première lettre du premier mot

            if init == lettre:
                nb_ok = nb_ok + 1
            nb = nb + 1
        num = num + 1

    # Fermeture des fichiers
    fic_in.close()

    proba = nb_ok/nb

    return proba

print("--- Probabilité initiale ---")
proba = probabilite_initiale("D",0,1000,"personnes_100000.csv")
print("Probabilité sur échantillon :",proba)
print("Nb d'occurences attendues pour 26 noms :",proba*26)
```

```
##############################
## Question 3 ##
def probabilite_groupe_sanguin(debut,fin,fichier):
    # sortie nb de A+, A-, B+, B-, AB+, AB-, O+, O-

    # Fichier à lire
    fic_in = open(fichier,"r")

    dico_nb = {}
    dico_nb['A+'] = 0
    dico_nb['A-'] = 0
    dico_nb['B+'] = 0
    dico_nb['B-'] = 0
    dico_nb['O+'] = 0
    dico_nb['O-'] = 0
    dico_nb['AB+'] = 0
    dico_nb['AB-'] = 0

    print(dico_nb)

    num = 0    # Numéro de ligne
    nb = 0     # Nb de personnes testées
    for ligne in fic_in:
        if debut <= num < fin:
            ligne = ligne.strip()  # Pour retirer la fin de ligne
            liste = ligne.split(",")
            groupe = liste[7]  # Le groupe
            dico_nb[groupe] += 1
            nb = nb + 1
        num = num + 1

    # Fermeture des fichiers
    fic_in.close()

    print(dico_nb)
    dico_proba = {}
    for cle in dico_nb:
        nb_groupe = dico_nb[cle]
        dico_proba[cle] = nb_groupe/nb

    return dico_proba

print("--- Probabilités groupe sanguin ---")
dico = probabilite_groupe_sanguin(0,10,"personnes_100.csv")
print("Probabilités groupe sanguin :",dico)
```

Activité 2

Activité 2 bigdata_chercher.py

```
##############################
# Big data
##############################

# Cours ordre alphabétique et Python
print("AA"<="AB")
print("CB"<="CA")
```

```
##############################
# Activité 2 - Chercher dans liste ordonnée
##############################

##############################
## Question 1 ##

def fichier_vers_liste_noms(fichier):
    fic_in = open(fichier,"r")

    liste_noms = []
    for ligne in fic_in:
        ligne = ligne.strip()  # Pour retirer la fin de ligne
        liste = ligne.split(",")
        nom = liste[1]
        liste_noms = liste_noms + [nom]

    # Fermeture des fichiers
    fic_in.close()

    liste_noms.sort()

    return liste_noms

print("--- Liste triée des noms ---")
liste_noms = fichier_vers_liste_noms("personnes_100.csv")
print(liste_noms[0:20])

##############################
## Question 2 ##

def est_debut(debut,chaine):
    if len(debut) > len(chaine):
        return False

    n = len(debut)
    if debut == chaine[:n]:
        return True
    else:
        return False

print("--- Début d'une chaîne ? ---")
print(est_debut("ABC","ABCDEF"))
print(est_debut("XYZ","ABCDEF"))
print(est_debut("ABCD","AB"))

##############################
## Question 3 ##

def chercher_1(liste,debut):
    N = 0               # Nb d'itérations
    for nom in liste:
        if est_debut(debut,nom):
            return nom, N
        N = N + 1
    return None, N

print("--- Chercher par dans la liste du premier au dernier ---")
print(chercher_1(liste_noms,"Bri"))
print(chercher_1(liste_noms,"Xyz"))

##############################
## Question 4 ##
```

```
def chercher_2(liste,debut):
    a = 0
    b = len(liste) - 1
    N = 0               # Nb d'itérations
    while b >= a:
        m = (b+a)//2
        # print(b,a,m,liste[m])

        if est_debut(debut,liste[m]):
            return liste[m], N

        if debut > liste[m]:
            a = m+1
        else:
            b = m-1
        N = N + 1

    # A la fin a=m=b
    else:
        return None, N

print("--- Chercher par dichotomie ---")
print(chercher_2(liste_noms,"Bri"))
print(chercher_2(liste_noms,"Xyz"))

##############################
## Question 5 ##

from math import *

print("--- Comparaisons ---")
liste_noms = fichier_vers_liste_noms("personnes_10000.csv")
print(liste_noms[0:20])
n = len(liste_noms)
print("Longueur de la liste :",n)
print("Itération max recherche 1 :",n)
print("Itération max recherche 2 :",floor(log(n,2))+1)
print(chercher_1(liste_noms,"Bri"))
print(chercher_2(liste_noms,"Bri"))
print(chercher_1(liste_noms,"Xyz"))
print(chercher_2(liste_noms,"Xyz"))
```

Activité 3

Activité 3 bigdata_tank.py

```
##############################
# Big data
##############################

from random import *

##############################
# Activité 3 - Tank
##############################

##############################
## Question 1 ##
```

```
def formule_tanks(echantillon):
    """ Applique la formule des tanks qui estime ... """
    k = len(echantillon)
    m = max(echantillon)
    N = m + m/k - 1
    return N

# Test
print("--- Formule des tanks ---")
echantillon = [143,77,198,32]
N = formule_tanks(echantillon)
print("Echantillon :",echantillon)
print("Nombre de tanks total estimé : ",round(N))

##############################
## Question 2 ##

def double_moyenne(echantillon):
    """ Applique la formule ... """
    k = len(echantillon)
    S = sum(echantillon)
    N = 2 * S/k
    return N

# Test
print("--- Formule des tanks ---")
echantillon = [143,77,198,32]
N1 = formule_tanks(echantillon)
N2 = double_moyenne(echantillon)
print("Echantillon :",echantillon)
print("Nombre de tanks total estimé, par la formule des tanks : ",round(N1))
print("Nombre de tanks total estimé, par le double de la moyenne : ",round(N2))

##############################
## Question 3 ##

def tirage_sans_remise(N,k):
    tirage = []
    while len(tirage) < k:
        n = randint(0,N)
        if n not in tirage:
            tirage.append(n)
    return tirage

# Test
print("--- Tirage sans remise ---")
N = 100
k = 4
tirage = tirage_sans_remise(N,k)
print("N, k :",N,k)
print("Tirage : ",tirage)

##############################
## Question 4 ##

def erreurs(N,k,nb_tirages=1000):
    erreur_tanks = 0
    erreur_double = 0
    for i in range(nb_tirages):
        echantillon = tirage_sans_remise(N,k)
        N1 = formule_tanks(echantillon)
```

```
        N2 = double_moyenne(echantillon)
        # print(N1,N2)
        erreur_tanks += abs(N-N1)
        erreur_double += abs(N-N2)
        moyenne_erreur_tank = erreur_tanks/nb_tirages
        moyenne_erreur_double = erreur_double/nb_tirages
    return moyenne_erreur_tank, moyenne_erreur_double

# Test
print("--- Erreurs formules des tanks/double de la moyenne ---")
N = 1000
k = 20
E1,E2 = erreurs(N,k)
print("N, k :",N,k)
print("Erreur moyenne formule des tanks :",E1)
print("Erreurs moyenne formule double de la moyenne :",E2)
```

Activité 4

Activité 4 bigdata_secretaire.py

```
################################
# Big data
################################

################################
# Activité 4 - Problème du secrétaire
################################

from random import *
from math import *

################################
## Question 1 ##

def genere_liste(k,N):
    """ Génère une liste de k éléments entre 0 et N. """
    liste = []
    for i in range(k):
        n = randint(0,N)
        liste.append(n)
    return liste

# Test
print("--- Génération d'une liste ---")
N = 100    # Note max
k = 20     # Nb de secrétaires
liste_aleatoire = genere_liste(k,N)
print("k, N :",k,N)
print("Liste : ",liste_aleatoire)

################################
## Question 2 ##

def choix_secretaire(liste,p):
    k = len(liste)                          # longueur de la liste
    j = min(ceil(p/100*k),100)     # longueur de l'échantillon
    # print("j",j)
    Me = max(liste[:j])                     # le meilleur score dans l'échantillon
```

```
    # print("Me",Me)
    for i in range(j,k):                    # on cherche un meilleur parmi les suivants
        if liste[i] >= Me:
            return liste[i]
    return None                             # on n'a pas trouvé, on ne prend personne

# Test
print("--- Choix d'un secrétaire ---")
N = 100    # max note
k = 100    # nb de secrétaire
liste_aleatoire = genere_liste(k,N)
pourcentage = 25
MM = choix_secretaire(liste_aleatoire,pourcentage)
M = max(liste_aleatoire)  # le meilleur score d'un sécretaire
print("k, N:",k, N)
# print("Liste : ",liste_aleatoire)
print("Meilleur valait :",M)
print("Choix vaut :",MM)

# Test
print("--- Choix d'un secrétaire : exemple cours ---")
liste = [2,5,3,4,1,6,4,5,8,3]
p = 25
MM = choix_secretaire(liste,p)
M = max(liste)  # le meilleur score d'un sécretaire
# print("Liste : ",liste)
print("Meilleur valait :",M)
print("Choix vaut :",MM)

##############################
## Question 3 ##

def meilleurs_secretaires(k,N,p,nb_tirages):
    nb_meilleurs = 0
    for i in range(nb_tirages):
        liste = genere_liste(k,N)
        M = max(liste)  # le meilleur score d'un sécretaire
        MM = choix_secretaire(liste,p)
        if M == MM:
            nb_meilleurs += 1

    return nb_meilleurs
# Test
print("--- Nb de fois où le choix d'un secrétaire est le meilleur ---")
N = 100
k = 100
pourcentage = 25
nb_tirages = 1000
nb_best = meilleurs_secretaires(k,N,pourcentage,nb_tirages)
print("k, N:",k, N)
print("Pourcentage : ",pourcentage)
print("Nb de meilleurs choisi : ",nb_best, "soit ",'{0:.2f}'.format(nb_best/nb_tirages*100),
    ↪ "%")


##############################
## Question 4 ##

# Variante liste des erreurs (avec les mêmes tirages pour tous)
def liste_meilleurs_secretaires(k,N,nb_tirages):
    liste_nb_meilleurs = [0 for i in range(1,100)]
    for i in range(nb_tirages):
```

```
        liste = genere_liste(k,N)
        M = max(liste)   # le meilleur score d'un sécretaire
        for p in range(1,100):
            MM = choix_secretaire(liste,p)
            if M == MM:
                liste_nb_meilleurs[p-1] += 1

    return liste_nb_meilleurs

# Test
print("--- Liste des erreurs choix d'un secrétaire ---")
N = 100
k = 100

liste_best = liste_meilleurs_secretaires(k,N,5000)  # Il faut au moins 1000 tirages pour
    ↪ obtenir 37%
print("k, N:",k, N)
print("Liste des nb de bons choix : ",liste_best)
maximum = max(liste_best)
print("Meilleure stratégie : pourcentage = ",liste_best.index(maximum))
```

Activité 5

Activité 5 bigdata_regression.py

```
################################
# Big data
################################

################################
# Activité 5 - Régression linéaire
################################

################################
## Question 1 ##

def moyenne(liste):
    n = len(liste)
    somme = sum(liste)
    if n==0:
        return 0
    else:
        return somme/n

def variance(liste):
    n = len(liste)
    m = moyenne(liste)
    somme = 0
    for x in liste:
        somme = somme + (x-m)**2
    return somme/n

def covariance(listex,listey):
    n = len(listex)  # = len(listey)
    mx = moyenne(listex)
    my = moyenne(listey)
    somme = 0
    for i in range(n):
        x = listex[i]
```

```
        y = listey[i]
        somme = somme + (x-mx)*(y-my)
    return somme/n

# Test
print("--- Moyenne, variance, covariance---")
liste = [1,2,3,4,5]
print("liste : ",liste)
print("moyenne :",moyenne(liste))
print("variance :",variance(liste))
print("covariance de la liste avec elle même = variance :",covariance(liste,liste))

listex = [1,2,3,4,5]
listey = [4,5,4,7,6]
print("liste des x : ",listex)
print("liste des y : ",listey)
print("covariance  :",covariance(listex,listey))

##############################
## Question 2 ##

def regression_lineaire(points):
    listex = [x for (x,y) in points]
    listey = [y for (x,y) in points]
    a = covariance(listex,listey)/variance(listex)
    b = moyenne(listey)-a*moyenne(listex)
    return a,b

# Heures/notes
print("--- Régression linéaire ---")
eleves = [(0.25,5),(0.5,4),(0.75,7),(1,6),(1,7),(1,10),(1.5,9),(1.75,14),(2,9),(2,11)
    ↪ ,(2.25,15),(2.5,10),(2.5,13),(2.75,18),(3,13),(3,16)]

# Pour le cours
cours = [(1,13),(2,10),(2,15),(3,8),(3,14),(5,7),(5,10),(6,3),(7,5),(7,9),(8,4),(9,1),(9,5)
    ↪ ,(10,2)]

a,b = regression_lineaire(eleves)
print("a,b",a,b)

##############################
## Question 3 ##

import matplotlib.pyplot as plt
import numpy as np  # juste pour la grille de numérotation !!

def afficher(points):

    # Points
    n = len(points)
    for i in range(n):
        x,y = points[i]
        plt.scatter(x,y,color='blue')

    # Droite
    a,b = regression_lineaire(points)
    # Deux points P1 = (x1,y1), P2=(x2,y2) de la droite y = ax+b
    listex = [x for (x,y) in points]
    x1 = min(listex)-1
    y1 = a*x1 + b
    x2 = max(listex)+1
    y2 = a*x2 + b
    plt.plot([x1,x2],[y1,y2],color='red')

    # Axes
```

```
    # plt.axis('equal')
    plt.xlim(0,4)
    plt.ylim(0,20)
    plt.yticks(range(0,21,2))

    # Pour cours
    # plt.xlim(0,11)
    # plt.ylim(0,16)
    # plt.xticks(range(0,12,1))
    # plt.yticks(range(0,17,1))
    plt.grid()


    plt.show()
    plt.close()
    return

# Test
afficher(eleves)

##############################
## Question 4 ##

def combien_de_temps():
    note_str = input("Quelle note voudrais-tu obtenir ? ")
    y = float(note_str)
    a,b = regression_lineaire(eleves)
    x = (y-b)/a

    # heure_str = '{0:.2f}'.format(x)
    heures = int(x)
    minutes = int((x-heures)*60)
    print("Tu dois travailler au moins",str(heures),"heures et",str(minutes),"minutes.")
    return

# Test
print("--- Travailler plus pour gagner plus ---")
combien_de_temps()
```

Activité 6

Activité 6 bigdata_bayes_1.py

```
##############################
# Big data
##############################

##############################
# Activité 5 - Classification bayésienne naïve
##############################

from math import *

##############################
## Question A.1 ##

def moyenne(liste):
    n = len(liste)
    somme = sum(liste)
    if n==0:
```

```
        return 0
    else:
        return somme/n

def variance(liste):
    n = len(liste)
    m = moyenne(liste)
    somme = 0
    for x in liste:
        somme = somme + (x-m)**2
    return somme/n

# Test
print("--- Moyenne, variance ---")
liste = [1,2,3,4,5]
print("liste : ",liste)
print("moyenne :",moyenne(liste))
print("variance :",variance(liste))

print("--- Echantillon hommes ---")
# on veut obtenir environ muh = 178, (sigmah = 8), sigma2h = 64
taille_hommes = [172,165,187,181,167,184,168,174,180,186]

# Pour le cours
# taille_hommes = [181,170,186,175,169]  # Pour le cours

print("taille hommes : ",taille_hommes)
print("moyenne :",moyenne(taille_hommes))
print("variance :",variance(taille_hommes))
print("écart-type :",sqrt(variance(taille_hommes)))

print("--- Echantillon femmes ---")
# on veut obtenir environ muf = 166, (sigmaf = 7), sigma2f = 49
taille_femmes = [172,156,164,182,171,164,162,170,161,167]

# Pour le cours
# taille_femmes = [162,174,160,171,162]  # Pour le cours

print("taille femmes : ",taille_femmes)
print("moyenne :",moyenne(taille_femmes))
print("variance :",variance(taille_femmes))
print("écart-type :",sqrt(variance(taille_femmes)))


##############################
## Question A.2 ##

def densite_gauss(x,mu,sigma2):
    p =  1/sqrt(2*pi*sigma2)*exp( -1/2 * (x-mu)**2 / sigma2 )
    return p

# Test
print("--- Fonction de densité de la loi normale ---")
mu = 178         # moyenne
sigma2 = 64      # variance (= écart-type au carré)
x = 183
print(densite_gauss(x,mu,sigma2))

##############################
## Question A.3 ##

# Valeur taille homme/femme
muh = 178         # moyenne
sigma2h = 64      # variance
```

```
muf = 166         # moyenne
sigma2f = 49      # variance

# Pour le cours
# muh = 176         # moyenne
# sigma2h = 42      # variance
# muf = 166         # moyenne
# sigma2f = 32      # variance

def homme_ou_femme(taille):
    ph = densite_gauss(taille,muh,sigma2h)
    pf = densite_gauss(taille,muf,sigma2f)
    print('--- Homme ou femme ? ---')
    print('--- Taille donnée :',taille)
    print("Probabilité homme ",'{0:.8f}'.format(ph))
    print("Probabilité femme ",'{0:.8f}'.format(pf))
    if ph>pf:
        print("C'est plus probablement un homme.")
    else:
        print("C'est plus probablement une femme.")
    return

# Test
print("--- Homme ou femme par la taille ---")
taille = 170
# taille = 169  # pour le cours
homme_ou_femme(taille)

##############################
## Question A.4 ##

print("--- Echantillon hommes ---")
# mu_taille_h = 178 cm, (sigma_taille_h = 8), sigma2_taille_h = 64
# mu_poids_h = 75 kg, (sigma_poids_h = 10), sigma2_poids_h = 100
hommes = [(172,68),(165,71),(187,85),(181,73),(167,75),(184,93),(168,67),(174,83),(180,70)
    ↪ ,(186,73)]
taille_hommes = [x for x,y in hommes]
mu_taille_h = moyenne(taille_hommes)
sigma2_taille_h = variance(taille_hommes)

# print("liste tailles: ",taille_hommes)
# print("moyenne :",moyenne(taille_hommes))
# print("variance :",variance(taille_hommes))
# print("écart-type :",sqrt(variance(taille_hommes)))

poids_hommes = [y for x,y in hommes]
mu_poids_h = moyenne(poids_hommes)
sigma2_poids_h = variance(poids_hommes)
# print("liste poids: ",poids_hommes)
# print("moyenne :",moyenne(poids_hommes))
# print("variance :",variance(poids_hommes))
# print("écart-type :",sqrt(variance(poids_hommes)))

print("--- Echantillon femmes ---")
# mu_taille_f = 166 cm, (sigma_taille_f = 7), sigma2_taille_f = 49
# mu_poids_f = 63 kg, (sigma_poids_f = 10), sigma2_poids_f = 100
femmes = [(172,66),(156,57),(164,48),(182,71),(171,55),(164,68),(162,52),(170,68),(161,76)
    ↪ ,(167,67)]
taille_femmes = [x for x,y in femmes]
mu_taille_f = moyenne(taille_femmes)
sigma2_taille_f = variance(taille_femmes)
```

```
# print("liste tailles: ",taille_femmes)
# print("moyenne :",moyenne(taille_femmes))
# print("variance :",variance(taille_femmes))
# print("écart-type :",sqrt(variance(taille_femmes)))

poids_femmes = [y for x,y in femmes]
mu_poids_f = moyenne(poids_femmes)
sigma2_poids_f = variance(poids_femmes)

# print("liste poids: ",poids_femmes)
# print("moyenne :",moyenne(poids_femmes))
# print("variance :",variance(poids_femmes))
# print("écart-type :",sqrt(variance(poids_femmes)))

def homme_ou_femme_bis(taille,poids):
    p_taille_h = densite_gauss(taille,mu_taille_h,sigma2_taille_h)
    p_poids_h = densite_gauss(poids,mu_poids_h,sigma2_poids_h)
    p_taille_f = densite_gauss(taille,mu_taille_f,sigma2_taille_f)
    p_poids_f = densite_gauss(poids,mu_poids_f,sigma2_poids_f)
    ph = p_taille_h*p_poids_h
    pf = p_taille_f*p_poids_f
    print('--- Homme ou femme ? ---')
    print('--- Taille donnée :',taille)
    print('--- Poids donné   :',poids)
    print("Probabilité homme ",ph)
    print("Probabilité femme ",pf)
    if ph>pf:
        print("C'est plus probablement un homme.")
    else:
        print("C'est plus probablement une femme.")
    return

# Test
print("--- Homme ou femme par la taille et le poids ---")
taille = 176
poids = 64
homme_ou_femme_bis(taille,poids)
```

Activité 7

Activité 7 bigdata_bayes_1.py

```
##############################
# Big data
##############################

##############################
# Activité 6 - Classification bayésienne naïve (suite)
##############################

# Inspiré par un post de Bruno Stecanella
# https://monkeylearn.com/blog/practical-explanation-naive-bayes-classifier/

titres_sport = [
"un beau match de championnat",
"victoire de Paris en finale",
"défaite à Marseille",
"le coach viré après la finale",
"Paris change de coach"
```

```
]

titres_passport = [
"un beau printemps à Paris",
"un robot écrase un chien à Marseille",
"célébration de la victoire de la grande guerre",
"grève finale au lycée"
]

# Sport ou pas ?
phrase = "victoire de Marseille"
# phrase = "un beau chien"
# phrase = "Paris écrase Barcelone en finale"

##############################
## Question B.1 ##

def liste_mots(titres):
    liste = []
    for titre in titres:
        liste = liste + titre.split()
    return liste

mots_sports = liste_mots(titres_sport)
mots_passports = liste_mots(titres_passport)

print(mots_sports)
print(mots_passports)

## Question B.2 ##

def probabilite_mot(mot,liste_mots):
    nb = liste_mots.count(mot)
    nb_total = len(liste_mots)
    # print(mot,nb,nb_total)
    return nb/nb_total

# Test
print("--- Probabilité mot sachant sports (ou pas sport) ---")
mot = "Paris"
p_mot_sport = probabilite_mot(mot,mots_sports)
p_mot_passport = probabilite_mot(mot,mots_passports)
print("mot :",mot)
print("Donc probabilité du mot sachant sport =",p_mot_sport)
print("Donc probabilité du mot sachant pas sport =",p_mot_passport)

## Question B.3 ##

def probabilite_phrase(phrase,liste_mots):
    # Produit des probas de chaque mot
    nb_total = len(liste_mots)
    liste = phrase.split()
    p = 1
    for mot in liste:
        proba_mot = probabilite_mot(mot,liste_mots)
        p = p*proba_mot

    return p

# Test
print("--- Probabilité classique ---")
# phrase = "victoire de Marseille"
# phrase = "un beau chien"
# phrase = "Paris écrase Barcelone en finale"
```

```
phrase = "la finale de Paris"
phrase = "le coach perd la finale"

p_sport = probabilite_phrase(phrase,mots_sports)
p_passport = probabilite_phrase(phrase,mots_passports)
print("proba phrase sport =",p_sport)
print("proba phrase pas sport =",p_passport)
# print("Rapport =",p_sport/p_passport)

## Question B.4 ##

def probabilite_mot_bis(mot,liste_mots):
    nb = liste_mots.count(mot) + 1  # on ajoute 1, donc jamais nul
    nb_total = len(liste_mots)
    return nb/nb_total              # c'est plus un vraie proba

def probabilite_phrase_bis(phrase,liste_mots):
    # Produit des probas de chaque mot
    nb_total = len(liste_mots)
    liste = phrase.split()
    p = 1
    for mot in liste:
        proba_mot = probabilite_mot_bis(mot,liste_mots)
        p = p*proba_mot

    return p

# Test
print("--- Probabilité modifiée ---")

p_sport = probabilite_phrase_bis(phrase,mots_sports)
p_passport = probabilite_phrase_bis(phrase,mots_passports)
print("proba phrase sport =",p_sport)
print("proba phrase pas sport =",p_passport)
print("Rapport =",p_sport/p_passport)
```

23. Big data II

Activité 1

Activité 1 bigdata_voisins.py

```
################################
# Big data
################################

from math import *
from random import *
import matplotlib.pyplot as plt

################################
# Activité - Voisins
################################

################################
## Question 1 ##

## Question 1.a ##
```

```
def afficher_cpoints(cpoints):

    for x,y,c in cpoints:
        if c == 0:
            plt.scatter(x,y,color='red',s=100)
        elif c == 1:
            plt.scatter(x,y,color='blue',marker="s",s=100)
        elif c == 2:
            plt.scatter(x,y,color='red',s=100)
        elif c == 3:
            plt.scatter(x,y,color='blue',marker="s",s=100)
        elif c == 4:
            plt.scatter(x,y,color='black',marker="*",s=100)

    plt.axes().set_aspect('equal')
    plt.xlim(xmin-0.5,xmax+0.5)
    plt.ylim(ymin-0.5,ymax+0.5)
    # plt.xticks(range(xmin,xmax+1,10))
    # plt.yticks(range(ymin,ymax+1,10))
    plt.grid()
    plt.show()
    plt.close()
    return

# Test
print("--- Affichage de cpoints ---")
xmin,xmax,ymin,ymax = 0,10,0,10 # Fenêtre des cpoints
cpoints = [(2,3,0),(5,7,1)]
# afficher_cpoints(cpoints)

## Question 1.b ##

def fonction_couleur(x,y):

    res = ((x**2+3*y**2) % 100) - 50

    # res = (0.1*(y-ymax/2))**2 - (0.1*(x-xmax/2))**3 + (0.1*(x-xmax/2)) - xmax/100
    # res = (x-xmax/2)**3-3*(x-xmax/2)*(y-ymax/2)**2 - xmax

    # res = randint(0,1)

    if res > 0:
        return 1
    else:
        return 0

## Question 1.c ##

def generer_cpoints(N):

    cpoints = []
    for __ in range(N):
        x = int((xmax-xmin)*random())
        y = int((ymax-ymin)*random())
        c = fonction_couleur(x,y)
        if ((x,y,0) not in cpoints) or ((x,y,1) not in cpoints):
            cpoints = cpoints + [(x,y,c)]

    return cpoints

# Test
xmin,xmax,ymin,ymax = 0,100,0,100 # Fenêtre des cpoints
cpoints = generer_cpoints(30)
# afficher_cpoints(cpoints)
```

```
##############################
## Question 2 ##

## Question 2.a ##

def distance(P,Q):
    x1,y1 = P
    x2,y2 = Q
    d = sqrt((x2-x1)**2 + (y2-y1)**2)
    return d

## Question 2.b ##

def un_voisin_proche(P,cpoints):
    dmin = 1000  # un très grand nb ou bien l'infini de Python : 'inf'

    for Qc in cpoints:
        d = distance(P,Qc[0:2])
        if d < dmin:
            dmin = d
            Qcmin = Qc

    return Qcmin

# Test
xmin,xmax,ymin,ymax = 0,10,0,10 # Fenêtre des cpoints
P = (4,3)
cpoints = [ (8,6,0), (1,2,0), (5,9,1), (6,2,1) ]
x,y,c = un_voisin_proche(P,cpoints)
Pc = (P[0],P[1],4)
# afficher_cpoints(cpoints+[Pc])
print(cpoints)
print(Pc)


##############################
## Question 3 ##

def colorier_par_un_voisin_proche(cpoints):

    for x,y,c in cpoints:
        if c == 0:
            plt.scatter(x,y,color='red',s=75)
        elif c == 1:
            plt.scatter(x,y,color='blue',s=75,marker="s")

    pts = [ (x,y) for x,y,c in cpoints]  # cpoints références

    for x in range(xmin,xmax+1):
        for y in range(ymin,ymax+1):
            if (x,y) not in pts:
                c = un_voisin_proche((x,y),cpoints)[2]
                if c == 0:
                    plt.scatter(x,y,color='red',s=20, alpha=0.7)
                elif c == 1:
                    plt.scatter(x,y,color='blue',s=20,marker="s", alpha=0.7)

    plt.axes().set_aspect('equal')
    plt.xlim(xmin-0.5,xmax+0.5)
    plt.ylim(ymin-0.5,ymax+0.5)
    plt.grid()
    plt.show()
    plt.close()
```

```
    return

# Test
xmin,xmax,ymin,ymax = 0,10,0,10
cpoints = [ (5,4,0), (5,9,1), (2,8,0), (6,1,1) ]
# afficher_cpoints(cpoints)
# colorier_par_un_voisin_proche(cpoints)

xmin,xmax,ymin,ymax = 0,40,0,40
cpoints = generer_cpoints(20)
# afficher_cpoints(cpoints)
# colorier_par_un_voisin_proche(cpoints)

##############################
## Question 4 ##

## Question 4.a ##

def les_voisins_proches(P,cpoints,k):

    liste_voisins = []
    liste_distances = []

    for Qc in cpoints:

        d = distance(P,Qc[0:2])

        # On commence par prendre les k premiers
        if len(liste_voisins) < k:
            liste_voisins = liste_voisins + [Qc]
            liste_distances = liste_distances + [d]

        else:
            dmax = max(liste_distances)  # le maximum parmi les plus petites distances !
            if d < dmax:
                imax = liste_distances.index(dmax)
                del liste_voisins[imax]
                del liste_distances[imax]
                liste_voisins = liste_voisins + [Qc]
                liste_distances = liste_distances + [d]

    return liste_voisins

# Test
print("--- Voisins proches ---")
P = (18,45)
cpoints = [ (20,10,0), (20,59,1), (15,50,0) ]
voisins = les_voisins_proches(P,cpoints,2)
print(voisins)

## Question 4.b ##

def couleur_majoritaire(cpoints):
    nb_zero = len([pt for pt in cpoints if pt[2]==0])
    nb_un = len(cpoints) - nb_zero
    if nb_zero >= nb_un:
        return 0
    else:
        return 1

# Test
print("--- Couleur majoritaire ---")
cpoints = [ (20,10,0), (20,59,1), (15,50,1), (5,5,0) ]
print(couleur_majoritaire(cpoints))
```

```
## Question 4.c ##
def colorier_par_les_voisins_proches(cpoints,k):
    for x,y,c in cpoints:
        if c == 0:
            plt.scatter(x,y,color='red',s=50)
        elif c == 1:
            plt.scatter(x,y,color='blue',s=50,marker="s")
    pts = [ (x,y) for x,y,c in cpoints]  # cPoints références
    for x in range(xmin,xmax+1):
        for y in range(ymin,ymax+1):
            if (x,y) not in pts:
                voisins = les_voisins_proches((x,y),cpoints,k)
                c = couleur_majoritaire(voisins)
                if c == 0:
                    plt.scatter(x,y,color='red',s=20, alpha=0.7)
                elif c == 1:
                    plt.scatter(x,y,color='blue',s=20,marker="s", alpha=0.7)
    plt.axes().set_aspect('equal')
    plt.xlim(xmin-0.5,xmax+0.5)
    plt.ylim(ymin-0.5,ymax+0.5)
    plt.grid()
    plt.show()
    plt.close()
    return
xmin,xmax,ymin,ymax = 0,10,0,10
cpoints = [ (3,2,0), (6,4,1), (3,5,1), (2,8,0), (3,6,0), (7,7,1), (9,4,0) ]
# voisins = les_voisins_proches((2,1),cpoints,3)
# print(voisins)
# print(couleur_majoritaire(voisins))
# afficher_cpoints(cpoints)
# colorier_par_les_voisins_proches(cpoints,1)
# colorier_par_les_voisins_proches(cpoints,2)
# colorier_par_les_voisins_proches(cpoints,3)
xmin,xmax,ymin,ymax = 0,50,0,50
cpoints = generer_cpoints(100)
afficher_cpoints(cpoints)
colorier_par_les_voisins_proches(cpoints,3)
colorier_par_les_voisins_proches(cpoints,5)
colorier_par_les_voisins_proches(cpoints,7)
```

Activité 2

Activité 2 bigdata_parentheses.py

```
##############################
# Big data
##############################

from random import *
```

```
##############################
# Activité - Parenthèses
##############################

##############################
# Rappels activité "Piles" du livre 1
##############################

def empile(element):
    global pile
    pile = pile + [element]
    return None

def depile():
    global pile
    sommet = pile[len(pile)-1]
    pile = pile[0:len(pile)-1]
    return sommet

def pile_est_vide():
    if len(pile) == 0:
        return True
    else:
        return False

def parentheses_correctes(expression):
    """ Teste si une expression est bien parenthésée
    Entrée : un expression (chaîne de caractère)
    Sortie : vrai/faux
    Action : utilise une pile """

    global pile
    pile = []    # On part d'une pile vide

    for car in expression:
        if car == "(":
            empile(car)

        if car == ")":
            if pile_est_vide():
                return False     # Problème : il manque une "("
            else:
                depile()

    # A la fin :
    if pile_est_vide():
        return True
    else:
        return False

def crochets_parentheses_correctes(expression):
    """ Teste si une expression a des crochets et des parenthèses bien placées
    Entrée : un expression (chaîne de caractère)
    Sortie : vrai/faux
    Action : utilise une pile """

    global pile
    pile = []    # On part d'une pile vide

    for car in expression:

        if car == "(" or car == "[":
            empile(car)

        if car == ")" or car == "]":
```

```
            if pile_est_vide():
                return False     # Problème : il manque "(" ou "["
            else:
                element = depile()
                if element == "[" and car == ")":
                    return False     # Problème du type [)
                if element == "(" and car == "]":
                    return False     # Problème du type (]

    # A la fin
    return pile_est_vide()

# Test
print("--- Vérification exacte - Parenthèses ---")
exp = "()()(())(()(()))"  # Vrai
# exp = "()()(())((((()))"  # Faux
print(exp)
print("Verification exacte ok ?:",parentheses_correctes(exp))

print("--- Vérification exacte - Parenthèses et crochets ---")
exp = "()()(())[[](())]"  # Vrai
# exp = "()[(())((([]))"  # Faux
print(exp)
print("Verification exacte ok ?:",crochets_parentheses_correctes(exp))


##############################
## Partie A - Parenthèses seules ##
##############################

print("\n=== Partie A. Parenthèses ===\n")

##############################
## Question 0 ##

def construction_parentheses(n):
    """ Constructions d'une expression aléatoire de longueur 2n
    avec parenthèses cohérentes. """
    p = 0.5    # probalibilités
    exp = ""   # Expression
    pile = []  # Pile
    k = 0      # Nombre de pairs
    while k<n:
        x = random()    # Nb aléatoire 0 <= x < 1

        if 0 <= x < p:  # Parenthèses
            exp += "("
            pile.append(")")
            k = k+1

        if p <= x < 1 and len(pile)>0:  # Dépiler
            el = pile.pop()
            exp += el

    # A la fin vider la pile
    while len(pile)>0:
        el = pile.pop()
        exp += el

    return exp

# Test
# print("--- Constructions d'une expression ---")
# exp = construction_parentheses(10)
# print(exp)
```

```
# print("Verif ok ?:",parentheses_correctes(exp))

##############################
## Question 2 ##

def test_parentheses(expression,p=101,a=2):
    """ Vérification probabiliste qu'une expression
    est correctement parenthésées """

    # p est un nombre premier
    # a est un entier pour les parenthèses

    S = 0    # somme de contrôle
    h = 0    # hauteur
    for car in expression:
        if car == "(":
            h = h + 1
            S = (S + a**h) % p
        if car == ")":
            S = (S - a**h) % p
            h = h - 1
        if h < 0:
            return False

    if S == 0:
        return True
    else:
        return False

 # Test
print("--- Test probabiliste d'une expression : parenthèses ---")
exp = construction_parentheses(20)
print(exp)
exp = "()()(())(()(()))"  # Vrai
# exp = "()()(())((((()))"  # Faux
exp = ")("
print("Test proba ok ?:",test_parentheses(exp))
print("Verification exacte ok ?:",parentheses_correctes(exp))

##############################
## Partie B - Parenthèses et crochets ##
##############################

print("\n=== Partie B. Parenthèses et crochets ===\n")

##############################
## Question 0 ##

def construction_crochets_parentheses(n):
    """ Constructions d'une expression aléatoire de longueur 2n
    avec parenthèses et crochets cohérents. """
    p = 0.33    # probalibilités
    q = 0.66
    exp = ""   # Expression
    pile = []  # Pile
    k = 0      # Nombre de pairs
    while k<n:
        x = random()    # Nb aléatoire 0 <= x < 1

        if 0 <= x < p:  # Parenthèses
            exp += "("
            pile.append(")")
            k = k+1
```

```
        if p <= x < q:  # Crochets
            exp += "["
            pile.append("]")
            k = k+1

        if q <= x < 1 and len(pile)>0:  # Dépiler
            el = pile.pop()
            exp += el

    # A la fin vider la pile
    while len(pile)>0:
        el = pile.pop()
        exp += el

    return exp

# Test
# print("--- Constructions d'une expression ---")
# exp = construction_crochets_parentheses(20)
# print(exp)
# print("Verif ok ?:",crochets_parentheses_correctes(exp))

##############################
## Question 1 ##

def test_crochets_parentheses(expression):
    p = 101  # un nombre premier
    a = 2    # entier pour les parenthèses
    b = 3    # entiers pour les crochets
    S = 0    # somme de contrôle
    h = 0    # hauteur
    for car in expression:
        if car == "(":
            h = h + 1
            S = (S + a**h) % p
        if car == ")":
            S = (S - a**h) % p
            h = h - 1
        if car == "[":
            h = h + 1
            S = (S + b**h) % p
        if car == "]":
            S = (S - b**h) % p
            h = h - 1
        if h < 0:
            return False

    if S == 0:
        return True
    else:
        return False

 # Test
print("--- Test probabiliste d'une expression : parenthèses et crochets ---")
exp = construction_crochets_parentheses(20)
exp = "[([[]][[[[(()[(()([((()))[[]])]))])]]]])]"  # vrai
exp = "[[[()](()[((([[]])()(([[[]][()]]))))])]]"  # Faux
#exp = "[[[()](()[((([[]])()([[[[]][()]]))))])])"  # Faux
#exp = "[)(]"  # Faux
print(exp)
```

```
print("Test proba ok ?:",test_crochets_parentheses(exp))
print("Verification exacte ok ?:",crochets_parentheses_correctes(exp))
```

Activité 3

Activité 3 bigdata_barycentres.py

```
##############################
# Big data
##############################

from math import *
from random import *
import matplotlib.pyplot as plt

##############################
# Activité 3 - Barycentres
##############################

##############################
## Question 1 ##

xmin,xmax,ymin,ymax = 0,100,0,100 # Fenêtre des points

def afficher_points(points,couleurs):

    liste_couleurs = ["red","blue","green","orange","cyan","chartreuse","purple","black","
    ↪ pink"]
    n = len(points)

    for i in range(n):
        x,y = points[i]
        coul = couleurs[i]
        plt.scatter(x,y,color=liste_couleurs[coul],s=100)

    plt.axes().set_aspect('equal')
    plt.xlim(xmin-1,xmax+1)
    plt.ylim(ymin-1,ymax+1)
    plt.grid()
    plt.show()
    plt.close()
    return

# Test
points = [(20,20),(60,40),(40,80),(70,70)]
couleurs = [0,1,2,0]
# afficher_points(points,couleurs)

##############################
## Question 2 ##

def generer_points(k=3,nb_points=50,dispersion=20):

    points = []
    for __ in range(k):
        # centre
        xc = xmin + (xmax-xmin)*(0.2+0.6*random())
        yc = ymin + (ymax-ymin)*(0.2+0.6*random())

        n = 0
```

```
        while n < nb_points:
            theta = 2*pi*random()
            r = dispersion * random()**2
            x = xc + r*cos(theta)
            y = yc + r*sin(theta)
            if (xmin <= x <= xmax) and (ymin <= y <= ymax):
                points = points + [(x,y)]
                n = n+1

    return points

# Test
points = generer_points(k=1,nb_points=100,dispersion=20)
couleurs = [0] * len(points)
# afficher_points(points,couleurs)

##############################
## Question 3 ##

def calcul_barycentre(points):
    """ Calcul les coordonnée (xG,yG) du barycentres des points) """
    n = len(points)
    if n == 0: return None
    xG = sum([pt[0] for pt in points ]) / n
    yG = sum([pt[1] for pt in points ]) / n
    return xG,yG

# Test
points = [(20,60),(40,20),(60,80)]
bar = calcul_barycentre(points)
# print("Points :",points)
# print("Barycentre :",bar)
# points = points + [bar]
# afficher_points(points,[0,0,0,1])

##############################
## Question 4 ##

## Question 4.a ##

def distance(P,Q):
    """ Calcule la distance entre deux points """
    x1,y1 = P
    x2,y2 = Q
    d = sqrt((x2-x1)**2 + (y2-y1)**2)
    return d

## Question 4.b ##

def barycentre_proche(P,barycentres):
    """ Renvoie le rang (c-à-d la couleur) du barycentre le plus proche """
    i0 = 0
    bar =  barycentres[0]
    d0 = distance(P,bar)
    for i in range(len(barycentres)):
        bar = barycentres[i]
        d = distance(P,bar)
        if d < d0:
            i0 = i
            d0 = d
    return i0

# Test
```

```
barycentres = [(80,40),(20,80),(60,60)]
point = (40,40)
# print(barycentre_proche(point,barycentres))
# afficher_points(barycentres+[point],[0,0,0,7])


##############################
## Question 4.c ##

def couleurs_barycentres_proches(points,barycentres):
    """ Pour chaque point renvoie la couleur c-à-d le rang du barycentre le plus proche """
    couleurs = []
    for pt in points:
        num_bar = barycentre_proche(pt,barycentres)
        couleurs = couleurs + [num_bar]
    return couleurs

# Test
barycentres = [(80,40),(20,80),(60,60)]
points = [(40,40),(20,40),(80,70),(50,10)]
# points = generer_points()
# afficher_points(points + barycentres,[7,7,7,7] + [0,1,2])
# couleurs = couleurs_barycentres_proches(points,barycentres)
# afficher_points(points + barycentres,couleurs + [0,1,2])


##############################
## Question 5 ##

def recalculer_barycentres(points,barycentres):
    k = len(barycentres)
    n = len(points)
    nouv_barycentres = []
    couleurs = couleurs_barycentres_proches(points,barycentres)
    for c in range(k):
        liste_partielle = []
        for i in range(n):
            if couleurs[i] == c:
                liste_partielle = liste_partielle + [points[i]]
        if len(liste_partielle) == 0:
            (x0,y0) = barycentres[c]
        else:
            (x0,y0) = calcul_barycentre(liste_partielle)
        nouv_barycentres = nouv_barycentres + [(x0,y0)]
    return nouv_barycentres

# Test
barycentres = [(80,40),(20,80),(40,20)]
# points = generer_points()
points = [(10,30),(30,20),(80,20),(30,40),(40,40),(20,50),(80,70),(40,70),(50,10),(70,60)]
n = len(points)
# afficher_points(points + barycentres,[7]*n + [3,4,5])
# couleurs = couleurs_barycentres_proches(points,barycentres)
# afficher_points(points + barycentres,couleurs + [3,4,5])
# nouv_barycentres = recalculer_barycentres(points,barycentres)
# afficher_points(points + nouv_barycentres,couleurs + [3,4,5])
# print(nouv_barycentres)

##############################
## Question 5 ##
```

```
def iterer_barycentres(points,barycentres_init):
    barycentres = list(barycentres_init)
    k = len(barycentres)
    couleurs = couleurs_barycentres_proches(points,barycentres)

    while True:
        afficher_points(points + barycentres,couleurs + [7]*k)

        nouv_barycentres = recalculer_barycentres(points,barycentres)
        nouv_couleurs = couleurs_barycentres_proches(points,nouv_barycentres)

        barycentres = list(nouv_barycentres)

        if couleurs == nouv_couleurs:
            break

        couleurs = list(nouv_couleurs)

    return barycentres

# Test
# mon_k = 5
# points = generer_points(k=mon_k,nb_points=30,dispersion=20)
# print(points)
# afficher_points(points,[0]*len(points))
# barycentres_init = generer_points(k=mon_k,nb_points=1)
# barycentres_init = [(20,40),(80,40),(65,70),(50,20),(25,70)]
# barycentres = iterer_barycentres(points,barycentres_init)
# couleurs = couleurs_barycentres_proches(points,barycentres)
# afficher_points(points + barycentres,couleurs + [7]*mon_k)
# print(barycentres)

# Pour le cours
# import exemple_barycentres
# points = exemple_barycentres.points
# barycentres_init = exemple_barycentres.barycentres_init
# afficher_points(points,[0]*len(points))
# barycentres = iterer_barycentres(points,barycentres_init)
# couleurs = couleurs_barycentres_proches(points,barycentres)
# afficher_points(points + barycentres,couleurs + [7]*3)

# Pour le cours
import exemple_barycentres_bis
points = exemple_barycentres_bis.points
barycentres_init = exemple_barycentres_bis.barycentres_init
afficher_points(points,[0]*len(points))
barycentres = iterer_barycentres(points,barycentres_init)
couleurs = couleurs_barycentres_proches(points,barycentres)
afficher_points(points + barycentres,couleurs + [7]*5)

##############################
## Question 6 - à virer ##

def erreur_barycentres(points,barycentres):

    n = len(points)
    k = len(barycentres)
    couleurs = couleurs_barycentres_proches(points,barycentres)
    err = 0

    for c in range(k):
        bar = barycentres[c]
        for i in range(n):
            if couleurs[i] == c:
                err = err + distance(points[i],bar)
```

```
    return err

# Test
# erreur  = erreur_barycentres(points,barycentres)
# print(erreur)
```

Activité 4

Activité 4 bigdata_neurone.py

```
##############################
# Big data II
##############################

##############################
# Activité 4 - Neurone
##############################

# Un neurone : 3 coeff réels
p1 = 1
p2 = 0.2
p3 = 0.2
neurone = [p1,p2,p3]

# le seuil vaut 1 : seuil = 1

# vitesse d'apprentissage
epsilon = 0.2

# Une entrée : 3 réels [r,g,b] entre 0 et 1
entree = [1,1,1]

##############################
## Question 1 ##

def activation(neurone,entree):
    p1,p2,p3 = neurone
    e1,e2,e3 = entree
    q = p1*e1 + p2*e2 + p3*e3
    if q >= 1:
        sortie = 1
    else:
        sortie = 0
    return sortie

# Test
print("--- Activation (ou pas) ---")
p1 = 1
p2 = 2
p3 = 3
# neurone = [2,-2,-2]
# entree = [0,1,0]
# entree = [0,0,1]
# Pour les exemples
neurone, entree = [1,2,3], [0.5,0,0]
neurone, entree = [1,2,3], [0,1,0.5]

neurone, entree = [1,0.5,2], [0.2,0.1,0.1]
neurone, entree = [1,0.5,2], [0.3,0.2,0.7]
```

```
print(activation(neurone,entree))

#####################################
## Question 2 ##
def apprentissage(neurone,entree,objectif):
    p1,p2,p3 = neurone
    e1,e2,e3 = entree
    # epsilon = 0.2
    sortie = activation(neurone,entree)
    if sortie == objectif:
        nouv_neurone = list(neurone)
    else:
        pp1 = p1 + epsilon * (objectif-sortie)*e1
        pp2 = p2 + epsilon * (objectif-sortie)*e2
        pp3 = p3 + epsilon * (objectif-sortie)*e3
        nouv_neurone = [pp1,pp2,pp3]
    return nouv_neurone
# Test
print("--- Apprentissage pas à pas ---")

# Cas où objectif atteint : on garde le neurone inchangé
neurone = [1,1,1]
entree = [1,0,2]
objectif = 1
print(neurone)
neurone = apprentissage(neurone,entree,objectif)
print(neurone)
# Cas où objectif atteint : on garde le neurone inchangé
neurone = [1,1,1]
entree = [0.5,0.1,0.2]
objectif = 0
print(neurone)
neurone = apprentissage(neurone,entree,objectif)
print(neurone)
# Cas où objectif pas atteint : on change le neurone
neurone = [1,1,1]
entree = [0,1,1]
objectif = 0
print(neurone)
neurone = apprentissage(neurone,entree,objectif)
print(neurone)  # Le neurone a changé
# Cas où objectif pas atteint : on change le neurone
neurone = [1,1,1]
entree = [0.5,0.2,0]
objectif = 1
print(neurone)
neurone = apprentissage(neurone,entree,objectif)
print(neurone)  # Le neurone a changé
# On peut itérer :
neurone = apprentissage(neurone,entree,objectif)
print(neurone)  # Le neurone a encore changé

# neurone = apprentissage(neurone,[0.9,0.9,0],0)
# print(neurone)
```

```
# neurone = apprentissage(neurone,[1,1,1],0)
# print(neurone)
# neurone = apprentissage(neurone,[0,1,1],0)
# print(neurone)
# neurone = apprentissage(neurone,[0,0,1],0)
# print(neurone)
# neurone = apprentissage(neurone,[0.9,0.9,0],0)
# print(neurone)
# neurone = apprentissage(neurone,[1,1,1],0)
# print(neurone)
# neurone = apprentissage(neurone,[0,0,1],0)
# print(neurone)
# neurone = apprentissage(neurone,[0.9,0.9,0],0)
# print(neurone)
# neurone = apprentissage(neurone,[1,1,1],0)
# print(neurone)

# neurone = apprentissage(neurone,[0.9,0.1,0],1)
# print(neurone)
# neurone = apprentissage(neurone,[1,0,0],1)
# print(neurone)
# neurone = apprentissage(neurone,[1,0,0],1)
# print(neurone)

#######################################
## Question 3 ##

def epoque_apprentissage(neurone_init,liste_entrees_objectifs):
    neurone = list(neurone_init)
    for entree,objectif in liste_entrees_objectifs:
        # print(entree,"neurone avant",neurone)
        neurone = apprentissage(neurone,entree,objectif)
        # print(" -> neurone après",neurone)
    return neurone

# Test
print("--- Apprentissage automatique ---")

neurone_init = [1,1,1]
liste_entrees_objectifs = [
([1,0,0],1),
([0,1,1],0),
([1,1,0],0),
([1,0,0.2],1),
([0,1,0],0),
([0,0,0],0),
([1,0,1],0),
([0.7,0,0],1),
([0.5,0.5,0.5],0),
([0.9,0.2,0],1),
([0.9,0,0],1),
([1,1,1],0),
([0.2,1,0],0),
([0.8,0.2,0],1),
([0.7,0.1,0.1],1)
]

print(neurone_init)
neurone = list(neurone_init)
for __ in range(10):
    neurone = epoque_apprentissage(neurone,liste_entrees_objectifs)
```

```
    print(neurone)

# Vérification du neurone
print("=== Vérification ===")
neurone = [1.66,-0.78,-0.66]  # donner par apprentissagege ci-dessus
liste_couleurs = [ [0.9,0,0], [1,0.2,0.2], [0,0,1], [1,0.5,0], [0.3,0.3,0.3], [0.7,0.5,0.4]
    ↪ ]
p1,p2,p3 = neurone
for couleur in liste_couleurs:
    e1,e2,e3 = couleur
    q = p1*e1 + p2*e2 + p3*e3
    print("couleur = ",couleur)
    print("q =",q)
    print("rouge ?",activation(neurone,couleur))
```