1. Premiers pas

Activité 1

Activité 1 premiers_pas_1.py

```
##############################
# Premiers pas
##############################

##############################
# Activité 1 - Nombres
##############################

##############################
# Cours
print("--- Cours ---")
# Afficher une phrase
print("Bonjour le monde !")

# Addition
5+7
print(5+7)

# Multiplication
print(6*7)

print(3*(12+5))

print(3*1.5)

# Puissance

print(3**2)

print(10**-3)

# Division réelle

print(14/4)

print(1/3)

# Division entière et modulo

print(14//4)

print(14%4)


##############################
# Questions

# Q1
# Nombre de secondes dans un siècle
print("--- Question 1 ---")
print(100 * 365 * 24 * 60 * 60)

# Q2
# A partir dans quand plus grand qu'un milliard
print("--- Question 2 ---")
print((1+2)*(3+4)*(5+6)*(7+8)*(9+10)*(11+12)*(13+14)*(14+15))

# Q3
# Trois derniers chiffres de 123456789 * 123456789 * ...
```

```
print("--- Question 3 ---")
print(123456789 ** 7)

# Q4
# Premier 1/n avec période 7
print("--- Question 4 ---")
print(1/7)     # Période 6
print(1/239)   # Période 7

# Q5
# Trouver un nb connaissant deux divisions et un reste
print("--- Question 5 ---")
# for n in range(1175,1190):
#     print(n,n//11,n//13,n%7)

print(1182//11)
print(1182//13)
print(1182%7)
```

Activité 2

Activité 2 premiers_pas_2.py

```
################################
# Premiers pas
################################

################################
# Activité 2 - Variables
################################

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

# C1 - variables

a = 3  # Une variable
b = 5  # Une autre variable

print("La somme vaut",a+b)   # Affiche la somme
print("Le produit vaut",a*b) # Affiche le produit

c = b**a     # Nouvelle variable...
print(c)     # ... qui s'affiche

# C2 - aire d'un triangle

base = 8
hauteur = 3
aire = base * hauteur / 2
print(aire)
# print(Aire)  # !! Erreur !!

# C3 - ajout

S = 1000
S = S + 100
S = S + 200
S = S - 50
```

```
print(S)

##############################
# Questions

# Q1

# Aires - Volumes

# Trapèze : bien nommé les choses
B, b, h = 7, 4, 3
aire = (B + b) * h / 2
print("L'aire vaut",aire)

# Boîtes
L, l, h = 10,8,3
volume = L * l * h
print(volume)

# Boules
PI = 3.14
R = 10
aire = PI * R**2
print(aire)

# Q2
# Remettre dans l'ordre de sorte qu'à la fin x = 46
x = 7
y = 2*x
y = y - 1
x = x + 3*y
print(x)

# Q3
# Intérêts de 10%
S = 1000
S = S * 1.1
S = S * 1.1
S = S * 1.1

# Q4

# Bon choix afin d'échanger a et b

# Mauvais
a = 11
b = 9
a = b
b = a
print(a,b)

# Mauvais
a = 11
b = 9
c = b
a = b
b = c
print(a,b)

# Mauvais
a = 11
b = 9
c = a
```

```
a = c
c = b
b = c

print(a,b)

# Bon
a = 11
b = 9
c = a
a = b
b = c
print(a,b)
```

Activité 3

Activité 3 premiers_pas_3.py

```
################################
# Premiers pas
################################

################################
# Activité 3 - Utiliser des fonctions
################################

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

# C1 - fonctions

print("Coucou")

x = float("+1.234567")
print(x)

# C2 - module math

from math import *

x = sqrt(2)
print(x)
print(x**2)

# C3 - fonction trigo

angle = pi/2
print(angle)
print(sin(angle))

# C4 - décimal vers entier

x = 3.6
print(round(x))
print(floor(x))
print(ceil(x))

################################
# Questions

# Q1
```

```
# pgcd
print(gcd(13*121,13*122))

a = 101*103
b = 102*103
print(a,b)
ppcm = a * b // gcd(a,b)
print(ppcm)

# Q2
# Valeur absolue
x = 3.85
print(abs(x**2-15))
print(round(2*x))
print(floor(3*x))
print(ceil(4*x))

# Q3
# Angle
angle = pi/7
x = cos(angle)**2 + sin(angle)**2
print(x)
```

Activité 4

Activité 4 premiers_pas_4.py

```
################################
# Premiers pas
################################

from math import *

################################
# Activité 4 - Boucle "pour"
################################

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

# C1 - Boucle "for"

for i in range(10):
    print(i*i)

# C2 - Boucle "for"

somme = 0
for i in range(20):
    somme = somme + i
print(somme)

# C3

print(list(range(10)))
print(list(range(10,20)))
print(list(range(10,20,2)))
```

```
# C4 - Imbrication de boucles

for x in [10,20,30,40,50]:
    for y in [3,7]:
        print(x+y)

##############################
# Questions

# Q1
# Cubes
for i in range(101):
    print(i**3)

for i in range(10,21):
    print(i**4)

for i in range(0,101,5):
    print(sqrt(i))

# Q2
# Puissances de 2
for k in range(1,11):
    print(2**k)

# Q3
# Minimum d'une fonction par balayage
for i in range(101):
    x = i/100
    y = x**3 - x**2 - 1/4*x + 1
    print("x =",x,"y =",y)

# Q4
# Aire d'un disque qui vaut 100
for i in range(50):
    R = i/10
    V = 4/3 * 3.14 * R**3
    print("R =",R,"V =",V)
```

2. Tortue (Scratch avec Python)

Activité 1

Activité 1 tortue_1.py

```
##############################
# Tortue
##############################

##############################
# Activité 1 - Bouge ta tortue !
##############################

# Début du code

from turtle import *

width(5)    # Epaisseur du trait

# Lettre "P"
```

```
color('red')

left(90)        # 90 degrés à gauche
forward(200)    # On avance

right(90)
forward(100)

right(90)
forward(100)

right(90)
forward(100)

up()

# Fin du code

# Lettre "Y"

color('blue')

goto(200,0)
down()
setheading(90)
forward(120)
left(30)
forward(75)
backward(75)
right(60)
forward(75)

exitonclick()
```

Activité 2

Activité 2 tortue_2.py

```
################################
# Tortue
################################

################################
# Activité 2 - Boucle "pour"
################################

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

from turtle import *

# Un pentagone
width(5)
color('blue')

for i in range(5):
    forward(100)
    left(72)

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

```
# Un autre pentagone
color('red')

longueur = 200
angle = 72
for i in range(5):
    forward(longueur)
    left(angle)

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

# Un dodecagone (12 côtés quoi)

color("purple")
n = 12
angle = 360/n
for i in range(n):
    forward(100)
    left(angle)

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

# Une spirale

color("green")
longueur = 10
for i in range(25):
    forward(longueur)
    left(40)
    longueur = longueur + 10

exitonclick()
```

Activité 3

Activité 3 tortue_3.py

```
################################
# Tortue
################################

################################
# Activité 3 - Graphe d'une fonction
################################

from turtle import *
from math import *

speed("fastest")
width(2)
color('blue')
up()

for x in range(-200,200):
    if x == -199: down()
    # y = 1/ 100 * x ** 2   # Parabole
    y = 100*sin(1/20*x)     # Sinus
```

```
    goto(x,y)

exitonclick()
```

Activité 4

Activité 4 tortue_4.py

```
##############################
# Tortue
##############################

##############################
# Activité 4 - Boucle pour itérée - Triangle de Sierpinski
##############################

from turtle import *

width(5)
up()
goto(-100,-100)
down()

for i in range(3):
    color("blue")
    forward(256)
    left(120)

    for i in range(3):
        color("red")
        forward(128)
        left(120)

        for i in range(3):
            color("green")
            forward(64)
            left(120)

            # for i in range(3):
            #     color("orange")
            #     forward(32)
            #     left(120)

exitonclick()
```

Activité 5

Activité 5 tortue_5.py

```
##############################
# Tortue
##############################

##############################
# Activité 5 - Table de multiplications
##############################
```

```
from turtle import *
from math import *

speed("fastest")

n = 100
b = 2
r = 200

for i in range(n):
    up()
    goto(r*cos(2*i*pi/n),r*sin(2*i*pi/n))
    down()
    j = (b*i) % n
    goto(r*cos(2*j*pi/n),r*sin(2*j*pi/n))

exitonclick()
```

Activité 6

Activité 6 tortue_6.py

```
################################
# Tortue
################################

################################
# Activité 4 - Plusieurs tortues - Courbe de poursuite
################################

# Préparation

from turtle import *

tortue1 = Turtle()
tortue2 = Turtle()
tortue3 = Turtle()
tortue4 = Turtle()

tortue1.speed("fastest")
tortue2.speed("fastest")
tortue3.speed("fastest")
tortue4.speed("fastest")

tortue1.color('red')
tortue2.color('blue')
tortue3.color('orange')
tortue4.color('green')

# tortue1.width(5)
# tortue2.width(5)
# tortue3.width(5)
# tortue4.width(5)

tortue1.up()
tortue1.goto(-200,-200)
tortue1.down()

tortue2.up()
tortue2.goto(200,-200)
tortue2.down()
```

```
tortue3.up()
tortue3.goto(200,200)
tortue3.down()

tortue4.up()
tortue4.goto(-200,200)
tortue4.down()

print(tortue1.position())
print(tortue1.towards(0,0))

# Boucle principale

for i in range(40):

    position1 = tortue1.position()
    position2 = tortue2.position()
    position3 = tortue3.position()
    position4 = tortue4.position()

    tortue1.goto(position2)  # Va à la tortue suivante
    tortue1.goto(position1)       # Revient à sa place

    tortue2.goto(position3)
    tortue2.goto(position2)

    tortue3.goto(position4)
    tortue3.goto(position3)

    tortue4.goto(position1)
    tortue4.goto(position4)

    angle1 = tortue1.towards(position2)  # Récupèrer l'angle
    tortue1.setheading(angle1)   # S'oriente selon cet angle

    angle2 = tortue2.towards(position3)
    tortue2.setheading(angle2)

    angle3 = tortue3.towards(position4)
    tortue3.setheading(angle3)

    angle4 = tortue4.towards(position1)
    tortue4.setheading(angle4)

    tortue1.forward(10)    # Avance
    tortue2.forward(10)
    tortue3.forward(10)
    tortue4.forward(10)

exitonclick()
```

3. Si ... alors ...

Activité 1

Activité 1 sialors_1.py

```
##############################
# Si ... alors ...
##############################
```

```
##############################
# Activité 1 - Quiz multiplications
##############################

from random import *

a = randint(1,10)
b = randint(1,10)

print("Combien font",a,"fois",b,"?")
reponse_str = input("Réponse : ")
reponse_int = int(reponse_str)

if reponse_int == a*b:
    print("Bravo !")
else:
    print("Non, la réponse était",a*b)
```

Activité 2

Activité 2 sialors_2.py

```
##############################
# Si ... alors ...
##############################

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

from turtle import *

width(5)
color('blue')

mot = "AagAgAdAgAAgaAA"

for c in mot:

    if c == "A":
        forward(100)

    if c == "a":
        up()
        forward(100)
        down()

    if c == "g":
        left(90)

    if c == "d":
        right(90)

exitonclick()
```

Activité 3

Activité 3 sialors_3.py

```
##############################
# Si ... alors ...
##############################

##############################
# Activité 3 - Chiffres d'un nombre
##############################

##############################
## Question 1 ##
for d in range(10):
    for u in range(10):
        n  = 10*d + u
        print(n)

##############################
## Question 2 ##
for c in range(10):
    for d in range(10):
        for u in range(10):
            n  = 100*c + 10*d + u
            if u == 3 and (c+d+u >= 15) and (d == 0 or d == 2 or d == 4 or d == 6 or d== 8):
                print(n)

##############################
## Question 3 ##
compteur = 0
for c in range(10):
    for d in range(10):
        for u in range(10):
            n  = 100*c + 10*d + u
            if u == 3 and (c+d+u >= 15) and (d == 0 or d == 2 or d == 4 or d == 6 or d== 8):
                compteur = compteur + 1
print("Nombre de solutions :",compteur)
```

Activité 4

Activité 4 sialors_4.py

```
##############################
# Si ... alors ...
##############################

##############################
# Activité 4 - Triangles
##############################

# a,b,c = 3,4,5
```

```
################################
## Question 1 ##

a = 4
b = 5
c = 8
print("Triangle",a,b,c)

# A-t-on a <= b <= c ?

if a <= b and b <= c:
    print("Longueurs dans le bon ordre.")
else:
    print("Longueurs dans le mauvais ordre.")

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

# Un triangle peut-il être construit à partir de ces longueurs ?

if a+b >= c:
    print("Un tel triangle existe.")
else:
    print("Un tel triangle n'existe pas.")

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

# Le triangle est-il rectangle ?

if a**2 + b**2 == c**2:
    print("Le triangle est rectangle.")
else:
    print("Le triangle n'est pas rectangle.")

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

# Le triangle est-il equilatéral ?

if (a == b) and (b == c) and (a == c):
    print("Le triangle est équilatéral.")
else:
    print("Le triangle n'est pas équilatéral.")

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

# Le triangle est-il isocèle ?

if (a == b) or (b == c) or (a == c):
    print("Le triangle est isocèle.")
else:
    print("Le triangle n'est pas isocèle.")

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

# Tous les angles sont-ils aigus ?

cosalpha = (-a**2 + b**2 + c**2)/(2*b*c)
cosbeta = (a**2 - b**2 + c**2)/(2*a*c)
cosgamma = (a**2 + b**2 - c**2)/(2*a*b)

if (cosalpha >= 0) and (cosbeta >= 0) and (cosgamma >= 0):
```

```
    print("Tous les angles sont aigus.")
else:
    print("Tous les angles ne sont pas aigus. (Il existe un angle obtu).")
```

Activité 5

Activité 5 sialors_5.py

```
##############################
# Si ... alors ...
##############################

##############################
# Activité 4 - Le jeu de la devinette
##############################

##############################
## Question 1 ##
from random import *

# Jeu classique de la devinette

nb_mystere = randint(0,99)

for essai in range(7):
    print("Quel est le nombre mystère ?")
    reponse_str = input("Ta proposition : ")
    reponse_int = int(reponse_str)
    if nb_mystere == reponse_int:
        print("Bravo !")
        break   # Arrête la boucle
    elif nb_mystere < reponse_int:
        print("Non, le nombre à trouver est plus petit !")
    elif nb_mystere > reponse_int:
        print("Non, le nombre à trouver est plus grand !")

# Lorsque c'est fini :
if nb_mystere != reponse_int:
    print("Perdu ! Le nombre mystère était",nb_mystere)

##############################
## Question 2 ##
# Variante : l'ordinateur ment (1 fois sur 4)

# nb_mystere = randint(0,99)

# for essai in range(7):
#     print("Quel est le nombre mystère ?")
#     reponse_str = input("Ta proposition : ")
#     reponse_int = int(reponse_str)

#     # 1 fois sur 4 (environ) l'ordinateur ment
#     verite = True
#     hasard = randint(1,4)
#     if hasard == 4:
#         verite = False

#     if nb_mystere == reponse_int:
#         print("Bravo !")
```

```
#         break   # Arrête la boucle
#     elif nb_mystere < reponse_int:
#         if verite == True:
#             print("Non, le nombre à trouver est plus petit !")
#         else:
#             print("Non, le nombre à trouver est plus grand !")
#     elif nb_mystere > reponse_int:
#         if verite == True:
#             print("Non, le nombre à trouver est plus grand !")
#         else:
#             print("Non, le nombre à trouver est plus petit !")

# # Lorsque c'est fini :
# if nb_mystere != reponse_int:
#     print("Perdu ! Le nombre mystère était",nb_mystere)

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

# Variante : le nombre mystère change un peu

# nb_mystere = randint(0,99)

# for essai in range(7):
#     print("Quel est le nombre mystère ?")
#     reponse_str = input("Ta proposition : ")
#     reponse_int = int(reponse_str)

#     # Modification du nb mystère
#     hasard = randint(-3,3)
#     nb_mystere = nb_mystere + hasard
#     if nb_mystere < 1:
#         nb_mystere = 1
#     if nb_mystere > 99:
#         nb_mystere = 99

#     if nb_mystere == reponse_int:
#         print("Bravo !")
#         break   # Arrête la boucle
#     elif nb_mystere < reponse_int:
#         print("Non, le nombre à trouver est plus petit !")
#     elif nb_mystere > reponse_int:
#         print("Non, le nombre à trouver est plus grand !")

# # Lorsque c'est fini :
# if nb_mystere != reponse_int:
#     print("Perdu ! Le nombre mystère était",nb_mystere)
```

4. Fonctions

Activité 1

Activité 1 fonctions_1.py

```
##############################
# Fonctions - Idées
##############################
```

```
##############################
# Activité 1 - Introduction aux fonctions
##############################

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

# Fonction sans paramètre, sans sortie

def affiche_table_de_7():
    """ Affiche la table de 7 """

    print("--- Table de 7 ---")
    for i in range(1,11):
        print(i,"x 7 =",str(i*7))

    return

# Test
affiche_table_de_7()

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

def affiche_bonjour():
    """ Dit bonjour """

    prenom = input("Comment t'appelles-tu ? ")
    print("Bonjour",prenom)

    return

# Test
affiche_bonjour()


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

# Fonction avec paramètre, sans sortie

def affiche_une_table(n):
    """ Affiche la table de n """

    print("--- Table de",n,"---")
    for i in range(1,11):
        print(i,"x",n,"=",str(i*n))

    return

# Test
affiche_une_table(5)

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

def affiche_salutation(formule):
    """ Dit bonjour, bonsoir, au revoir... """

    prenom = input("Comment t'appelles-tu ? ")
    print(formule,prenom)

    return

# Test
affiche_salutation("Coucou")
```

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

# Fonction sans paramètre, avec sortie

def demande_prenom_nom():
    """ Demande et renvoie le prénom et le nom """

    prenom = input("Quel est ton prénom ? ")
    nom = input("Quel est ton nom ? ")

    nom_complet = prenom + " " + nom.upper()

    return nom_complet

# Test
identite = demande_prenom_nom()
print("Identité :",identite)
```

Activité 2

Activité 2 fonctions_2.py

```
##############################
# Fonctions
##############################


##############################
# Activité 2 - Fonctions
##############################

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

# Fonction avec paramètre, avec sortie

def trinome_1(x):
    """ Calcule 3x^2-7x+4 """

    resultat = 3*x**2 - 7*x + 4

    return resultat

# Test
print("--- Trinôme ---")
for i in range(10):
    print("La valeur en x =",i,"est",trinome_1(i))

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

def trinome_2(a,b,c,x):
    """ Calcule ax^2+bx+c """

    resultat = a*x**2 + b*x + c

    return resultat

# Test
a = 2 ; b = -1 ; c = 0
```

```
print("Trinôme pour a,b,c =",a,b,c)
for i in range(10):
    print("La valeur en x =",i,"est",trinome_2(a,b,c,i))


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

# Fonction avec paramètre, avec sortie

def conversion_euros_vers_dollars(montant):
    """ Calcule la valeur en dollars d'un montant donné en euros """

    montant_dollar = 1.15 * montant

    return montant_dollar

# Test
print("--- Devises ---")
x = 20
print(x,"euros valent", conversion_euros_vers_dollars(x),"dollars")

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

def conversion_euros(montant,devise):
    """ Calcule la valeur dans une monnaie d'un montant donné en euros """

    if devise == "dollar":
        taux = 1.15
    if devise == "livre":
        taux = 0.81
    if devise == "yen":
        taux = 130

    montant_devise = montant * taux

    return montant_devise

# Test
x = 100
for madevise in ["yen","dollar","livre"]:
    print(x,"euros valent", conversion_euros(x,madevise),madevise)


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

from math import *

# Calculs de différents volumes

def volume_cube(a):
    return a**3

def volume_boule(r):
    return 4/3 * pi * r**3

def volume_cylindre(r,h):
    return pi * r**2 * h

def volume_boite(a,b,c):
    return a * b * c

# Test
print("--- Volumes ---")
print(volume_cube(3))
```

```
print(volume_boule(3))
print(volume_cylindre(2,5))
print(volume_boite(3,4,5))

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

def perimetre_aire_rectangle(a,b):
    """ Calcule le périmètre et l'aire
    d'un rectangle de côtés a et b """

    p = 2*a+2*b
    A = a * b

    return p, A

def perimetre_aire_disque(r):
    """ Calcule le périmètre et l'aire
    d'un disque de rayon r """

    p = 2 * pi * r
    A = pi * r**2

    return p, A

print("--- Périmètres et aires ---")
print(perimetre_aire_rectangle(4,5))
print(perimetre_aire_disque(3))

# Recherche expérimentale : comparaison périmètre/aire d'un disque

for rayon in range(0,30):
    perimetre, aire = perimetre_aire_disque(rayon/10)
    print(rayon/10, perimetre - aire)

# Conclusion expérimentale :
#   pour 0 < r < 2, le perimètre est strictement plus grand que l'aire
#   pour r = 2, le périmètre égale l'aire,
#   pour r > 2, le périmètre est strictement plus petit que l'aire
```

Activité 3

Activité 3 — fonctions_3.py

```
###############################
# Fonctions
###############################

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

from turtle import *
width(5)    # Epaisseur du trait

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

def triangle():
    color('red')
    forward(200)
```

```
    left(120)
    forward(200)
    left(120)
    forward(200)
    return

# Test
# triangle()
# exitonclick()

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

def carre():
    color('green')
    for i in range(4):
        forward(200)
        left(90)
    return

# Test
# carre()
# exitonclick()

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

def hexagone(longueur):
    color('blue')
    for i in range(6):
        forward(longueur)
        left(60)
    return

# Test
# hexagone(100)
# exitonclick()

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

def polygone(n,longueur):
    color('purple')
    angle = 360/n
    for i in range(n):
        forward(longueur)
        left(angle)
    return

# Test
# polygone(10,70)
# exitonclick()

# Test tout
up()
goto(-450,0)
down()
triangle()
up()
goto(-200,0)
setheading(0)
```

```
down()
carre()
up()
goto(100,0)
setheading(0)
down()
hexagone(100)
up()
goto(320,0)
setheading(0)
down()
polygone(8,70)
up()
exitonclick()
```

Activité 4

Activité 4 fonctions_4.py

```
##############################
# Fonctions
##############################

##############################
# Activité 4 - Fonctions
##############################

####################################################
## Question 1 ##
def reduction(age):
    """ Renvoie le pourcentage de réduction en fonction de l'âge """

    if age < 10:
        reduc = 50
    elif age <= 18:
        reduc = 30
    elif age >= 60:
        reduc = 20
    else:
        reduc = 0

    return reduc

# Test
print("--- Réduction ---")
mon_age = 16
print("J'ai",mon_age,"ans et ma réduction est de",reduction(mon_age),"%.")

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

def montant(tarif_normal,age):
    """ Calcule le montant dû, en fonction du tarif normal et de l'âge """

    reduc = reduction(age)
    tarif = tarif_normal * (100-reduc)/100

    return tarif
```

```
# Test
print("--- Coût total des billets ---")
montant_famille = montant(30,9) + 2*montant(20,16) + 2*montant(35,40)
print(montant_famille)

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

def calcul_est_exact(a,b,reponse):
    """ Teste si le résultat de a*b est correct """

    if reponse == a * b:
        return True
    else:
        return False

# Test
print("--- Test résultat multiplication ---")
print(calcul_est_exact(6,7,42))

def test_multiplication(a,b,lang):
    """ Pose une multiplication en français ou en anglais
    et affiche si la réponse est correcte ou pas """

    # Phrases en français et en anglais
    if lang == "francais":
        question = "Combien vaut le produit a x b ? "
        reponse_juste = "Bravo !"
        reponse_fausse = "Eh non !"
    elif lang == "anglais":
        question = "How much is the product a x b? "
        reponse_juste = "Well done!"
        reponse_fausse = "It's wrong!"

    # Interrogation
    print("--- Question ---")
    print("a =",a)
    print("b =",b)
    reponse = int(input(question))

    if calcul_est_exact(a,b,reponse):
        print(reponse_juste)
    else:
        print(reponse_fausse)

    return

# Test
print("--- Quiz multiplication français/anglais ---")
test_multiplication(6,7,"anglais")
```

Activité 5

Activité 5 fonctions_5.py

```
##############################
# Fonctions
##############################
```

```
##############################
# Activité 5 - Egalité expérimentale
##############################

from math import *

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

def valeur_absolue(x):
    if x >= 0:
        return x
    else:
        return -x

####################################################
# def valeur_absolue_moins(x):
#     return valeur_absolue(-x)

####################################################
def racine_du_carre(x):
    return sqrt(x**2)

####################################################
def egalite_experimentale_1(f,g):
    """ Teste si deux fonctions sont expérimentalement égales """
    for i in range(-100,101):
        if f(i) != g(i):
            return False
    return True

# Test
print("--- Egalité experimentale, une variable ---")
# print(egalite_experimentale_1(valeur_absolue,valeur_absolue_moins))  # Vrai
print(egalite_experimentale_1(valeur_absolue,racine_du_carre))       # Vrai

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

def F1(a,b):
    return (a+b)**2

####################################################
def F2(a,b):
    return a**2 + 2*a*b + b**2

####################################################
def F3(a,b):
    return (a-b)**3

####################################################
def F4(a,b):
    return a**3 - 3*a**2*b  - 3*a * b**2 + b**3

####################################################
def F5(a,b):
    return a**3 - 3*a**2*b  + 3*a * b**2 - b**3

####################################################
def egalite_experimentale_2(F,G):
```

```
    """ Teste si deux fonctions de deux variables sont expérimentalement égales """

    for i in range(-100,101):
        for j in range(-100,101):
            if F(i,j) != G(i,j):
                # print(i,j,F(i,j),G(i,j))
                return False
    return True

# Test
print("--- Egalité experimentale, deux variables ---")
print(egalite_experimentale_2(F1,F2))   # Vrai
print(egalite_experimentale_2(F3,F4))   # Faux
print(egalite_experimentale_2(F3,F5))   # Vrai

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

def sincos(x):
    return sin(x)**2 + cos(x)**2

################################################
def un(x):
    return 1

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

precision = 0.00001   # = 10**-5

def egalite_experimentale_3(f,g):
    """ Teste si deux fonctions sont expérimentalement égales
    en autorisant une marge d'erreur """

    for i in range(-100,101):
        if abs(f(i) - g(i))> precision :
            return False
    return True

# Test
print("--- Egalité experimentale approchée ---")
print(egalite_experimentale_1(sincos,un))  # Faux !! Mais pourtant devrait être vrai !
print(sin(3)**2+cos(3)**2)         #  Explication : Python ne renvoie par exactement 1
print(egalite_experimentale_3(sincos,un))  # Vrai !

# Test avec d'autres inégalités, exemples :
# sin(2x) = 2 sin(x)cos(x)
# cos(pi/2 - x) = sin(x)
# et qq unes fausses ...

################################################
# Une égalité fausse mais experimentalement vraie

def g1(x):
    return sin(pi*x)

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

def g2(x):
    return 0

print("--- Une égalité fausse mais experimentalement vraie ---")
print(egalite_experimentale_3(g1,g2))  # Vrai (on a toujours égalité pour i entier)
print(g1(1/2))  # et pourtant g1(0.5) n'est pas nul, donc l'égalité n'est pas vrai en
    ↪ général
```

5. Arithmétique – Boucle tant que – I

Activité 1

Activité 1 tantque_1.py

```
##############################
# Tant que - Booléen - Arithmétiques
##############################

##############################
# Activité 1 - Divisibilité, quotient, reste
##############################

##############################
## Question 1 ##
def quotient_reste(a,b):
    """ Affiche le quotient et le reste et vérifie
    la validité de la division euclidienne """

    q = a // b
    r = a % b
    print("Division de a =",a,"par b =",b)
    print("Le quotient vaut q =",q)
    print("Le reste vaut r =",r)

    if (r >=0) and (r < b):
        verif_reste = True
    else:
        verif_reste = False
    print("Vérification reste 0 <= r < b ?",verif_reste)

    if a == b*q +r:
        verif_egal = True
    else:
        verif_egal = False
    print("Vérification égalité a = bq + r ?",verif_egal)

    return q,r

# Test
print("--- Quotient et reste ---")
quotient_reste(100,7)

##############################
## Question 2 ##
def est_pair(n):
    """ Teste si l'entier n est pair ou pas (renvoie vrai ou faux) """
    reste = n % 2
    if reste == 0:
        return True
    else:
        return False

def est_pair_bis(n):
    """ Teste si l'entier n est pair ou pas (renvoie vrai ou faux) """
    chiffre = n % 10
    if (chiffre==0) or (chiffre==2) or (chiffre==4) or (chiffre==6) or (chiffre==8):
```

```
        return True
    else:
        return False

# En deux lignes !
def est_pair_ter(n):
    return (n % 2) == 0

# Test
print("--- Parité  ---")
print(est_pair(1023))

################################
## Question 3 ##
def est_divisible(a,b):
    """ Teste si a est divisible par b """
    if a % b == 0:
        return True
    else:
        return False

# Test
print("--- Divisibilité  ---")
print(est_divisible(125,5))
```

Activité 2

Activité 2 tantque_2.py

```
################################
# Tant que - Booléen - Arithmétiques
################################

################################
# Activité 2 - Diviseur, nombres premiers - Boucle while
################################

################################
## Question 1 ##
def plus_petit_diviseur(n):
    """ Cherche le plus petit diviseur de n """
    d = 2
    while n % d != 0:
        d = d + 1
    return d

# Test
print("--- Plus petit diviseur ---")
print(plus_petit_diviseur(7*13))

################################
## Question 2 ##
def est_premier_1(n):
    """ Teste si n est un nombre premier """
```

```
    d = 2

    while n % d != 0:
        d = d + 1

    if d == n:
        return True
    else:
        return False

# Test
print("--- Est premier (1) ---")
print(est_premier_1(97))

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

# Nombre de Fermat

def contre_exemple_fermat():
    for n in range(6):
        fermat = 2**(2**n)+1
        print(n,fermat,est_premier_1(fermat))
    return

# Test
print("--- Test conjecture nombres de Fermat ---")
contre_exemple_fermat()

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

def est_premier_2(n):
    """ Teste si n est un nombre premier """

    if n < 2:
        return False

    d = 2

    while (n % d != 0) and (d**2 <= n):
        d = d + 1

    if d** 2 > n:
        return True
    else:
        return False

# Test
print("--- Est premier (2) ---")
print(est_premier_2(97))

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

def est_premier_3(n):
    """ Teste si n est un nombre premier """

    if n < 2: return False
    if n == 2: return True
    if n % 2 == 0: return False

    d = 3
    while (n % d != 0) and (d**2 <= n):
```

```
        d = d + 2

    if d ** 2 > n:
        return True
    else:
        return False

 # Test
print("--- Est premier (3) ---")
print(est_premier_3(97))

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

## Question 5 ##

# Calcul les temps d'éxécution moyens des différents fonction est_premier()

import timeit
print(timeit.timeit("est_premier_1(97)", setup="from __main__ import est_premier_1", number
    ↪ =10000))
print(timeit.timeit("est_premier_2(97)", setup="from __main__ import est_premier_2", number
    ↪ =10000))
print(timeit.timeit("est_premier_3(97)", setup="from __main__ import est_premier_3", number
    ↪ =10000))

##############################
# On garde la plus efficace !

def est_premier(n):
    return est_premier_3(n)
```

Activité 3

Activité 3 tantque_3.py

```
##############################
# Tant que - Booléen - Arithmétiques
##############################

##############################
# Activité 3 - Diviseur, nombres premiers - Boucle while (suite)
##############################

##############################
# Rappel : activité 2

def est_premier(n):
    if n < 2: return False
    if n == 2: return True
    if n % 2 == 0: return False

    d = 3
    while (n % d != 0) and (d**2 <= n):
        d = d + 2

    if d ** 2 > n:
        return True
    else:
        return False
```

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

##############################
## Question 1 ##
def nombre_premier_apres(n):
    """ Cherche le prochain nombre premier après n """
    p = n
    while not est_premier(p):
        p = p + 1
    return p

# Test
print("--- Premier nombre premier après un entier ---")
print(nombre_premier_apres(60))
print(nombre_premier_apres(100000))

##############################
## Question 2 ##
def nombres_jumeaux_apres(n):
    """ Trouve deux nombre premiers jumeaux après n """
    p = n
    q = p + 2
    while (not est_premier(p)) or (not est_premier(q)):
        p = p + 1
        q = p + 2
    return p,q

# Test
print("--- Premiers nombres jumeaux après un entier ---")
print(nombres_jumeaux_apres(60))
print(nombres_jumeaux_apres(100000))

##############################
## Question 3 ##
def nombre_germain_apres(n):
    """ Trouve deux nombre premiers de Germain après n """
    p = n
    q = 2*p + 1
    while (not est_premier(p)) or (not est_premier(q)):
        p = p + 1
        q = 2*p + 1
    return p,q

# Test
print("--- Premiers nombres premiers de Germain après un entier ---")
print(nombre_germain_apres(60))
print(nombre_germain_apres(100000))
```

6. Chaînes de caractères – Analyse d'un texte

Activité 1

Activité 1 chaines_1.py

```
##############################
# Chaînes de caractères - Analyse statistique d'un texte
##############################

##############################
# Activité 1 - Pluriels
##############################

## Question 1 ##

mot = "chat"
pluriel = mot + "s"
print("Mon mot : ",mot)
print("Au pluriel : ",pluriel)

## Question 2 ##

# mot = "chat"
mot = "souris"

derniere_lettre = mot[len(mot)-1]

if derniere_lettre == "s":
    pluriel = mot
else:
    pluriel = mot + "s"

print("Mon mot : ",mot)
print("Au pluriel : ",pluriel)

## Question 3 ##

# mot = "chat"
# mot = "souris"
mot = "journal"

derniere_lettre = mot[len(mot)-1]
avant_derniere_lettre = mot[len(mot)-2]

if derniere_lettre == "s":
    pluriel = mot
elif avant_derniere_lettre == "a" and derniere_lettre == "l":
    debut_mot = mot[0:len(mot)-2]
    pluriel = debut_mot + "aux"
else:
    pluriel = mot + 's'

print("Mon mot : ",mot)
print("Au pluriel : ",pluriel)


## Question 4 ##

# C'est mieux avec une fonction !

def met_au_pluriel(mot):
    """ Met un mot au pluriel.
```

```
    Entrée : un mot
    Sortie : le mot au pluriel (sauf exceptions) """

    derniere_lettre = mot[len(mot)-1]
    avant_derniere_lettre = mot[len(mot)-2]

    if derniere_lettre == "s":
        pluriel = mot
    elif avant_derniere_lettre == "a" and derniere_lettre == "l":
        debut_mot = mot[0:len(mot)-2]
        pluriel = debut_mot + "aux"
    else:
        pluriel = mot + "s"

    return pluriel

# Test
#mot = input("Dis moi un mot : ")
#pluriel = met_au_pluriel(mot)
#print("Son pluriel est :",pluriel)

## Question 5 ##

def affiche_conjugaison(verbe):
    """ Conjugue au présent.
    Entrée : un verbe du premier groupe
    Sortie : affiche la conjugaison au présent"""

    debut_verbe = verbe[0:len(verbe)-2]
    fin_verbe = verbe[len(verbe)-2:len(verbe)]

    if fin_verbe == "er":
        print("Je " + debut_verbe + "e\n")
        print("Tu " + debut_verbe + "es\n")
        print("Il/elle " + debut_verbe + "e\n")
        print("Nous " + debut_verbe + "ons\n")
        print("Vous " + debut_verbe + "ez\n")
        print("Ils/elles " + debut_verbe + "ent\n")
    else:
        print("Ce n'est pas un verbe du premier groupe !")

    return

# Test
verbe = input("Donne-moi un verbe du premier groupe : ")
affiche_conjugaison(verbe)
```

Activité 2

Activité 2 chaines_2.py

```
################################
# Chaînes de caractères - Analyse statistique d'un texte
################################

################################
# Activité 2 - Jeux de mots
################################
```

```
## Question 1 ##

def distance_hamming(mot1,mot2):
    """ Calcule la distance de Hamming
    Entrée : deux mots de même longueur
    Sortie : la distance entre ces mots """

    distance = 0
    for i in range(len(mot1)):
        if mot1[i] != mot2[i]:
            distance = distance + 1

    return distance

# Test
premier_mot = "JAPON"
second_mot = "SAVON"
dist = distance_hamming(premier_mot,second_mot)
print("La distance entre",premier_mot,"et",second_mot, "est",dist)


## Question 2 ##

def transforme_en_latin_cochon(mot):
    """ Transforme un mot en latin-cochon
    Entrée : un mot (une chaîne de caractères)
    Sortie : le mot transformé en latin cochon s'il commence par une consonne. """

    premiere_lettre = mot[0]
    reste_mot = mot[1:len(mot)]

    if premiere_lettre not in ["A", "E", "I", "O", "U", "Y"]:
        latin_cochon = reste_mot + premiere_lettre + "UM"
    else:
        latin_cochon = mot

    return latin_cochon

# Test
mot = "SALOPETTE"
latin = transforme_en_latin_cochon(mot)
print("Le mot",mot,"devient",latin,"!")


## Question 3 ##

def transforme_en_verlan(mot):
    """ Transforme un mot en verlan
    Entrée : un mot (une chaîne de caractères)
    Sortie : le mot transformé verlan """

    verlan = ""
    for carac in mot:
        verlan = carac + verlan

    return verlan

# Test
mot = "BONJOUR"
verlan = transforme_en_verlan(mot)
print("Le mot",mot,"devient",verlan,"!")


## Question 4 ##

def est_un_palindrome(mot):
```

```
    """ Détermine si un mot est un palindrome
    Entrée : un mot (une chaîne de caractères)
    Sortie : vrai si le mot est un palindrome, faux sinon """

    verlan = transforme_en_verlan(mot)
    if mot == verlan:
        return True
    else:
        return False

# Test
mot = "KAYAK"
print("Le mot",mot,"est-il un palindrome ? : ",est_un_palindrome(mot))
```

Activité 3

Activité 3 chaines_3.py

```
################################
# Chaînes de caractères - Analyse statistique d'un texte
################################

################################
# Activité 3 - Séquences d'ADN
################################

## Question 1 ##

def presence_de_A(sequence):
    """ Détermine la présence du nucléotides A
    Entrée : une séquence de A,C,T,G (chaîne de caractères)
    Sortie : vrai ou faux """

    for nucleotide in sequence:
        if nucleotide == 'A':
            return True

    return False

# Test
sequence = "AGACAGCGAGCATATGCAGGAAG"
reponse = presence_de_A(sequence)
print("Est-ce qu'il y a A dans la séquence",sequence," : ",reponse)

## Question 2 ##

def position_de_AT(sequence):
    """ Détermine la position de la première séquence AT
    Entrée : une séquence de A,C,T,G (chaîne de caractères)
    Sortie : la position dans cette séquence (commence à 0) """

    for i in range(len(sequence)-1):
        if sequence[i] == 'A' and sequence[i+1] == 'T':
            return i  # Si trouvé

    return None  # Si pas du tout trouvé

# Test
sequence = "GTGGTTTGACCTCCCATGGCCAT"
```

```
position = position_de_AT(sequence)
print("Dans la séquence",sequence,"le code AT apparaît en position",position)

## Question 3 ##

def position(code,sequence):
    """ Détermine la position du code dans la séquence
    Entrée : une séquence de A,C,T,G (chaîne de caractères)
    Sortie : la position dans cette séquence (commence à 0) """

    for i in range(len(sequence)-len(code)):
        if code == sequence[i:i+len(code)]:
            return i  # Si trouvé, c'est fini

    return None  # Si pas du tout trouvé

# Test
sequence = "GAAGACCTTCTCCTCCTGC"
code = "CCTC"
pos = position(code,sequence)
print("Dans la séquence",sequence,"le code",code,"apparaît en position",pos)

## Question 4 ##

def enquete():
    moutarde =  "CCTGGAGGGTGGCCCCACCGGCCGAGACAGCGAGCATATGCAGGAAGCGGCAGGAATAAGGAAAAGCAGC"
    rose =      "CTCCTGATGCTCCTCGCTTGGTGGTTTGAGTGGACCTCCCAGGCCAGTGCCGGGCCCCTCATAGGAGAGG"
    pervenche = "AAGCTCGGGAGGTGGCCAGGCGGCAGGAAGGCGCACCCCCCCAGTACTCCGCGCGCCGGGACAGAATGCC"
    leblanc =   "CTGCAGGAACTTCTTCTGGAAGTACTTCTCCTCCTGCAAATAAAACCTCACCCATGAATGCTCACGCAAG"

    code1 = "CATA"
    code2 = "ATGC"

    for suspect in [moutarde,rose,pervenche,leblanc]:
        print(position(code1,suspect))
        print(position(code2,suspect))

    return

# Execution de l'enquête
enquete()
```

Activité 4

Activité 4 chaines_4.py

```
################################
# Chaînes de caractères - Analyse statistique d'un texte
################################

################################
# Activité 4 - Majuscules/minuscules
################################

## Question 1 ##

# A la machine
code = [80,121,116,104,111,110,32,101,115,116,32,115,121,109,112,64]
phrase = ""
```

```
for c in code:
    phrase = phrase + chr(c)
print(phrase)

## Question 2 ##

for i in range(33,128):
    print(i," : ",chr(i))

## Question 3 ##

exp1 = 'chr(ord("a")-32)'
exp2 = 'chr(ord("B")+32)'

print(exp1," donne ",eval(exp1))
print(exp2," donne ",eval(exp2))

## Question 4 ##

def lettre_majuscule(car):
    """ Transforme une lettre minuscule en majuscule
    Entrée : un caractère minuscule parmi "a",...,"z"
    Sortie : le même caractère en majuscule """

    ordre = ord(car)
    nouv_ordre = ordre - 32
    nouv_car = chr(nouv_ordre)

    return nouv_car

# Test
print("La majuscule de a est",lettre_majuscule("a"))


## Question 5 ##

def majuscules(phrase):
    """ Transforme une phrase en majuscules
    Entrée : une phrase (une chaîne de caractères)
    Sortie : la même phrase avec les lettres en majuscules """

    nouv_phrase = ""
    for car in phrase:
        ordre = ord(car)
        if ordre >= 97 and ordre <= 122:
             # transformation en majuscule
            nouv_phrase = nouv_phrase + chr(ordre-32)
        else:
            # conserver le caractère
            nouv_phrase = nouv_phrase + car

    return nouv_phrase

# Test
phrase = "Bonjour le monde !"
print("La phrase",phrase,"devient",majuscules(phrase))

# On aura aussi besoin de :
def minuscules(phrase):
    """ Transforme une phrase en minuscules
    Entrée : une phrase (une chaîne de caractères)
    Sortie : la même phrase avec les lettres en minuscules """

    nouv_phrase = ""
```

```
    for car in phrase:
        ordre = ord(car)
        if ordre >= 65 and ordre <= 90:
             # transformation en minuscules
            nouv_phrase = nouv_phrase + chr(ordre+32)
        else:
            # conserver le caractère
            nouv_phrase = nouv_phrase + car

    return nouv_phrase

## Question 6 ##

def formate_prenom_nom(personne):
    """ Transforme le nom d'une personne au format "Prenom NOM"
    Entrée : le prénom et le nom d'une personne (sans accent, séparé par une espace)
    Sortie : le nom complet au format "Prenom NOM" """

    # On sépare le prénom du nom
    prenom = ""
    nom = ""
    dans_le_prenom = True     # On est dans le prénom
    for car in personne:
        if dans_le_prenom:
            prenom = prenom + car
        else:
            nom = nom + car
        if car == " ":
            dans_le_prenom =  False    # Le prénom est fini

    # On formate le prénom
    nouv_prenom = majuscules(prenom[0])+minuscules(prenom[1:len(prenom)])

    # On formate le nom
    nouv_nom = majuscules(nom)

    return nouv_prenom+nouv_nom

# Test
personne = "harry POTTER"
print(personne,"devient",formate_prenom_nom(personne))
personne = "LORD Voldemort"
print(personne,"devient",formate_prenom_nom(personne))
```

Activité 5

Activité 5 chaines_5.py

```
################################
# Chaînes de caractères - Analyse statistique d'un texte
################################

import random # Uniquement pour créer l'énigme

################################
# Activité 5 - Analyse statistique d'un texte
################################

## Question supprmiée ##
```

```
def nombre_E(phrase):
    """ Compte le nombre de "E"
    Entrée : une phrase en majuscule
    Sortie : le nombre de "E" """

    nb = 0
    for car in phrase:
        if car == "E":
            nb = nb + 1

    return nb

# Test
phrase = "ESPRIT ES TU LA"
print("La phrase",phrase,"contient",nombre_E(phrase),"fois la lettre E")


## Question 1 ##

def occurences_lettre(lettre,phrase):
    """ Compte le nombre d'occurrences d'une lettre donnée dans phrase
    Entrée : une lettre et une phrase en majuscule
    Sortie : le nombre de lettres """

    nb = 0
    for car in phrase:
        if car == lettre:
            nb = nb + 1

    return nb

# Test
phrase = "ESPRIT ES TU LA"
print("La phrase",phrase,"contient",occurences_lettre("S",phrase),"fois la lettre S")


## Question 2 ##

def nombre_lettres(phrase):
    """ Compte le nombre total de lettres
    Entrée : une phrase en majuscule
    Sortie : le nombre total de lettres de "A" à "Z" """

    alphabet = list("ABCDEFGHIJKLMNOPQRSTUVWXYZ")

    nb = 0
    for car in phrase:
        if car in alphabet:
            nb = nb + 1

    return nb

# Test
phrase = "ESPRIT ES TU LA"
print("La phrase",phrase,"contient",nombre_lettres(phrase),"lettres")


## Question 3 ##

def pourcentage_lettre(lettre,phrase):
    """ Calcule le ratio d'une lettre donnée dans phrase
    Entrée : une lettre et une phrase en majuscule
    Sortie : le pourcentage d'apparition de la lettre """
```

```
    nb_lettres = occurences_lettre(lettre,phrase)
    nb_total = nombre_lettres(phrase)
    pourcentage = nb_lettres/nb_total*100

    return pourcentage

# Test
phrase = "ESPRIT ES TU LA"
pourcentage = pourcentage_lettre("S",phrase)
print("La phrase",phrase,"contient",pourcentage,"% de lettre S")
print("Pourcentage arrondi :","{0:.2f}".format(pourcentage))

## Question 4 ##

def affiche_frequences(phrase):
    """ Calcule le ratio de toutes les lettres dans une phrase
    Entrée : uune phrase en majuscule
    Sortie : l'affichage des pourcentage d'apparition des lettres """

    alphabet = list("ABCDEFGHIJKLMNOPQRSTUVWXYZ")
    for lettre in alphabet:
        pourcentage = pourcentage_lettre(lettre,phrase)
        print(lettre," : ","{0:.2f}".format(pourcentage))

    return

# SECRET ------------------
# Création de l'énigme

def myshuffle(x):
    x = list(x)
    random.shuffle(x)
    return x

#for mot in phrase.split():
#  print(list(mot))
#  print(myshuffle(list(mot)))

# Le corbeau et le renard - Jean de la Fontaine
phrase1 = "MAITRE CORBEAU SUR UN ARBRE PERCHE TENAIT EN SON BEC UN FROMAGE MAITRE RENARD PAR
    ↪  L ODEUR ALLECHE LUI TINT A PEU PRES CE LANGAGE ET BONJOUR MONSIEUR DU CORBEAU QUE
    ↪ VOUS ETES JOLI QUE VOUS ME SEMBLEZ BEAU SANS MENTIR SI VOTRE RAMAGE SE RAPPORTE A
    ↪ VOTRE PLUMAGE VOUS ETES LE PHENIX DES HOTES DE CES BOIS A CES MOTS LE CORBEAU NE SE
    ↪ SENT PAS DE JOIE ET POUR MONTRER SA BELLE VOIX IL OUVRE UN LARGE BEC LAISSE TOMBER SA
    ↪  PROIE LE RENARD S EN SAISIT ET DIT MON BON MONSIEUR APPRENEZ QUE TOUT FLATTEUR VIT
    ↪ AUX DEPENS DE CELUI QUI L ECOUTE CETTE LECON VAUT BIEN UN FROMAGE SANS DOUTE LE
    ↪ CORBEAU HONTEUX ET CONFUS JURA MAIS UN PEU TARD QU ON NE L Y PRENDRAIT PLUS"

#phrase_mystere1 = ' '.join([''.join(myshuffle(list(mot))) for mot in phrase1.split() ])

# Le roi de aulnes - Goethe
phrase2 = "WER REITET SO SPAT DURCH NACHT UND WIND ES IST DER VATER MIT SEINEM KIND ER HAT
    ↪ DEN KNABEN WOHL IN DEM ARM ER FASST IHN SICHER ER HALT IHN WARM MEIN SOHN WAS BIRGST
    ↪ DU SO BANG DEIN GESICHT SIEHST VATER DU DEN ERLKONIG NICHT DEN ERLENKONIG MIT KRON
    ↪ UND SCHWEIF  MEIN SOHN ES IST EIN NEBELSTREIF DU LIEBES KIND KOMM GEH MIT MIR GAR
    ↪ SCHONE SPIELE SPIEL ICH MIT DIR MANCH BUNTE BLUMEN SIND AN DEM STRAND MEINE MUTTER
    ↪ HAT MANCH GULDEN GEWAND MEIN VATER MEIN VATER UND HOREST DU NICHT WAS ERLENKONIG MIR
    ↪ LEISE VERSPRICHT SEI RUHIG BLEIBE RUHIG MEIN KIND IN DURREN BLATTERN SAUSELT DER WIND
    ↪ "

#phrase_mystere2 = ' '.join([''.join(myshuffle(list(mot))) for mot in phrase2.split() ])

# Cent ans de solitude - Gabriel Garcia Marquez
phrase3 = "FASCINADO POR UNA REALIDAD INMEDIATA QUE ENTONCES LE RESULTO MAS FANTASTICA QUE
```

```
    ↪ EL VASTO UNIVERSO DE SU IMAGINACION PERDIO TODO INTERES POR EL LABORATORIO DE
    ↪ ALQUIMIA PUSO A DESCANSAR LA MATERIA EXTENUADA POR LARGOS MESES DE MANIPULACION Y
    ↪ VOLVIO A SER EL HOMBRE EMPRENDEDOR DE LOS PRIMEROS TIEMPOS QUE DECIDIA EL TRAZADO DE
    ↪ LAS CALLES Y LA POSICION DE LAS NUEVAS CASAS Y SE DETERMINO QUE FUERA EL QUIEN
    ↪ DIRIGIERA LA REPARTICION DE LA TIERRA"

phrase_mystere3 = ' '.join([''.join(myshuffle(list(mot))) for mot in phrase3.split() ])

# Sumertimes - Elle Fitzgerald
phrase4 = "SUMMERTIME AND THE LIVING IS EASY FISH ARE JUMPING AND THE COTTON IS HIGH OH YOUR
    ↪  DADDY IS RICH AND YOUR MA IS GOOD LOOKING SO HUSH LITTLE BABY DONT YOU CRY ONE OF
    ↪ THESE MORNINGS YOU RE GONNA RISE UP SINGING AND YOULL SPREAD YOUR WINGS AND YOULL
    ↪ TAKE TO THE SKY BUT TILL THAT MORNING THERE AINT NOTHING CAN HARM YOU WITH DADDY AND
    ↪ MAMMY STANDING BY"

phrase_mystere4 = ' '.join([''.join(myshuffle(list(mot))) for mot in phrase4.split() ])


# FIN SECRET ------------------

# Choix des phrases mystères

phrase_mystere1 = "TMAIER BERACUO RSU NU REBRA PRCEEH EIANTT NE ONS EBC NU GAOFREM EIMATR
    ↪ RERNAD APR L RDUOE LAHECLE UIL TTNI A EUP SREP EC LGNGAEA TE RBONUJO ERMNOUSI DU
    ↪ UBRACEO QUE OVSU EEST LIJO UQE OUVS EM MSZELBE BAEU ASNS MIERNT IS RVETO AGRAME ES
    ↪ PRARPTOE A OEVTR AMGUPLE VUOS SEET EL PNIHXE DSE OSHET ED CSE BIOS A ESC MSOT LE
    ↪ OUBRCEA NE ES ESTN ASP DE IEJO TE OUPR ERRNOTM AS BELEL XOVI IL OREVU NU RGLEA ECB
    ↪ ILESSA EBOMTR AS PIOER EL NRDAER S EN ISIAST TE ITD MNO NOB EUSRMNOI NRPEEAZP QEU
    ↪ UTOT EUTLRFTA IVT XUA SPNEDE DE UECIL UQI L TECEOU TECET NEOCL VATU BNEI UN GMAEORF
    ↪ SNAS TUOED LE EOABURC OHENTXU TE NSCOFU UJRA SMIA UN EPU TRDA UQ NO EN L Y ARRPEIDNT
    ↪ ULSP"

print(phrase_mystere1)

affiche_frequences(phrase_mystere1)

phrase_mystere2 = "WRE TREITE SO TSPA CUDHR AHNCT UND WIND SE STI RED AEVRT MTI ESEIMN IDNK
    ↪ RE ATH END NEABNK WLOH IN EMD AMR ER AFTSS HIN IHSERC RE AHTL HIN MRWA EINM SHNO SAW
    ↪ SRTIBG UD SO NGBA DNEI EIHSGTC ESISTH RAETV UD DEN LERNIOKG NITHC NDE LOENINKGRE TIM
    ↪ OKRN UDN CHWFSEI NEIM NSOH ES STI IEN BIFTRLSEEEN DU BILESE IKDN OMKM EHG MIT MIR RAG
    ↪  ECHNOS EPELSI EIPSL IHC ITM RDI HNCMA BEUTN MBLUNE DINS NA DEM TNDRAS NMIEE UTETMR
    ↪ AHT CAMHN UDNGEL GDAWEN MIEN EATRV MENI VEART DUN OSTHER DU CINTH SAW KNNOEIREGL RIM
    ↪ ILEES PRSTVRCIEH ISE IHGRU BEEILB RIGUH MNEI KNDI NI RDNEUR NATBRLET STAESUL EDR WNID
    ↪ "

print(phrase_mystere2)

affiche_frequences(phrase_mystere2)

phrase_mystere3 = "DSNOACAIF ORP ANU DAEDALRI DNAAEIMTI EQU NNCOSETE EL RSTEOUL SMA
    ↪ AACTFAITNS UQE EL TSVAO OINSRVUE DE US ANIGIICANOM EIORDP TOOD RTEIENS RPO LE
    ↪ ITOABOLRROA ED QIUAMALI USOP A NSSRCAEAD LA TMREAAI NXTADAUEE ROP GOARLS EMESS DE
    ↪ NNAMICLUIAPO Y LOVOIV A RES LE RHMEOB EOMDNEERPRD DE LOS RSOPMRIE OMTSIPE UEQ CIIDADE
    ↪  LE RTDAAOZ ED LSA CELSAL Y LA NICOIOPS ED LAS UESVNA SSACA Y ES ITRMNEEOD QEU AERFU
    ↪ EL UEQIN IIIRDEGAR LA NAIORTREICP DE AL RRTEIA"

print(phrase_mystere3)

affiche_frequences(phrase_mystere3)

phrase_mystere4 = "IMTRUESMME DNA TEH LNGIIV SI EYAS SIFH REA GJPNUIM DNA HET TTNOCO IS GHIH
    ↪  OH OUYR DDADY SI IRHC DAN ROUY MA SI DOGO GKOILON OS USHH LTLIET BBYA NDOT OUY CYR
    ↪ NEO OF HESET GNSRONIM YUO RE NANGO SIER PU SNIGING NAD OULLY EPADRS YUOR GINSW DAN
    ↪ LYOLU KATE OT HET KSY TUB ITLL TATH MGNIRNO EREHT NATI INTGOHN ACN AHMR OYU TWIH
    ↪ DADYD NDA MYMMA NSTIDGAN YB"

print(phrase_mystere4)
```

```
affiche_frequences(phrase_mystere4)
```

7. Listes I

Activité 1

Activité 1 listes_I_1.py

```
##############################
# Listes I
##############################

##############################
# Cours 1

maliste = [11,13,17,19]
maliste.append(23)
maliste.append(29)
print(maliste[5])
len(maliste)

##############################
# Activité 1 - Intérêts
##############################

## Question 1 ##

##############################
def interets_simples(S0,p,n):
    liste = [S0]
    interets = S0 * p/100
    S = S0
    for i in range(n):
        S = S + interets
        liste.append(S)
    return liste

# Test
print("--- Intérêts simples ---")
liste_interets_simples = interets_simples(1000,10,12)
print(liste_interets_simples)
print(liste_interets_simples[11])


## Question 2 ##

##############################
def interets_composes(S0,p,n):
    liste = [S0]
    S = S0
    for i in range(n):
        interets = S * p/100
        S = S + interets
        liste.append(S)
    return liste

# Test
```

```
print("--- Intérêts composés ---")
liste_interets_composes = interets_composes(1000,7,12)
print(liste_interets_composes)
print(liste_interets_composes[11])
```

Activité 2

Activité 2 listes_I_2.py

```
################################
# Listes I
################################

################################
# Cours 1

maliste = [11,13,17,19]
maliste.append(23)
maliste.append(29)
print(maliste[5])
len(maliste)

################################
# Activité 1 - Intérêts
################################

## Question 1 ##

################################
def interets_simples(S0,p,n):
    liste = [S0]
    interets = S0 * p/100
    S = S0
    for i in range(n):
        S = S + interets
        liste.append(S)
    return liste

# Test
print("--- Intérêts simples ---")
liste_interets_simples = interets_simples(1000,10,12)
print(liste_interets_simples)
print(liste_interets_simples[11])


## Question 2 ##

################################
def interets_composes(S0,p,n):
    liste = [S0]
    S = S0
    for i in range(n):
        interets = S * p/100
        S = S + interets
        liste.append(S)
    return liste

# Test
```

```
print("--- Intérêts composés ---")
liste_interets_composes = interets_composes(1000,7,12)
print(liste_interets_composes)
print(liste_interets_composes[11])
```

Activité 3

Activité 3 listes_I_3.py

```
################################
# Listes I
################################

################################
# Cours 1

maliste = [11,13,17,19]
maliste.append(23)
maliste.append(29)
print(maliste[5])
len(maliste)

################################
# Activité 1 - Intérêts
################################

## Question 1 ##

################################
def interets_simples(S0,p,n):
    liste = [S0]
    interets = S0 * p/100
    S = S0
    for i in range(n):
        S = S + interets
        liste.append(S)
    return liste

# Test
print("--- Intérêts simples ---")
liste_interets_simples = interets_simples(1000,10,12)
print(liste_interets_simples)
print(liste_interets_simples[11])


## Question 2 ##

################################
def interets_composes(S0,p,n):
    liste = [S0]
    S = S0
    for i in range(n):
        interets = S * p/100
        S = S + interets
        liste.append(S)
    return liste

# Test
```

```
print("--- Intérêts composés ---")
liste_interets_composes = interets_composes(1000,7,12)
print(liste_interets_composes)
print(liste_interets_composes[11])
```

Activité 4

Activité 4 listes_I_4.py

```
################################
# Listes I
################################

################################
# Cours 1

maliste = [11,13,17,19]
maliste.append(23)
maliste.append(29)
print(maliste[5])
len(maliste)

################################
# Activité 1 - Intérêts
################################

## Question 1 ##

################################
def interets_simples(S0,p,n):
    liste = [S0]
    interets = S0 * p/100
    S = S0
    for i in range(n):
        S = S + interets
        liste.append(S)
    return liste

# Test
print("--- Intérêts simples ---")
liste_interets_simples = interets_simples(1000,10,12)
print(liste_interets_simples)
print(liste_interets_simples[11])


## Question 2 ##

################################
def interets_composes(S0,p,n):
    liste = [S0]
    S = S0
    for i in range(n):
        interets = S * p/100
        S = S + interets
        liste.append(S)
    return liste

# Test
```

```
print("--- Intérêts composés ---")
liste_interets_composes = interets_composes(1000,7,12)
print(liste_interets_composes)
print(liste_interets_composes[11])
```

8. Statistique – Visualisation de données

Activité 1

Activité 1 statistique_1.py

```
##############################
# Statistique - Visualisation de données - tkinter
##############################

##############################
# Activité 1 - Calculs statistiques
##############################

from math import *

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

def somme(liste):
    """ Calcule la somme des éléments
    Entrée : une liste de nombres
    Sortie : leur somme """

    som = 0
    for x in liste:
        som = som + x
    return som

# Test
print("--- Somme ---")
liste = [5,18,6,3]
print(liste)
print(somme(liste))
print(sum(liste))

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

def moyenne(liste):
    """ Calcule la moyenne des éléments
    Entrée : une liste de nombres
    Sortie : leur moyenne """

    nbliste = len(liste)

    if nbliste == 0:
        moy = 0
    else:
        moy = somme(liste) / nbliste

    return moy
```

```
# Test
print("--- Moyenne ---")
liste = [5,18,6,3]
print(liste)
print(moyenne(liste))


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

def minimum(liste):
    """ Calcule le minimum des éléments
    Entrée : une liste de nombres
    Sortie : leur minimum """

    if len(liste) == 0:
        return None

    mini = liste[0]
    for i in range(1,len(liste)):
        if liste[i] < mini:
            mini = liste[i]

    return mini

# Test
print("--- Minimum ---")
liste = [6,8,2,10]
print(liste)
print(minimum(liste))
print(min(liste))

##################################################
def maximum(liste):
    """ Calcule le maximum des éléments
    Entrée : une liste de nombres
    Sortie : leur maximum """

    if len(liste) == 0:
        return None

    maxi = liste[0]
    for i in range(1,len(liste)):
        if liste[i] > maxi:
            maxi = liste[i]

    return maxi

# Test
print("--- Maximum ---")
liste = [6,8,2,10]
print(liste)
print(maximum(liste))
print(max(liste))


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

def variance(liste):
    """ Calcule la variance des éléments
    Entrée : une liste de nombres
    Sortie : leur variance """
```

```
    if len(liste) == 0:
        return 0

    moy = moyenne(liste)

    somme_carres = 0
    for x in liste:
        somme_carres = somme_carres + (x-moy)**2

    var = somme_carres / len(liste)

    return var

# Test
print("--- Variance ---")
liste = [6,8,2,10]
print(liste)
print(variance(liste))

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

def ecart_type(liste):
    """ Calcule l'écart-type des éléments
    Entrée : une liste de nombres
    Sortie : leur écart-type """

    eca = sqrt(variance(liste))

    return eca

# Test
print("--- Ecart-type ---")
liste = [6,8,2,10]
print(liste)
print(ecart_type(liste))

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

temp_brest = [6.4,6.5,8.5,9.7,11.9,14.6,15.9,16.3,15.1,12.2,9.2,7.1]
temp_strasbourg = [0.9,2.4,6.1,9.7,13.8,17.2,19.2,18.6,15.7,10.7,5.3,2.1]

print(moyenne(temp_brest))
print(moyenne(temp_strasbourg))
print(ecart_type(temp_brest))
print(ecart_type(temp_strasbourg))
```

Activité 2

Activité 2 statistique_2.py

```
###############################
# Statistique - Visualisation de données - tkinter
###############################

###############################
# Activité 2 - Visualisation des données
###############################
```

```
##############################
## Question 0 ##

from math import *
from random import *
from tkinter import *

# Fenêtre tkinter
root = Tk()

canvas = Canvas(root, width=800, height=600, background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

def une_couleur():
    """ Renvoie une couleur au hasard
    Entrée : rien
    Sortie : une couleur """

    # Méthode 1 - Choix limité
    # couleurs = ["red","orange","yellow","green","cyan","blue","violet","purple"]
    # coul = random.choice(couleurs)

    # Méthode 2 - Choix "infini"
    R = randint(0,255)
    V = randint(0,255)
    B = randint(0,255)
    coul = '#%02x%02x%02x' % (R, V, B)

    return coul


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

def graphique_barres(liste):
    """Graphique avec une barre par élément de la liste"""
    posx = 100
    for x in liste:
        hauteur = x * echelle
        canvas.create_rectangle(posx,400,posx+10,400-hauteur,fill="red")
        posx = posx + 30

    # Bonus : Coordonnées verticales à gauche
    max_liste = max(liste)
    canvas.create_line(90,400,90,400-echelle*max_liste)
    for j in range(max_liste+1):
        canvas.create_line(85,400-echelle*j,90,400-echelle*j)
        canvas.create_text(80,400-echelle*j,text=str(j))

    return

# Test
# echelle = 20
# liste = [5,8,6,3,7,10,4]
# graphique_barres(liste)
# root.mainloop()

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

def graphique_cumulatif(liste):
    """Graphique avec rectangles superposés"""
    bas = 500
```

```
    for x in liste:
        hauteur = x * echelle
        canvas.create_rectangle(100,bas,200,bas-hauteur,fill=une_couleur())
        bas = bas - hauteur

    # Bonus : Coordonnées verticales à gauche
    max_liste = sum(liste)
    canvas.create_line(90,500,90,500-echelle*max_liste)
    for j in range(0,max_liste+1,5):
        canvas.create_line(85,500-echelle*j,90,500-echelle*j)
        canvas.create_text(80,500-echelle*j,text=str(j))

    return

# Test
# echelle = 5
# liste = [5,8,6,3,7,10,4,12]
# graphique_cumulatif(liste)
# root.mainloop()

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

def graphique_pourcentage(liste):
    """Graphique rectangulaire divisé en sous-rectangles"""
    somme = sum(liste)
    posx = 100
    for x in liste:
        largeur = x/somme*100 * 5
        canvas.create_rectangle(posx,300,posx+largeur,200,fill=une_couleur())
        posx = posx + largeur

    # Bonus : Coordonnées horizontales en dessous
    canvas.create_line(100,325,600,325)
    for i in range(0,11):
        canvas.create_line(100+i*50,325,100+i*50,330)
        canvas.create_text(100+i*50,340,text=str(i*10)+"%")
    return

# Test
# echelle = 5
# liste = [5,8,6,3,7,10,4,12]
# graphique_pourcentage(liste)
# root.mainloop()

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

def graphique_secteurs(liste):
    """Graphique en camenbert"""
    somme = sum(liste)
    debut_angle = 0
    for x in liste:
        angle = x/somme*360
        canvas.create_arc(200,100,550,450,start=debut_angle,extent=angle,style=PIESLICE,fill
    ↪ =une_couleur())
        debut_angle = debut_angle + angle
    return

# Test
# echelle = 5
# liste = [5,8,6,3,7,10,4,12]
# graphique_secteurs(liste)
```

```
# root.mainloop()

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

longueurs = [randint(5,15) for i in range(103)]

liste = [15,8,6,13,17,10,14,12]

def action_bouton1():
    global echelle
    echelle = 15
    canvas.delete("all")
    graphique_barres(liste)
    return

def action_bouton2():
    global echelle
    echelle = 4
    canvas.delete("all")
    graphique_cumulatif(liste)
    return

def action_bouton3():
    canvas.delete("all")
    graphique_pourcentage(liste)
    return

def action_bouton4():
    canvas.delete("all")
    graphique_secteurs(liste)
    return

def nouvelle_liste():
    """Génère une nouvelle liste aléatoire"""
    global liste
    n = randint(3,10)
    liste = [randint(1,20) for i in range(n)]
    canvas.delete("all")
    return

# Titre
root.title("Visualisation de données")

# Boutons
bouton_quitter = Button(root,text="Quitter", width=8, command=root.quit)
bouton_quitter.pack(side=BOTTOM, padx=5, pady=20)

bouton_changer = Button(root,text="Changer les données", width=30, command=nouvelle_liste)
bouton_changer.pack(side=BOTTOM, padx=5, pady=20)

bouton1 = Button(root,text="Graphique en barres", width=30, command=action_bouton1)
bouton1.pack(padx=5, pady=20)

bouton2 = Button(root,text="Graphique cumulatif", width=30, command=action_bouton2)
bouton2.pack(padx=5, pady=20)

bouton3 = Button(root,text="Graphique en pourcentage", width=30, command=action_bouton3)
bouton3.pack(padx=5, pady=20)

bouton4 = Button(root,text="Graphique en secteurs", width=30, command=action_bouton4)
bouton4.pack(padx=5, pady=20)

root.mainloop()
```

Activité 3

Activité 3 statistique_3.py

```
##############################
# Statistique - Visualisation de données - tkinter
##############################

##############################
# Activité 3 - Calculs statistiques (bis)
##############################

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

from math import *
from random import *

def 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
        med = (liste_triee[indice_milieu-1]+liste_triee[indice_milieu]) / 2
    else:
        indice_milieu = (n-1)//2
        med = liste_triee[indice_milieu]

    return med

# Test
print("--- Médiane ---")
liste = [5,18,6,3]
print(liste)
print(mediane(liste))


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

def notes_vers_liste(effectif_notes):
    """ Convertit un effectif de notes en une liste de notes
    Entrée : une liste d'effectif de notes
    Sortie : la liste des notes """
    liste = []
    for i in range(len(effectif_notes)):
        nb = effectif_notes[i]
        liste = liste + [i]*nb

    return liste

# Test
print("--- Liste à partir d'un effectif ---")
effectif_notes = [0,0,0,0,0,1,0,2,0,1,5,1,2,3,2,4,1,2,0,1,0]
# effectif_notes = [randint(1,5) for i in range(21)]
print(effectif_notes)
print(notes_vers_liste(effectif_notes))
```

```
def mediane_notes(effectif_notes):
    """ Calcule la médiane des notes
    Entrée : une liste d'effectif de notes
    Sortie : la médiane """
    liste = notes_vers_liste(effectif_notes)
    med = mediane(liste)

    return med

# Test
print("--- Médiane des notes ---")
effectif_notes = [0,0,0,0,0,1,0,2,0,1,5,1,2,3,2,4,1,2,0,1,0]
print(effectif_notes)
print(mediane_notes(effectif_notes))


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

def calcule_quartiles(liste):
    """ Calcule les quartiles de la liste
    Entrée : une liste de nombre
    Sortie : leur quartile Q1, Q2=med, Q3 """
    med = mediane(liste)

    liste_triee = sorted(liste)
    n = len(liste_triee)
    indice_milieu = n//2
    if n%2 == 0: # si n pair
        liste_inf = liste[:indice_milieu]
        liste_sup = liste[indice_milieu:]
    else: # n impair
        liste_inf = liste[:indice_milieu+1]
        liste_sup = liste[indice_milieu:]
    Q1 =  mediane(liste_inf)
    Q3 =  mediane(liste_sup)

    return Q1, med, Q3

# Test
print("--- Quartiles ---")
liste = [3,4,5,7,12,50,100]
print(liste)
print(calcule_quartiles(liste))

##################################################
def quartiles_notes(effectif_notes):
    """ Calcule les quartiles des notes
    Entrée : une liste d'effectif de notes
    Sortie : les quartiles """
    liste = notes_vers_liste(effectif_notes)
    Q1,Q2,Q3 = calcule_quartiles(liste)

    return Q1, Q2, Q3

# Test
print("--- Quartiles des notes ---")
effectif_notes = [0,0,0,0,0,1,0,2,0,1,5,1,2,3,2,4,1,2,0,1,0]
print(effectif_notes)
print(quartiles_notes(effectif_notes))
```

Activité 4

Activité 4 statistique_4.py

```
##############################
# Statistique - Visualisation de données - tkinter
##############################

##############################
# Activité 4 - Visualisation des données (bis)
##############################

from math import *
from tkinter import *
from random import *

from statistique_3 import *

root = Tk()  # Fenêtre tkinter

canvas = Canvas(root, width=800, height=600, background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

echelle = 15  # Echelle pour mieux voir les données

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

def diagramme_boite(effectif_notes):
    """ Boîte à moustaches """

    # Graphique en barres
    for i in range(len(effectif_notes)):  # i varie de 0 à 20
        hauteur = effectif_notes[i] * echelle
        canvas.create_rectangle(100+2*i*10,300,100+(2*i+1)*10,300-hauteur,fill="red")
        canvas.create_text(100+2*i*10+5,320,text=str(i),fill="red")

    # Coordonnées verticale à gauche
    max_effectifs = max(effectif_notes)
    canvas.create_line(95,300,95,300-echelle*max_effectifs)
    for j in range(max_effectifs+1):
        canvas.create_line(90,300-echelle*j,95,300-echelle*j)
        canvas.create_text(85,300-echelle*j,text=str(j))

    # Passage à une liste
    liste = notes_vers_liste(effectif_notes)

    # Calculs des quartiles & co
    mini = min(liste)
    maxi = max(liste)
    Q1,Q2,Q3 = calcule_quartiles(liste)
    #  Diagramme
    canvas.create_rectangle(100+2*mini*10+5,197,100+2*Q1*10+5,203,fill="blue")
    canvas.create_rectangle(100+2*Q1*10+5,185,100+2*Q3*10+5,215,width=3,outline="blue")
    canvas.create_rectangle(100+2*Q2*10+5-2,185,100+2*Q2*10+5+2,215,fill="blue")
    canvas.create_rectangle(100+2*Q3*10+5,197,100+2*maxi*10+5,203,fill="blue")

    return

# Test
effectif_notes = [0,0,0,0,0,1,0,2,0,1,5,1,2,3,2,4,1,2,0,1,0]
diagramme_boite(effectif_notes)
root.mainloop()
```

Activité 5

Activité 5 statistique_5.py

```
##############################
# Statistique - Visualisation de données - tkinter
##############################

##############################
# Activité 5 - Visualisation des données (bis)
##############################

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

from random import *

def cours_bourse(n):
    """ Simulation de n jours de bourse """
    val = 1000
    liste_val = [val]
    for i in range(n-1):
        val = val + randint(-10,12)/3
        liste_val = liste_val + [val]

    return liste_val

# Test
print(cours_bourse(100))

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

def graphique_point(liste):
    """ Affiche la courbe des cours """
    # Base 1000 en y = 300
    canvas.create_line(100,300,100+365,300,width=3)

    # Coordonnées verticale à gauche
    canvas.create_line(95,420,95,80)
    for j in range(-1,3):
        canvas.create_line(90,300-100*j,95,300-100*j)
        canvas.create_text(72,300-100*j,text=str(1000+j*100))

    # Un point par jour
    for i in range(len(liste)):
        canvas.create_rectangle(100+i,300+1000-liste[i],100+i+1,300+1000-liste[i],outline="
    ↪ red")

    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)

# Test
liste = cours_bourse(365)
# graphique_point(liste)
# root.mainloop()
```

```
##################################################
## Question 3 ##
def moyenne_mobile(liste,duree):
    """ Calcule la moyenne mobile """
    moy_mob = []
    for i in range(len(liste)-duree+1):
        moy = sum(liste[i:i+duree])/duree
        moy_mob = moy_mob + [moy]

    return moy_mob

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

##################################################
## Question 4 ##
def graphique_moyenne_mobile(liste):
    """ Affiche les moyennes mobiles à 7 et 30 jours """
    # moyenne 7 derniers jours
    moyenne_7 = moyenne_mobile(liste,7)
    for i in range(len(moyenne_7)):
        canvas.create_rectangle(100+i+6,300+1000-moyenne_7[i],100+i+6+1,300+1000-moyenne_7[i
    ↪ ],outline="blue")

    # moyenne 30 derniers jours
    moyenne_30 = moyenne_mobile(liste,30)
    for i in range(len(moyenne_30)):
        canvas.create_rectangle(100+i+29,300+1000-moyenne_30[i],100+i+29+1,300+1000-
    ↪ moyenne_30[i],outline="sienna")

    return

# Test
liste = cours_bourse(365)
graphique_point(liste)           # Le cours au jour le jour
graphique_moyenne_mobile(liste)  # La moyenne à 7 et 30 jours
root.mainloop()
```

9. Fichiers

Activité 1

Activité 1 fichier_1.py

```
##################################
# Fichier
##################################
from random import *

##################################
# Activité 1 - Ecrire un fichier
##################################
```

```
## Question 1 ##

def ecrire_fichier_notes():

    # Création d'un fichier en écriture
    nom_fichier = "notes.txt"
    fic = open(nom_fichier,"w")

    # Listes nom
    liste_prenoms = ["Gargamel","Robin","Hermione","Arsène","Alice","James","Tintin"]
    liste_noms = ["Skywalker","Lupin","Voldemort","Tchoupi","Bond","Tartuffe","Dubois"]

    for i in range(7):
        prenom = choice(liste_prenoms)
        nom = choice(liste_noms)
        notes = str(randint(10,40)/2) + " " + str(randint(10,40)/2) + " " + str(randint
    ↪ (10,40)/2)
        ligne = prenom + " " + nom + " " + notes + "\n"

        # Ecriture dans le fichier
        fic.write(ligne)

    # Fermeture du fichier
    fic.close()

    return

# Test

print("--- Fichier 'notes.txt' ---")
ecrire_fichier_notes()

## Question 2 ##

def ecrire_fichier_moyennes():

    # Fichier à lire
    fichier_notes = "notes.txt"
    fic_in = open(fichier_notes,"r")

    # Fichier à écrire
    fichier_moyennes = "moyennes.txt"
    fic_out = open(fichier_moyennes,"w")

    for ligne in fic_in:
        liste = ligne.split()
        moyenne = (float(liste[2])+float(liste[3])+float(liste[4]))/3
        #moyenne_str = str(moyenne)
        moyenne_str = '{0:.2f}'.format(moyenne)
        nouv_ligne = liste[0] + " " + liste[1] + " " + moyenne_str + "\n"
        fic_out.write(nouv_ligne)

    # Fermeture des fichiers
    fic_in.close()
    fic_out.close()
    return

print("--- Fichier 'moyenne.txt' ---")
ecrire_fichier_moyennes()
```

Activité 2

Activité 2 fichier_2.py

```
##############################
# Fichier
##############################

from random import *
import matplotlib.pyplot as plt

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

## Question 1 ##

def fichier_ventes():

    # Création d'un fichier en écriture
    nom_fichiers = "ventes.csv"
    fic = open(nom_fichiers,"w")

    # Listes nom
    liste_produits = ["Vélo VTT","Planche de surf","Chaussures de courses","Raquette de
    ↪ badminton","Ballon de volley"]

    # Lignes du haut
    fic.write("Meilleures ventes de la société 'Pentathlon'\n\n")
    fic.write(",2013,2014,2015,2016,2017,2018\n\n")

    for produit in liste_produits:

        # Génération des chiffres de vente au hasard
        ventes = ""
        for i in range(6):
            ventes = ventes + "," + str(randint(50,100)*10)

        ligne = produit + ventes  + "\n"

        # Ecriture dans le fichier
        fic.write(ligne)

    # Fermeture du fichier
    fic.close()

    return

print("--- Fichier 'ventes.csv' ---")
fichier_ventes()

## Question 2 ##

def afficher_ventes():

    # Fichier à lire
    fichier_ventes = "ventes.csv"
    fic_in = open(fichier_ventes,"r")

    num_ligne = 0
    for ligne in fic_in:
        if num_ligne > 3: # on oublie les lignes de titres
            liste = ligne.split(",")
            donnees = [float(x) for x in liste[1:]]
            plt.plot(donnees)

        num_ligne += 1
```

```
    # Fermeture des fichiers
    fic_in.close()

    # Affichage
    plt.grid()
    plt.show()

    return

print("--- Affichages graphique de 'ventes.csv' ---")
afficher_ventes()
```

Activité 3

Activité 3 fichier_3.py

```
##############################
# Fichier
##############################

import os

##############################
# Activité 3 - Images
##############################

## Question 1 ##

def ecrire_fichier_image_nb():

    # Création d'un fichier en écriture
    nom_fichier = "image_nb.pbm"
    fic = open(nom_fichier,"w")

    # Entete
    fic.write("P1\n")  # Image noir et blanc
    nb_col = 300
    nb_lig = 200
    fic.write(str(nb_col) + " " + str(nb_lig) + "\n")

    for i in range(nb_lig):
        ligne = ""
        for j in range(nb_col):
            coul = (i+j)//10 % 2
            ligne = ligne + str(coul) + " "
        ligne = ligne + "\n"

        # Ecriture dans le fichier
        fic.write(ligne)

    # Fermeture du fichier
    fic.close()

    return
# Test

print("--- Fichier 'image.pbm' ---")
ecrire_fichier_image_nb()

## Question 2 ##
```

```
def ecrire_fichier_image_gris():

    # Création d'un fichier en écriture
    nom_fichier = "image_gris.pgm"
    fic = open(nom_fichier,"w")

    # Entete
    fic.write("P2\n")  # Image en niveaux de gris
    nb_col = 200
    nb_lig = 200
    fic.write(str(nb_col) + " " + str(nb_lig) + "\n")
    niveaux = 255
    fic.write(str(niveaux) + "\n")

    for i in range(nb_lig):
        ligne = ""
        for j in range(nb_col):
            coul = (i**2 + j**2) % 256   # un niveau de gris en fonction de i et j
            ligne = ligne + str(coul) + " "
        ligne = ligne + "\n"

        # Ecriture dans le fichier
        fic.write(ligne)

    # Fermeture du fichier
    fic.close()

    return

# Test

print("--- Fichier 'image.pgm' ---")
ecrire_fichier_image_gris()

## Question 3 ##

def ecrire_fichier_image_coul():

    # Création d'un fichier en écriture
    nom_fichier = "image_coul.ppm"
    fic = open(nom_fichier,"w")

    # Entete
    fic.write("P3\n")  # Image en couleurs
    nb_col = 200
    nb_lig = 200
    fic.write(str(nb_col) + " " + str(nb_lig) + "\n")
    niveaux = 255
    fic.write(str(niveaux) + "\n")

    for i in range(nb_lig):
        ligne = ""
        for j in range(nb_col):
            R = (i*j) % 256   # niveau de rouge
            V = i % 256        # niveau de vert
            B = (i+j)//3 % 256   # niveau de bleu

            ligne = ligne + str(R) + " " + str(V) + " "  + str(B) + "   "
        ligne = ligne + "\n"

        # Ecriture dans le fichier
        fic.write(ligne)

    # Fermeture du fichier
    fic.close()

    return
```

```
# Test
print("--- Fichier 'image.ppm' ---")
ecrire_fichier_image_coul()


## Question 4 ##
def inverser_couleurs_nb(fichier):
    # Fichier à lire
    fic_in = open(fichier,"r")
    # Fichier à écrire
    nom, extension = os.path.splitext(fichier)
    nouv_nom = nom + "_inverse"+extension
    fic_out = open(nouv_nom,"w")

    i = 0    # Numéro de ligne
    for ligne in fic_in:
        if i<2:     # Garder les 2 premières lignes
            fic_out.write(ligne)
        else:
            liste = ligne.split()
            nouv_ligne = ""
            for l in liste:
                if l == "1":
                    nouv_ligne = nouv_ligne + "0 "
                else:
                    nouv_ligne = nouv_ligne + "1 "
            nouv_ligne = nouv_ligne + "\n"
            fic_out.write(nouv_ligne)
        i = i + 1
    # Fermeture des fichiers
    fic_in.close()
    fic_out.close()
    return
print("--- Inversion des noirs et blancs ---")
inverser_couleurs_nb("simple_nb.pbm")

## Question 4 ##

def formule_couleur_vers_gris(R,V,B):
    gris = round(0.21*R + 0.72*V + 0.07*R)
    return gris
def couleurs_vers_gris(fichier):
    # Fichier à lire
    fic_in = open(fichier,"r")
    # Fichier à écrire
    nom, extension = os.path.splitext(fichier)
    nouv_nom = nom + "_gris"+".pgm"
    fic_out = open(nouv_nom,"w")

    i = 0    # Numéro de ligne
    for ligne in fic_in:
        if i == 0:
```

```
            fic_out.write("P2\n")  # Image en niveaux de gris
        elif i == 1 or i == 2:     # Garder les lignes 2 et 3
            fic_out.write(ligne)
        else:
            liste = ligne.split()
            nouv_ligne = ""

            j = 0  # Numéro de colonne
            while j < len(liste):
                R = int(liste[j])
                V = int(liste[j+1])
                B = int(liste[j+2])
                gris = formule_couleur_vers_gris(R,V,B)
                nouv_ligne = nouv_ligne + str(gris) + " "

                j = j + 3

            nouv_ligne = nouv_ligne + "\n"
            fic_out.write(nouv_ligne)

        i = i + 1

    # Fermeture des fichiers
    fic_in.close()
    fic_out.close()
    return

print("--- Couleurs vers niveaux de gris ---")
couleurs_vers_gris("image_coul.ppm")
```

Activité 4

Activité 4 fichier_4.py

```
################################
# Fichier
################################

from math import *
import os

################################
# Activité 4 - Distances entre les villes
################################

from math import *

## Question 1 ##

def distance_xy(x1,y1,x2,y2):
    return sqrt( (x2-x1)**2 + (y2-y1)**2 )

## Question 2 ##

def fichier_distances_xy(fichier):
    # Fichier à lire
    fic_in = open(fichier,"r")

    liste_villes = []

    for ligne in fic_in:
        liste_villes = liste_villes + [ligne.split()]
```

```
    # Fermeture du fichier
    fic_in.close()

    # Fichier à écrire
    nom, extension = os.path.splitext(fichier)
    nouv_nom = nom + "_distances"+".txt"
    fic_out = open(nouv_nom,"w")

    ligne = '{:>10s}'.format("")
    for v in liste_villes:
        nom = v[0]
        ligne = ligne + '{:>10s}'.format(nom) +" "

    fic_out.write(ligne + "\n")

    for v1 in liste_villes:
        nom1 = v1[0]
        x1 = float(v1[1])
        y1 = float(v1[2])

        ligne = '{:10s}'.format(nom1)

        for v2 in liste_villes:
            nom2 = v2[0]
            x2 = float(v2[1])
            y2 = float(v2[2])

            d = distance_xy(x1,y1,x2,y2)

            ligne = ligne + '{:10d}'.format(round(d)) + " "

        fic_out.write(ligne + "\n")

    return

print("--- Villes xy ---")
fichier_distances_xy("villes_xy.txt")

## Question 3 ##

def degres_vers_radians(a):
    return 2*pi*a/360

def distance_approx_lat_long(lat1,long1,lat2,long2):
    R = 6371  # rayon (moyen) de la Terre
    x = (long2-long1)*cos( (lat1+lat2)/2 )
    y = lat2-lat1
    d = R * sqrt( x**2 + y**2 )
    return d

# Test
Paris = (48.853,2.350)
Paris_radians = (degres_vers_radians(Paris[0]),degres_vers_radians(Paris[1]))

New_York = (40.713,-74.006)
New_York_radians = (degres_vers_radians(New_York[0]),degres_vers_radians(New_York[1]))

print("--- Distances approchées Terre ---")
d = distance_approx_lat_long(*Paris_radians,*New_York_radians)
print(d)

def distance_lat_long(lat1,long1,lat2,long2):
    R = 6371  # rayon (moyen) de la Terre
    a = (sin((lat2-lat1)/2))**2 + cos(lat1)*cos(lat2)*(sin((long2-long1)/2))**2
    d = 2 * R * atan2(sqrt(a),sqrt(1-a))
    return d
```

```
# Test
print("--- Distances exactes Terre ---")
d = distance_lat_long(*Paris_radians,*New_York_radians)
print(d)

## Question 3 ##

def fichier_distances_lat_long(fichier):
    # Fichier à lire
    fic_in = open(fichier,"r")

    liste_villes = []

    for ligne in fic_in:
        liste_villes = liste_villes + [ligne.split()]

    # Fermeture du fichier
    fic_in.close()

    # Fichier à écrire
    nom, extension = os.path.splitext(fichier)
    nouv_nom = nom + "_distances"+".txt"
    fic_out = open(nouv_nom,"w")

    ligne = '{:>12s}'.format("")
    for v in liste_villes:
        nom = v[0]
        ligne = ligne + '{:>12s}'.format(nom) +" "

    fic_out.write(ligne + "\n")

    for v1 in liste_villes:
        nom1 = v1[0]
        lat1 = degres_vers_radians(float(v1[1]))
        long1 = degres_vers_radians(float(v1[2]))

        ligne = '{:12s}'.format(nom1)

        for v2 in liste_villes:
            nom2 = v2[0]
            lat2 = degres_vers_radians(float(v2[1]))
            long2 = degres_vers_radians(float(v2[2]))

            d = distance_lat_long(lat1,long1,lat2,long2)

            ligne = ligne + '{:12d}'.format(round(d)) + " "

        fic_out.write(ligne + "\n")

    return

print("--- Villes lat_long ---")
fichier_distances_lat_long("villes_lat_long.txt")
```

10. Arithmétique – Boucle tant que – II

Activité 1

Activité 1 tantque_4.py

```
##############################
# Tant que - Booléen - Arithmétiques
##############################

##############################
# Activité 4 - Conjecture(s) de Goldbach
##############################

from math import *

##############################
# Rappel : activité 2

def est_premier(n):
    if n < 2: return False
    if n == 2: return True
    if n % 2 == 0: return False

    d = 3
    while (n % d != 0) and (d**2 <= n):
        d = d + 2

    if d ** 2 > n:
        return True
    else:
        return False

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

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

# La (vraie) conjecture de Goldbach (1742) :
# tout entier pair plus grand que 3 est la somme de deux nombres premiers

def nombre_solutions_goldbach(n):
    """ Calcule le nb de décompositions n = p + q avec
    n pair ; p, q premiers et q >= p """

    # Si entier pas pair, c'est terminé
    if n % 2 == 1:
        print("Attention ! Entier non pair.")
        return None

    nb_sol = 0

    for p in range(2,n//2+1):
        q = n - p
        if (q>=p) and (est_premier(p)) and (est_premier(q)):
            print("n =",n,"somme de p =",p,", q = ",q)
            nb_sol = nb_sol + 1

    return nb_sol

# Test
print("--- Conjecture de Goldbach ---")
```

```
print(nombre_solutions_goldbach(100))

def test_conjecture_goldbach(nmax):
    """ Vérifie la validité de la conjecture de Goldbach
    pour les entiers pairs de 4 jusqu'à nmax """

    print("Début du test")
    for n in range(4,nmax,2):
        if nombre_solutions_goldbach(n) == 0:
            print("Problème avec n = ",n)
    print("Fin du test")
    return

# Test
print("--- Conjecture de Goldbach ---")
test_conjecture_goldbach(10000)

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

# Goldbach 1752 :
# tout entier impair n peut s'écrire sous la forme
# n = p + 2k^2
# avec p premier et k entier (éventuellement nul)

def existe_decomposition_goldbach(n):
    """ Teste si n impair peut se décomposer n = p + 2k^2
    avec p premier et k entier """

    maxk = ceil(sqrt(n/2))+1
    for k in range(maxk):
        p = n - 2 * k**2
        if est_premier(p):
            # print(n,p,k,n-p-2*k**2)
            return True
    return False

def contre_exemple_goldbach(nmax):
    """ Cherche un contre-exemple à la seconde conjecture de Goldbach """
    print("--- Début de la recherche ---")
    for m in range(1,nmax):
        n = 2 * m + 1
        if existe_decomposition_goldbach(n) == False:
            print("Contre-exemple :",n)

    print("--- Fin de la recherche ---")
    return

# Test
print("--- Test conjecture fausse de Goldbach ---")
print("Avec 5777 : ",existe_decomposition_goldbach(5777))
contre_exemple_goldbach(10000)
```

Activité 2

Activité 2 tantque_5.py

```
################################
# Tant que - Booléen - Arithmétiques
################################

################################
# Activité 5 - Nombres ayant 4 ou 8 diviseurs
################################

# Conjecture : entre 1 et N, il y a toujours plus d'entiers ayant
# 4 diviseurs que d'entier ayant 8 diviseurs

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

def nombre_de_diviseurs_1(n):
    """ Nombre de diviseurs de n (y compris 1  et n) """
    nb = 0
    for d in range(1,n+1):
        if n % d == 0:
            nb = nb + 1
    return nb

def nombre_de_diviseurs_2(n):
    """ Nombre de diviseurs de n (y compris 1  et n) """
    nb = 2  # on compte déjà 1 et n
    for d in range(2,n//2+1):
        if n % d == 0:
            nb = nb + 1
    return nb

# On garde la méthode la plus efficace
def nombre_de_diviseurs(n):
    return nombre_de_diviseurs_2(n)

# Test
print("--- Nombre de diviseurs ---")
print(nombre_de_diviseurs(100))

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

## Question 2 ##

def quatre_et_huit_diviseurs(Nmin,Nmax):
    nb_quatre = 0
    nb_huit = 0
    for n in range(Nmin,Nmax):
        nb = nombre_de_diviseurs(n)
        if nb == 4:
            nb_quatre = nb_quatre + 1
        if nb == 8:
            nb_huit = nb_huit + 1
    return nb_quatre, nb_huit

# Test
print("--- Nombre ayant 4 puis 8 diviseurs ---")
print(quatre_et_huit_diviseurs(1,100))
```

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

## Question 3 ##

# Recherche de contre-exemple à la conjecture
# Il faut prendre N assez grand par exempel
# entre 1 et N = 250000 il y a plus d'entiers
# ayant 8 diviseurs que 4 diviseurs

# Par tranche de 50 000 (les calculs sont longs !)

# print(quatre_et_huit_diviseurs(1,50000))
# print(quatre_et_huit_diviseurs(50000,100000))
# print(quatre_et_huit_diviseurs(100000,150000))
# print(quatre_et_huit_diviseurs(150000,200000))
# print(quatre_et_huit_diviseurs(200000,250000))

# Tranche 1 : 12073, 10957
# Tranche 2 : 11254, 11224
# Tranche 3 : 10995, 11229
# Tranche 4 : 10838, 11218
# Tranche 5 : 10690, 11260

# 4 diviseurs 12073+11254+10995+10838+10690   = 55850
# 8 diviseurs 10957+11224+11229+11218+11260   = 55888
```

Activité 3

Activité 3 tantque_6.py

```
##############################
# Tant que - Booléen - Arithmétiques
##############################

##############################
# Activité 6 - Conjecture fausse : 1211111... n'est jamais premier
##############################

##############################
# Rappel : activité 2
def est_premier(n):
    if n < 2: return False
    if n == 2: return True
    if n % 2 == 0: return False

    d = 3
    while (n % d != 0) and (d**2 <= n):
        d = d + 2

    if d ** 2 > n:
        return True
    else:
        return False

##############################
## Question 1 ##
def un_deux_un(k):
```

```
    """ Calcule un entier 121111... """
    u = 12
    for i in range(k):
        u = 10*u + 1
    return u

# Test
print("--- 121111.... ---")
u = un_deux_un(10)
print(u)


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

# Conjecture 1211111... n'est jamais premier

def test_premier_un_deux_un(kmax):
    """ Teste si 121111... est premier ou pas """
    for k in range(kmax):
        uk = un_deux_un(k)
        print(uk,est_premier(uk))
    return

# Test
print("--- Test conj 121111.... jamais premier ---")
test_premier_un_deux_un(21)

# N'aboutira pas au contre-exemple
# Les calculs sont trop longs

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

def est_presque_premier(n,r):
    """ Teste si n n'a pas de diviseurs <= r """

    if n < 2: return False
    if n == 2: return True
    if n % 2 == 0: return False

    d = 3
    while (n % d != 0) and (d ** 2 <= n) and (d <= r):
        d = d + 2

    if (d ** 2 > n) or (d > r):
        return True
    else:
        return False

# Test
print("--- Test presque premier ---")
print(est_presque_premier(101,13))

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

## Question 4 ##

def test_presque_un_deux_un(kmax):
    """ Teste si 121111... est presque premier """

    n = 12
    for k in range(kmax):
        if est_presque_premier(n,1000000):
            print(k,'Presque premier',n)
```

```
        n = 10*n + 1
    return

# Test
print("--- Test conj 121111.... jamais presque premier ---")
test_presque_un_deux_un(151)
```

11. Binaire I

Activités binaire_I.py

```
##############################
# Binaire - partie I
##############################

##############################
# Activité 1 - Decimale vers entier
##############################

## Question 1 ##

def decimale_vers_entier_1(liste_decimale):
    nombre = 0
    p = len(liste_decimale)
    for i in range(p):
        d = liste_decimale[p-1-i]
        nombre = nombre + d*10**i

    return nombre

## Question 1bis ##

def decimale_vers_entier_2(liste_decimale):
    nombre = 0
    i = 0
    for d in reversed(liste_decimale):
        nombre = nombre + d*10**i
        i = i + 1

    return nombre

# Test
print("--- Ecriture décimale vers entier ---")
liste = [1,2,3,4]
print(decimale_vers_entier_1(liste))
print(decimale_vers_entier_2(liste))

##############################
# Activité 2 - Binaire vers entier
##############################


## Question 1 ##

def binaire_vers_entier_1(liste_binaire):
    nombre = 0
    p = len(liste_binaire)
    for i in range(p):
        if liste_binaire[p-1-i] == 1:
```

```
            nombre = nombre + 2**i

    return nombre

## Question 1bis ##

def binaire_vers_entier_2(liste_binaire):
    nombre = 0
    i = 0
    for b in reversed(liste_binaire):
        if b == 1:
            nombre = nombre + 2**i
        i = i + 1

    return nombre


## Question 2 ##

def binaire_vers_entier_bis(liste_binaire):
    nombre = 0
    for b in liste_binaire:
        if b == 0:
            nombre = nombre*2
        else:
            nombre = nombre*2 + 1

    return nombre


# Test
print("--- Ecriture binaire vers entier ---")
liste = [1,1,0,1,1,0,0,1]
print(binaire_vers_entier_1(liste))
print(binaire_vers_entier_2(liste))
print(binaire_vers_entier_bis(liste))

## Question 3 (à virer) ##

# Utilise bin() [pas très fantastique]

def liste_vers_chaine(liste_binaire):
    liste_chaine = [str(b) for b in liste_binaire]
    chaine = "".join(liste_chaine)
    chaine = "0b" + chaine
    return chaine

def binaire_vers_entier_4(liste_binaire):

    chaine = liste_vers_chaine(liste_binaire)
    nombre = int(chaine,2)

    return nombre

# print("--- Ecriture binaire vers entier avec bin() ---")
# liste = [1,1,0,1,1,0,0,1]
# print(liste_vers_chaine(liste))
# print(binaire_vers_entier_4(liste))


##############################
# Activité 3 - Ecriture décimale
##############################

def entier_vers_decimale(n):
    if n==0: return [0]
```

```
    liste_decimale = []
    while n != 0:
        liste_decimale = [n%10] + liste_decimale
        n = n//10
    return liste_decimale

# Test
print("--- Entier vers écriture décimale ---")
n = 1234
liste = entier_vers_decimale(n)
entier = decimale_vers_entier_1(liste)
print(n)
print(liste)
print(entier)


##############################
# Activité 4 - Ecriture binaire
##############################

def entier_vers_binaire(n):
    if n==0: return [0]
    liste_binaire = []
    while n != 0:
        liste_binaire = [n%2] + liste_binaire
        n = n//2
    return liste_binaire

# Test
print("--- Entier vers écriture binaire ---")
n = 1234
liste = entier_vers_binaire(n)
entier = binaire_vers_entier_1(liste)
print(n)
print(liste)
print(entier)
```

12. Listes II

Activité 1

Activité 1 listes_II_1.py

```
##############################
# Listes II
##############################

##############################
# Activité 1 - Manipulation de listes
##############################

## Question 1 ##

##############################
def multiplier(liste,k):
    return [k*x for x in liste]
```

```
## Question 2 ##
################################
def puissance(liste,k):
    return [x**k for x in liste]

## Question 3 ##
################################
def addition(liste1,liste2):
    liste_add = []
    for i in range(len(liste1)):
        liste_add.append(liste1[i]+liste2[i])
    return liste_add

## Question 4 ##
################################
def non_zero(liste):
    return [x for x in liste if x != 0]

## Question 5 ##
################################
def pairs(liste):
    return [x for x in liste if x % 2 == 0]

# Test
print("--- Multiplier ---")
print(multiplier([1,2,3,4,5],2))
print("--- Puissance ---")
print(puissance([1,2,3,4,5],3))
print("--- Addition ---")
print(addition([1,2,3],[4,5,6]))
print("--- Sans zéro ---")
print(non_zero([1,0,2,3,0,4,5,0]))
print("--- Pairs ---")
print(pairs([1,0,2,3,0,4,5,0]))
```

Activité 2

Activité 2 listes_II_2.py

```
################################
# Listes II
################################
from random import *

################################
# Activité 2 - Somme fixée
################################
################################

from random import *
liste_20 = [randint(1,99) for i in range(20)]
```

```
liste_20 = [16, 2, 85, 27, 9, 45, 98, 73, 12, 26, 46, 25, 26, 49, 18, 99, 10, 86, 7, 42]
print(liste_20)

liste_200 = [randint(1,99) for i in range(200)]
print(liste_200)

## Question 1 ##

##############################
# Trouver deux élément consécutifs dont la somme vaut 100

def somme_deux_consecutifs_100(liste):
    n = len(liste)
    for i in range(n-1):
        if liste[i]+liste[i+1] == 100:
            # print(i,i+1,liste[i],liste[i+1])
            return True
    return False

## Question 2 ##

##############################
# Trouver deux élément différents dont la somme vaut 100
def somme_deux_100(liste):
    n = len(liste)
    for i in range(n-1):
        for j in range(i+1,n):
            if liste[i]+liste[j] == 100:
                # print(i,j,liste[i],liste[j])
                return True
    return False

## Question 3 ##

##############################
# Suite de termes consécutifs qui font 100
def somme_suite_100(liste):
    n = len(liste)
    for i in range(n):
        somme = 0
        j = i
        while somme < 100 and j < n:
            somme = somme  + liste[j]
            j = j + 1
        if somme == 100:
            # print(i,j-1,liste[i:j])
            return True
    return False

##############################
print("--- Somme deux consécutifs ---")
print(somme_deux_consecutifs_100(liste_20))
print(somme_deux_consecutifs_100(liste_200))

print("--- Somme deux qcq ---")
print(somme_deux_100(liste_20))
print(somme_deux_100(liste_200))

print("--- Somme suites ---")
print(somme_suite_100(liste_20))
print(somme_suite_100(liste_200))

## Question 3 ##
```

```
##############################
# Idées proba : Quelle long donne prob > 1/2

##############################
def proba_1(n,N):
    nb = 0
    for k in range(N):
        liste_n = liste_n = [randint(1,99) for i in range(n)]
        trouve = somme_deux_consecutifs_100(liste_n)
        if trouve:
            nb += 1
    return nb/N

##############################
def proba_2(n,N):
    nb = 0
    for k in range(N):
        liste_n = liste_n = [randint(1,99) for i in range(n)]
        trouve = somme_deux_100(liste_n)
        if trouve:
            nb += 1
    return nb/N

##############################
def proba_3(n,N):
    nb = 0
    for k in range(N):
        liste_n = liste_n = [randint(1,99) for i in range(n)]
        trouve = somme_suite_100(liste_n)
        if trouve:
            nb += 1
    return nb/N


print("--- Proba deux consécutifs ---")
# Proba ~ 1/2 pour longueur n ~ 67
print(proba_1(67,10000))

print("--- Proba deux ---")
# Proba ~ 1/2 pour longueur n ~ 12
print(proba_2(12,10000))

print("--- Proba suite ---")
# Proba ~ 1/2 pour longueur n ~ 42
print(proba_3(42,10000))
```

Activité 3

Activité 3 listes_II_3.py

```
##############################
# Listes II
##############################

##############################
# Activité 3 - Tableau à deux dimensions
##############################
```

```
## Question 1 ##

##############################
def somme_diagonale(tableau):
    n = len(tableau)
    somme = 0
    for i in range(n):
        somme = somme + tableau[i][i]
    return somme

## Question 2 ##

##############################
def somme_anti_diagonale(tableau):
    n = len(tableau)
    somme = 0
    for i in range(n):
        somme = somme + tableau[n-1-i][i]
    return somme

## Question 3 ##

##############################
def somme_tout(tableau):
    n = len(tableau)
    somme = 0
    for i in range(n):
        for j in range(n):
            somme = somme + tableau[i][i]
    return somme

## Question 4 ##

##############################
def affiche_tableau(tableau):
    """
    Affiche un tableau carré à l'écran
    Entrée : un tableau de taille n x n
    Sortie : rien (affichage à l'écran)
    """

    n = len(tableau)
    for i in range(n):
        for j in range(n):
            print('{:>3d}'.format(tableau[i][j]),end="")
        print()
    return


##############################
tableau = [ [1,2,3], [4,5,6], [7,8,9] ]

print("--- Somme diagonale ---")
print(somme_diagonale(tableau))

print("--- Somme anti-diagonale ---")
print(somme_anti_diagonale(tableau))

print("--- Somme tout ---")
print(somme_tout(tableau))

print("--- Affichage ---")
```

```
affiche_tableau(tableau)
```

Activité 4

Activité 4 listes_II_4.py

```
##############################
# Listes II - Idées
##############################

from random import *

##############################
# Activité 3 - Rappels
##############################

######################################################
def affiche_tableau(tableau):
    """
    Affiche un carré à l'écran
    Entrée : un tableau de taille n x n
    Sortie : rien (affichage à l'écran)
    """

    n = len(tableau)

    for i in range(n):
        for j in range(n):
            print('{:>3d}'.format(tableau[i][j])," ", end="")
        print()

    return

##############################
def somme_diagonale(tableau):
    n = len(tableau)
    somme = 0
    for i in range(n):
        somme = somme + tableau[i][i]
    return somme

##############################
def somme_anti_diagonale(tableau):
    n = len(tableau)
    somme = 0
    for i in range(n):
        somme = somme + tableau[n-1-i][i]
    return somme

##############################
# Activité 4 - Carrés magiques
##############################

## Question 1 ##

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

print("--- Carré magique ---")
# carre = [ [1,2,3], [4,5,6], [7,8,9] ]
carre_3x3 = [ [4,9,2], [3,5,7], [8,1,6] ]
```

```
carre_4x4 = [ [1,14,15,4], [7,9,6,12], [10,8,11,5], [16,3,2,13] ]
print("--- Carré 3x3 ---")
affiche_tableau(carre_3x3)
print("--- Carré 4x4 ---")
affiche_tableau(carre_4x4)

## Question 2 ##

##############################
def est_carre_magique(carre):
    n = len(carre)
    total = n * (n**2 + 1) // 2

    if somme_diagonale(carre) != total:
        return False

    if somme_anti_diagonale(carre) != total:
        return False

    for ligne in carre:
        if sum(ligne) != total:
            return False

    for j in range(n):
        somme = 0
        for i in range(n):
            somme = somme + carre[i][j]
        if somme != total:
            return False

    return True

print("--- Vérification carré magique ---")
print(est_carre_magique(carre_3x3))
print(est_carre_magique(carre_4x4))

## Question 3 ##

##############################
def carre_aleatoire(n):

    entiers = list(range(1,n**2+1))
    shuffle(entiers)
    carre = [ entiers[i*n:(i+1)*n] for i in range(n) ]
    return carre

print("--- Carré aléatoire ---")
carre = carre_aleatoire(4)
affiche_tableau(carre)
print(est_carre_magique(carre))

## Question 4 ##

##############################
def addition_carre(carre,k):

    n = len(carre)
    nouv_carre = [[0 for j in range(n)] for i in range(n)]

    for i in range(n):
        for j in range(n):
            nouv_carre[i][j] = carre[i][j] + k

    return nouv_carre
```

```
## Question 4 ##

##############################
def multiplication_carre(carre,k):

    n = len(carre)
    nouv_carre = [[0 for j in range(n)] for i in range(n)]

    for i in range(n):
        for j in range(n):
            nouv_carre[i][j] = carre[i][j] * k

    return nouv_carre

# Test
print("--- Addition, multiplication de carrés magiques ---")
# carre = [ [1,2,3], [4,5,6], [7,8,9] ]
carre = [ [4,9,2], [3,5,7], [8,1,6] ]
somme_carre = addition_carre(carre,-1)
produit_carre = multiplication_carre(carre,9)
affiche_tableau(carre)
affiche_tableau(somme_carre)
affiche_tableau(produit_carre)

## Question 5 ##

##############################
def homothetie_carre(carre,k):

    n = len(carre)
    nouv_carre = [[0 for j in range(k*n)] for i in range(k*n)]

    for i in range(k*n):
        for j in range(k*n):
            nouv_carre[i][j] = carre[i//k][j//k]

    return nouv_carre

# Test
print("--- Homothétie carré magique ---")
# carre = [ [1,2,3], [4,5,6], [7,8,9] ]
# carre = [ [4,9,2], [3,5,7], [8,1,6] ]
# grand_carre = homothetie_carre(carre,3)
# affiche_tableau(grand_carre)

grand_carre = homothetie_carre(carre_3x3,3)
affiche_tableau(grand_carre)

grand_carre = homothetie_carre(carre_4x4,2)
affiche_tableau(grand_carre)

## Question 6 ##

##############################
def addition_bloc_carre(grand_carre,petit_carre):

    N = len(grand_carre)
    n = len(petit_carre)
    # k = N//n

    nouv_carre = [[0 for j in range(N)] for i in range(N)]

    for i in range(N):
        for j in range(N):
            nouv_carre[i][j] = grand_carre[i][j] + petit_carre[i%n][j%n]

    return nouv_carre
```

```
    # Test
print("--- Addition de blocs - Carré magique ---")
petit_carre = [ [1,2], [3,4] ]
carre = [ [1,2,3], [4,5,6], [7,8,9] ]
grand_carre = homothetie_carre(carre,2)
nouv_grand_carre = addition_bloc_carre(grand_carre,petit_carre)
affiche_tableau(petit_carre)
print("---")
affiche_tableau(grand_carre)
print("---")
affiche_tableau(nouv_grand_carre)

## Question 7 ##

##############################
def produit_carres(carre1,carre2):
    n = len(carre1)
    # m = len(carre2)

    carre3a = addition_carre(carre2,-1)
    # print("---")
    # affiche_tableau(carre3a)
    carre3b = homothetie_carre(carre3a,n)
    # print("---")
    # affiche_tableau(carre3b)
    carre3c = multiplication_carre(carre3b,n**2)
    # print("---")
    # affiche_tableau(carre3c)
    carre3d = addition_bloc_carre(carre3c,carre1)
    # print("---")
    # affiche_tableau(carre3d)

    return carre3d

#### Test ####
carre1 = [ [4,9,2], [3,5,7], [8,1,6] ]
carre2 = [ [4,14,15,1], [9,7,6,12], [5,11,10,8], [16,2,3,13] ]
carre3 = produit_carres(carre1,carre2)
print("--- Produit carrés ---")
affiche_tableau(carre1)
print("---")
affiche_tableau(carre2)
print("---")
affiche_tableau(carre3)
print(est_carre_magique(carre3))

#### Produit non commutatif ####
carre4 = produit_carres(carre2,carre1)
print("--- Produit carrés ---")
affiche_tableau(carre1)
print("---")
affiche_tableau(carre2)
print("---")
affiche_tableau(carre4)
print(est_carre_magique(carre4))

#### Carré de taille 36 x 36 ####
carre5 = produit_carres(carre1,carre4)
```

13. Binaire II

Activités binaire_II.py

```
##############################
# Binaire - partie II
##############################

from binaire_I import *

##############################
# Activité 1 - Palindrome en binaire
##############################

## Question 1 ##

def est_palindrome_1(liste):

    p = len(liste)

    drapeau = True
    for i in range(p):
        if liste[i] != liste[p-1-i]:
            drapeau =  False

    return drapeau

# Version optimisée :
def est_palindrome_1_bis(liste):

    p = len(liste)

    for i in range(p//2):
        if liste[i] != liste[p-1-i]:
            return False

    return True

def est_palindrome_2(liste):
    liste_inverse = list(reversed(liste))
    return liste == liste_inverse

# Test
print("--- Test d'un palindrome ---")
liste = [1,0,1,0,0,1,0,1]
print(est_palindrome_1(liste))
print(est_palindrome_1_bis(liste))
print(est_palindrome_2(liste))


## Question 2 ##

def cherche_palindrome_binaire(N):
    num = 0
    for n in range(N):
        liste_binaire = entier_vers_binaire(n)
        if est_palindrome_1(liste_binaire) == True:
            num = num + 1
            print(num,":",n,"=",entier_vers_binaire(n))
    return

# Test
print("--- Palindromes binaires ---")
```

```
cherche_palindrome_binaire(1000)

# Le 1000ème palindrome en binaire est :
#249903 = [1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1]

## Question 3 ##

def cherche_palindrome_decimale(N):
    num = 0
    for n in range(N):
        liste_decimale = entier_vers_decimale(n)
        if est_palindrome_1(liste_decimale) == True:
            num = num + 1
            print(num,":",n)
    return

# Test
print("--- Palindromes avec décimales ---")
cherche_palindrome_decimale(1000)

# Le 1000ème palindrome en décimales est :
# 90009

## Question 4 ##

def cherche_bi_palindrome(N):
    num = 0
    for n in range(N):
        liste_binaire = entier_vers_binaire(n)
        liste_decimale = entier_vers_decimale(n)
        if est_palindrome_1(liste_binaire) == True and est_palindrome_1(liste_decimale):
            num = num + 1
            print(num,":",n,"=",entier_vers_binaire(n))
    return

# Test
print("--- Bi-palindromes ---")
cherche_bi_palindrome(1000)

# Le 20ème bi-palindrome est
#  585585 = [1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1]


##############################
# Activité 2 - Opérations logiques
##############################

## Question 1 ##

def OUeg(l1,l2):
    n = len(l1)
    l = []
    for i in range(n):
        if l1[i]==1 or l2[i]==1:
            l = l + [1]
        else:
            l = l + [0]

    return l

def ETeg(l1,l2):
    n = len(l1)
    l = []
    for i in range(n):
        if l1[i]==1 and l2[i]==1:
```

```
            l = l + [1]
        else:
            l = l + [0]
    return l

def NON(l1):
    l = []
    for b in l1:
        if b==1:
            l = l + [0]
        else:
            l = l + [1]
    return l

# Test
print("--- Opérations logiques (même longueur) ---")
l1 = [1,0,1,0,1,0,1]
l2 = [1,0,0,1,0,0,1]
print(l1)
print(l2)
print(OUeg(l1,l2))
print(ETeg(l1,l2))
print(NON(l1))

## Question 2 ##

# Rajouter des zéros non significatifs si besoins

def ajouter_zeros(liste,p):
    while len(liste)< p:
        liste = [0] + liste

    return liste

# Test
print("--- Zeros non significatifs ---")
print(ajouter_zeros([1,0,1,1],8))

## Question 3 ##

# Opérations logiques avec des listes de tailles différentes

def OU(l1,l2):
    p = len(l1)
    q = len(l2)
    if p>q:
        ll2 = ajouter_zeros(l2,p)
        return OUeg(l1,ll2)
    else:
        ll1 = ajouter_zeros(l1,q)
        return OUeg(ll1,l2)

def ET(l1,l2):
    p = len(l1)
    q = len(l2)
    if p>q:
        ll2 = ajouter_zeros(l2,p)
        return ETeg(l1,ll2)
    else:
        ll1 = ajouter_zeros(l1,q)
        return ETeg(ll1,l2)
```

```
# Test
print("--- Opérations logiques (cas général) ---")
l1 = [1,0,1,0,1,0,1]
l2 = [1,0,0,1,0,]
print(l1)
print(l2)
print(OU(l1,l2))
print(ET(l1,l2))

##############################
# Activité 3 - Loi de Morgan
##############################

## Question 1 ##

def tous_les_binaires(p):

    liste_p = []
    for n in range(2**p):
        liste_p = liste_p + [entier_vers_binaire(n)]
    return liste_p

# Test
print("--- Tous les binaires ---")
print(tous_les_binaires(3))

## Question 2 ##

def toutes_les_listes(p):
    if p == 0:
      return []
    if p == 1:
      return [[0],[1]]

    liste_p_1 = toutes_les_listes(p-1)

    liste_p = [ [0] + l for l in liste_p_1] + [ [1] + l for l in liste_p_1]

    return liste_p

# Test
print("--- Toutes les listes ---")
print(toutes_les_listes(3))


## Question 3 ##

# Lois de Morgan

def test_loi_de_morgan(p):
    liste_tous = [ajouter_zeros(l,p) for l in tous_les_binaires(p)]
    #liste_tous = toutes_les_listes(p)
    for l1 in liste_tous:
        for l2 in liste_tous:
            non_l1_ou_l2 = NON(OU(l1,l2))
            non_l1_et_non_l2 = ET(NON(l1),NON(l2))
            if non_l1_ou_l2 == non_l1_et_non_l2:
                print("Vrai")
                # pass
            else:
                print("Faux",l1,l2)

    return

# Test
```

```
print("--- Test loi de Morgan ---")
test_loi_de_morgan(2)
```

14. Probabilités – Paradoxe de Parrondo

Activités proba.py

```
##############################
# Probablité - Paradoxe de Parrondo
##############################

# Référence : "Paradoxe de Parrondo", La gazette des mathématiciens, juillet 2017

from random import *

##############################
# Activité 1 - Jeu A : premier jeu perdant
##############################

## Question 1 ##

def tirage_jeu_A():
    x = random()
    if x <= 0.49:
        return +1
    else:
        return -1

## Question 2 ##

def gain_jeu_A(N):
    gain = 0
    for i in range(N):
        gain = gain + tirage_jeu_A()

    return gain

## Question 3 ##

def esperance_jeu_A(N):
    esperance = gain_jeu_A(N)/N
    return esperance

# Test
print("--- Jeu A ---")
N = 1000000
print(esperance_jeu_A(N))

##############################
# Activité 2 - Jeu B : premier jeu perdant
##############################

## Question 1 ##

def tirage_jeu_B(g):
    if g%3 == 0:
        x = random()
        if x <= 0.09:
            return +1
```

```
        else:
            return -1
    else:
        x = random()
        if x <= 0.74:
            return +1
        else:
            return -1

## Question 2 ##

def gain_jeu_B(N):
    gain = 0
    for i in range(N):
        gain = gain + tirage_jeu_B(gain)

    return gain

## Question 3 ##

def esperance_jeu_B(N):
    esperance = gain_jeu_B(N)/N
    return esperance

# Test
print("--- Jeu B ---")
N = 1000000
print(esperance_jeu_B(N))

##############################
# Activité 3 -  Paradoxe de Parrondo
##############################

## Question 1 ##
def tirage_jeu_AB(g):
    x = random()
    if x < 0.5:
        return tirage_jeu_A()
    else:
        return tirage_jeu_B(g)

## Question 2 ##
def gain_jeu_AB(N):
    gain = 0
    for i in range(N):
        gain = gain + tirage_jeu_AB(gain)
    return gain

## Question 3 ##
def esperance_jeu_AB(N):
    esperance = gain_jeu_AB(N)/N
    return esperance

# Test
print("--- Jeu AB ---")
N = 1000000
print(esperance_jeu_AB(N))
```

15. Chercher et remplacer

Activité 1

Activité 1 chercher_1.py

```
##############################
# Chercher et remplacer
##############################

##############################
# Activité 1 - Chercher
##############################

## Question 1 ##

def chercher_in(chaine,sous_chaine):
    return sous_chaine in chaine

# Test
print("--- Avec 'in' ---")

chaine = "ETRE OU NE PAS ETRE"
sous_chaine = "PAS"
print(chercher_in(chaine,sous_chaine))

## Question 2 ##

def chercher_find(chaine,sous_chaine):
    position = chaine.find(sous_chaine)
    return position

# Test
print("--- Avec find() ---")

position = chercher_find(chaine,sous_chaine)
print(position)

position = chercher_find(chaine,"XYZ")
print(position)

## Question 3 ##

def chercher_index(chaine,sous_chaine):
    position = chaine.index(sous_chaine)
    return position

# Test
print("--- Avec index() ---")

position = chercher_index(chaine,sous_chaine)
print(position)

# position = chercher_index(chaine,"XYZ")
# print(position)

## Question 4 ##

def chercher(chaine,sous_chaine):
    long_chaine = len(chaine)
    long_sous_chaine = len(sous_chaine)
    for i in range(long_chaine-long_sous_chaine+1):
        trouve = True
```

```
        for j in range(long_sous_chaine):
            if chaine[i+j] != sous_chaine[j]:
                trouve = False
                break
        if trouve == True:
            return i
    return None

# Test

print("--- A la main ---")

position = chercher(chaine,sous_chaine)
print(position)

position = chercher(chaine,"XYZ")
print(position)
```

Activité 2

Activité 2 chercher_2.py

```
##############################
# Chercher et remplacer
##############################

##############################
##############################
# Rappel de l'activité 1

def chercher(chaine,sous_chaine):
    long_chaine = len(chaine)
    long_sous_chaine = len(sous_chaine)
    for i in range(long_chaine-long_sous_chaine+1):
        trouve = True
        for j in range(long_sous_chaine):
            if chaine[i+j] != sous_chaine[j]:
                trouve = False
                break
        if trouve == True:
            return i
    return None


##############################
# Activité 2 - Remplacer
##############################

chaine = "ETRE OU NE PAS ETRE"
sous_chaine = "PAS"
nouv_sous_chaine = "PLUS"

## Question 1 ##

print("--- Avec replace() ---")
nouv_chaine = chaine.replace(sous_chaine,nouv_sous_chaine)
print(nouv_chaine)

## Question 2 ##
```

```
#
# remplacer_une_fois() A la main en utilisant chercher()
def remplacer(chaine,sous_chaine,nouv_sous_chaine):
    pos = chercher(chaine,sous_chaine)

    if pos is not None:  # Si trouvé
        finpos = pos+len(sous_chaine)
        chaine = chaine[:pos]+nouv_sous_chaine+chaine[finpos:]

    return chaine

print("--- Remplacer : à la main ---")
nouv_chaine = remplacer(chaine,sous_chaine,nouv_sous_chaine)
print(nouv_chaine)

## Question 3 ##

#
# remplacer() remplace toutes les occurences

def remplacer_tout(chaine,sous_chaine,nouv_sous_chaine):

    pos = chercher(chaine,sous_chaine)

    while pos is not None:  # Tant que trouvé
        finpos = pos+len(sous_chaine)
        chaine = chaine[:pos]+nouv_sous_chaine+chaine[finpos:]
        pos = chercher(chaine,sous_chaine)

    return chaine

# Attention fonction un peu trop basique car A -> AB devrait boucler indféiniment.

print("--- Remplacer tout : à la main ---")
chaine = "ETRE OU NE PAS ETRE"
sous_chaine = "ETRE"
nouv_sous_chaine = "AVOIR"
nouv_chaine = remplacer_tout(chaine,sous_chaine,nouv_sous_chaine)
print(nouv_chaine)
```

Activité 3

Activité 3 chercher_3.py

```
##############################
# Chercher et remplacer
##############################

##############################
# Activité 3 - Regex - Expressions rationnelles
##############################

## Question 1 ##

from re import *

def python_regex_chercher(chaine,exp):
    trouve = search(exp,chaine)
    if trouve:
        return trouve.group(), trouve.start(), trouve.end()
```

```
    else:
        return None

# Test
print("--- Avec regex search() ---")
chaine = "ETRE OU NE PAS ETRE"
exp = "P.S"
print(python_regex_chercher(chaine,exp))

exp = "E..E"
print(python_regex_chercher(chaine,exp))

exp = "[OT]U"
print(python_regex_chercher(chaine,exp))

exp = "[MN]..P[AI]S"
print(python_regex_chercher(chaine,exp))


## Question 2 ##

# Le joker "."

def regex_chercher_joker(chaine,exp):

    long_chaine = len(chaine)
    long_exp = len(exp)

    for i in range(long_chaine-long_exp+1):
        trouve = True
        for j in range(long_exp):
            if exp[j] != "." and chaine[i+j] != exp[j]:
                trouve = False
                break
        if trouve == True:
            return chaine[i:i+long_exp],i,i+long_exp
    return None

# Test
print("--- regex joker ---")
chaine = "ETRE OU NE PAS ETRE"
exp = "T.E"
print(regex_chercher_joker(chaine,exp))

## Question 3 ##

# Le choix "[AB]", voir [ABC]

def genere_choix(exp):
    liste_exp = [""]
    mode_choix = False
    for c in exp:
        if c == "[":
            mode_choix = True
            old_liste_exp = list(liste_exp)
            nouv_list_exp = []
        elif c == "]":
            mode_choix = False
            liste_exp = nouv_list_exp
        else:
            if mode_choix == False:  # Mode normal
                liste_exp = [ l + c for l in liste_exp]
            else:  # Mode choix
                nouv_list_exp = nouv_list_exp + [ l + c for l in old_liste_exp]

    return liste_exp
```

```
# Test
print("--- regex choix ---")
exp = "ET[XYZ]RE[UVW]"
print(genere_choix(exp))

def regex_chercher_choix(chaine,exp):
    liste_exp = genere_choix(exp)
    for mon_exp in liste_exp:
        resultat = python_regex_chercher(chaine,mon_exp)
        if resultat is not None:
            return resultat

    return None

# Test
print("--- regex choix ---")
chaine = "ETRE OU NE PAS ETRE"
exp = "P[ABC]S"
print(regex_chercher_choix(chaine,exp))

## Question 4 ##

# La négation [^A] voir [^AB]
```

Activité 4

Activité 4 chercher_4.py

```
################################
# Chercher et remplacer
################################

################################
# Rappel de l'activité 1 - Chercher
################################
def chercher(chaine,sous_chaine):
    long_chaine = len(chaine)
    long_sous_chaine = len(sous_chaine)
    for i in range(long_chaine-long_sous_chaine+1):
        trouve = True
        for j in range(long_sous_chaine):
            if chaine[i+j] != sous_chaine[j]:
                trouve = False
                break
        if trouve == True:
            return i
    return None

################################
# Rappel de l'activité 2 - Remplacer
################################
def remplacer(chaine,sous_chaine,nouv_sous_chaine):
    pos = chercher(chaine,sous_chaine)

    if pos is not None:  # Si trouvé
        finpos = pos+len(sous_chaine)
```

```
        chaine = chaine[:pos]+nouv_sous_chaine+chaine[finpos:]
    return chaine

##############################
# Activité 4 - Itérations
##############################

## Question 1 ##

 # Test

print("--- Une itération ---")

print("-- Ex 1 --")
phrase = "01001110"
motif = "01"
nouv_motif = "10"
nouv_phrase = remplacer(phrase,motif,nouv_motif)
print(phrase)
print(nouv_phrase)

print("-- Ex 2 --")
phrase = "01001110"
motif = "0011"
nouv_motif = "1100"
nouv_phrase = remplacer(phrase,motif,nouv_motif)
print(phrase)
print(nouv_phrase)

print("-- Ex 3 --")
phrase = "01001110"
motif = "0011"
nouv_motif = "111000"
nouv_phrase = remplacer(phrase,motif,nouv_motif)
print(phrase)
print(nouv_phrase)

print("-- Ex 4 --")
phrase = "0001"
motif = "01"
nouv_motif = "1100"
print(phrase)
phrase = remplacer(phrase,motif,nouv_motif)
print(phrase)
phrase = remplacer(phrase,motif,nouv_motif)
print(phrase)
phrase = remplacer(phrase,motif,nouv_motif)
print(phrase)


## Question 2 ##

# Constante globale du maximum d'itérations considérées
MAX_ITER = 1000

def iterations(phrase,motif,nouv_motif):
    i = 0
    while i <= MAX_ITER:
        nouv_phrase = remplacer(phrase,motif,nouv_motif)
        if phrase == nouv_phrase:
            return i, phrase
        else:
```

```
            phrase = nouv_phrase
            i = i+1
    return None

print("--- Itérations ---")
print("-- Ex 1 --")
phrase = "000011011"
motif = "0011"
nouv_motif = "1100"
resultat = iterations(phrase,motif,nouv_motif)
print(resultat)

phrase = "000011011"
print(phrase)
phrase = remplacer(phrase,motif,nouv_motif)
print(phrase)
phrase = remplacer(phrase,motif,nouv_motif)
print(phrase)
phrase = remplacer(phrase,motif,nouv_motif)
print(phrase)
phrase = remplacer(phrase,motif,nouv_motif)
print(phrase)
phrase = remplacer(phrase,motif,nouv_motif)
print(phrase)
phrase = remplacer(phrase,motif,nouv_motif)
print(phrase)

print("-- Ex 2 --")

phrase = "000011011"
motif = "001"
nouv_motif = "11000"
resultat = iterations(phrase,motif,nouv_motif)
print(resultat)

## Rappel sur binaire ##

def decimal_vers_binaire(n,p):
    chaine_b = bin(n)  # Conversion en une chaîne écriture binaire
    chaine_b = chaine_b[2:]  # On enlève le préfixe

    # On rajoute des zéros au début si besoin
    nb_zeros = p - len(chaine_b)
    for i in range(nb_zeros):
        chaine_b = "0" + chaine_b

    return chaine_b

# Test
print(decimal_vers_binaire(33,8))

## Question 3 ##

def iteration_maximale(p,motif,nouv_motif):

    maxi_iter = 0
    phrase_maxi_iter = ""
    nouv_phrase_maxi_iter = ""

    for n in range(2**p):
        phrase = decimal_vers_binaire(n,p)
        resultat = iterations(phrase,motif,nouv_motif)
        #print(resultat)
        if resultat is None:
```

```
            return None, phrase
        else:
            nb_iter = resultat[0]
            if nb_iter > maxi_iter:
                maxi_iter = nb_iter
                phrase_maxi_iter = phrase
                nouv_phrase_maxi_iter = resultat[1]
    return maxi_iter, phrase_maxi_iter, nouv_phrase_maxi_iter

print("--- Itérations maximales ---")

# Exemple
motif = "01"
nouv_motif = "100"
print(iteration_maximale(4,motif,nouv_motif))

## Question 4 ##

# Linéaire
motif = "0011"
nouv_motif = "110"
print("- Linéaire -")
print(iteration_maximale(10,motif,nouv_motif))

# Quadratique
motif = "01"
nouv_motif = "10"
print("- Quadratique -")
print(iteration_maximale(10,motif,nouv_motif))

# Exponentiel
motif = "01"
nouv_motif = "110"
print("- Exponentiel -")
print(iteration_maximale(10,motif,nouv_motif))

# Ne termine pas
motif = "01"
nouv_motif = "1100"
print("- Ne se termine pas -")
print(iteration_maximale(4,motif,nouv_motif))
```

16. Calculatrice polonaise – Piles

Activité 1

Activité 1 piles_1.py

```
##############################
# Piles - Calculatrice polonaise
##############################

##############################
# Activité 1 - Opération sur la pile
##############################

# "pile" est une variable globale
```

```
## Question 1 ##
def empile(element):
    """  Ajoute un élément au sommet de la pile
    Entrée : un objet
    Sortie : rien
    Action : la pile contient un élément en plus """
    global pile     # Pour pouvoir modifier la pile
    pile = pile + [element]
    return None

# Test
print("--- Empiler ---")
pile = [4,5,6]
print('Pile avant :',pile)
empile(7)
print('Pile après :',pile)

## Question 2 ##
def depile():
    """ Lit l'élément au sommet de la pile et l'enlève
    Entrée : rien
    Sortie : l'élément du sommet
    Action : la pile contient un élément de moins """
    global pile
    sommet = pile[len(pile)-1]
    pile = pile[0:len(pile)-1]
    return sommet

# Test
print("--- Dépiler ---")
pile = [4,5,6]
print('Pile avant :',pile)
val = depile()
print('Valeur dépilée :',val,'\nPile après :',pile)

## Question 3 ##
def pile_est_vide():
    """ Détermine si la pile est vide ou pas
    Entrée : rien
    Sortie : vrai/faux
    Action : ne modifie pas la pile """
    if len(pile) == 0:
        return True
    else:
        return False

# Tests
print("--- Tester si pile vide ---")
# Test 1
pile = [4,5,6]
vide = pile_est_vide()
print(pile,'pile vide ?',vide)
```

```
# Test 2
pile = []
vide = pile_est_vide()
print(pile,'pile vide ?',vide)
```

Activité 2

Activité 2 piles_2.py

```
##############################
# Piles - Calculatrice polonaise
##############################

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

##############################
# Activité 2 - Manipulation de la pile
##############################

## Question 1 ##

print("--- Manipulation ---")
pile = []
empile(5)
empile(7)
empile(2)
empile(4)
print(pile)
depile()
empile(8)
empile(1)
empile(3)
print(pile)
val =  depile()
print('Valeur :',val)

## Question 2 ##

def pile_contient(element):
    """  Détermine si la pile contient l'élément
    Entrée : rien
```

```
    Sortie : vrai/faux
    Action : modifie la pile """

    while not pile_est_vide():
        el = depile()
        if el == element:
            return True    # Si on trouve l'élément c'est bon

    return False     # On arrive au bas sans trouver l'élément

# Tests
print("--- Test si pile contient 7 ---")

# Test 1
pile = [4,5,6]
print(pile,'pile contient 7 ?',pile_contient(7))

# Test 2
pile = [4,7,12,99]
print(pile,'pile contient 7 ? ',pile_contient(7))

## Question 3 ##

def somme_pile():
    """ Calcule la somme de la pile
    Entrée : rien
    Sortie : la somme
    Action : vide la pile """

    somme = 0
    while not pile_est_vide():
        element = depile()
        somme = somme + element

    return somme

# Test
print("--- Somme des valeurs de la pile ---")
pile = [4,5,6]
print('La somme de',pile,'est ',somme_pile())

## Question 4 ##

def avant_dernier():
    """ Renvoie l'avant-dernier élément en bas de la pile
    Entrée : rien
    Sortie : l'avant-dernier élément
    Action : vide la pile """

    dernier = None
    avant_dernier = None

    while not pile_est_vide():
        avant_dernier = dernier  # Le dernier devient avant-dernier
        dernier = depile()       # Nouveau dernier

    return avant_dernier

# Tests
pile = [4,5,6,13]
print('L\'avant-dernier élément de',pile,'est',avant_dernier())

pile = [4,6]
print('L\'avant-dernier élément de',pile,'est',avant_dernier())
```

```
pile = [6]
print('L\'avant-dernier élément de',pile,'est',avant_dernier())

pile = []
print('L\'avant-dernier élément de',pile,'est',avant_dernier())
```

Activité 3

Activité 3 piles_3.py

```
##############################
# Piles - Calculatrice polonaise
##############################

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

##############################
# Activité 3 - La gare de triage
##############################

def tri_wagons(train):
    """ Trie les wagons rouges/bleus d'un train
    Entrée : un train avec des wagons bleus (nombre) et rouges (lettres)
    Sortie : les wagons triés avec les bleus d'abord et les rouges ensuite
    Action : utilise une pile """

    global pile   # Doit être globale pour pouvoir être modifiée
    pile = []

    nouv_train = ""

    for wagon in train.split():
        if wagon.isdigit():   # Wagon bleu directement dans le nouveau train
            nouv_train = nouv_train + wagon + " "
        else:                 # Wagon rouge en attente
            empile(wagon)

    # Tous les wagon bleus sont maintenant rangés
    # On s'occupe des wagons rouges en attente
    while not pile_est_vide():
        wagon = depile()
```

```
        nouv_train = nouv_train + wagon + " "
    return nouv_train

# Tests
print("--- Tri rouge/bleu ---")
train = "A 4 C 12"
train_trie = tri_wagons(train)
print(train,' -> ',train_trie)
train = "9 K 8 P 17 L B R 3 10 2 N"
train_trie = tri_wagons(train)
print(train,' -> ',train_trie)
```

Activité 4

Activité 4 piles_4.py

```
################################
# Piles - Calculatrice polonaise
################################

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

################################
# Activité 4 - Calculatrice polonaise
################################

## Question 1 ##
def operation(a,b,op):
    """ Calcule l'opération 'a + b 'ou 'a * b'...
    Entrée : a,b (nombres) et 'op' un ccaractère '+'' ou '*'
    Sortie : le résultat du calcul """
    if op == '+':
        return a + b
    if op == '*':
        return a * b
```

```
# Tests
print("--- Opérations ---")
a=5 ; b=7
print("La somme de",a,"et",b,"vaut",operation(a,b,'+'))
print("Le produit de",a,"et",b,"vaut",operation(a,b,'*'))

## Question 2 ##

def calculatrice_polonaise(expression):
    """ Calcule l'expression codée en notation polonaise
    Entrée : une expression en notation polonaise
    Sortie : le résultat du calcul
    Action : utilise la pile """

    global pile
    pile = []

    liste_expression = expression.split()

    for car in liste_expression:
        if (car == '+') or (car == '*'):
            b = depile()
            a = depile()
            calcul_partiel = operation(a,b,car)
            empile(calcul_partiel)
        else:
            val = int(car)
            empile(val)

    return depile()

# Tests
print("--- Calculatrice polonaise ---")

exp = "2 3 +"
print("L'expression",exp,"vaut",calculatrice_polonaise(exp))

exp = "2 3 + 5 *"
print("L'expression",exp,"vaut",calculatrice_polonaise(exp))

exp = "8 7 3 + *"
print("L'expression",exp,"vaut",calculatrice_polonaise(exp))

exp = "8 7 3 * +"
print("L'expression",exp,"vaut",calculatrice_polonaise(exp))
```

Activité 5

Activité 5 piles_5.py

```
##############################
# Piles - Calculatrice polonaise
##############################

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

##############################
# Activité 5 - Expression bien parenthésée
##############################

## Question 1 ##

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

# Test
print("--- Expression correctemment parenthésée ---")

expression = "(a+b)^2 = a^2 + (b^2+2(ab))"
print("L'expression",expression,"est bien parenthésées ?",parentheses_correctes(expression))

expression = "((a+b)^3 = (a+b)"
print("L'expression",expression,"est bien parenthésées ?",parentheses_correctes(expression))

expression = "(a+b)^4) = ((a+b)"
print("L'expression",expression,"est bien parenthésées ?",parentheses_correctes(expression))

## Question 2 ##

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("--- Expression avec crochets et parenthèses corrects ---")

expression = "(a+b)^2 = (a^2 + [b^2+[2(ab)]])"
print("L'expression",expression,"est bien parenthésées et crochetées ?",
    ↪ crochets_parentheses_correctes(expression))

expression = "((a+b)]^3 = [a+b]"
print("L'expression",expression,"est bien parenthésées et crochetées ?",
    ↪ crochets_parentheses_correctes(expression))

expression = "[a+b)^4] = (a+b)"
print("L'expression",expression,"est bien parenthésées et crochetées ?",
    ↪ crochets_parentheses_correctes(expression))
```

Activité 6

Activité 6 piles_6.py

```
################################
# Piles - Calculatrice polonaise
################################

global pile

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

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

def operation(a,b,op):
    if op == '+':
        return a + b
    if op == '*':
        return a * b

def calculatrice_polonaise(expression):
    global pile
    pile = []

    liste_expression = expression.split()

    for car in liste_expression:
        if (car == '+') or (car == '*'):
            b = depile()
            a = depile()
            calcul_partiel = operation(a,b,car)
            empile(calcul_partiel)
        else:
            val = int(car)
            empile(val)

    return depile()

##############################
# Activité 6 - Conversion vers l'écriture polonaise
##############################

def ecriture_polonaise(expression):
    """ Convertit une expression classique en notation polonaise
    Entrée : une expression classique
    Sortie : l'expression en notation polonaise
    Action : utilise une pile """

    global pile
    pile = []

    liste_expression = expression.split()

    polonaise = ""    # L'écriture polonaise

    for car in liste_expression:
        if car.isdigit():
            polonaise = polonaise + car + " "

        if car == "(":
            empile(car)

        if car == "*":
            empile(car)

        if car == "+":
            while not pile_est_vide():
                element = depile()
                if element == "*":
                    polonaise = polonaise + element + " "
                else:
```

```
                    empile(element)    # Remettre l'élément
                    break
            empile(car)

        if car == ")":
            while not pile_est_vide():
                element = depile()
                if element == "(":
                    break
                else:
                    polonaise = polonaise + element + " "

    while not pile_est_vide():
        element = depile()
        polonaise = polonaise + element + " "

    return polonaise

# Tests
print("--- Conversion en écriture polonaise ---")

exp = "2 + 3"
print("l'expression",exp,"s'écrit",ecriture_polonaise(exp))

exp = "2 * 3"
print("l'expression",exp,"s'écrit",ecriture_polonaise(exp))

exp = "( 2 + 3 ) * 4"
print("l'expression",exp,"s'écrit",ecriture_polonaise(exp))

exp = "4 * ( 2 + 3 )"
print("l'expression",exp,"s'écrit",ecriture_polonaise(exp))

exp = "2 + 4 * 5"
print("l'expression",exp,"s'écrit",ecriture_polonaise(exp))

exp = "2 * 4 * 5"
print("l'expression",exp,"s'écrit",ecriture_polonaise(exp))

exp = "( 2 + 3 ) * ( 4 + 8 )"
print("l'expression",exp,"s'écrit",ecriture_polonaise(exp))

##
# Automatisation des tests et des vérifications

def test_polonaise(expression):
    classique = eval(expression)
    print("---\n",classique)
    conversion = ecriture_polonaise(expression)
    print(conversion)
    polonaise = calculatrice_polonaise(conversion)
    print(polonaise)
    return classique == polonaise

exp = "2 + 3"
print(exp, "OK ?",test_polonaise(exp))

exp = "2 * 3 * 7"
print(exp, "OK ?",test_polonaise(exp))

exp = "( 2 + 3 ) * ( 4 + 8 )"
print(exp, "OK ?",test_polonaise(exp))

exp = "( ( 2 + 3 ) * 11 ) * ( 4 + ( 8 + 5 ) )"
print(exp, "OK ?",test_polonaise(exp))

exp = "( 17 * ( 2 + 3 ) ) + ( 4 + ( 8 * 5 ) )"
```

```
print(exp, "OK ?",test_polonaise(exp))
```

17. Visualiseur de texte – Markdown

Activité 1

Activité 1 markdown_1.py

```
##############################
# Visualiseur de texte - Markdown
##############################

##############################
# Activité 1 - Afficher du texte
##############################

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

from tkinter import *
from tkinter.font import Font

# Fenêtre tkinter
root = Tk()

canvas = Canvas(root, width=800, height=600, background="white")
canvas.pack(fill="both", expand=True)

# Format de la page de texte
largeur = 700
hauteur = 500

# Couleurs
couleur_fond = "lightgray"
couleur_texte = "black"

# Cadre
canvas.create_rectangle(10,10,largeur,hauteur,width=2,fill=couleur_fond)

# Fontes
fonte_texte = Font(family="Times", size=12)
fonte_italique = Font(family="Times", slant="italic", size=12)
fonte_gras = Font(family="Times", weight="bold", size=12)
fonte_titre = Font(family="Times", weight="bold", size=20)
fonte_sous_titre = Font(family="Times", weight="bold", size=16)

# Test
# canvas.create_text(100,100, text="Vive les maths !",anchor=NW,font=fonte_titre,fill=
    ↪ couleur_texte)
# canvas.create_text(200,200, text="Vive Python !",anchor=NW,font=fonte_sous_titre,fill="red
    ↪ ")
# root.mainloop()


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

def encadre_mot(mot,fonte):
    """ Encadre un mot
    Entrée : une chaîne et sa fonte
```

```
    Sortie : affichage du mot et d'un cadre (bounding box) """

    # Affiche un texte
    mot_canvas = canvas.create_text(100,100, text=mot,anchor=NW,font=fonte,fill=
    ↪ couleur_texte)

    # Coordonnées du rectangle (x1,y1,x2,y2)
    x1,y1,x2,y2 = canvas.bbox(mot_canvas)
    # print(cadre)

    # Affichage du cadre
    canvas.create_rectangle(x1,y1,x2,y2,width=2)

    return

# Test
# encadre_mot("Du texte avec Python",fonte_titre)
# root.mainloop()


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

def longueur_mot(mot,fonte):
    """ Longueur d'un mot
    Entrée : une chaîne et sa fonte
    Sortie : la longuer de ce mot """

    # Affiche un texte invisible juste pour récupérer sa longueur
    mot_canvas = canvas.create_text(100,100, text=mot,anchor=NW,font=fonte,fill=couleur_fond
    ↪ )

    # En extraire les extrémités
    x1,y1,x2,y2 = canvas.bbox(mot_canvas)

    return x2 - x1

# Test
# print("Longueur du mot 'Coucou' :",longueur_mot("Coucou",fonte_titre),"pixels")
# encadre_mot("Coucou",fonte_titre)
# root.mainloop()


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

def choix_fonte(mode,en_gras,en_italique):
    """ Renvoie une fonte selon les paramètres
    Entrée : un mode (texte ou liste, titre, sous-titre), gras ou pas, italique ou pas
    Sortie : la fonte """

    if mode == "titre":
        fonte = fonte_titre
    elif mode == "sous_titre":
        fonte = fonte_sous_titre
    else:       # Mode texte ou liste
        if en_gras:
            fonte = fonte_gras
        elif en_italique:
            fonte = fonte_italique
        else:
            fonte = fonte_texte

    return fonte

# Test
```

```
fonte = choix_fonte("texte",False,True)
canvas.create_text(100,100, text="Ceci est en italique",anchor=NW,font=fonte,fill=
    ↪ couleur_texte)
root.mainloop()

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

Activité 2

Activité 2 markdown_2.py

```
###############################
# Visualiseur de texte - Markdown
###############################

###############################
# Activité 2 - Afficher du markdown
###############################

from tkinter import *
from tkinter.font import Font

###############################
# A garder de l'activité 1
###############################

# Fenêtre tkinter
root = Tk()

canvas = Canvas(root, width=800, height=600, background="white")
canvas.pack(fill="both", expand=True)

# Format de la page de texte
largeur = 700
hauteur = 500

# Couleurs
couleur_fond = "lightgray"
couleur_texte = "black"

# Cadre
canvas.create_rectangle(10,10,largeur,hauteur,width=2,fill=couleur_fond)

# Fontes
fonte_texte = Font(family="Times", size=12)
fonte_italique = Font(family="Times", slant="italic", size=12)
fonte_gras =  Font(family="Times", weight="bold", size=12)
fonte_titre = Font(family="Times", weight="bold", size=20)
fonte_sous_titre = Font(family="Times", weight="bold", size=16)

#####################################################
def choix_fonte(mode,en_gras,en_italique):
    """
    Renvoie une fonte selon les paramètres
    Entrée : un mode (texte ou  liste,titre,sous-titre), gras ou pas, italique ou pas
    Sortie : la fonte
    """

    if mode == "titre":
```

```
        fonte = fonte_titre
    elif mode == "sous_titre":
        fonte = fonte_sous_titre
    else:       # Mode texte ou liste
        if en_gras:
            fonte = fonte_gras
        elif en_italique:
            fonte = fonte_italique
        else:
            fonte = fonte_texte

    return fonte


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

def afficher_ligne_v1(par,posy):
    """ Affiche le texte sur une seul ligne sans mise en forme
    Entrée : un paragraphe (c-à-d une longue ligne), la position verticale
    Sortie : affichage """

    posx = 20  # Début de la ligne tout à gauche

    liste_mots = par.split()
    for mot in liste_mots:

        mot = mot + " " # Rajoute espace qui sépare les mots

        mot_canvas = canvas.create_text(posx,posy, text=mot,anchor=NW,font=fonte_titre,fill=
    ↪ couleur_texte)
        canvas.create_rectangle(canvas.bbox(mot_canvas),width=2)

        # On place le nouveau mot après le précédent
        posx = canvas.bbox(mot_canvas)[2]

    return

# Tests
# afficher_ligne_v1("Bonjour, voici mon premier texte !",100)
# root.mainloop()

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

def afficher_ligne_v2(par,posy):
    """ Affiche le texte selon le mode titre, sous-titre, texte, liste
    Entrée : un paragraphe (c-à-d une longue ligne), la position verticale
    Sortie : affichage """

    # Par défaut : texte, sans indentation
    mode = "texte"
    indentation = 20

    if par[0:2] == "##":       # Sous_titre
        mode = "sous_titre"
        par = par[2:]          # Supprime les ##
    elif par[0] == "#":        # Titre
        mode = "titre"
        par = par[1:]          # Supprime le #
    elif par[0] == "+":        # liste
        mode = "liste"
        par = u'\u2022' + par[1:]          # Remplace le "+" par un rond
        indentation = 40
```

```
    # Début de la ligne (décalé si liste)
    posx = indentation

    liste_mots = par.split()
    for mot in liste_mots:

        fonte = choix_fonte(mode,False,False)

        mot = mot + " " # Rajoute espace qui sépare les mots
        mot_canvas = canvas.create_text(posx,posy, text=mot,anchor=NW,font=fonte,fill=
    ↪ couleur_texte)
        posx = canvas.bbox(mot_canvas)[2]

    return

# Tests
# afficher_ligne_v2("# Voici un titre",80)
# afficher_ligne_v2("## Et ici un sous titre",115)
# afficher_ligne_v2("Du texte normal, et une liste ci-dessous :",150)
# afficher_ligne_v2("+ Pomme",175)
# afficher_ligne_v2("+ Poire",200)
# afficher_ligne_v2("+ Scoubidou",225)
# root.mainloop()

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

def afficher_ligne_v3(par,posy):
    """ Affiche le texte selon gras et italique et selon le mode
    Entrée : un paragraphe (c-à-d une longue ligne), la position verticale
    Sortie : affichage """

    # Par défaut : texte, sans indentation
    mode = "texte"
    indentation = 20

    if par[0:2] == "##":       # Sous_titre
        mode = "sous_titre"
        par = par[2:]          # Supprime les ##
    elif par[0] == "#":        # Titre
        mode = "titre"
        par = par[1:]          # Supprime le #
    elif par[0] == "+":        # liste
        mode = "liste"
        par = u'\u2022' + par[1:]          # Remplace le "+" par un rond
        indentation = 40

    # Gras / pas gras (par défaut ni gras, ni italique)
    en_gras = False
    en_italique = False

    # Début de la ligne (décalé si liste)
    posx = indentation

    liste_mots = par.split()
    for mot in liste_mots:

        if mot == "**":  # Bascule gras / pas gras
            en_gras = not(en_gras)
            mot = ""

        if en_gras:
            fonte = fonte_gras

        if mot == "*":       # Bascule italique / pas italique
```

```
            en_italique = not(en_italique)
            mot = ""

        fonte = choix_fonte(mode,en_gras,en_italique)

        if mot != "":
            mot = mot + " " # Rajoute espace qui sépare les mots

        mot_canvas = canvas.create_text(posx,posy, text=mot,anchor=NW,font=fonte,fill=
    ↪ couleur_texte)
        posx = canvas.bbox(mot_canvas)[2]

    return

# Tests
# afficher_ligne_v3("Mot ** en gras ** et lui en * italique *",100)
# afficher_ligne_v3("+ Pommes et surtout ** poires ** et * ananas *",125)
# root.mainloop()


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

# Interligne
espace_entre_lignes = 18

def afficher_paragraphe(par,posy):
    """ Affiche le texte selon gras et italique et selon le mode
    Entrée : un paragraphe (c-à-d une longue ligne), la position verticale
    Sortie : affichage """

    # Par défaut : texte, sans indentation
    mode = "texte"
    indentation = 20

    if par[0:2] == "##":       # Sous_titre
        mode = "sous_titre"
        par = par[2:]          # Supprime les ##
    elif par[0] == "#":        # Titre
        mode = "titre"
        par = par[1:]          # Supprime le #
    elif par[0] == "+":        # liste
        mode = "liste"
        par = u'\u2022' + par[1:]          # Remplace le "+" par un rond
        indentation = 40

    # Gras / pas gras (par défaut ni gras, ni italique)
    en_gras = False
    en_italique = False

    # Début de la ligne (décalé si liste)
    posx = indentation

    liste_mots = par.split()
    for mot in liste_mots:

        if mot == "**":    # Bascule gras / pas gras
            en_gras = not(en_gras)
            mot = ""

        if en_gras:
            fonte = fonte_gras

        if mot == "*":    # Bascule italique / pas italique
            en_italique = not(en_italique)
```

```
            mot = ""

        fonte = choix_fonte(mode,en_gras,en_italique)

        if mot != "":
            mot = mot + " "    # Rajoute espace qui sépare les mots

        mot_canvas = canvas.create_text(posx,posy, text=mot,anchor=NW,font=fonte,fill=
    ↪ couleur_texte)
        posx = canvas.bbox(mot_canvas)[2]

        if posx > largeur:
            posx = indentation
            posy = posy + espace_entre_lignes

    return posy

# Tests
# afficher_paragraphe("# Titre Hello ! World",100)
# afficher_paragraphe("## Sous_titre Hello "*5,150)
# afficher_paragraphe("Hello Bonjour "*30,200)
# afficher_paragraphe("Hello ! ** GRAS **   * Italique *  ** TRES GRAS ** Rien  * Très
    ↪ italique * ** SUPER GRAS **",300)
# afficher_paragraphe("+ Pomme",350)
# afficher_paragraphe("+ Poire",370)
# afficher_paragraphe("Des mots, toujours de mots, encore des mots. "*10,10)
# root.mainloop()


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

def afficher_fichier(nom):
    """ Affiche les paragraphes d'un fichier
    Entrée : un nom de fichier au format markdown
    Sortie : affichage des paragraphes """

    # Ouvrir le fichier
    fichier = open(nom,"r")
    liste_paragraphes = fichier.readlines()
    fichier.close()

    posy = 50

    # Traiter chaque paragraphe
    for par in liste_paragraphes:
        newposy = afficher_paragraphe(par,posy)
        posy = newposy + espace_entre_lignes

    root.mainloop()

    return

# Tests
# afficher_fichier("markdown1.md")
# afficher_fichier("markdown2.md")
```

Activité 3

Activité 3 markdown_3.py

```
##############################
# Visualiseur de texte - Markdown
##############################

##############################
# Activité 3 - Justification
##############################


################################################
## Question 1 ##
from random import randint

# longueurs = [randint(5,15) for i in range(103)]
# longueurs = [14, 3, 16, 9, 2, 11, 13, 5, 4, 19, 16, 6, 17, 16, 15, 5, 14, 12, 17, 7]
longueurs = [8, 11, 9, 14, 8, 8, 15, 10, 14, 11, 15, 15, 5, 12, 9, 9, 15, 10, 14, 5, 12, 8,
    ↪ 8, 13, 10, 11, 8, 13, 7, 5, 6, 11, 7, 7, 13, 6, 6, 9, 8, 12, 5, 8, 7, 6, 6, 15, 13,
    ↪ 11, 7, 12]

longueur_ligne = 100
longueur_espace = 1

def coupures_simples(long):
    """ Calcule les coupures des mots pour un alignement à gauche (sans espaces)
    Entrée : une suite de longueurs (une liste d'entiers)
    Sortie : la liste des indices où effectuer la coupure """

    coupures = [0]

    i = 1
    while i < len(long):
        somme = long[i-1]

        while (i < len(long)) and (somme <= longueur_ligne):
            somme +=  long[i]
            i += 1

        if somme > longueur_ligne:
            coupures +=  [i-1]

    coupures += [len(long)]

    return coupures

################################################
def afficher_coupures_simples():
    """ Test : affiche les coupures simples """

    print("\n--- Coupures sans espaces ---")
    print("Longueurs des mots :",longueurs)

    coupures = coupures_simples(longueurs)
    print("coupures",coupures)

    for i in range(len(coupures)-1):
        ligne = longueurs[coupures[i]:coupures[i+1]]
        somme = sum(ligne)
        print("\nLigne",i,":",ligne,"\nIndices",coupures[i],"à",coupures[i+1]-1,"= longueur[
    ↪ ",coupures[i],":",coupures[i+1],"]","\nSomme =",somme,"Reste =",longueur_ligne-somme
```

```
    ↪ ,)

    return

# Test
afficher_coupures_simples()


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

def coupures_espaces(long):
    """ Calcule les coupures des mots pour un alignement à gauche (avec espaces)
    Entrée : une suite de longueurs (une liste d'entiers)
    Sortie : la liste des indices où effectuer la coupure """

    coupures = [0]

    i = 1

    while i < len(long):
        somme = long[i-1]

        while (i < len(long)) and (somme <= longueur_ligne):
            somme +=  longueur_espace + long[i]
            i += 1

        if somme > longueur_ligne:
            coupures +=  [i-1]

    coupures += [len(long)]

    return coupures


################################################
def afficher_coupures_espaces():
    """ Test : affiche les coupures avec espaces """

    print("\n--- Coupures avec espaces ---")
    print("Longueurs des mots :",longueurs)

    coupures = coupures_espaces(longueurs)
    print("Coupures :",coupures)

    for i in range(len(coupures)-1):
        ligne = longueurs[coupures[i]:coupures[i+1]]
        nb_espaces = len(ligne)-1
        somme =  sum(ligne) + nb_espaces*longueur_espace
        print("\nLigne",i,":",ligne,"\nIndices",coupures[i],"à",coupures[i+1]-1,"= longueur[
    ↪ ",coupures[i],":",coupures[i+1],"]","\nSomme avec espaces =",somme,"Reste =",
    ↪ longueur_ligne-somme,)

    return

# Test
afficher_coupures_espaces()


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

def calcul_longueur_espaces(long,coupures):
    """ Calcule les longueurs des espaces pour justification
    Entrée : une suite de longueurs avec les coupures
    Sortie : la longueurs des espaces pour chaque ligne """

    longueur_espaces_ligne = []
```

```
    for i in range(len(coupures)-2):

        ligne = long[coupures[i]:coupures[i+1] ]
        nb_espaces = len(ligne)-1
        somme =  sum(ligne) + nb_espaces*longueur_espace
        restant = longueur_ligne - somme

        if nb_espaces > 0:
            nouvel_espace = longueur_espace + restant / nb_espaces
        else:
            nouvel_espace = longueur_espace

        longueur_espaces_ligne += [nouvel_espace]

    # Dernière ligne du paragraphe pas justifiée
    longueur_espaces_ligne += [longueur_espace]

    return longueur_espaces_ligne


################################################
def afficher_calcul_longueur_espaces():
    """ Test : affiche les longueurs des espaces """

    print("\n--- Coupures avec espaces et justification ---")
    print("Longueurs des mots :",longueurs)

    coupures = coupures_espaces(longueurs)
    print("Coupures :",coupures)

    longueur_espaces_ligne = calcul_longueur_espaces(longueurs,coupures)
    print("Longueur des espaces de chaque ligne :",[float("{0:0.2f}".format(l)) for l in
    ↪ longueur_espaces_ligne])

    for i in range(len(coupures)-1):
        ligne = longueurs[coupures[i]:coupures[i+1]]
        nb_espaces = len(ligne) - 1
        somme =  sum(ligne) + nb_espaces*longueur_espaces_ligne[i]

        print("\nLigne",i,":",ligne,"\nIndices",coupures[i],"à",coupures[i+1]-1,"= longueur[
    ↪ ",coupures[i],":",coupures[i+1],"]","\nSomme avec espaces =",somme,"Reste =",
    ↪ longueur_ligne-somme,)
        print("Longueur espace de cette ligne",longueur_espaces_ligne[i])

    return

# Test
afficher_calcul_longueur_espaces()
```

18. L-système

Activités `lsysteme.py`

```
################################
# L-système
################################

from turtle import *

################################
# Activité 1 - Tracer un L-système
```

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

def trace_lsysteme(mot,angle=90,echelle=1):
    speed("fastest")
    width(2)
    color('blue')
    up()
    goto(-150,-150)
    down()
    for c in mot:
        if c == "A" or c == "B":
            forward(100*echelle)
        if c == "g":
            left(angle)
        if c == "d":
            right(angle)
    exitonclick()
    return
## Test ##
# trace_lsysteme("AgAdAAdAdA")

##############################
# Activité 2 - Une seule règle : le flocon de Koch
##############################

# Un L-système
# un mot de départ
# des règles de remplacement

####################################################
## Question 1 ##
def remplacer_1(mot,lettre,motif):
    nouv_mot = ""
    for l in mot:
        if l == lettre:
            nouv_mot = nouv_mot + motif
        else:
            nouv_mot = nouv_mot + l
    return nouv_mot
## Test ##
print("--- Remplacer une lettre ---")
mot = "AdAAg"
nouv_mot = remplacer_1(mot,"A","Ag")
print(mot)
print(nouv_mot)

####################################################
## Question 2 ##
def iterer_lsysteme_1(depart,regle,k):
    mot = depart
    lettre = regle[0]
    motif = regle[1]
    for i in range(k):
        mot = remplacer_1(mot,lettre,motif)
```

```
    return mot

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

## Flocon de Koch
depart_Koch = "A"
regle_Koch = ("A","AgAdAdAgA")

## Test
for k in range(4):
    print(k,iterer_lsysteme_1(depart_Koch,regle_Koch,k))
    print()

k = 3
mot = iterer_lsysteme_1(depart_Koch,regle_Koch,k)
# trace_lsysteme(mot,echelle=5/3**k)

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

######################
## Autres exemples ##

######################
depart = "AdAdAdA"
regle = ("A"," AdAgAgAAdAdAgA")
k = 3
mot = iterer_lsysteme_1(depart,regle,k)
#trace_lsysteme(mot,echelle=0.05)

######################
depart = "AdAdAdA"
regle = ("A","AgAAdAAdAdAgAgAAdAdAgAgAAgAAdA")
k = 2
mot = iterer_lsysteme_1(depart,regle,k)
#trace_lsysteme(mot,echelle=0.07)

######################
depart = "AdAdAdA"
regle = ("A","AAdAdAdAdAA")
k = 3
mot = iterer_lsysteme_1(depart,regle,k)
# trace_lsysteme(mot,echelle=0.1)

######################
depart = "AdAdAdA"
regle = ("A","AAdAddAdA")
k = 3
mot = iterer_lsysteme_1(depart,regle,k)
# trace_lsysteme(mot,echelle=0.1)

######################
depart = "AdAdAdA"
regle = ("A","AAdAdAdAdAdAgA")
k = 3
mot = iterer_lsysteme_1(depart,regle,k)
# trace_lsysteme(mot,echelle=0.1)

######################
depart = "AdAdAdA"
regle = ("A","AAdAgAdAdAA")
k = 3
mot = iterer_lsysteme_1(depart,regle,k)
```

```
# trace_lsysteme(mot,echelle=0.15)

######################
depart = "AdAdAdA"
regle = ("A","AdAAddAdA")
k = 3
mot = iterer_lsysteme_1(depart,regle,k)
# trace_lsysteme(mot,echelle=0.15)

######################
depart = "AdAdAdA"
regle = ("A","AdAgAdAdA")
k = 4
mot = iterer_lsysteme_1(depart,regle,k)
# trace_lsysteme(mot,echelle=0.15)


##############################
# Activité 3 - Deux règles : Triangle de Sierpinski
##############################

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

def remplacer_2(mot,lettre1,motif1,lettre2,motif2):
    nouv_mot = ""
    for l in mot:
        if l == lettre1:
            nouv_mot = nouv_mot + motif1
        elif l == lettre2:
            nouv_mot = nouv_mot + motif2
        else:
            nouv_mot = nouv_mot + l

    return nouv_mot

## Test ##
print("--- Remplacer deux lettres ---")
mot = "AdBgA"
nouv_mot = remplacer_2(mot,"A","ABg","B","Bd")
print(mot)
print(nouv_mot)

# mot1 = remplacer_1(mot,"A","ABg")
# mot2 = remplacer_1(mot1,"B","Bd")
# print(mot2)

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

def iterer_lsysteme_2(depart,regle1,regle2,k):
    mot = depart
    lettre1 = regle1[0]
    motif1 = regle1[1]
    lettre2 = regle2[0]
    motif2 = regle2[1]

    for i in range(k):
        mot = remplacer_2(mot,lettre1,motif1,lettre2,motif2)

    return mot

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

```
## Triangle de Sierpinski
depart_Sierp = "AdBdB"
regle_Sierp_1 = ("A","AdBgAgBdA")
regle_Sierp_2 = ("B","BB")

## Test
print("--- Sierpinski ---")
for k in range(3):
    print(iterer_lsysteme_2(depart_Sierp,regle_Sierp_1,regle_Sierp_2,k))
    print()

k = 4
mot = iterer_lsysteme_2(depart_Sierp,regle_Sierp_1,regle_Sierp_2,k)
# trace_lsysteme(mot,angle=-120,echelle=5/2**k)

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

######################
## Autres exemples ##

######################
## Courbe du dragon
depart_dragon = "AX"
regle_dragon_1 = ("X","XgYAg")
regle_dragon_2 = ("Y","dAXdY")

k = 9
mot = iterer_lsysteme_2(depart_dragon,regle_dragon_1,regle_dragon_2,k)
# trace_lsysteme(mot,echelle=2/k)

######################
## Variante Sierpinski (angle = 60)
depart = "A"
regle1 = ("A","BdAdB")
regle2 = ("B","AgBgA")
# angle = 60

k = 3
mot = iterer_lsysteme_2(depart,regle1,regle2,k)
# trace_lsysteme(mot,angle=60,echelle=2/k**2)

######################
## Courbe de Gosper
depart = "A"
regle1 = ("A","AgBggBdAddAAdBg")
regle2 = ("B","dAgBBggBgAddAdB")
k = 3
mot = iterer_lsysteme_2(depart,regle1,regle2,k)
# trace_lsysteme(mot,angle=60,echelle=2/k**2)

##############################
# Activité 4 - Tracer un L-système avec pile
##############################

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

def trace_lsysteme_pile(mot,angle=90,echelle=1):

    speed("fastest")
    width(3)
    color('blue')
    up()
```

```
    goto(0,-300)
    down()

    pile = []

    for c in mot:
        if c == "A" or c == "B":
            forward(100*echelle)
        if c == "a":
            up()
            forward(100*echelle)
            down()
        if c == "g":
            left(angle)
        if c == "d":
            right(angle)
        if c == "[":
            pile = pile + [(position(),heading())]
        if c == "]":
            up()
            pos,direc = pile.pop()
            goto(pos)
            setheading(direc)
            down()

    exitonclick()

    return

## Test
# trace_lsysteme_pile("AaAgAA[gAAA][dAA]A",angle=90,echelle=1)
# trace_lsysteme_pile("AgA[gAAA]A[dAA]A",angle=90,echelle=1)

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

# Plante
depart_plante = "gggX"
regle_plante_1 = ("X","A[gX][X]A[gX]dAX")
regle_plante_2 = ("A","AA")

k = 5
mot = iterer_lsysteme_2(depart_plante,regle_plante_1,regle_plante_2,k)
# trace_lsysteme_pile(mot,angle=30,echelle=1/k**(3/2))


######################
## Exemples avec up-down ##
depart = "AdAdAdA"
regle1 = ("A","AgadAAgAgAAgAagAAdagAAdAdAAdAadAAA")
regle2 = ("a","aaaaaa")
k = 2
mot = iterer_lsysteme_2(depart,regle1,regle2,k)
# trace_lsysteme_pile(mot,echelle=0.1)


######################
## Autres exemples de plantes ##

# ######################
# angle = 22.5
# F
depart = "gggA"
```

```
regle = ("A","A[gA]A[dA][A]")
k = 4
mot = iterer_lsysteme_1(depart,regle,k)
# trace_lsysteme_pile(mot,angle=30,echelle=0.2)

# ######################
# angle = 20
depart = "A"
regle = ("A","A[gA]A[dA]A")
k = 4
mot = iterer_lsysteme_1(depart,regle,k)
# trace_lsysteme_pile(mot,angle=30,echelle=0.075)

# ####################
# angle = 22.5
depart = "A"
regle = ("A","AAd[dAgAgA]g[gAdAdA]")
k = 3
mot = iterer_lsysteme_1(depart,regle,k)
# trace_lsysteme_pile(mot,angle=30,echelle=0.2)

# ######################
# angle = 25.7
depart = "X"
regle1 = ("X","A[gX]A[dX]AX")
regle2 = ("A","AA")
k = 5
mot = iterer_lsysteme_2(depart,regle1,regle2,k)
# trace_lsysteme_pile(mot,angle=30,echelle=0.07)

# ######################
# angle = 30
depart = "A"
regle1 = ("A","A[dB][gB]")
regle2 = ("B","A[dB]A[gAdB]")
k = 5
mot = iterer_lsysteme_2(depart,regle1,regle2,k)
# trace_lsysteme_pile(mot,angle=30,echelle=0.25)

####################
# angle = 30
depart = "X"
regle1 = ("X","Ad[[X]gX]gA[gAX]dX")
regle2 = ("A","AA")
k = 4
mot = iterer_lsysteme_2(depart,regle1,regle2,k)
# trace_lsysteme_pile(mot,angle=30,echelle=0.15)


####################
####################
# Courbe de Hilbert
# Pour les illustrations de livre
  # \rule{L -> +RF-LFL-FR+}
  # \rule{R -> -LF+RFR+FL-}}
# angle = 30
depart = "X"
regle1 = ("X","gYAdXAXdAYg")
regle2 = ("Y","dXAgYAYgAXd")
k = 4
```

```
mot = iterer_lsysteme_2(depart,regle1,regle2,k)
trace_lsysteme_pile(mot,angle=90,echelle=0.15)
```

19. Images dynamiques

Activités images.py

```
##############################
# Images dynamiques
##############################
import os   # pour les fichiers images

##############################
# Activité 1 - Photomaton
##############################


##############################
## Depuis autres fiches ##
def afficher_tableau(tableau):
    n = len(tableau)
    m = len(tableau[0])
    for i in range(n):
        for j in range(m):
            print('{:>3d}'.format(tableau[i][j])," ", end="")
        print()
    return

##############################
## Question 1 ##
def transformation(i,j,n):
    if i%2 == 0 and j%2 == 0:
        ii = i//2
        jj = j//2
    if i%2 == 0 and j%2 == 1:
        ii = i//2
        jj = (n+j)//2
    if i%2 == 1 and j%2 == 0:
        ii = (n+i)//2
        jj = j//2
    if i%2 == 1 and j%2 == 1:
        ii = (n+i)//2
        jj = (n+j)//2
    return ii,jj
## Test ##
print("--- Transformation du photomaton ---")
print(transformation(1,1,6))

##############################
## Question 2 ##
def photomaton(tableau):
```

```
    n = len(tableau)
    nouv_tableau = [[0 for j in range(n)] for i in range(n)]

    for i in range(n):
        for j in range(n):
            ii, jj = transformation(i,j,n)
            nouv_tableau[ii][jj] = tableau[i][j]

    return nouv_tableau

## Test ##
print("--- Transformation du photomaton ---")
tableau = [ [1,2,3,4], [5,6,7,8],[9,10,11,12], [13,14,15,16] ]
tableau_transforme = photomaton(tableau)
afficher_tableau(tableau)
print("---")
afficher_tableau(tableau_transforme)

##############################
## Question 3 ##
def photomaton_iterer(tableau,k):
    n = len(tableau)
    tab = [[tableau[i][j] for j in range(n)] for i in range(n)]

    for i in range(k):
        tab = photomaton(tab)

    return tab

## Test ##
print("--- Itération de la transformation du photomaton ---")
tableau = [ [1,2,3,4], [5,6,7,8],[9,10,11,12], [13,14,15,16] ]
afficher_tableau(tableau)
for k in range(1,10):
    tableau_itere = photomaton_iterer(tableau,k)
    # Pas très malin, car repart du début à chaque fois
    print("--- k =",k,"---")
    afficher_tableau(tableau_itere)


##############################
# Activité 2 - Conversion tableau/image
##############################

##############################
## Question 1 ##
def tableau_vers_image(tableau,nom_image):

    # Création d'un fichier en écriture
    nom_fichier = "output/" + nom_image + ".pgm"
    fic = open(nom_fichier,"w")

    # Entete
    fic.write("P2\n")  # Image en niveaux de gris

    nb_lig = len(tableau)
    nb_col = len(tableau[0])

    fic.write(str(nb_col) + " " + str(nb_lig) + "\n")
    niveaux = 255
    fic.write(str(niveaux) + "\n")

    for i in range(nb_lig):
        ligne = ""
```

```
        for j in range(nb_col):
            coul = tableau[i][j]
            ligne = ligne + str(coul) + " "
        ligne = ligne + "\n"

        # Ecriture dans le fichier
        fic.write(ligne)

    # Fermeture du fichier
    fic.close()

    return

## Test ##
print("--- Tableau vers image ---")
tableau =  [[128, 192, 128, 192, 128], [224, 0, 228, 0, 224], [228, 228, 228, 228, 228],
    ↪ [224, 64, 64, 64, 224], [192, 192, 192, 192, 192]]
tableau_vers_image(tableau,"test")

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

def image_vers_tableau(nom_image):

    # Création d'un fichier en écriture
    nom_fichier = "input/" + nom_image + ".pgm"
    fic = open(nom_fichier,"r")

    i = 0    # Numéro de ligne
    for ligne in fic:
        if i == 1:     # Garder les 2 premières lignes
            liste_ligne = ligne.split()
            nb_col = int(liste_ligne[0])
            nb_lig = int(liste_ligne[1])

            tableau = [[ 0 for j in range(nb_col)] for i in range(nb_lig)]
        elif i > 2:
            liste = ligne.split()
            for j in range(nb_col):
                tableau[i-3][j] = int(liste[j])

        i = i + 1

    # Fermeture du fichier
    fic.close()

    return tableau

print("--- Image vers tableau ---")

test_tableau = image_vers_tableau("test")
print(test_tableau)
afficher_tableau(test_tableau)

##############################
## Depuis la fiche "Fichiers" ##
## Permet d'avoir un exemple de fichier

def ecrire_fichier_image_gris():

    # Création d'un fichier en écriture
    nom_fichier = "input/image_gris.pgm"
    fic = open(nom_fichier,"w")

    # Entete
    fic.write("P2\n")  # Image en niveaux de gris
```

```
    nb_col = 256
    nb_lig = 256
    fic.write(str(nb_col) + " " + str(nb_lig) + "\n")
    niveaux = 255
    fic.write(str(niveaux) + "\n")

    for i in range(nb_lig):
        ligne = ""
        for j in range(nb_col):
            coul = (i**2 + j**2) % 256   # un niveau de gris en fonction de i et j
            ligne = ligne + str(coul) + " "
        ligne = ligne + "\n"

        # Ecriture dans le fichier
        fic.write(ligne)

    # Fermeture du fichier
    fic.close()

    return

# Test

print("--- Fichier 'image.pgm' ---")
# ecrire_fichier_image_gris()

##############################
# Activité 1bis - Photomaton
##############################

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

def photomaton_images(nom_image,kmax):
    tableau = image_vers_tableau(nom_image)
    tableau_vers_image(tableau,nom_image+"_photo_"+str(0)) # image initiale

    n = len(tableau)
    tab = [[tableau[i][j] for j in range(n)] for i in range(n)]

    for k in range(1,kmax+1):
        tab = photomaton(tab)
        tableau_vers_image(tab,nom_image+"_photo_"+str(k))

    return

## Test ##
# photomaton_images("image_gris",8)
# photomaton_images("pi_gimp_new",8)
# photomaton_images("chat_gimp_new",8)


##############################
# Activité 3 - Transfomation du boulanger
##############################

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

def boulanger_etirer(tableau):
    n = len(tableau)
    nouv_tableau = [[0 for j in range(2*n)] for i in range(n//2)]

    for i in range(n//2):
        for j in range(2*n):
            if j%2 == 0:
                nouv_tableau[i][j] = tableau[2*i][j//2]
```

```
            else:
                nouv_tableau[i][j] = tableau[2*i+1][j//2]

    return nouv_tableau

print("--- Boulanger : étirer tableau ---")
tableau = [ [1,2,3,4], [5,6,7,8],[9,10,11,12], [13,14,15,16] ]
tableau_etire = boulanger_etirer(tableau)
afficher_tableau(tableau)
print("---")
afficher_tableau(tableau_etire)

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

def boulanger_replier(tableau):
    n = 2*len(tableau)

    nouv_tableau = [[0 for j in range(n)] for i in range(n)]

    # partie haute
    for i in range(n//2):
        for j in range(n):
            nouv_tableau[i][j] = tableau[i][j]

     # partie basse
    for i in range(n//2,n):
        for j in range(n):
            nouv_tableau[i][j] = tableau[n//2 - i - 1][2*n-1-j]

    # for i in range(n//2):
    #     for j in range(n):
    #         nouv_tableau[n-i-1][j] = tableau[i][2*n-1-j]

    return nouv_tableau

print("--- Boulanger : replier tableau ---")
tableau_replie = boulanger_replier(tableau_etire)
afficher_tableau(tableau_etire)
print("---")
afficher_tableau(tableau_replie)

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

def boulanger_iterer(tableau,k):
    n = len(tableau)
    tab = [[tableau[i][j] for j in range(n)] for i in range(n)]

    for i in range(k):
        tabb = boulanger_etirer(tab)
        tab = boulanger_replier(tabb)

    return tab

print("--- Boulanger : itérer tranformation tableau ---")
tableau = [ [1,2,3,4], [5,6,7,8],[9,10,11,12], [13,14,15,16] ]
afficher_tableau(tableau)
for k in range(1,10):
    tableau_itere = boulanger_iterer(tableau,k)
    print("--- k =",k,"---")
    afficher_tableau(tableau_itere)
```

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

def boulanger_images(nom_image,kmax):

    tableau = image_vers_tableau(nom_image)
    tableau_vers_image(tableau,nom_image+"_boul_"+str(0)) # image initiale

    n = len(tableau)
    tab = [[tableau[i][j] for j in range(n)] for i in range(n)]

    for k in range(1,kmax+1):
        tabb = boulanger_etirer(tab)
        tab = boulanger_replier(tabb)
        tableau_vers_image(tab,nom_image+"_boul_"+str(k))

    return

## Test ##
# boulanger_images("image_gris",17)
# boulanger_images("pi_gimp_new",17)
# boulanger_images("chat_gimp_new",17)
# boulanger_images("reveil_gimp_new",17)
# boulanger_images("surf_gimp_new",15)
```

20. Jeu de la vie

Activité 1

Activité 1 vie_1.py

```
##############################
# Jeu de la vie
##############################

##############################
# Activité 1 - Tableau
##############################

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

n, p = 5, 8;
tableau = [[0 for j in range(p)] for i in range(n)]

# Clignotant
tableau[2][2] = 1
tableau[2][3] = 1
tableau[2][4] = 1

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

def voir_tableau(tab):
    """ Affiche un tableau à l'écran
    Entrée : un tableau à deux dimension
    Sortie : rien (affichage à l'écran) """

    for i in range(n):
        for j in range(p):
```

```
            print(tab[i][j], end="")
        print()

    return

# Test
voir_tableau(tableau)
```

Activité 2

Activité 2 vie_2.py

```
##############################
# Jeu de la vie
##############################

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

n, p = 5, 8;
tableau = [[0 for j in range(p)] for i in range(n)]

# Clignotant
tableau[2][2] = 1
tableau[2][3] = 1
tableau[2][4] = 1

##############################
# Activité 2 - Affichage graphique
##############################

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

from tkinter import *

# Fenêtre tkinter
root = Tk()

canvas = Canvas(root, width=800, height=600, background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

# Echelle
echelle = 100

def afficher_lignes():
    """ Affiche la grille à l'écran """

    for i in range(n+1):
        canvas.create_line(0,i*echelle,p*echelle,i*echelle)

    for j in range(p+1):
        canvas.create_line(j*echelle,0,j*echelle,n*echelle)

    for i in range(n):
        canvas.create_text(echelle//3,i*echelle+echelle//2,text=str(i))

    for j in range(p):
        canvas.create_text(j*echelle+echelle//2,echelle//3,text=str(j))

    return
```

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

def afficher_tableau(tab):
    """ Affiche un tableau à l'écran
    Entrée : un tableau à deux dimension
    Sortie : rien (affichage à l'écran) """

    for i in range(n):
        for j in range(p):
            if tab[i][j] != 0:
                canvas.create_rectangle(j*echelle,i*echelle,(j+1)*echelle,(i+1)*echelle,fill
    ↪ ="red")

    return

# Boutons
def action_bouton_afficher():
    canvas.delete("all")
    afficher_lignes()
    afficher_tableau(tableau)
    return

bouton_quitter = Button(root,text="Quitter", width=8, command=root.quit)
bouton_quitter.pack(side=BOTTOM, padx=5, pady=20)

bouton_afficher = Button(root,text="Afficher", width=30, command=action_bouton_afficher)
bouton_afficher.pack(side=BOTTOM, padx=5, pady=20)

# Test
afficher_lignes()
# afficher_tableau(tableau)
root.mainloop()
```

Activité 3

Activité 3 — vie_3.py

```
##############################
# Jeu de la vie
##############################

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

from vie_1 import *

n, p = 5, 8;
tableau = [[0 for j in range(p)] for i in range(n)]

# Clignotant
tableau[2][2] = 1
tableau[2][3] = 1
tableau[2][4] = 1

##############################
# Activité 3 - Evolution
```

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

################################################
## Question 1 ##
def nombre_voisins(i,j,tab):
    """ Calcule le nb de voisins de la cellule(i,j)
    Entrée : une cellule dans un tableau à deux dimension
    Sortie : le nb de cellules voisines """
    nb = 0
    # Voisin en haut à gauche
    if (i>0) and (j>0) and (tab[i-1][j-1] != 0):
        nb += 1
    # Voisin juste au-dessus
    if (i>0) and (tab[i-1][j] != 0):
        nb += 1
    # Voisin en haut à droite
    if (i>0) and (j<p-1) and (tab[i-1][j+1] != 0):
        nb += 1
    # Voisin juste à gauche
    if (j>0) and (tab[i][j-1] != 0):
        nb += 1
    # Voisin juste à droite
    if (j<p-1) and (tab[i][j+1] != 0):
        nb += 1
   # Voisin en bas à gauche
    if (i<n-1) and (j>0) and (tab[i+1][j-1] != 0):
        nb += 1
    # Voisin juste en-dessous
    if (i<n-1) and (tab[i+1][j] != 0):
        nb += 1
    # Voisin en bas à droite
    if (i<n-1) and (j<p-1) and (tab[i+1][j+1] != 0):
        nb += 1
    return nb
# Test
print("--- Nombre de voisins ---")
print(nombre_voisins(1,1,tableau))
print(nombre_voisins(2,1,tableau))
print(nombre_voisins(3,1,tableau))
print(nombre_voisins(2,0,tableau))
print(nombre_voisins(2,2,tableau))
print(nombre_voisins(3,3,tableau))
################################################
def voir_voisins(tab):
    """ Affiche le nb de voisins à l'écran
    Entrée : un tableau à deux dimension
    Sortie : rien (affichage à l'écran) """
    for i in range(n):
        for j in range(p):
            print(nombre_voisins(i,j,tab), end='')
        print()
    return

# Test
print("--- Position de départ ---")
```

```
voir_tableau(tableau)
print("--- Nombre de voisins (tableau) ---")
voir_voisins(tableau)

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

def evolution(tab):
    """ Calcule l'évolution en un jour
    Entrée : un tableau à deux dimension
    Sortie : un tableau à deux dimension """

    nouv_tab = [[0 for j in range(p)] for i in range(n)]

    for j in range(p):
        for i in range(n):
            # Cellule vivante ou pas ?
            if tab[i][j] != 0:
                cellule_vivante = True
            else:
                cellule_vivante = False

            # Nombres de voisins
            nb_voisins = nombre_voisins(i,j,tab)

            # Règle du jeu de la vie
            if cellule_vivante == True and (nb_voisins == 2 or nb_voisins == 3):
                nouv_tab[i][j] = 1
            if cellule_vivante == False and nb_voisins == 3:
                nouv_tab[i][j] = 1

    return nouv_tab

# Test
print("--- Position de départ ---")
voir_tableau(tableau)
print("--- Nombre de voisins ---")
voir_voisins(tableau)
print("--- Après évolution ---")
tableau = evolution(tableau)
voir_tableau(tableau)
```

Activité 4

Activité 4 vie_4.py

```
################################
# Jeu de la vie
################################

################################
# Rappels - Activité précédentes
################################

from tkinter import *

# Fenêtre tkinter
root = Tk()

canvas = Canvas(root, width=800, height=600, background="white")
```

```
canvas.pack(side=LEFT, padx=5, pady=5)

# Par défaut : rien
n, p = 25, 25
echelle = 40
# n, p = 8, 5
# echelle = 80

tableau = [[0 for j in range(p)] for i in range(n)]

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

def nombre_voisins(i,j,tab):
    """ Calcule le nb de voisins de la cellule(i,j)
    Entrée : une cellule dans un tableau à deux dimension
    Sortie : le nb de cellules voisines """

    nb = 0
    # Voisin en haut à gauche
    if (i>0) and (j>0) and (tab[i-1][j-1] != 0):
        nb += 1
    # Voisin juste au-dessus
    if (i>0) and (tab[i-1][j] != 0):
        nb += 1
    # Voisin en haut à droite
    if (i>0) and (j<p-1) and (tab[i-1][j+1] != 0):
        nb += 1
    # Voisin juste à gauche
    if (j>0) and (tab[i][j-1] != 0):
        nb += 1
    # Voisin juste à droite
    if (j<p-1) and (tab[i][j+1] != 0):
        nb += 1
   # Voisin en bas à gauche
    if (i<n-1) and (j>0) and (tab[i+1][j-1] != 0):
        nb += 1
    # Voisin juste en-dessous
    if (i<n-1) and (tab[i+1][j] != 0):
        nb += 1
    # Voisin en bas à droite
    if (i<n-1) and (j<p-1) and (tab[i+1][j+1] != 0):
        nb += 1

    return nb

def evolution(tab):
    """ Calcule l'évolution en un jour
    Entrée : un tableau à deux dimension
    Sortie : un tableau à deux dimension """

    nouv_tab = [[0 for j in range(p)] for i in range(n)]

    for j in range(p):
        for i in range(n):
            # Cellule vivante ou pas ?
            if tab[i][j] != 0:
                cellule_vivante = True
            else:
                cellule_vivante = False

            # Nombres de voisins
            nb_voisins = nombre_voisins(i,j,tab)

            # Règle du jeu de la vie
```

```
            if cellule_vivante == True and (nb_voisins == 2 or nb_voisins == 3):
                nouv_tab[i][j] = 1
            if cellule_vivante == False and nb_voisins == 3:
                nouv_tab[i][j] = 1

    return nouv_tab

def afficher_lignes():
    """ Affiche la grille à l'écran """

    for i in range(n+1):
        canvas.create_line(0,i*echelle,p*echelle,i*echelle)

    for j in range(p+1):
        canvas.create_line(j*echelle,0,j*echelle,n*echelle)

    for i in range(n):
        canvas.create_text(echelle//3,i*echelle+echelle//2,text=str(i))

    for j in range(p):
        canvas.create_text(j*echelle+echelle//2,echelle//3,text=str(j))

    return

def afficher_tableau(tab):
    """ Affiche un tableau à l'écran
    Entrée : un tableau à deux dimension
    Sortie : rien (affichage à l'écran) """

    for i in range(n):
        for j in range(p):
            if tab[i][j] != 0:
                canvas.create_rectangle(j*echelle,i*echelle,(j+1)*echelle,(i+1)*echelle,fill
    ↪ ="red")

    return

##############################
# Activité 4 - Jeu de la vie en entier
##############################

#####################################################
## Question 0 ##

# Clignotant
def clignotant():
    """ Définition du clignotant """
    global tableau
    tableau = [[0 for j in range(p)] for i in range(n)]
    tableau[4][7] = 1
    tableau[4][8] = 1
    tableau[4][9] = 1
    canvas.delete("all")
    afficher_lignes()
    afficher_tableau(tableau)
    return

# Vaisseau
def vaisseau():
    """ Définition du vasseau spatial """
    global tableau
    tableau = [[0 for j in range(p)] for i in range(n)]
    tableau[3][4] = 1
    tableau[3][5] = 1
```

```
    tableau[3][6] = 1
    tableau[2][6] = 1
    tableau[1][5] = 1
    canvas.delete("all")
    afficher_lignes()
    afficher_tableau(tableau)
    return

# Pentadecathlon
def pentadecathlon():
    """ Définition du pentadecathlon """
    global tableau
    tableau = [[0 for j in range(p)] for i in range(n)]
    tableau[6][4] = 1
    tableau[6][5] = 1
    tableau[6][7] = 1
    tableau[6][8] = 1
    tableau[6][9] = 1
    tableau[6][10] = 1
    tableau[6][12] = 1
    tableau[6][13] = 1
    tableau[5][6] = 1
    tableau[7][6] = 1
    tableau[5][11] = 1
    tableau[7][11] = 1
    canvas.delete("all")
    afficher_lignes()
    afficher_tableau(tableau)
    return

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

# Boutons

def action_bouton_evolution():
    global tableau
    tableau = evolution(tableau)
    canvas.delete("all")
    afficher_lignes()
    afficher_tableau(tableau)
    return

bouton_quitter = Button(root,text="Quitter", width=8, command=root.quit)
bouton_quitter.pack(side=BOTTOM, padx=5, pady=20)

bouton_afficher = Button(root,text="Évoluer", width=20, command=action_bouton_evolution)
bouton_afficher.pack(side=BOTTOM, padx=5, pady=20)

bouton_clignotant = Button(root,text="Clignotant", width=20, command=clignotant)
bouton_clignotant.pack(side=TOP, padx=5, pady=5)

bouton_vaisseau = Button(root,text="Vaisseau", width=20, command=vaisseau)
bouton_vaisseau.pack(side=TOP, padx=5, pady=5)

bouton_pentadecathlon = Button(root,text="Pentadecathlon", width=20, command=pentadecathlon)
bouton_pentadecathlon.pack(side=TOP, padx=5, pady=5)

# root.mainloop()

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

```
def allumer_eteindre(i,j):
    """ Commute une cellule """
    global tableau
    if tableau[i][j] == 0:
        tableau[i][j] = 1
    else:
        tableau[i][j] = 0
    return

def xy_vers_ij(x,y):
    """ Coordonnées (x,y) vers coordonnées (i,j) """
    i = y // echelle
    j = x // echelle
    return i, j

def action_clic_souris(event):
    canvas.focus_set()
    # print("Clic à", event.x, event.y)
    x = event.x
    y = event.y
    allumer_eteindre(*xy_vers_ij(x,y))
    canvas.delete("all")
    afficher_lignes()
    afficher_tableau(tableau)
    return

# Liaison clic de souris/action
canvas.bind("<Button-1>",action_clic_souris)

afficher_lignes()
afficher_tableau(tableau)
root.mainloop()
```

21. Graphes et combinatoire de Ramsey

Activité 1

Activité 1 ramsey_1.py

```
##############################
# Graphes et combinatoire de Ramsey
##############################

##############################
# Activité 1 - Définition et amis/étrangers
##############################

## Question 1 ##

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

# Exemple 1
n = 3
exemple_graphe_1 = [[0 for j in range(n)] for i in range(n)]
exemple_graphe_1[0][1] = 1; exemple_graphe_1[1][0] = 1
exemple_graphe_1[0][2] = 1; exemple_graphe_1[2][0] = 1
```

```
# Exemple 2
n = 4

exemple_graphe_2 = [[0 for j in range(n)] for i in range(n)]

exemple_graphe_2[0][2] = 1; exemple_graphe_2[2][0] = 1
exemple_graphe_2[0][3] = 1; exemple_graphe_2[3][0] = 1
exemple_graphe_2[1][2] = 1; exemple_graphe_2[2][1] = 1

# Exemple 3
n = 5

exemple_graphe_3 = [[0 for j in range(n)] for i in range(n)]

exemple_graphe_3[0][2] = 1; exemple_graphe_3[2][0] = 1
exemple_graphe_3[0][3] = 1; exemple_graphe_3[3][0] = 1
exemple_graphe_3[1][2] = 1; exemple_graphe_3[2][1] = 1
exemple_graphe_3[1][4] = 1; exemple_graphe_3[4][1] = 1
exemple_graphe_3[3][4] = 1; exemple_graphe_3[4][3] = 1

# Exemple 4
n = 6

exemple_graphe_4 = [[0 for j in range(n)] for i in range(n)]

exemple_graphe_4[3][2] = 1; exemple_graphe_4[2][3] = 1;
exemple_graphe_4[1][2] = 1; exemple_graphe_4[2][1] = 1
exemple_graphe_4[3][4] = 1; exemple_graphe_4[4][3] = 1
exemple_graphe_4[4][1] = 1; exemple_graphe_4[1][4] = 1
exemple_graphe_4[0][2] = 1; exemple_graphe_4[2][0] = 1
exemple_graphe_4[5][0] = 1; exemple_graphe_4[0][5] = 1
exemple_graphe_4[5][1] = 1; exemple_graphe_4[1][5] = 1
exemple_graphe_4[0][3] = 1; exemple_graphe_4[3][0] = 1

# Exemple cours
n = 4

exemple_graphe_cours_1 = [[0 for j in range(n)] for i in range(n)]

exemple_graphe_cours_1[0][2] = 1; exemple_graphe_cours_1[2][0] = 1
exemple_graphe_cours_1[1][3] = 1; exemple_graphe_cours_1[3][1] = 1

## Question 2 ##

#####################################################
def voir_graphe(graphe):
    """
    Affiche un graphe à l'écran
    Entrée : un graphe comme tableau à deux dimension
    Sortie : rien (affichage à l'écran)
    """

    n = len(graphe)

    for j in range(n):
        for i in range(n):
            print(graphe[i][j], end="")
        print()

    return

# Test
if __name__ == '__main__':
    print("--- Matrice du graphe ---")
    print("--- Exemple 1 ---")
```

```
    voir_graphe(exemple_graphe_1)
    print("--- Exemple 2 ---")
    voir_graphe(exemple_graphe_2)
    print("--- Exemple 3 ---")
    voir_graphe(exemple_graphe_3)
    print("--- Exemple 4 ---")
    voir_graphe(exemple_graphe_4)

    print("--- Cours 1 ---")
    voir_graphe(exemple_graphe_cours_1)

##################################################
##################################################
# Test si un graphe est contient
# 3 amis/étranger dont les positions sont données

##################################################
def contient_3_amis_fixes(graphe,i,j,k):
    """Cherche si sommets i, j, k sont tous reliés entre eux comme amis"""
    if graphe[i][j] == 1 and graphe[i][k] == 1 and graphe[j][k] == 1:
        return True
    else:
        return False

##################################################
def contient_3_etrangers_fixes(graphe,i,j,k):
    """Cherche si sommets i, j, k sont tous reliés entre eux comme étrangers"""
    if graphe[i][j] == 0 and graphe[i][k] == 0 and graphe[j][k] == 0:
        return True
    else:
        return False

# Test
if __name__ == '__main__':
    print("--- Sous-graphe fixé de 0 et de 1 ---")
    print(contient_3_amis_fixes(exemple_graphe_4,1,3,4))
    print(contient_3_etrangers_fixes(exemple_graphe_4,1,3,4))
```

Activité 2

Activité 2 ramsey_2.py

```
################################
# Graphes et combinatoire de Ramsey
################################


################################
# Activité 2 - Affichage graphique
################################

from tkinter import *
from math import *
from tkinter.font import Font

from ramsey_1 import *  # Pour les exemples

# Fenêtre tkinter
```

```
root = Tk()

canvas = Canvas(root, width=800, height=500, background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

# Echelle
echelle = 200

#####################################################
# Un graphe

## Question 1 ##

# Version basique (recalcule plein de fois la même chose)
#####################################################
def afficher_graphe_basic(graphe):
    """
    Affiche un graphe à l'écran
    Entrée : un graphe
    Sortie : rien (affichage à l'écran)
    """
    n = len(graphe)

    # Arêtes
    for j in range(n):
        for i in range(n):
            xi = 2*echelle + cos(2*i*pi/n)*echelle
            yi = 1.5*echelle + sin(2*i*pi/n)*echelle
            xj = 2*echelle + cos(2*j*pi/n)*echelle
            yj = 1.5*echelle + sin(2*j*pi/n)*echelle
            if graphe[i][j] == 0:
                canvas.create_line(xi,yi,xj,yj,width=4,fill="red")
            if graphe[i][j] == 1:
                canvas.create_line(xi,yi,xj,yj,width=4,fill="green")

    # Sommets
    for i in range(n):
        x = 2*echelle + cos(2*i*pi/n)*echelle
        y = echelle + sin(2*i*pi/n)*echelle
        canvas.create_oval(x-5,y-5,x+5,y+5,fill="black")

    return

# Version optimale
#####################################################
def afficher_graphe(graphe):
    """
    Affiche un graphe à l'écran
    Entrée : un graphe
    Sortie : rien (affichage à l'écran)
    """
    n = len(graphe)  # Nombre de sommets

    # Liste des coordonnées (x,y) des sommets
    coord = [(2*echelle + cos(2*i*pi/n)*echelle,1.2*echelle + sin(2*i*pi/n)*echelle) for i
    ↪ in range(n)]

    # Arêtes
    for j in range(n):
        for i in range(j+1,n):  # i>j
            if graphe[i][j] == 0:
                canvas.create_line(coord[i],coord[j],width=4,fill="red",dash=(6, 2))
            if graphe[i][j] == 1:
```

```
                canvas.create_line(coord[i],coord[j],width=4,fill="green")

    mafonte = Font(family="Courier", weight="bold",size=18)
     # Sommets
    for i in range(n):
        x,y = coord[i]
        canvas.create_oval(x-15,y-15,x+15,y+15,fill="black")
        canvas.create_text(x,y,text=str(i),font=mafonte,fill="white")

    # Numéro

    return

# Lancement de la fenêtre
if __name__ == '__main__':
    bouton_quitter = Button(root,text="Quitter", width=8, command=root.quit)
    bouton_quitter.pack(side=BOTTOM, padx=5, pady=20)

    # Test d'un exemple
    afficher_graphe(exemple_graphe_2)
    root.mainloop()
```

Activité 3

Activité 3 ramsey_3.py

```
##############################
# Graphes et combinatoire de Ramsey
##############################

##############################
# Activité 3 - Binaire
##############################

## Echanger n <-> p pour être cohérent avec fiche binaire

def decimal_vers_binaire(p,n):
    chaine_b = bin(p)  # Conversion en une chaîne écriture binaire
    chaine_bb = chaine_b[2:]  # On enlève le préfixe
    # On transforme la chaine en une liste **d'entiers** 0 ou 1
    liste_binaire = []
    for b in chaine_bb:
        liste_binaire = liste_binaire + [int(b)]

    # On rajoute des zéros au début si besoin
    nb_zeros = n - len(liste_binaire)
    for i in range(nb_zeros):
        liste_binaire = [0] + liste_binaire

    return liste_binaire

# Version plus courte qui utilise "format()"
def decimal_vers_binaire_bis(p,n):
    modele = '{:0'+str(n)+'b}'
    chaine_binaire = modele.format(p)
    liste_binaire = [int(b) for b in list(chaine_binaire)]
    return liste_binaire

# Test
```

```
if __name__ == '__main__':
    n = 8
    p = 37
    print(decimal_vers_binaire(p,n))
    print(decimal_vers_binaire_bis(p,n))
```

Activité 4

Activité 4 ramsey_4.py

```
##############################
# Graphes et combinatoire de Ramsey
##############################

##############################
# Activité 4 - Sous-ensembles
##############################
from ramsey_3 import decimal_vers_binaire

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


## Question 1 ##

# Génère tous les sous-ensembles

#####################################################
def sous_ensembles(n):
    """Trouve tous les sous-ensembles de l'ensemble à n éléments [0,1,2,...n-1]"""

    tous_sous_ens = []

    for p in range(2**n):

        # Conversion binaire
        liste_binaire = decimal_vers_binaire(p,n)
        #print(liste_binaire)

        sous_ens = []
        for j in range(n):
            # if liste_binaire[n-j-1] == 1:
            if liste_binaire[j] == 1:
                sous_ens = sous_ens + [j]

        tous_sous_ens = tous_sous_ens + [sous_ens]

    return tous_sous_ens

# Test
if __name__ == '__main__':
    print("--- Sous-ensembles ---")
    n = 3
    SS_ENS = sous_ensembles(n)
    print("Pour n = ",n)
    print("Nombre de sous-ensembles = ",len(SS_ENS))
    print(SS_ENS)

## Question 2 ##
```

```
##################################################
def sous_ensembles_fixe(n,k):
    tous_sous_ens_fixe = []
    for sous_ens in sous_ensembles(n):
        if len(sous_ens) == k:
            tous_sous_ens_fixe = tous_sous_ens_fixe + [sous_ens]
    return tous_sous_ens_fixe

# Test (suite)
if __name__ == '__main__':
    print("--- Sous-ensembles à 3 éléments ---")

    n = 6
    k = 3
    SS_ENS_3 = sous_ensembles_fixe(n,k)
    print("Pour n = ",n," k = ",k)
    print("Nombre de sous-ensembles à 3 éléments = ",len(SS_ENS_3))
    print(SS_ENS_3)
```

Activité 5

Activité 5 ramsey_5.py

```
##############################
# Graphes et combinatoire de Ramsey
##############################

##############################
# Activité 5 - Preuve n=6
##############################

from ramsey_1 import *
from ramsey_1 import contient_3_amis_fixes
from ramsey_1 import contient_3_etrangers_fixes
from ramsey_3 import decimal_vers_binaire
from ramsey_4 import sous_ensembles
from ramsey_4 import sous_ensembles_fixe

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

# Sous-ensembles
n = 6
k = 3
SS_ENS_6_3 = sous_ensembles_fixe(n,k)

## Question 1 ##

##################################################
def graphe_contient_3(graphe):
    """Cherche si trois sommets sont tous reliés entre eux"""

    n = len(graphe)

    #for sous_ens in SS_ENS_6_3:  # Pour n=6, k=3
    for sous_ens in sous_ensembles_fixe(n,3):  # Pour n quelconque
        #print(sous_ens)
        contient_3_amis = contient_3_amis_fixes(graphe,*sous_ens)
        contient_3_etrangers = contient_3_etrangers_fixes(graphe,*sous_ens)
```

```
        contient = contient_3_amis or contient_3_etrangers
        if contient == True:
            break

    # Affichage
    # if contient == True:
    #     print("Validé par l'exemple:",sous_ens)
    # else:
    #     print("Problème")
    # if contient == False:
    #     print("Problème")
    #     voir_graphe(graphe)

    return contient

 # Test
# Un exemple

if __name__ == '__main__':
    print("--- Test conjecture un seul graphe ---")
    print("--- Exemple 1 ---")
    print(graphe_contient_3(exemple_graphe_1))
    print("--- Exemple 2 ---")
    print(graphe_contient_3(exemple_graphe_2))
    print("--- Exemple 3 ---")
    print(graphe_contient_3(exemple_graphe_3))
    print("--- Exemple 4 ---")
    print(graphe_contient_3(exemple_graphe_4))

## Question 2 ##

##################################################
##################################################
# Calcul de tous les graphes possible à n sommets
# Il y a 2^( (n-1)*n/2 )
def voir_tous_graphes(n):

    N = ((n-1) * n)//2
    print("Nombre total de graphes :",2**N)

    for p in range(2**N):
        # Conversion binaire
        liste_binaire = decimal_vers_binaire(p,N)

        print("p =",p,liste_binaire)

        graphe = [[0 for j in range(n)] for i in range(n)]

        for j in range(0,n):
            for i in range(j+1,n):
                b = liste_binaire.pop()
                graphe[i][j] = b
                graphe[j][i] = b

        voir_graphe(graphe)

    return

 # Test
# n = 4
# print("--- Affiche tous les graphes possibles ---")
# print("n = ",n)
# voir_tous_graphes(n)

## Question 3 ##
```

```
##################################################
##################################################
# Test de tous les graphes possible à n sommets
# Il y a 2^( (n-1)*n/2 )

def test_tous_graphes(n):

    N = ((n-1) * n)//2
    print("Nombre total de graphes :",2**N)

    for p in range(2**N):
        # Conversion binaire
        liste_binaire = decimal_vers_binaire(p,N)

        # print("p =",p,liste_binaire)

        graphe = [[0 for j in range(n)] for i in range(n)]

        for j in range(0,n):
            for i in range(j+1,n):
                b = liste_binaire.pop()
                graphe[i][j] = b
                graphe[j][i] = b

        # voir_graphe(graphe)
        test = graphe_contient_3(graphe)
        if test == False:
            print("Problème avec",p)

    return

# Test
n = 6
print("\n\n--- Preuve du théorème de Ramsey, n = 6 ---")
print("n = ",n)
print("--- Recherche de graphe ne vérifiant pas l'énonce...")
test_tous_graphes(n)
print("fin des calculs ---")
print("Si rien ne s'est affiché, c'est que c'est bon !")
```

Activité 6

Activité 6 — ramsey_6.py

```
################################
# Graphes et combinatoire de Ramsey
################################

################################
# Activité 6 - Pour aller plus loin
################################

from ramsey_1 import *
from ramsey_1 import contient_3_amis_fixes
from ramsey_1 import contient_3_etrangers_fixes
from ramsey_3 import decimal_vers_binaire
from ramsey_4 import sous_ensembles
from ramsey_4 import sous_ensembles_fixe

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

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


## Question 1 ##

# Sous-ensembles
n = 6
k = 3
SS_ENS_3 = sous_ensembles_fixe(n,k)


################################################
def graphe_contient_3(graphe):
    """Cherche si un graphe possède 3 amis ou 3 étrangers."""

    n = len(graphe)

    for sous_ens in SS_ENS_3:
        contient_3_amis = contient_3_amis_fixes(graphe,*sous_ens)
        contient_3_etrangers = contient_3_etrangers_fixes(graphe,*sous_ens)
        contient = contient_3_amis or contient_3_etrangers
        if contient == True:
            break

    # Affichage
    # if contient == True:
    #     print("Validé par l'exemple:",sous_ens)
    # else:
    #     print("Problème")
    # if contient == False:
    #     print("Problème")
    #     voir_graphe(graphe)

    return contient

# Test de tous les graphes possible à n sommets
# Il y a 2^( (n-1)*n/2 )

def test_tous_graphes(n):

    N = ((n-1) * n)//2
    print("Nombre total de graphes :",2**N)

    for p in range( ((2**N) // 2)):
        # Conversion binaire
        liste_binaire = decimal_vers_binaire(p,N)

        # print("p =",p,liste_binaire)

        graphe = [[0 for j in range(n)] for i in range(n)]

        for j in range(0,n):
            for i in range(j+1,n):
                b = liste_binaire.pop()
                graphe[i][j] = b
                graphe[j][i] = b

        # voir_graphe(graphe)
        test = graphe_contient_3(graphe)
        if test == False:
            print("Problème avec graphe p =",p)

    return

# Test
print("\n\n--- Preuve du théorème de Ramsey, n =",n,"---")
print("n = ",n)
```

```
print("--- Recherche de graphe ne vérifiant pas l'énonce...")
test_tous_graphes(n)
print("fin des calculs ---")
print("Si rien ne s'est affiché, c'est que c'est bon !")

# n = 6 : 0.5 secondes
# n = 7 : 20 secondes
# n = 8 : 2500 secondes = 40 min (extrapolation sur échantillon de 10^-2)
# n = 9 : 800 000 secondes = 9 jours (extrapolation sur échantillon de 10^-4)

## Question 2 ##

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

# Sous-ensembles
n = 9
SS_ENS_3 = sous_ensembles_fixe(n,3)
SS_ENS_4 = sous_ensembles_fixe(n,4)

#####################################################
def contient_4_amis_fixes(graphe,i,j,k,l):
    """Cherche si sommets i, j, k sont tous reliés entre eux comme amis"""
    if graphe[i][j] == 1 and graphe[i][k] == 1 and graphe[i][l] == 1 and graphe[j][k] == 1
    ↪ and graphe[j][l] == 1 and graphe[k][l] == 1:
        return True
    else:
        return False

# Test de tous les graphes possible à n sommets
# pour savoir s'il existe 4 amis ou 3 étrangers

def graphe_contient_3_4(graphe):
    """Cherche si trois ou quatre sommets sont tous reliés entre eux"""

    n = len(graphe)

    # Cherche 3 étrangers
    for sous_ens in SS_ENS_3:
        contient_3_etrangers = contient_3_etrangers_fixes(graphe,*sous_ens)
        if contient_3_etrangers == True:
            break

    # Si pas 3 étrangers, cherche 4 amis
    if contient_3_etrangers == False:
        for sous_ens in SS_ENS_4:
            contient_4_amis = contient_4_amis_fixes(graphe,*sous_ens)
            if contient_4_amis == True:
                break
    else:
        contient_4_amis = True # Peu importe vu que déjà 3 étrangers

    contient = contient_3_etrangers or contient_4_amis

    return contient


def ramsey_4_3(n):

    N = ((n-1) * n)//2
    print("Nombre total de graphes :",2**N)

    # for p in range(  ((2**N)) // 100000 ):
    for p in range( 1000000 ):
```

```
        # Conversion binaire
        liste_binaire = decimal_vers_binaire(p,N)

        # print("p =",p,liste_binaire)

        graphe = [[0 for j in range(n)] for i in range(n)]

        for j in range(0,n):
            for i in range(j+1,n):
                b = liste_binaire.pop()
                graphe[i][j] = b
                graphe[j][i] = b

        test = graphe_contient_3_4(graphe)
        if test == False:
            print("Problème avec graphe p =",p)

    return

# Test
print("\n\n--- Preuve du théorème de Ramsey avec 4 amis ou 3 étrangers, n =",n,"---")
print("n = ",n)
print("--- Recherche de graphe ne vérifiant pas l'énonce...")
ramsey_4_3(n)
print("fin des calculs ---")
print("Si rien ne s'est affiché, c'est que c'est bon !")

# n = 7, contre-exemples facile
# n = 8 contre-exemple par exemple p=111121101
# n = 9 est vrai ! Mais doit prendre 18 jours de calculs
```

22. Bitcoin

Activités bitcoin.py

```
################################
# Bitcoin
################################


from random import randint
from time import *


################################
# Activité 2 - Outils pour les listes
################################

# Constante globale de longueur des blocs
N = 6

# Constante pour preuve de travail
Max = [0,0,25]

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

## Question 1 ##

# Addition des termes de deux listes de même longueur
def addition(liste1,liste2):
```

```
    liste_somme = []
    for i in range(len(liste1)):
        liste_somme = liste_somme + [ (liste1[i]+liste2[i]) % 100 ]

    return liste_somme

# Test
print("--- Test somme liste ---")
print(addition([1,2,3,4,5,6],[1,1,1,1,1,1]))

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

## Question 2 ##

# Test si une liste est plus petite que liste_max
def est_plus_petit(liste,liste_max):
    i = 0
    n = len(liste_max)
    while (i < n) and (liste[i] <= liste_max[i]):
        i = i + 1
    if i == n:
        return True
    else:
        return False

# Test
print("--- Test plus petit liste ---")
print(est_plus_petit([0,0,24,4,5,6],[0,0,50]))

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

## Question 3 ##

def phrase_vers_liste(phrase):
    # Transforme lettres en nombre modulo 100
    liste = [ord(c) % 100 for c in phrase]

    # Rajoute des 0 devant si besoin
    while len(liste) % N > 0:
        liste = [0] + liste

    return liste

# Test
print("--- Phrase vers liste ---")
phrase = "Vive moi !"
print(phrase)
print(phrase_vers_liste(phrase))


##############################
# Activité 3 - Fonction de hachage
##############################


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

## Question 3 ##
p = [7,11,13,17,19,23] # nb premiers

def un_tour(bloc):
    # Addition
    bloc[1] = (bloc[1]+bloc[0]) % 100
    bloc[3] = (bloc[3]+bloc[2]) % 100
    bloc[5] = (bloc[5]+bloc[4]) % 100
```

```
    # m = p*m + 1 (modulo 100)
    for i in range(N):
        bloc[i] = (p[i]*bloc[i]+1) %  100
    # permutation
    bloc = [bloc[N-1]] + bloc[:N-1]
    return bloc

# Test
print("--- Test un tour ---")
bloc = [0,1,2,3,4,5]
print(bloc)
print(un_tour(bloc))

bloc = [1,1,2,3,4,5]
print(bloc)
print(un_tour(bloc))

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

## Question 3 ##

def dix_tours(bloc):
    for i in range(10):
        bloc = un_tour(bloc)
    return bloc

# Test
print("--- Test dix tours ---")
bloc = [0,1,2,3,4,5]
print(bloc)
print(dix_tours(bloc))

bloc = [1,1,2,3,4,5]
print(bloc)
print(dix_tours(bloc))

bloc = [99,96,87,56,67,76]
print(bloc)
print(dix_tours(bloc))

bloc = [70,92,22,4,16,90]
print(bloc)
print(dix_tours(bloc))

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

## Question 3 ##
def hachage(liste):

    while len(liste)>N:
        bloc1 = liste[0:N]  # Premier bloc
        bloc2 = liste[N:2*N] # Second bloc
        fin_liste = liste[2*N:] # Le reste
        # print(bloc1)
        # print(bloc2)
        # print(fin_liste)

        #bloc1 = un_tour(bloc1)   # Un tour
        bloc1 = dix_tours(bloc1) # Dix tours

        #print(bloc1)

        nouv_bloc_deb = addition(bloc1,bloc2)

        liste = nouv_bloc_deb + fin_liste
```

```
    # Tours de fin pour la liste (qui ne contient plus que N nb)
    # liste = un_tour(liste) # Un tour
    liste = dix_tours(liste) # Un tour

    return liste

# Test
print("--- Hachage d'une liste ---")

liste = [1,2,3,4,5,6,1,2,3,4,5,6]
hach = hachage(liste)
print(liste)
print(hach)

liste = [1,1,3,4,5,6,1,2,3,4,5,6]
hach = hachage(liste)
print(liste)
print(hach)

liste = [0,1,2,3,4,5,1,1,1,1,1,1,10,10,10,10,10,10]
hach = hachage(liste)
print(liste)
print(hach)


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

################################
# Activité 4 - Preuve de travail - Minage
################################

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

## Question 3 ##

def verification_preuve_de_travail(liste,preuve):

    liste_test = liste + preuve
    hach = hachage(liste_test)
    # print(preuve,hach)
    if est_plus_petit(hach,Max):
        return True
    else:
        return False

# Test
print("--- Verif Preuve de travail ---")

liste = [0,1,2,3,4,5]
preuve = [12, 3, 24, 72, 47, 77]
# Max = [0,0,7]

start_time = time()
print(verification_preuve_de_travail(liste,preuve))
end_time = time()
duree = end_time-start_time

print("Temps de calcul :",duree)

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

## Question 2 ##

def preuve_de_travail(liste):

    hach = [1,1,1,1,1,1]

    while not(est_plus_petit(hach,Max)):
```

```
        preuve = [randint(0,99) for i in range(N)]
        liste_test = liste + preuve
        hach = hachage(liste_test)
    print(preuve,hach)

    return preuve

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

## Question 2 bis ##

from itertools import product

def preuve_de_travail_bis(liste):
    for preuve in product(range(100),range(100),range(100),range(100),range(100),range(100))
    ↪ :
        preuve = list(preuve)
        liste_test = liste + preuve
        hach = hachage(liste_test)
        if est_plus_petit(hach,Max):
            break

    print(preuve,hach)
    return preuve

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

## Question 3 ##

# Test
print("--- Preuve de travail ---")

start_time = time()
liste = [0,1,2,3,4,5]
# preuve = preuve_de_travail(liste)
# preuve = preuve_de_travail_bis(liste)
end_time = time()
duree = end_time-start_time

print("Temps de calcul :",duree)

##############################
# Activité 5 - Tes bitcoins
##############################

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

## Question 1 ##

preuve_init = [0,0,0,0,0,0]   # valeur au pif
Livre = [preuve_init]

def ajout_transaction(transaction):
    global Livre
    Livre = Livre + [transaction]
    return Livre

# Test
print("--- Création du livre et ajout d'une transaction ---")
print(Livre)
ajout_transaction("Bob +135")
print(Livre)

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

## Question 2 ##

def minage():
```

```
    global Livre
    transaction = Livre[-1]
    prec_preuve = Livre[-2]
    # print(transaction)
    # print(prec_hach)
    # print(phrase_vers_liste(transaction))
    liste = prec_preuve + phrase_vers_liste(transaction)

    preuve = preuve_de_travail(liste)

    Livre = Livre + [preuve]

    return Livre

# Test
print("--- Minage  ---")
print(Livre)
minage()
print(Livre)

# Exemple pour fiche
print("--- Exemple pour fiche  ---")
Max = [0,0,7]
hach_init = [3,1,4,1,5,9]   # valeur au pif
Livre = [hach_init]
ajout_transaction("Abel +35")
print(Livre)
minage()
print(Livre)

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

## Question 3 ##

def verification_livre():
    prec_preuve = Livre[-3]
    transaction = Livre[-2]
    preuve = Livre[-1]
    hach = hachage(prec_preuve+phrase_vers_liste(transaction)+preuve)
    if est_plus_petit(hach,Max):
        return True
    else:
        return False

# Test
print("--- Vérification du livre  ---")
print(Livre)
print(verification_livre())


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

## Question 4 ##
# Exemple complet

# Constante pour preuve de travail
Max = [0,0,7]

start_time = time()  # début chrono

hach_init = [0,0,0,0,0,0]   # valeur au pif
Livre = [hach_init]

# print(Livre)
# ajout_transaction("Abel +135")
```

```
# print(Livre)
# minage()
# print(Livre)
# print(verification_livre())

# ajout_transaction("Bob -77")
# print(Livre)
# minage()
# print(Livre)
# print(verification_livre())

# ajout_transaction("Camille -25")
# print(Livre)
# minage()
# print(Livre)
# print(verification_livre())

end_time = time()
duree = end_time-start_time
print("Temps de calcul :",duree)
```

23. Constructions aléatoires

Activités 1 et 2

Activités 1 et 2 aleatoire_vertical.py

```
################################
# Aléatoire
################################

from random import *
from tkinter import *
import time

################################
# Activité 1 - Faire tomber des blocs
################################

n = 4   # nb de lignes
p = 6   # nb de colonnes

tableau = [[0 for j in range(p)] for i in range(n)]

tableau[3][3] = 1
tableau[3][2] = 1
tableau[2][2] = 1
tableau[1][2] = 1
tableau[0][4] = 1

#####################################################
def voir_tableau():

    for i in range(n):
        for j in range(p):
            print(tableau[i][j], end="")
        print()

    return
```

```
voir_tableau()

##################################################
def peut_tomber(i,j):
    if i == n-1:      # tout en bas
        return False

    if tableau[i+1][j]:  # case juste en-dessous
        return False

    if j>0 and tableau[i][j-1]:   # à gauche
        return False

    if j<p-1 and tableau[i][j+1]:  # à droite
        return False

    return True

##################################################
def faire_tomber_un_bloc(j):
    # j = nouveau_bloc()
    i = 0
    while peut_tomber(i,j):
        i = i + 1

    tableau[i][j] = 1

    return i,j

##################################################
def faire_tomber_des_blocs(k):
    # voir_tableau()
    # print()
    for __ in range(k):
        j = randint(0,p-1)
        faire_tomber_un_bloc(j)
        # voir_tableau()
        # print()

    return

# faire_tomber_des_blocs(7)
# print()
# voir_tableau()

# exit()

##############################
# Activité 2 - Affichage tkinter statique
##############################

n = 125   # nb de lignes
p = 250   # nb de colonnes

tableau = [[0 for j in range(p)] for i in range(n)]

echelle = 5  # échelle
nb_blocs = 500

root = Tk()

canvas = Canvas(root, width=p*echelle, height=n*echelle, background="white")
canvas.pack(fill="both", expand=True)


def afficher_tableau():
```

```
    canvas.delete("all")   # Efface tout

    for i in range(n):
        for j in range(p):
            if tableau[i][j]:
                canvas.create_rectangle(j*echelle,i*echelle,j*echelle+echelle-1,i*echelle+
    ↪ echelle-1,width=1,fill='green')
    return

# Test
# afficher_tableau()

def action_bloc():
    faire_tomber_des_blocs(nb_blocs)
    afficher_tableau()

    return

bouton_bloc = Button(root,text="Afficher blocs", width=20, command=action_bloc)
bouton_bloc.pack(pady=10)

bouton_quitter = Button(root,text="Quitter", width=20, command=root.quit)
bouton_quitter.pack(side=BOTTOM, pady=10)

root.mainloop()
```

Activité 3

Activité 3 aleatoire_circulaire.py

```
##############################
# Aléatoire - Idées
##############################

from random import *
from tkinter import *
import time

##############################
# Activité 1 - Faire tourner des blocs
##############################

n = 10   # nb de lignes
p = 10   # nb de colonnes
bord = min(n,p)//5  # distance au bord

tableau = [[0 for j in range(p)] for i in range(n)]

tableau[(n-1)//2][(p-1)//2] = 1  # Centre

##################################################
def voir_tableau():

    for i in range(n):
        for j in range(p):
            print(tableau[i][j], end="")
        print()

    return

# voir_tableau()

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

```
def est_libre(i,j):
    # if tableau[i][j]:   # sur un bloc existant
    #     return False

    if i>0 and tableau[i-1][j]:   # au dessus
        return False

    if i<n-1 and tableau[i+1][j]:  # en-dessous
        return False

    if j>0 and tableau[i][j-1]:   # à gauche
        return False

    if j<p-1 and tableau[i][j+1]:  # à droite
        return False

    return True

#####################################################
def est_dedans(i,j):
    if (0 <= i < n) and (0 <= j < p):
        return True
    else:
        return False

#####################################################
def lancer_un_bloc():
    i = randint(0+bord,n-1-bord)
    j = randint(0+bord,p-1-bord)

    while est_dedans(i,j) and est_libre(i,j):
        dx = randint(-1,1)
        dy = randint(-1,1)
        i = i + dx
        j = j + dy

    if est_dedans(i,j):
        tableau[i][j] = 1

    return i,j

#####################################################
def lancer_des_blocs(k):
    # voir_tableau()
    # print()
    for __ in range(k):
        lancer_un_bloc()
        # voir_tableau()
        # print()

    return

lancer_des_blocs(5)


##############################
# Activité 2 - Affichage tkinter statique
##############################

n = 170   # nb de lignes
p = 200   # nb de colonnes

bord = min(n,p)//10  # distance au bord pour lancement
echelle = 5

tableau = [[0 for j in range(p)] for i in range(n)]
```

```
tableau[(n-1)//2][(p-1)//2] = 1  # Centre

nb_blocs = 500

root = Tk()

canvas = Canvas(root, width=p*echelle, height=n*echelle, background="white")
canvas.pack(fill="both", expand=True)

def afficher_tableau():
    canvas.delete("all")   # Efface tout

    for i in range(n):
        for j in range(p):
            if tableau[i][j]:
                canvas.create_rectangle(j*echelle,i*echelle,j*echelle+echelle-1,i*echelle+
    ↪ echelle-1,width=1,fill='green')
    return

def action_bloc():
    lancer_des_blocs(nb_blocs)
    afficher_tableau()

    return

bouton_bloc = Button(root,text="Lancer des blocs", width=20, command=action_bloc)
bouton_bloc.pack(pady=10)

bouton_quitter = Button(root,text="Quitter", width=20, command=root.quit)
bouton_quitter.pack(side=BOTTOM, pady=10)

root.mainloop()
```