1. Hello world!

Activity 1

Activity 1 hello_world_1.py

```
##############################
# Hello world!
##############################

##############################
# Activity 1 - Numbers
##############################

##############################
# Cours
print("--- Lesson ---")
# Display a sentence
print("Hello world!")

# Addition
5+7
print(5+7)

# Multiplication
print(6*7)

print(3*(12+5))

print(3*1.5)

# Power

print(3**2)

print(10**-3)

# Division of real number

print(14/4)

print(1/3)

# Euclidean division and modulo

print(14//4)

print(14%4)


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

# Q1
# Number of second in a century
print("--- Question 1 ---")
print(100 * 365 * 24 * 60 * 60)

# Q2
# When it is greater than one billion
print("--- Question 2 ---")
print((1+2)*(3+4)*(5+6)*(7+8)*(9+10)*(11+12)*(13+14)*(15+16))

# Q3
# Last three digits of 123456789 * 123456789 * ...
```

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

# Q4
# First 1/n with a period of 7
print("--- Question 4 ---")
print(1/7)     # Période 6
print(1/239)   # Période 7

# Q5
# Find a number knowing two divisions and a remainder
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)
```

Activity 2

Activity 2 — hello_world_2.py

```
################################
# Hello world!
################################

################################
# Activity 2 - Variables
################################

################################
# Lesson

# C1 - variables

a = 3  # One variable
b = 5  # Another variable

print("The sum is",a+b)   # Display the sum
print("The product",a*b)  # Display the product

c = b**a     # New variable...
print(c)     # ... that is displayed

# C2 - area of a triangle

base = 8
height = 3
area = base * height / 2
print(area)
# print(Area)  # !! Error !!

# C3 - Re-assignement

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

```
print(S)

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

# Q1

# Areas - Volumes

# Trapezoid: right namming
B, b, h = 7, 4, 3
area = (B + b) * h / 2
print("The area is",area)

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

# Disc
PI = 3.14
R = 10
area = PI * R**2
print(area)

# Q2
# Put the lines back in order so that at the end x has the value 46.
x = 7
y = 2*x
y = y - 1
x = x + 3*y
print(x)

# Q3
# Interest of 10%
S = 1000
S = S * 1.1
S = S * 1.1
S = S * 1.1

# Q4

# Good way to exchange the values of a and b

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

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

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

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

print(a,b)

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

Activity 3

Activity 3 hello_world_3.py

```
################################
# Hello world!
################################

################################
# Activity 3 - Use functions
################################

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

# C1 - functions

print("Hi there.")

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

# C2 - math module

from math import *

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

# C3 - sine and cosine

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

# C4 - decimal number to integer

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

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

# Q1
```

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

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

# Q2
# Absolute value
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)
```

Activity 4

Activity 4 hello_world_4.py

```
################################
# Hello world!
################################

from math import *

################################
# Activity 4 - Loop "for"
################################

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

# C1 - Loop "for"

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

# C2 - Loop "for"

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

# C3

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

```
# C4 - Nesting of loops

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
# Powers of 2
for k in range(1,11):
    print(2**k)

# Q3
# Minimum of a function by scanning
for i in range(101):
    x = i/100
    y = x**3 - x**2 - 1/4*x + 1
    print("x =",x,"y =",y)

# Q4
# Volume of ball equal to 100
for i in range(50):
    R = i/10
    V = 4/3 * 3.14 * R**3
    print("R =",R,"V =",V)
```

2. Turtle (Scratch with Python)

Activity 1

Activity 1 turtle_1.py

```
##############################
# Turtle
##############################

##############################
# Activity 1 - Move your turtle!
##############################

# Begining of the code

from turtle import *

width(5)    # ize of the pencil

# Letter "P"
```

```
color('red')

left(90)        # 90 degree to the left
forward(200)    # Go ahead!

right(90)
forward(100)

right(90)
forward(100)

right(90)
forward(100)

up()

# End of the code

# Letter "Y"

color('blue')

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

exitonclick()
```

Activity 2

Activity 2 turtle_2.py

```
################################
# Turtle
################################

################################
# Activity 2 - Loop "for"
################################

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

from turtle import *

# A pentagon
width(5)
color('blue')

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

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

```
# An other pentagon
color('red')

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

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

# A dodecagon (12 edges)

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

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

# A spiral

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

exitonclick()
```

Activity 3

Activity 3 turtle_3.py

```
################################
# Turtle
################################

################################
# Activity 3 - Graph of a function
################################

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   # Parabola
    y = 100*sin(1/20*x)     # Sine
```

```
    goto(x,y)

exitonclick()
```

Activity 4

Activity 4 turtle_4.py

```
##############################
# Turtle
##############################

##############################
# Activity 4 - Nested "for" loop -Sierpinski triangle
##############################

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()
```

Activity 5

Activity 5 turtle_5.py

```
##############################
# Turtle
##############################

##############################
# Activity 5 - Multiplication tables
##############################
```

```
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()
```

Activity 6

Activity 6 turtle_6.py

```
################################
# Turtle
################################

################################
# Activity 4 - Several turtles - The pursuit
################################

# Preparation

from turtle import *

turtle1 = Turtle()
turtle2 = Turtle()
turtle3 = Turtle()
turtle4 = Turtle()

turtle1.speed("fastest")
turtle2.speed("fastest")
turtle3.speed("fastest")
turtle4.speed("fastest")

turtle1.color('red')
turtle2.color('blue')
turtle3.color('orange')
turtle4.color('green')

# turtle1.width(5)
# turtle2.width(5)
# turtle3.width(5)
# turtle4.width(5)

turtle1.up()
turtle1.goto(-200,-200)
turtle1.down()

turtle2.up()
turtle2.goto(200,-200)
turtle2.down()
```

```
turtle3.up()
turtle3.goto(200,200)
turtle3.down()

turtle4.up()
turtle4.goto(-200,200)
turtle4.down()

print(turtle1.position())
print(turtle1.towards(0,0))

# Main loop

for i in range(40):

    position1 = turtle1.position()
    position2 = turtle2.position()
    position3 = turtle3.position()
    position4 = turtle4.position()

    turtle1.goto(position2)  # Go where is the next turtle
    turtle1.goto(position1)  # and go back to its position

    turtle2.goto(position3)
    turtle2.goto(position2)

    turtle3.goto(position4)
    turtle3.goto(position3)

    turtle4.goto(position1)
    turtle4.goto(position4)

    angle1 = turtle1.towards(position2)  # Memorize the angle
    turtle1.setheading(angle1)   # Set the direction with this angle

    angle2 = turtle2.towards(position3)
    turtle2.setheading(angle2)

    angle3 = turtle3.towards(position4)
    turtle3.setheading(angle3)

    angle4 = turtle4.towards(position1)
    turtle4.setheading(angle4)

    turtle1.forward(10)    # Move
    turtle2.forward(10)
    turtle3.forward(10)
    turtle4.forward(10)

exitonclick()
```

3. If ... then ...

Activity 1

Activity 1 ifthen_1.py

```
##############################
# If ... then ...
##############################
```

```
##############################
# Activity 1 - Quiz multiplications
##############################

from random import *

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

print("How much is the product",a,"times",b,"?")
answer_str = input("Your answer: ")
answer_int = int(answer_str)

if answer_int == a*b:
    print("Well done!")
else:
    print("Lost! The correct answer was",a*b)
```

Activity 2

Activity 2 ifthen_2.py

```
##############################
# If ... then ...
##############################

##############################
# Fctivity 2 - Turtle
##############################

from turtle import *

width(5)
color('blue')

mot = "FflFlFrFlFFlfFF"

for c in mot:

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

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

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

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

exitonclick()
```

Activity 3

Activity 3 ifthen_3.py

```
##############################
# If ... then ...
##############################

##############################
# Activity 3 - Digits of a integer
##############################

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

for t in range(10):
    for u in range(10):
        n  = 10*t + u
        print(n)

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

for h in range(10):
    for t in range(10):
        for u in range(10):
            n  = 100*h + 10*t + u
            if u == 3 and (h+t+u >= 15) and (t == 0 or t == 2 or t == 4 or t == 6 or t == 8)
    ↪ :
                print(n)

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

count = 0

for h in range(10):
    for t in range(10):
        for u in range(10):
            n  = 100*h + 10*t + u
            if u == 3 and (h+t+u >= 15) and (t == 0 or t == 2 or t == 4 or t == 6 or t == 8)
    ↪ :
                count = count + 1

print("Total number of solutions:",count)
```

Activity 4

Activity 4 ifthen_4.py

```
##############################
# If ... then ...
##############################

##############################
# Activity 4 - Triangles
##############################
```

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

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

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

# Do we have a <= b <= c?

if a <= b and b <= c:
    print("Lengths in the right order.")
else:
    print("Lengths in the wrong order.")

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

# Can we construct a triangle from these three lengths?

if a+b >= c:
    print("Such a triangle exists.")
else:
    print("Such a triangle doesn't exist.")

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

# Is the triangle recantgle?

if a**2 + b**2 == c**2:
    print("The triangle is rectangle.")
else:
    print("The triangle is not rectangle.")

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

# Is the triangle equilateral?

if (a == b) and (b == c) and (a == c):
    print("The triangle is equilateral.")
else:
    print("The triangle is not equilateral.")

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

# Is the triangle isosceles?

if (a == b) or (b == c) or (a == c):
    print("The triangle is isosceles.")
else:
    print("The triangle is not isosceles.")

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

# Are all the angles acute?

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("All the angles are acute.")
else:
    print("Not all the angles are acute. (At least one of them is obtuse).")
```

Activity 5

Activity 5 ifthen_5.py

```
##############################
# If ... then ...
##############################

##############################
# Activity 5 - The Mytery number
##############################

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

# Classic mystery number

mystery_nb = randint(0,99)

for trial in range(7):
    print("What is the mystery number?")
    answer_str = input("Your proposal:")
    answer_int = int(answer_str)
    if mystery_nb == answer_int:
        print("Bravo!")
        break # Stop the loop

    if mystery_nb < answer_int:
        print("No, the number to find is smaller!")

    if mystery_nb > answer_int:
        print("No, the number to find is bigger!")

# When it's over:
if mystery_nb != answer_int:
    print("Lost! The mystery number was",mystery_nb)

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

# Variant: the computer lies (1 time out of 4)

# mystery_nb = randint(0,99)

# for trial in range(7):
#     print("What's the mystery number?")
#     answer_str = input("Your proposal:")
#     answer_int = int(answer_str)

#     # # 1 time out of 4 (about) the computer lies
#     tell_truth = True
#     chance = randint(1,4)
#     if chance == 4:
```

```
#         tell_truth = False

#     if mystery_nb == answer_int:
#         print("Bravo!")
#         break # Stop the loop

#     if mystery_nb < answer_int:
#         if tell_truth == True:
#             print("No, the number to find is smaller!")
#         else:
#             print("No, the number to find is bigger!")

#     if mystery_nb > answer_int:
#         if tell_truth == True:
#             print("No, the number to find is bigger!")
#         else:
#             print("No, the number to find is smaller!")

# # When it's over:
# if mystery_nb != answer_int:
#     print("Lost! The mystery number was",mystery_nb)

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

# Variant: the mystery number changes a little

# mystery_nb = randint(0,99)
# print(mystery_nb)
# for trial in range(7):
#     print("What's the mystery number?")
#     answer_str = input("Your proposal:")
#     answer_int = int(answer_str)

#     # Modification of the mystery number
#     chance = randint(-3,3)
#     mystery_nb = mystery_nb + chance
#     if mystery_nb < 1:
#         mystery_nb = 1
#     if mystery_nb > 99:
#         mystery_nb = 99

#     if mystery_nb == answer_int:
#         print("Bravo!")
#         break # Stop the loop

#     if mystery_nb < answer_int:
#         print("No, the number to find is smaller!")

#     if mystery_nb > answer_int:
#         print("No, the number to find is bigger!")

# # When it's over:
# if mystery_nb != answer_int:
#     print("Lost! The mystery number was",mystery_nb)
```

4. Functions

Activity 1

Activity 1 functions_1.py

```
##############################
# Functions
##############################

##############################
# Activity 1 - Introduction to functions
##############################

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

# Function without parameter, without output

def print_table_of_7():
    """ Display the table of 7 """

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

    return

# Test
print_table_of_7()

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

def print_hello():
    """ Say hello """

    prenom = input("What's your name? ")
    print("Hello",prenom)

    return

# Test
print_hello()

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

# Function with parameter, sans sortie

def print_a_table(n):
    """ Print the table of n """

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

    return

# Test
print_a_table(5)

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

def say_greeting(sentence):
```

```
    """ Say hello, hi, goodbye... """

    name = input("What's your name? ")
    print(sentence,name)

    return

# Test
say_greeting("Goodbye")


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

# Function without parameter, with output

def ask_name():
    """ Ask and return the first and last names """

    first_name = input("What is your first name? ")
    last_name = input("What is your last name? ")

    full_name = first_name + " " + last_name.upper()

    return full_name

# Test
identity = ask_name()
print("Identity:",identity)
```

Activity 2

Activity 2 functions_2.py

```
##############################
# Functions
##############################

##############################
# Activity 2 - Functions
##############################

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

# Function with parameter, with output

def trinomial_1(x):
    """ Compute 3x^2-7x+4 """

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

    return result

# Test
print("--- Trinomial ---")
for i in range(10):
    print("The value at x =",i,"is",trinomial_1(i))

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

def trinomial_2(a,b,c,x):
```

```
    """ Compute ax^2+bx+c """

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

    return result

# Test
a = 2 ; b = -1 ; c = 0
print("Trinomial for a,b,c =",a,b,c)
for i in range(10):
    print("The value at x =",i,"is",trinomial_2(a,b,c,i))


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

# Function with parameter, with sortie

def conversion_dollars_to_euros(amount):
    """ Convert an amount given in dollars to euros """

    amount_euro = 0.89 * amount

    return amount_euro

# Test
print("--- Currency ---")
x = 20
print(x,"dollars is", conversion_dollars_to_euros(x),"euros")

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

def conversion_dollars(amount,currency):
    """ Convert an amount given in dollars to another currency """

    if currency == "euro":
        rate = 0.89
    if currency == "pound":
        rate = 0.77
    if currency == "yen":
        rate = 110

    amount_currency = amount * rate

    return amount_currency

# Test
x = 100
for mycurrency in ["yen","euro","pound"]:
    print(x,"dollars equal", conversion_dollars(x,mycurrency),mycurrency)


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

from math import *

# Compute several volumes

def volume_cube(a):
    return a**3

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

def volume_cylinder(r,h):
    return pi * r**2 * h
```

```
def volume_box(a,b,c):
    return a * b * c

# Test
print("--- Volumes ---")
print(volume_cube(3))
print(volume_ball(3))
print(volume_cylinder(2,5))
print(volume_box(3,4,5))

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

def perimeter_area_rectangle(a,b):
    """ Compute the perimeter and the area
    of a rectangle of side a and b """

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

    return p, A

def perimeter_area_disc(r):
    """ Compute perimeter and area
    of a disc of radius r """

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

    return p, A

print("--- Perimeters et areas ---")
print(perimeter_area_rectangle(4,5))
print(perimeter_area_disc(3))

# Experimental research: comparison perimeter/area of a disc

for R in range(0,30):
    perimeter, area = perimeter_area_disc(R/10)
    print(R/10, perimeter - area)

# Experimental conclusion:
#   for 0 < r < 2, the perimeter is strictly greater than its area
#   pour r = 2, the perimeter is equal to area,
#   pour r > 2, the perimeter is strictly lower than the area
```

Activity 3

Activity 3 functions_3.py

```
##############################
# Functions
##############################

##############################
# Activity 3 - Turtle
##############################

from turtle import *
width(5)    # Width of the pencil
```

```
##############################
## Question 1 ##
def triangle():
    color('red')
    forward(200)
    left(120)
    forward(200)
    left(120)
    forward(200)
    return

# Test
# triangle()
# exitonclick()

##############################
## Question 2 ##
def square():
    color('green')
    for i in range(4):
        forward(200)
        left(90)
    return

# Test
# square()
# exitonclick()

##############################
## Question 3 ##
def hexagon(length):
    color('blue')
    for i in range(6):
        forward(length)
        left(60)
    return

# Test
# hexagon(100)
# exitonclick()

##############################
## Question 4 ##
def polygon(n,length):
    color('purple')
    angle = 360/n
    for i in range(n):
        forward(length)
        left(angle)
    return

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

# Test all
up()
```

```
goto(-450,0)
down()
triangle()
up()
goto(-200,0)
setheading(0)
down()
square()
up()
goto(100,0)
setheading(0)
down()
hexagon(100)
up()
goto(320,0)
setheading(0)
down()
polygon(8,70)
up()
exitonclick()
```

Activity 4

Activity 4 functions_4.py

```
##############################
# Functions
##############################

##############################
# Activity 4 - Functions
##############################

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

def reduction(age):
    """ Return the percentage of discount with respect to the age """

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

    return disc

# Test
print("--- Reduction ---")
my_age = 16
print("I'm",my_age,"years old and my reduction is of",reduction(my_age),"%.")

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

def amount(normal_rate,age):
```

```
    """ Compute the amount to pay with respect to normal rate and the age """

    reduc = reduction(age)
    rate = normal_rate * (100-reduc)/100

    return rate

# Test
print("--- Total amount to pay ---")
amout_family = amount(30,9) + 2*amount(20,16) + 2*amount(35,40)
print(amout_family)

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

def is_calculation_correct(a,b,answer):
    """ Test if the result of a*b is correct """

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

# Test
print("--- Test answer multiplication ---")
print(is_calculation_correct(6,7,42))

def test_multiplication(a,b,lang):
    """ Ask a multiplication in English or another language
    and print if the answer is correct """

    # Sentence in English or French
    if lang == "english":
        question = "How much is the product a x b? "
        correct_answer = "Well done!"
        wrong_answer = "It's wrong!"

    if lang == "french":
        question = "Combien vaut le produit a x b ? "
        correct_answer = "Bravo !"
        wrong_answer = "Ce n'est pas cela !"

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

    if is_calculation_correct(a,b,answer):
        print(correct_answer)
    else:
        print(wrong_answer)

    return

# Test
print("--- Quiz multiplication ---")
test_multiplication(6,7,"french")
```

Activity 5

Activity 5 functions_5.py

```
##############################
# Functions
##############################

##############################
# Activity 5 - Egalité expérimentale
##############################

from math import *

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

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

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

######################################################
def experimental_equality_1(f,g):
    """ Teste si deux Functions 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(experimental_equality_1(absolute_value,absolute_value_moins))  # True
print(experimental_equality_1(absolute_value,root_of_square))       # True

######################################################
## 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 experimental_equality_2(F,G):
    """ Test if two functions of two variables are experimentally equal """

    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("--- Experimental equality, two variables ---")
print(experimental_equality_2(F1,F2))   # True
print(experimental_equality_2(F3,F4))   # False
print(experimental_equality_2(F3,F5))   # True

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

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

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

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

precision = 0.00001   # = 10**-5

def experimental_equality_3(f,g):
    """ Test if two functions are experimentally equalt
    with an error margin """

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

# Test
print("--- Approximate experimental equality ---")
print(experimental_equality_1(sincos,un))  # False !! But yould be True !
print(sin(3)**2+cos(3)**2)         #  Explenation: Python does not exactly return 1
print(experimental_equality_3(sincos,un))  # True !

# Test with with other equalities, examples :
# sin(2x) = 2 sin(x)cos(x)
# cos(pi/2 - x) = sin(x)

################################################
# A wrong equality but experimentally true

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

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

def g2(x):
    return 0

print("--- A wrong equality but experimentally true ---")
```

```
print(experimental_equality_3(g1,g2))  # True (we always have equality for all integer i)
print(g1(1/2))  # however g1(0.5) is not zero, hence the equality is not true in general
```

5. Arithmetic – While loop – I

Activity 1

Activity 1 while_1.py

```
##############################
# While - Boolean - Arithmetic
##############################

##############################
# Activity 1 - Divisibility, quotient, remainder
##############################

##############################
## Question 1 ##
def quotient_remainder(a,b):
    """ Displays the quotient and remainder and checks
    the validity of the Euclidean division """

    q = a // b
    r = a % b
    print("Division of a =",a,"by b =",b)
    print("The quotient is q =",q)
    print("The remainder is r =",r)

    if (r >=0) and (r < b):
        check_remainder = True
    else:
        check_remainder = False
    print("Check remainder: 0 <= r < b ?",check_remainder)

    if a == b*q +r:
        check_equal = True
    else:
        check_equal = False
    print("Check a = bq + r ?",check_equal)

    return q,r

# Test
print("--- Quotient and remainder ---")
quotient_remainder(100,7)

##############################
## Question 2 ##
def is_even(n):
    """ Tests if the integer n is even or not (returns true or false) """
    remainder = n % 2
    if remainder == 0:
        return True
    else:
```

```
        return False

def is_even_bis(n):
    """ Tests if the integer n is even or not (returns true or false) """
    unit = n % 10
    if (unit==0) or (unit==2) or (unit==4) or (unit==6) or (unit==8):
        return True
    else:
        return False

# With two lines!
def is_even_ter(n):
    return (n % 2) == 0

# Test
print("--- Even/odd  ---")
print(is_even(1023))

##############################
## Question 3 ##
def is_divisible(a,b):
    """ Test if a is divisible per b """
    if a % b == 0:
        return True
    else:
        return False

# Test
print("--- Divisibility  ---")
print(is_divisible(125,5))
```

Activity 2

Activity 2 while_2.py

```
##############################
# While - Boolean - Arithmetic
##############################

##############################
# Activity 2 - Divisor, prime number - Loop while
##############################

##############################
## Question 1 ##
def smallest_divisor(n):
    """ Find the smallest divisor of n """
    d = 2
    while n % d != 0:
        d = d + 1
    return d

# Test
print("--- Smallest divisor ---")
```

```
print(smallest_divisor(7*13))

##############################
## Question 2 ##
def is_prime_1(n):
    """ Test if n is a prime number """
    d = 2
    while n % d != 0:
        d = d + 1
    if d == n:
        return True
    else:
        return False

# Test
print("--- Is prime (1) ---")
print(is_prime_1(97))

##############################
## Question 3 ##
# Fermat numbers
def counter_example_fermat():
    for n in range(6):
        fermat = 2**(2**n)+1
        print(n,fermat,is_prime_1(fermat))
    return
# Test
print("--- Test Fermat numbers conjecture ---")
counter_example_fermat()

##############################
## Question 4 ##
def is_prime_2(n):
    """ Test if n is a prime number """
    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("--- Is prime (2) ---")
print(is_prime_2(97))

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

def is_prime_3(n):
    """ Test if n is a prime number """
```

```
    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("--- Is prime (3) ---")
print(is_prime_3(97))

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

## Question 5 ##

# Calculation of the execution times of the different functions is_prime()

import timeit
print(timeit.timeit("is_prime_1(97)", setup="from __main__ import is_prime_1", number
    ↪ =100000))
print(timeit.timeit("is_prime_2(97)", setup="from __main__ import is_prime_2", number
    ↪ =100000))
print(timeit.timeit("is_prime_3(97)", setup="from __main__ import is_prime_3", number
    ↪ =100000))

################################
# We keep the best function!

def is_prime(n):
    return is_prime_3(n)
```

Activity 3

Activity 3 while_3.py

```
################################
# While - Boolean - Arithmetic
################################

################################
# Activity 3 - Divisor, prime numbers - Loop while (continued)
################################

################################
# Reminders from activity 2

def is_prime(n):
    """ Test if n is a prime number """
    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 prime_number_after(n):
    """ Look for the first prime number after n """
    p = n
    while not is_prime(p):
        p = p + 1
    return p

# Test
print("--- First prime number after an intger ---")
print(prime_number_after(60))
print(prime_number_after(100000))

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

def twin_prime_after(n):
    """ Find twin primes after n """
    p = n
    q = p + 2
    while (not is_prime(p)) or (not is_prime(q)):
        p = p + 1
        q = p + 2
    return p,q

# Test
print("--- First twin primes after an integer ---")
print(twin_prime_after(60))
print(twin_prime_after(100000))

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

def germain_prime_after(n):
    """ Find two Germain primes after n """
    p = n
    q = 2*p + 1
    while (not is_prime(p)) or (not is_prime(q)):
        p = p + 1
        q = 2*p + 1
    return p,q

# Test
print("--- First Germain primes after an integer ---")
print(germain_prime_after(60))
print(germain_prime_after(100000))
```

6. Strings – Analysis of a text

Activity 1

Activity 1 strings_1.py

```
##############################
# Strings - Text analysis
##############################

##############################
# Activity 1 - Plurals
##############################

## Question 1 ##

word = "cat"
plural = word + "s"
print("Singular:",word)
print("Plural:",plural)

## Question 2 ##

# word = "cat"
word = "bus"

last_letter = word[len(word)-1]

if last_letter == "s":
    plural = word + "es"
else:
    plural = word + "s"

print("Singular:",word)
print("Plural:",plural)

## Question 3 ##

# word = "cat"
# word = "bus"
word = "city"

last_letter = word[len(word)-1]

if last_letter == "s":
    plural = word
elif last_letter == "y":
    begin_word = word[0:len(word)-1]
    plural = begin_word + "ies"
else:
    plural = word + 's'

print("Singular:",word)
print("Plural:",plural)


## Question 4 ##

# It's better with a function!

def plural(word):
    """ Return the plural of the given word.
    Input: a word
```

```
    Output: the plural of the word (except exceptions) """

    last_letter = word[len(word)-1]

    if last_letter == "s":
        plural = word

    elif last_letter == "y":
        begin_word = word[0:len(word)-1]
        plural = begin_word + "ies"

    else:
        plural = word + 's'

    return plural

# Test
#word = input("Give me a noun: ")
#plural = plural(word)
#print("The plural is:",plural)

## Question 5 ##

def conjugation(verb):
    """ Conjugate a verb to present continuous tense.
    Input: a verb
    Output: print its conjugation """

    print("I'm " + verb + "ing")
    print("You're " + verb + "ing")
    print("He/she is " + verb + "ing")
    print("We are " + verb + "ing")
    print("You are " + verb + "ing")
    print("They are " + verb + "ing")

    return

# Test
verb = "sing"
#verb = input("Give me a verb: ")
conjugation(verb)
```

Activity 2

Activity 2 strings_2.py

```
##############################
# Strings - Text analysis
##############################

##############################
# Activity 2 - Word games
##############################

## Question 1 ##

def hamming_distance(word1,word2):
    """ Compute the Hamming distance
    Input: two words of same length
```

```
    Output: the distance """

    distance = 0
    for i in range(len(word1)):
        if word1[i] != word2[i]:
            distance = distance + 1

    return distance

# Test
first_word = "SNAKE"
second_word = "STACK"
dist = hamming_distance(first_word,second_word)
print("The distance between",first_word,"and",second_word,"is",dist)

## Question 2 ##

def upsidedown(word):
    """ Reverse a word
    Input: a word (a string)
    Output: the reversed word  """

    revword = ""
    for charac in word:
        revword = charac + revword

    return revword

# Test
word = "PYTHON"
revword = upsidedown(word)
print("The word",word,"becomes",revword,"!")


## Question 4 ##

def is_palindrome(word):
    """ Determined if a word is a palindrome
    Input: un word (a string)
    Output: true if it is a palindrome, false otherwise """

    revword = upsidedown(word)
    if word == revword:
        return True
    else:
        return False

# Test
word = "KAYAK"
print("Is the word",word,"a palindrome?",is_palindrome(word))


## Question 2 ##

def pig_latin(word):
    """ Transform a word to pig-latin
    Input: un word (une chaîne de charactères)
    Output: le word transformed to pig-lation. """

    # Case: start with a vowels
    first_letter = word[0]
    if first_letter in ["A", "E", "I", "O", "U", "Y"]:
        pig_latin = word + "WAY"
```

```
    # Case: start with consonants
    else:
        i = 0
        while word[i] not in ["A", "E", "I", "O", "U", "Y"]:
            i = i + 1

        begin = word[0:i]
        end = word[i:]

        pig_latin = end + begin + "AY"

    return pig_latin

# Test
word = "OMELET"
latin = pig_latin(word)
print("The pig-latin of",word,"is",latin,"!")
word = "STUPID"
latin = pig_latin(word)
print("The pig-latin of",word,"is",latin,"!")
```

Activity 3

Activity 3 strings_3.py

```
##############################
# Strings - Text analysis
##############################

##############################
# Activity 3 - DNA sequences
##############################

## Question 1 ##

def presence_of_A(sequence):
    """ Determine if the there is a nucleotid  A
    Input: a sequence of A,C,T,G (a string)
    Output: true or false """

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

    return False

# Test
sequence = "AGACAGCGAGCATATGCAGGAAG"
answer = presence_of_A(sequence)
print("Is there a 'A'  in the sequence",sequence," : ",answer)

## Question 2 ##

def position_of_AT(sequence):
    """ Determine the position of the code AT
    Input: a sequence of A,C,T,G (a string)
    Output: the position of this subsequence (start at 0) """

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

```
            return i  # If found
    return None  # If nowhere found

# Test
sequence = "GTGGTTTGACCTCCCATGGCCAT"
pos = position_of_AT(sequence)
print("In the sequence",sequence,"the code AT appears in position",pos)

## Question 3 ##

def position(code,sequence):
    """ Determine the position of the code in the given sequence
    Input: a code and a sequence of A,C,T,G (two strings)
    Output: the position of this code (start at 0) """

    for i in range(len(sequence)-len(code)):
        if code == sequence[i:i+len(code)]:
            return i  # If found, it's over

    return None  # If nowhere found

# Test
sequence = "GAAGACCTTCTCCTCCTGC"
code = "CCTC"
pos = position(code,sequence)
print("In the sequence",sequence,"the code',code,'appears in position",pos)

## Question 4 ##

def investigation():
    mustard = "CCTGGAGGGTGGCCCCACCGGCCGAGACAGCGAGCATATGCAGGAAGCGGCAGGAATAAGGAAAAGCAGC"
    scarlet = "CTCCTGATGCTCCTCGCTTGGTGGTTTGAGTGGACCTCCCAGGCCAGTGCCGGGCCCCTCATAGGAGAGG"
    peacock = "AAGCTCGGGAGGTGGCCAGGCGGCAGGAAGGCGCACCCCCCCAGTACTCCGCGCGCCGGGACAGAATGCC"
    plum    = "CTGCAGGAACTTCTTCTGGAAGTACTTCTCCTCCTGCAAATAAAACCTCACCCATGAATGCTCACGCAAG"

    code1 = "CATA"
    code2 = "ATGC"

    for suspect in [mustard,scarlet,peacock,plum]:
        print(position(code1,suspect))
        print(position(code2,suspect))

    return

# Investigation
print("--- Investigation ---")
investigation()
```

Activity 4

Activity 4 strings_4.py

```
##############################
# Strings - Text analysis
##############################

##############################
# Activity 4 - Uppercase/lowercase
```

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

sentence = "Python is c@@l"
code = [ord(c) for c in sentence]
print(code)

## Question 1 ##

# With the machine
code = [80, 121, 116, 104, 111, 110, 32, 105, 115, 32, 99, 64, 64, 108]
sentence = ""
for c in code:
    sentence = sentence + chr(c)
print(sentence)


## 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," gives ",eval(exp1))
print(exp2," gives ",eval(exp2))

## Question 4 ##

def upper_letter(charac):
    """ Change a lower case to uppercase
    Input: a lowercase character among "a",...,"z"
    Output: the same letter but in uppercase """

    order = ord(charac)
    new_order = order - 32
    new_car = chr(new_order)

    return new_car

# Test
print("The capital letter of 'a' is",upper_letter("a"))


## Question 5 ##

def uppercase(sentence):
    """ Transform a sentence to uppercase
    Input: a sentence (a string)
    Output: the same sentence in capital letters """

    new_sentence = ""
    for charac in sentence:
        order = ord(charac)
        if order >= 97 and order <= 122:
             # transformation to uppercase
            new_sentence = new_sentence + chr(order-32)
        else:
            # keep the character
            new_sentence = new_sentence + charac

    return new_sentence
```

```
# Test
sentence = "Hello world!"
print("The sentence",sentence,"becomes",uppercase(sentence))

# We will also need
def lowercase(sentence):
    """ Transform a sentence to lowercase
    Input: a sentence (a string)
    Output: the same sentence in lowercase """

    new_sentence = ""
    for charac in sentence:
        order = ord(charac)
        if order >= 65 and order <= 90:
             # transformation to lowercase
            new_sentence = new_sentence + chr(order+32)
        else:
            # keep the character
            new_sentence = new_sentence + charac

    return new_sentence

## Question 6 ##

def format_full_name(somebody):
    """ Transform some name to the style "First LAST"
    Input: the first ans last name of someone (separated by a space)
    Output: the formated full name "First LAST" """

    # We split first and last name
    first = ""
    last = ""
    in_first = True     # We are in the first name
    for charac in somebody:
        if in_first:
            first = first + charac
        else:
            last = last + charac
        if charac == " ":
            in_first =  False    # End of teh first name

    # Format of the first name
    new_first = uppercase(first[0])+lowercase(first[1:len(first)])

    # Format of last name
    new_last = uppercase(last)

    return new_first+new_last

# Test
somebody = "harry POTTER"
print(somebody,"devient",format_full_name(somebody))
somebody = "LORD Voldemort"
print(somebody,"devient",format_full_name(somebody))
```

Activity 5

Activity 5 strings_5.py

```
##############################
# Strings - Text analysis
##############################

import random # Just to create the enigma

##############################
# Activity 5 - Statistical analysis of a text
##############################


## Question 1 ##

def occurrences_letter(letter,sentence):
    """ Coutn the number of appearance of the given letter in the sentence
    Input: a letter, a sentence in uppercase
    Output: the number of this letter """

    nb = 0
    for charac in sentence:
        if charac == letter:
            nb = nb + 1

    return nb

# Test
sentence = "IS THERE ANYBODY OUT THERE"
print("The sentence",sentence,"contains",occurrences_letter("E",sentence),"times the letters
    ↪  E")


## Question 2 ##

def number_letters(sentence):
    """ Count the total number of letters
    Input: a sentence in uppercase
    Output: the total number of letters from "A" to "Z" """

    alphabet = list("ABCDEFGHIJKLMNOPQRSTUVWXYZ")

    nb = 0
    for charac in sentence:
        if charac in alphabet:
            nb = nb + 1

    return nb

# Test
sentence = "IS THERE ANYBODY OUT THERE"
print("The sentence",sentence,"contains",number_letters(sentence),"letters")


## Question 3 ##

def percentage_letter(letter,sentence):
    """ Calcule le ratio d'une letter donnée dans sentence
    Input: une letter et une sentence en majuscule
    Output: le percentage d'apparition de la letter """

    nb_letters = occurrences_letter(letter,sentence)
```

```
    nb_total = number_letters(sentence)
    percentage = nb_letters/nb_total*100

    return percentage

# Test
sentence = "IS THERE ANYBODY OUT THERE"
percentage = percentage_letter("E",sentence)
print("The sentence",sentence,"contains",percentage,"% of letter E")
print("Round percentage:","{0:.2f}".format(percentage))

## Question 4 ##

def display_frequency(sentence):
    """ Count and display the ration of all letters in the sentence
    Input: a sentence in uppercase
    Output: the display of percentage of letters appearance """

    alphabet = list("ABCDEFGHIJKLMNOPQRSTUVWXYZ")
    for letter in alphabet:
        percentage = percentage_letter(letter,sentence)
        print(letter," : ","{0:.2f}".format(percentage))

    return

# SECRET ------------------
# Creation of the enigma

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

#for word in sentence.split():
#  print(list(word))
#  print(myshuffle(list(word)))

# Le corbeau et le renard - Jean de la Fontaine
sentence1 = "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"

#sentence_mystere1 = ' '.join([''.join(myshuffle(list(word))) for word in sentence1.split()
    ↪ ])

# Le roi de aulnes - Goethe
sentence2 = "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
    ↪ "

#sentence_mystere2 = ' '.join([''.join(myshuffle(list(word))) for word in sentence2.split()
    ↪ ])

# Cent ans de solitude - Gabriel Garcia Marquez
```

```
sentence3 = "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"

sentence_mystere3 = ' '.join([''.join(myshuffle(list(word))) for word in sentence3.split()
    ↪ ])

# Sumertimes - Elle Fitzgerald
sentence4 = "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"

sentence_mystere4 = ' '.join([''.join(myshuffle(list(word))) for word in sentence4.split()
    ↪ ])


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

# Mystery sentences

sentence_mystery1 = "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(sentence_mystery1)

display_frequency(sentence_mystery1)

sentence_mystery2 = "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(sentence_mystery2)

display_frequency(sentence_mystery2)

sentence_mystery3 = "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(sentence_mystery3)

display_frequency(sentence_mystery3)

sentence_mystery4 = "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(sentence_mystery4)
display_frequency(sentence_mystere4)
```

7. Lists I

Activity 1

Activity 1 lists_I_1.py

```
################################
# Lists I
################################

################################
# Cours 1
mylist = [11,13,17,19]
mylist.append(23)
mylist.append(29)
print(mylist[5])
len(mylist)

################################
# Activity 1 - Interst
################################
## Question 1 ##
################################
def simple_interests(S0,p,n):
    mylist = [S0]
    interest = S0 * p/100
    S = S0
    for i in range(n):
        S = S + interest
        mylist.append(S)
    return mylist

# Test
print("--- Simple interests ---")
list_simple_interests = simple_interests(1000,10,12)
print(list_simple_interests)
print(list_simple_interests[11])


## Question 2 ##
################################
def compound_interests(S0,p,n):
    mylist = [S0]
    S = S0
    for i in range(n):
        interest = S * p/100
        S = S + interest
        mylist.append(S)
```

```
    return mylist

# Test
print("--- Compound interests ---")
list_compound_interests = compound_interests(1000,7,12)
print(list_compound_interests)
print(list_compound_interests[11])
```

Activity 2

Activity 2 lists_I_2.py

```
################################
# Lists I
################################

################################
# Cours 1
mylist = [11,13,17,19]
mylist.append(23)
mylist.append(29)
print(mylist[5])
len(mylist)

################################
# Activity 1 - Interst
################################
## Question 1 ##
################################
def simple_interests(S0,p,n):
    mylist = [S0]
    interest = S0 * p/100
    S = S0
    for i in range(n):
        S = S + interest
        mylist.append(S)
    return mylist

# Test
print("--- Simple interests ---")
list_simple_interests = simple_interests(1000,10,12)
print(list_simple_interests)
print(list_simple_interests[11])

## Question 2 ##
################################
def compound_interests(S0,p,n):
    mylist = [S0]
    S = S0
    for i in range(n):
        interest = S * p/100
        S = S + interest
        mylist.append(S)
```

```
    return mylist

# Test
print("--- Compound interests ---")
list_compound_interests = compound_interests(1000,7,12)
print(list_compound_interests)
print(list_compound_interests[11])
```

Activity 3

Activity 3 lists_I_3.py

```
################################
# Lists I
################################

################################
# Cours 1
mylist = [11,13,17,19]
mylist.append(23)
mylist.append(29)
print(mylist[5])
len(mylist)

################################
# Activity 1 - Interst
################################
## Question 1 ##
################################
def simple_interests(S0,p,n):
    mylist = [S0]
    interest = S0 * p/100
    S = S0
    for i in range(n):
        S = S + interest
        mylist.append(S)
    return mylist

# Test
print("--- Simple interests ---")
list_simple_interests = simple_interests(1000,10,12)
print(list_simple_interests)
print(list_simple_interests[11])


## Question 2 ##
################################
def compound_interests(S0,p,n):
    mylist = [S0]
    S = S0
    for i in range(n):
        interest = S * p/100
        S = S + interest
        mylist.append(S)
```

```
    return mylist

# Test
print("--- Compound interests ---")
list_compound_interests = compound_interests(1000,7,12)
print(list_compound_interests)
print(list_compound_interests[11])
```

Activity 4

Activity 4 lists_I_4.py

```
################################
# Lists I
################################

################################
# Cours 1
mylist = [11,13,17,19]
mylist.append(23)
mylist.append(29)
print(mylist[5])
len(mylist)

################################
# Activity 1 - Interst
################################
## Question 1 ##
################################
def simple_interests(S0,p,n):
    mylist = [S0]
    interest = S0 * p/100
    S = S0
    for i in range(n):
        S = S + interest
        mylist.append(S)
    return mylist

# Test
print("--- Simple interests ---")
list_simple_interests = simple_interests(1000,10,12)
print(list_simple_interests)
print(list_simple_interests[11])

## Question 2 ##
################################
def compound_interests(S0,p,n):
    mylist = [S0]
    S = S0
    for i in range(n):
        interest = S * p/100
        S = S + interest
        mylist.append(S)
```

```
    return mylist

# Test
print("--- Compound interests ---")
list_compound_interests = compound_interests(1000,7,12)
print(list_compound_interests)
print(list_compound_interests[11])
```

8. Statistics – Data visualization

Activity 1

Activity 1 statistics_1.py

```
##############################
# Statistics -- Data visualization -- tkinter
##############################

##############################
# Activity 1 - Statistics
##############################

from math import *

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

def mysum(mylist):
    """ Compute the um of elements
    Input: a list of numbers
    Output: their sum """

    S = 0
    for x in mylist:
        S = S + x
    return S

# Test
print("--- Sum ---")
mylist = [5,18,6,3]
print(mylist)
print(mysum(mylist))
print(sum(mylist))

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

def mean(mylist):
    """ Compute the average of the items of the list
    Input: a list of numbers
    Output: the average """

    nblist = len(mylist)

    if nblist == 0:
        M = 0
    else:
        M = mysum(mylist) / nblist
```

```
    return M

# Test
print("--- Mean ---")
mylist = [5,18,6,3]
print(mylist)
print(mean(mylist))


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

def minimum(mylist):
    """ Return the minimum among the elements
    Input: a list of numbers
    Output: their minimum """

    if len(mylist) == 0:
        return None

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

    return mini

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

##################################################
def maximum(mylist):
    """ Return the maximum among the elements
    Input: a list of numbers
    Output: their maximum """

    if len(mylist) == 0:
        return None

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

    return maxi

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


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

def variance(mylist):
    """ Return the variance of the elements
```

```
    Input: a list of numbers
    Output: their variance """

    if len(mylist) == 0:
        return 0

    M = mean(mylist)

    sum_square = 0
    for x in mylist:
        sum_square = sum_square + (x-M)**2

    var = sum_square / len(mylist)

    return var

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

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

def standard_deviation(mylist):
    """ Return the standard deviation of the elements
    Input: a list of numbers
    Output: their standard deviation """

    eca = sqrt(variance(mylist))

    return eca

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

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

temp_london = [4.9,5,7.2,9.7,13.1,16.6,18.7,18.2,15.5,11.6,7.7,5.6 ]

temp_chicago = [-5 ,-2.7,2.8,9.2,15.2,20.7,23.5,22.6,18.4,12.1,4.8,-1.9]

#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(mean(temp_london))
print(mean(temp_chicago))
print(standard_deviation(temp_london))
print(standard_deviation(temp_chicago))
```

Activity 2

Activity 2 statistics_2.py

```
##############################
# Statistics -- Data visualization -- tkinter
##############################

##############################
# Activity 2 - Data visualization
##############################

##############################
## Question 0 ##

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

# tkinter window
root = Tk()

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

def one_color():
    """ Return a random color
    Input: nothing
    Output: a color """

    # Method 1 - Limited choice
    # colors = ["red","orange","yellow","green","cyan","blue","violet","purple"]
    # col = random.choice(colors)

    # Methode 2 - "Infinite" choice
    R = randint(0,255)
    V = randint(0,255)
    B = randint(0,255)
    col = '#%02x%02x%02x' % (R, V, B)

    return col


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

def bar_graphics(mylist):
    """Bars graphics with one bar for each element of the list """
    posx = 100
    for x in mylist:
        height = x * scale
        canvas.create_rectangle(posx,400,posx+10,400-height,fill="red")
        posx = posx + 30

    # Bonus: vertical axis on the left
    max_mylist = max(mylist)
    canvas.create_line(90,400,90,400-scale*max_mylist)
    for j in range(max_mylist+1):
        canvas.create_line(85,400-scale*j,90,400-scale*j)
        canvas.create_text(80,400-scale*j,text=str(j))

    return
```

```
# Test
# scale = 20
# mylist = [5,8,6,3,7,10,4]
# bar_graphics(mylist)
# root.mainloop()

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

def cumulative_graphics(mylist):
    """ Graphics with rectangles one above the others """
    bas = 500
    for x in mylist:
        height = x * scale
        canvas.create_rectangle(100,bas,200,bas-height,fill=one_color())
        bas = bas - height

    # Bonus: vertical axis on the left
    max_mylist = sum(mylist)
    canvas.create_line(90,500,90,500-scale*max_mylist)
    for j in range(0,max_mylist+1,5):
        canvas.create_line(85,500-scale*j,90,500-scale*j)
        canvas.create_text(80,500-scale*j,text=str(j))

    return

# Test
# scale = 5
# mylist = [5,8,6,3,7,10,4,12]
# cumulative_graphics(mylist)
# root.mainloop()

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

def percentage_graphics(mylist):
    """ Rectangular graphics divided by sub-recatngles """
    mysum = sum(mylist)
    posx = 100
    for x in mylist:
        largeur = x/mysum*100 * 5
        canvas.create_rectangle(posx,300,posx+largeur,200,fill=one_color())
        posx = posx + largeur

    # Bonus: horizontal axis below
    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
# scale = 5
# mylist = [5,8,6,3,7,10,4,12]
# percentage_graphics(mylist)
# root.mainloop()

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

def sector_graphics(mylist):
    """ Circular graphics divided by sectors """
    mysum = sum(mylist)
    start_angle = 0
```

```
    for x in mylist:
        angle = x/mysum*360
        canvas.create_arc(200,100,550,450,start=start_angle,extent=angle,style=PIESLICE,fill
    ↪ =one_color())
        start_angle = start_angle + angle
    return

# Test
# scale = 5
# mylist = [5,8,6,3,7,10,4,12]
# sector_graphics(mylist)
# root.mainloop()

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

# mylist = [randint(5,15) for i in range(10)]

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

def action_bouton1():
    global scale
    scale = 15
    canvas.delete("all")
    bar_graphics(mylist)
    return

def action_bouton2():
    global scale
    scale = 4
    canvas.delete("all")
    cumulative_graphics(mylist)
    return

def action_bouton3():
    canvas.delete("all")
    percentage_graphics(mylist)
    return

def action_bouton4():
    canvas.delete("all")
    sector_graphics(mylist)
    return

def new_list():
    """ Create a new random list of data """
    global mylist
    n = randint(3,10)
    mylist = [randint(1,20) for i in range(n)]
    canvas.delete("all")
    return

# Titre
root.title("Data visualization")

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

bouton_new = Button(root,text="New data", width=30, command=new_list)
bouton_new.pack(side=BOTTOM, padx=5, pady=20)

bouton1 = Button(root,text="Bar graphics", width=30, command=action_bouton1)
bouton1.pack(padx=5, pady=20)
```

```
bouton2 = Button(root,text="Cumulative graphics", width=30, command=action_bouton2)
bouton2.pack(padx=5, pady=20)

bouton3 = Button(root,text="Percentage graphics", width=30, command=action_bouton3)
bouton3.pack(padx=5, pady=20)

bouton4 = Button(root,text="Sector graphics", width=30, command=action_bouton4)
bouton4.pack(padx=5, pady=20)

root.mainloop()
```

Activity 3

Activity 3 statistics_3.py

```
##############################
# Statistics -- Data visualization -- tkinter
##############################

##############################
# Activity 3 - Statistics (bis)
##############################

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

from math import *
from random import *

def median(mylist):
    """ Compute the median of the elements
    Input: a list of numbers
    Output: their median """
    my_sorted_list = sorted(mylist)

    n = len(my_sorted_list)

    if n%2 == 0:   # n est pair
        rank_middle = n//2
        med = (my_sorted_list[rank_middle-1]+my_sorted_list[rank_middle]) / 2
    else:
        rank_middle = (n-1)//2
        med = my_sorted_list[rank_middle]

    return med

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


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

def grades_to_list(grade_count):
    """ Takes a number of gradess as input and returns the list of grades
    Input: a list that counts each grade
    Output: the list of all grades """
    mylist = []
```

```
    for i in range(len(grade_count)):
        nb = grade_count[i]
        mylist = mylist + [i]*nb

    return mylist

# Test
print("--- List from a grade count ---")
grade_count = [0,0,1,2,5,2,3,5,4,1,2]
# grade_count = [0,0,0,0,0,1,0,2,0,1,5,1,2,3,2,4,1,2,0,1,0]
# grade_count = [randint(1,5) for i in range(21)]
print(grade_count)
print(grades_to_list(grade_count))

def median_grades(grade_count):
    """ Calcule la médiane des grades
    Input : une mylist d'effectif de grades
    Output : la médiane """
    mylist = grades_to_list(grade_count)
    med = median(mylist)

    return med

# Test
print("--- Médiane des grades ---")
# grade_count = [0,0,0,0,0,1,0,2,0,1,5,1,2,3,2,4,1,2,0,1,0]
grade_count = [0,0,1,2,5,2,3,5,4,1,2]
print(grade_count)
print(median_grades(grade_count))


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

def quartiles(mylist):
    """ Compute tes quartiles of the list
    Input: a list of numbers
    Output: their three quartiles Q1, Q2=median, Q3 """
    med = median(mylist)

    my_sorted_list = sorted(mylist)
    n = len(my_sorted_list)
    rank_middle = n//2
    if n%2 == 0: # si n pair
        mylist_inf = mylist[:rank_middle]
        mylist_sup = mylist[rank_middle:]
    else: # n impair
        mylist_inf = mylist[:rank_middle+1]
        mylist_sup = mylist[rank_middle:]
    Q1 =  median(mylist_inf)
    Q3 =  median(mylist_sup)

    return Q1, med, Q3

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

##################################################
def quartiles_grades(grade_count):
```

```
    """ Compute the quartiles of the grades
    Input: a list of grade count
    Output: the quartiles """
    mylist = grades_to_list(grade_count)
    Q1,Q2,Q3 = quartiles(mylist)

    return Q1, Q2, Q3

# Test
print("--- Quartiles of the grades ---")
grade_count = [0,0,1,2,5,2,3,5,4,1,2]
print(grade_count)
print(quartiles_grades(grade_count))
```

Activity 4

Activity 4 statistics_4.py

```
##############################
# Statistics -- Data visualization -- tkinter
##############################

##############################
# Activity 4 - Data visualization (bis)
##############################

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

from statistics_3 import *

root = Tk()  # tkinter window

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

scale = 15  # Scale to enlarge picture

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

def box_plot(grade_count):
    """ Box plot! """

    # Bar graphics
    for i in range(len(grade_count)):  # i range from 0 to 10
        hauteur = grade_count[i] * scale
        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")

    # Vertical axis on the left
    max_count = max(grade_count)
    canvas.create_line(95,300,95,300-scale*max_count)
    for j in range(max_count+1):
        canvas.create_line(90,300-scale*j,95,300-scale*j)
        canvas.create_text(85,300-scale*j,text=str(j))

    # Data to a list of grades
    mylist = grades_to_list(grade_count)
```

```
    # Computes the quartiles & co
    mini = min(mylist)
    maxi = max(mylist)
    Q1,Q2,Q3 = quartiles(mylist)

    #  Diagram
    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
# grade_count = [0,0,0,0,0,1,0,2,0,1,5,1,2,3,2,4,1,2,0,1,0]
grade_count = [0,0,1,2,5,2,3,5,4,1,2]
box_plot(grade_count)
root.mainloop()
```

Activity 5

Activity 5 statistics_5.py

```
##############################
# Statistics -- Data visualization -- tkinter
##############################

##############################
# Activity 5 - Data visualization (bis)
##############################

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

from random import *

def index_stock_exchange(n):
    """ Simulate n days of market """
    val = 1000
    list_val = [val]
    for i in range(n-1):
        val = val + randint(-10,12)/3
        list_val = list_val + [val]

    return list_val

# Test
print(index_stock_exchange(100))

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

def graphic_point(mylist):
    """ Display the curve of the indices of the stock exchange """

    # Base 1000 at y = 300
    canvas.create_line(100,300,100+365,300,width=3)

    # Vertical axis on the left
```

```
    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))

    # One point per day
    for i in range(len(mylist)):
        canvas.create_rectangle(100+i,300+1000-mylist[i],100+i+1,300+1000-mylist[i],outline=
    ↪ "red")

    return

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

# Test
mylist = index_stock_exchange(365)
#graphic_point(mylist)
#root.mainloop()

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

def moving_average(mylist,duration):
    """ Compute the moving average """
    mov_av = []
    for i in range(len(mylist)-duration+1):
        moy = sum(mylist[i:i+duration])/duration
        mov_av = mov_av + [moy]

    return mov_av

# Test
mylist = [1,2,3,4,5,6]
print(mylist)
print(moving_average(mylist,2))

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

def graphic_moving_average(mylist):
    """ Display the moving averages at 7 and 30 days """
    # average 7 last days
    average_7 = moving_average(mylist,7)
    for i in range(len(average_7)):
        canvas.create_rectangle(100+i+6,300+1000-average_7[i],100+i+6+1,300+1000-average_7[i
    ↪ ],outline="blue")

    # average 30 last days
    average_30 = moving_average(mylist,30)
    for i in range(len(average_30)):
        canvas.create_rectangle(100+i+29,300+1000-average_30[i],100+i+29+1,300+1000-
    ↪ average_30[i],outline="sienna")

    return

# Test
mylist = index_stock_exchange(365)
graphic_point(mylist)           # The index of the stock exchange
graphic_moving_average(mylist)  # The moving average at 7 and 30 days
```

```
root.mainloop()
```

9. Files

Activity 1

Activity 1 files_1.py

```
##############################
# Files
##############################

from random import *

##############################
# Activity 1 - Write to a file
##############################

## Question 1 ##

def write_file_grades():

    # Creation of a file to write
    filename = "grades.txt"
    fi = open(filename,"w")

    # Lists names
    list_firstnames = ["Lara","Robin","Hermione","Bill","Alice","James","Tintin"]
    list_names = ["Skywalker","Croft","Voldemort","Vador","Bond","Parker"]

    for i in range(6):
        firstname = choice(list_firstnames)
        name = choice(list_names)
        grade = str(randint(10,40)/2) + " " + str(randint(10,40)/2) + " " + str(randint
    ↪ (10,40)/2)
        line = firstname + " " + name + " " + grade + "\n"

        # Write line
        fi.write(line)

    # Close file
    fi.close()

    return

# Test

print("--- File 'grades.txt' ---")
write_file_grades()

## Question 2 ##

def write_file_averages():

    # File to read
    file_grades = "grades.txt"
    fi_in = open(file_grades,"r")

    # File to write
    file_averages = "averages.txt"
    fi_out = open(file_averages,"w")
```

```
    for line in fi_in:
        mylist = line.split()
        average = (float(mylist[2])+float(mylist[3])+float(mylist[4]))/3
        average_str = '{0:.2f}'.format(average)
        new_line = mylist[0] + " " + mylist[1] + " " + average_str + "\n"
        fi_out.write(new_line)

    # Fermeture des fichiers
    fi_in.close()
    fi_out.close()
    return

print("--- File 'averages.txt' ---")
write_file_averages()
```

Activity 2

Activity 2 files_2.py

```
################################
# Files
################################

from random import *
import matplotlib.pyplot as plt

################################
# Activity 2 -
################################

## Question 1 ##

def write_file_sales():

    # Create a file to write
    filename = "sales.csv"
    fi = open(filename,"w")

    # List of products nom
    list_products = ["Mountain bike","Surfboard","Running shoes","Badminton racket","Volley
    ↪ ball"]

    # Top lines
    fi.write("Best sales of the brand 'Pentathlon'\n\n")
    fi.write(",2015,2016,2017,2018,2019,2020\n\n")

    for product in list_products:

        # Generate random sales
        sales = ""
        for i in range(6):
            sales = sales + "," + str(randint(50,100)*10)

        line = product + sales  + "\n"

        # Write
        fi.write(line)

    # Close file
    fi.close()

    return
```

```
print("--- Files 'sales.csv' ---")
write_file_sales()

## Question 2 ##

def display_sales():

    # File to read
    file_sales = "sales.csv"
    fi_in = open(file_sales,"r")

    num_line = 0
    for line in fi_in:
        if num_line > 3: # we forget the top lines
            mylist = line.split(",")
            data = [float(x) for x in mylist[1:]]
            plt.plot(data)

        num_line += 1

    # Close file
    fi_in.close()

    # Display
    plt.grid()
    plt.show()

    return

print("--- Display 'sales.csv' ---")
display_sales()
```

Activity 3

Activity 3 files_3.py

```
################################
# Files
################################

import os

################################
# Activity 3 - Images
################################

## Question 1 ##

def write_image_bw():

    # Create a file to write
    filename = "image_bw.pbm"
    fi = open(filename,"w")

    # Header
    fi.write("P1\n")  # Black and white image
    nb_col = 300
    nb_lin = 200
    fi.write(str(nb_col) + " " + str(nb_lin) + "\n")

    for i in range(nb_lin):
        line = ""
```

```
        for j in range(nb_col):
            col = (i+j)//10 % 2
            line = line + str(col) + " "
        line = line + "\n"

        # Write the line
        fi.write(line)

    # Close file
    fi.close()

    return
# Test

print("--- File 'image_bw.pbm' ---")
write_image_bw()

## Question 2 ##

def write_image_gray():

    # Create a file to write
    filename = "image_gray.pgm"
    fi = open(filename,"w")

    # Header
    fi.write("P2\n")  # Grayscale image
    nb_col = 200
    nb_lin = 200
    fi.write(str(nb_col) + " " + str(nb_lin) + "\n")
    levels = 255
    fi.write(str(levels) + "\n")

    for i in range(nb_lin):
        line = ""
        for j in range(nb_col):
            col = (i**2 + j**2) % 256   # a level of gray: a function of i and j
            line = line + str(col) + " "
        line = line + "\n"

        # Write line
        fi.write(line)

    # Close file
    fi.close()

    return

# Test

print("--- File 'image_gray.pgm' ---")
write_image_gray()

## Question 3 ##

def ecrire_fichier_image_col():

    # Create a file to write
    filename = "image_col.ppm"
    fic = open(filename,"w")

    # Header
    fic.write("P3\n")  # Color image
    nb_col = 200
    nb_lin = 200
    fic.write(str(nb_col) + " " + str(nb_lin) + "\n")
    levels = 255
```

```
    fic.write(str(levels) + "\n")

    for i in range(nb_lin):
        line = ""
        for j in range(nb_col):
            R = (i*j) % 256      # red level
            G = i % 256          # green level
            B = (i+j)//3 % 256   # blue level

            line = line + str(R) + " " + str(G) + " "  + str(B) + "   "
        line = line + "\n"

        # Write line
        fic.write(line)

    # Close file
    fic.close()

    return

# Test

print("--- File 'image_col.ppm' ---")
ecrire_fichier_image_col()


## Question 4 ##

def inverse_black_white(filename):

    # Input file
    fi_in = open(filename,"r")

    # Output file
    name, extension = os.path.splitext(filename)
    new_name = name + "_inverse" + extension
    fi_out = open(new_name,"w")

    i = 0    # Line number
    for line in fi_in:

        if i<2:     # Keep first 2 lines
            fi_out.write(line)
        else:
            mylist = line.split()
            new_line = ""
            for l in mylist:
                if l == "1":
                    new_line = new_line + "0 "
                else:
                    new_line = new_line + "1 "

            new_line = new_line + "\n"
            fi_out.write(new_line)

        i = i + 1

    # Close all files
    fi_in.close()
    fi_out.close()
    return

print("--- Inverse black and white ---")
inverse_black_white("simple_bw.pbm")

## Question 4 ##
```

```
def formula_color_to_gray(R,G,B):
    gray = round(0.21*R + 0.72*G + 0.07*R)
    return gray

def color_to_gray(filename):

    # Input file
    fi_in = open(filename,"r")

    # Output
    name, extension = os.path.splitext(filename)
    new_name = name + "_gray" + ".pgm"
    fi_out = open(new_name,"w")

    i = 0    # Line number
    for line in fi_in:
        if i == 0:
            fi_out.write("P2\n")  # Grayscale image
        elif i == 1 or i == 2:    # Keep line 2 and 3
            fi_out.write(line)
        else:
            mylist = line.split()
            new_line = ""

            j = 0  # Column number
            while j < len(mylist):
                R = int(mylist[j])
                G = int(mylist[j+1])
                B = int(mylist[j+2])
                gray = formula_color_to_gray(R,G,B)
                new_line = new_line + str(gray) + " "

                j = j + 3

            new_line = new_line + "\n"
            fi_out.write(new_line)

        i = i + 1

    # Close all files
    fi_in.close()
    fi_out.close()
    return

print("--- Color to grayscale ---")
color_to_gray("image_col.ppm")
```

Activity 4

Activity 4 files_4.py

```
################################
# Files
################################

from math import *
import os

################################
# Activity 4 - Distances between to town
```

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

from math import *

## Question 1 ##

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

## Question 2 ##

def file_distances_xy(filename):
    # Input file
    fi_in = open(filename,"r")

    list_cities = []

    for line in fi_in:
        list_cities = list_cities + [line.split()]

    # Close input file
    fi_in.close()

    # Files à écrire
    name, extension = os.path.splitext(filename)
    new_name = name + "_distances" + ".txt"
    fi_out = open(new_name,"w")

    line = '{:>10s}'.format("")
    for c in list_cities:
        name = c[0]
        line = line + '{:>10s}'.format(name) +" "

    fi_out.write(line + "\n")

    for c1 in list_cities:
        name1 = c1[0]
        x1 = float(c1[1])
        y1 = float(c1[2])

        line = '{:10s}'.format(name1)

        for c2 in list_cities:
            name2 = c2[0]
            x2 = float(c2[1])
            y2 = float(c2[2])

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

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

        fi_out.write(line + "\n")

    return

print("--- Cities xy ---")
file_distances_xy("cities_xy.txt")


## Question 3 ##

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

def distance_approx_lat_long(lat1,long1,lat2,long2):
    R = 6371  # radius (average) of Earth in km
    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)  # (lat,long) in degrees
Paris_radians = (degrees_to_radians(Paris[0]),degrees_to_radians(Paris[1]))

New_York = (40.713,-74.006)  # (lat,long) in degrees
New_York_radians = (degrees_to_radians(New_York[0]),degrees_to_radians(New_York[1]))

print("--- Approximate distances Earth ---")
d = distance_approx_lat_long(*Paris_radians,*New_York_radians)
print(d)

def distance_lat_long(lat1,long1,lat2,long2):
    R = 6371  # radius (average) of Earth in km
    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("--- Exact distances Earth ---")
d = distance_lat_long(*Paris_radians,*New_York_radians)
print(d)

## Question 3 ##

def file_distances_lat_long(filename):
    # Input file
    fi_in = open(filename,"r")

    list_cities = []

    for line in fi_in:
        list_cities = list_cities + [line.split()]

    # Close input file
    fi_in.close()

    # Output file
    name, extension = os.path.splitext(filename)
    new_name = name + "_distances" + ".txt"
    fi_out = open(new_name,"w")

    line = '{:>12s}'.format("")
    for v in list_cities:
        name = v[0]
        line = line + '{:>12s}'.format(name) +" "

    fi_out.write(line + "\n")

    for c1 in list_cities:
        name1 = c1[0]
        lat1 = degrees_to_radians(float(c1[1]))
        long1 = degrees_to_radians(float(c1[2]))

        line = '{:12s}'.format(name1)

        for c2 in list_cities:
            name2 = c2[0]
            lat2 = degrees_to_radians(float(c2[1]))
            long2 = degrees_to_radians(float(c2[2]))

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

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

        fi_out.write(line + "\n")

    return
```

```
print("--- Cities lat_long ---")
file_distances_lat_long("cities_lat_long.txt")
```

10. Arithmetic – While loop – II

Activity 1

Activity 1 while_4.py

```
##############################
# While - Boolean - Arithmetic
##############################

##############################
# Activity 4 - Goldbach conjecture(s)
##############################
from math import *

##############################
# From activity 2
def is_prime(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 ##
# Goldbach's (good) conjecture (1742):
# any even integer greater than 3 is the sum of two prime numbers
def number_solutions_goldbach(n):
    """ Compute the number of decomposition n = p + q with
    n even ; p, q primes and q >= p """

    # If n odd, it's over
    if n % 2 == 1:
        print("Attention! Integer not even.")
        return None

    nb_sol = 0

    for p in range(2,n//2+1):
        q = n - p
        if (q>=p) and (is_prime(p)) and (is_prime(q)):
            print("n =",n,"sum of p =",p,", q = ",q)
```

```
            nb_sol = nb_sol + 1
    return nb_sol

# Test
print("--- Goldbach's conjecture  ---")
print(number_solutions_goldbach(100))

def test_goldbach_conjecture(nmax):
    """ Checks the validity of the Goldbach conjecture
    for even integers from 4 to nmax """
    print("Start of the test")
    for n in range(4,nmax,2):
        if number_solutions_goldbach(n) == 0:
            print("Problem with n = ",n)
    print("End of the test")
    return

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

##############################
## Question 2 ##
# Goldbach 1752 :
# every odd integer can be written as
# n = p + 2k^2
# with p prime and k integer (k maybe 0)
def is_decomposition_goldbach(n):
    """ Test if the odd n can be written n = p + 2k^2
    with p prime and k integer """
    maxk = ceil(sqrt(n/2))+1
    for k in range(maxk):
        p = n - 2 * k**2
        if is_prime(p):
            # print(n,p,k,n-p-2*k**2)
            return True
    return False

def counter_example_goldbach(nmax):
    """ Cherche un contre-exemple à la seconde conjecture de Goldbach """
    print("--- Start of the search ---")
    for m in range(1,nmax):
        n = 2 * m + 1
        if is_decomposition_goldbach(n) == False:
            print("Counter-example :",n)
    print("--- End of the search ---")
    return
# Test
print("--- Test wrong Goldbach's conjecture ---")
print("Avec 5777 : ",is_decomposition_goldbach(5777))
counter_example_goldbach(10000)
```

Activity 2

Activity 2 while_5.py

```
##############################
# While - Boolean - Arithmetic
##############################

##############################
# Activity 5 - Integers ayant 4 ou 8 divisors
##############################

# Conjecture: between 1 and N, there are more integers with (exactly)
# 4 divisors than integers with 8 divisors

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

def number_of_divisors_1(n):
    """ Number of divisors of n (1 and n are included) """
    nb = 0
    for d in range(1,n+1):
        if n % d == 0:
            nb = nb + 1
    return nb

def number_of_divisors_2(n):
    """ Number of divisors of n (1 and n are included) """
    nb = 2  # we already count 1 and n
    for d in range(2,n//2+1):
        if n % d == 0:
            nb = nb + 1
    return nb

# We choose the best method
def number_of_divisors(n):
    return number_of_divisors_2(n)

# Test
print("--- Number of divisors ---")
print(number_of_divisors(100))

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

## Question 2 ##

def four_and_eight_divisors(Nmin,Nmax):
    nb_four = 0
    nb_eight = 0
    for n in range(Nmin,Nmax):
        nb = number_of_divisors(n)
        if nb == 4:
            nb_four = nb_four + 1
        if nb == 8:
            nb_eight = nb_eight + 1
    return nb_four, nb_eight

# Test
print("--- Number having 4, then 8 divisors ---")
print(four_and_eight_divisors(1,100))
```

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

## Question 3 ##

# Search of a couter-example to the conjecture
# N must be large enough, for instance
# between 1 and N = 250000 there are more integers
# having 8 divisors than 4 divisors

# By slices of of 50 000 (computations are long!)

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

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

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

Activity 3

Activity 3 — while_6.py

```
##############################
# While - Boolean - Arithmetic
##############################

##############################
# Activity 6 - Wrong conjecture: 1211111... is never prime
##############################

##############################
# From activity 2

def is_prime(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 one_two_one(k):
```

```
    """ Compute an integer 121111... """
    u = 12
    for i in range(k):
        u = 10*u + 1
    return u

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


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

# Conjecture 1211111... is never prime

def test_prime_one_two_one(kmax):
    """ Test if 121111... is prime or not """
    for k in range(kmax):
        uk = one_two_one(k)
        print(k,"  ",uk,"is prime ?",is_prime(uk))
    return

# Test
print("--- Test conjecture 121111.... never prime ---")
test_prime_one_two_one(21)

# Will not yields to a counter-example
# Because computations are too long

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

def is_almost_prime(n,r):
    """ Test if n has no divisor <= 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 almost prime ---")
print(is_almost_prime(101,13))

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

## Question 4 ##

def test_almost_one_two_one(kmax):
    """ Test if 121111... is almost prime """

    n = 12
    for k in range(kmax):
        if is_almost_prime(n,1000000):
            print(k,"  ",n,'is almost prime')
```

```
        n = 10*n + 1
    return

# Test
print("--- Test conj 121111.... never almost prime ---")
test_almost_one_two_one(151)
```

11. Binary I

Activities binary_I.py

```
##############################
# Binary - part I
##############################

##############################
# Activity 1 - From decimal notation to integer
##############################

## Question 1 ##

def decimal_to_integer_1(list_decimal):
    number = 0
    p = len(list_decimal)
    for i in range(p):
        d = list_decimal[p-1-i]
        number = number + d*10**i

    return number

## Question 1bis ##

def decimal_to_integer_2(list_decimal):
    number = 0
    i = 0
    for d in reversed(list_decimal):
        number = number + d*10**i
        i = i + 1

    return number

# Test
print("--- Decimal notation to integer ---")
mylist = [1,2,3,4]
print(decimal_to_integer_1(mylist))
print(decimal_to_integer_2(mylist))

##############################
# Activity 2 - Binary notation to  integer
##############################


## Question 1 ##

def binary_to_integer_1(list_binary):
    number = 0
    p = len(list_binary)
    for i in range(p):
        if list_binary[p-1-i] == 1:
```

```
            number = number + 2**i

    return number

## Question 1bis ##

def binary_to_integer_2(list_binary):
    number = 0
    i = 0
    for b in reversed(list_binary):
        if b == 1:
            number = number + 2**i
        i = i + 1

    return number


## Question 2 ##

def binary_to_integer_bis(list_binary):
    number = 0
    for b in list_binary:
        if b == 0:
            number = number*2
        else:
            number = number*2 + 1

    return number


# Test
print("--- Binary notation to integer ---")
mylist = [1,1,0,1,1,0,0,1]
print(binary_to_integer_1(mylist))
print(binary_to_integer_2(mylist))
print(binary_to_integer_bis(mylist))


##############################
# Activity 3 - Decimal notation
##############################

def integer_to_decimal(n):
    if n==0: return [0]
    list_decimal = []
    while n != 0:
        list_decimal = [n%10] + list_decimal
        n = n//10
    return list_decimal

# Test
print("--- Integer to decimal notation ---")
n = 1234
mylist = integer_to_decimal(n)
integer = decimal_to_integer_1(mylist)
print(n)
print(mylist)
print(integer)


##############################
# Activity 4 - Binary notation
##############################
```

```
def integer_to_binary(n):
    if n==0: return [0]
    list_binary = []
    while n != 0:
        list_binary = [n%2] + list_binary
        n = n//2
    return list_binary

# Test
print("--- Integer to binary notation ---")
n = 1234
mylist = integer_to_binary(n)
integer = binary_to_integer_1(mylist)
print(n)
print(mylist)
print(integer)
```

12. Lists II

Activity 1

Activity 1 lists_II_1.py

```
################################
# Lists II
################################

################################
# Activity 1 - List comprehension
################################

## Question 1 ##

################################
def multiplication(list,k):
    return [k*x for x in list]

## Question 2 ##

################################
def power(list,k):
    return [x**k for x in list]

## Question 3 ##

################################
def addition(list1,list2):
    list_add = []
    for i in range(len(list1)):
        list_add.append(list1[i]+list2[i])
    return list_add

## Question 4 ##

################################
def non_zero(list):
    return [x for x in list if x != 0]
```

```
## Question 5 ##

##############################
def even(list):
    return [x for x in list if x % 2 == 0]

# Test
print("--- Multiplication ---")
print(multiplication([1,2,3,4,5],2))
print("--- Power ---")
print(power([1,2,3,4,5],3))
print("--- Addition ---")
print(addition([1,2,3],[4,5,6]))
print("--- Without zeros ---")
print(non_zero([1,0,2,3,0,4,5,0]))
print("--- Pairs ---")
print(even([1,0,2,3,0,4,5,0]))
```

Activity 2

Activity 2 lists_II_2.py

```
##############################
# Lists II
##############################

from random import *

##############################
# Activity 2 - Reach a sum
##############################

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

from random import *
list_20 = [randint(1,99) for i in range(20)]
list_20 = [16, 2, 85, 27, 9, 45, 98, 73, 12, 26, 46, 25, 26, 49, 18, 99, 10, 86, 7, 42]
print(list_20)

list_200 = [randint(1,99) for i in range(200)]
print(list_200)

## Question 1 ##

##############################
# Find two elements in a row so that their sum is 100

def sum_twoinarow_100(mylist):
    n = len(mylist)
    for i in range(n-1):
        if mylist[i]+mylist[i+1] == 100:
            # print(i,i+1,mylist[i],mylist[i+1])
            return True
    return False

## Question 2 ##
```

```
##############################
# Find two distcint elements whose sum is 100
def sum_two_100(mylist):
    n = len(mylist)
    for i in range(n-1):
        for j in range(i+1,n):
            if mylist[i]+mylist[j] == 100:
                # print(i,j,mylist[i],mylist[j])
                return True
    return False

## Question 3 ##

##############################
# Sequence of terms whos sum is 100
def sum_seq_100(mylist):
    n = len(mylist)
    for i in range(n):
        mysum = 0
        j = i
        while mysum < 100 and j < n:
            mysum = mysum  + mylist[j]
            j = j + 1
        if mysum == 100:
            # print(i,j-1,mylist[i:j])
            return True
    return False

##############################
print("--- Sum: two in a row ---")
print(sum_twoinarow_100(list_20))
print(sum_twoinarow_100(list_200))

print("--- Sum: two anywhere ---")
print(sum_two_100(list_20))
print(sum_two_100(list_200))

print("--- Sum: sequence ---")
print(sum_seq_100(list_20))
print(sum_seq_100(list_200))

## Question 3 ##

##############################
# Proba: which length n give proba > 1/2

def proba_1(n,N):
    nb = 0
    for k in range(N):
        mylist_n = mylist_n = [randint(1,99) for i in range(n)]
        found = sum_twoinarow_100(mylist_n)
        if found:
            nb += 1
    return nb/N

##############################
def proba_2(n,N):
    nb = 0
    for k in range(N):
        mylist_n = mylist_n = [randint(1,99) for i in range(n)]
        found = sum_two_100(mylist_n)
        if found:
```

```
            nb += 1
    return nb/N

##############################
def proba_3(n,N):
    nb = 0
    for k in range(N):
        mylist_n = mylist_n = [randint(1,99) for i in range(n)]
        found = sum_seq_100(mylist_n)
        if found:
            nb += 1
    return nb/N


print("--- Proba two in a row ---")
# Proba ~ 1/2 for length n ~ 67
print(proba_1(67,10000))

print("--- Proba two anywhere ---")
# Proba ~ 1/2 for length n ~ 12
print(proba_2(12,10000))

print("--- Proba sequence ---")
# Proba ~ 1/2 for length n ~ 42
print(proba_3(42,10000))
```

Activity 3

Activity 3 lists_II_3.py

```
##############################
# Lists II
##############################

##############################
# Activity 3 - Array
##############################

## Question 1 ##

##############################
def sum_diagonal(array):
    n = len(array)
    S = 0
    for i in range(n):
        S = S + array[i][i]
    return S

## Question 2 ##

##############################
def sum_antidiagonal(array):
    n = len(array)
    S = 0
    for i in range(n):
        S = S + array[n-1-i][i]
    return S
```

```
## Question 3 ##

##############################
def sum_all(array):
    n = len(array)
    S = 0
    for i in range(n):
        for j in range(n):
            S = S + array[i][i]
    return S

## Question 4 ##

##############################
def print_array(array):
    """
    Print a square on screen
    Input: an array of size n x n
    Output: nothing (display on terminal)
    """

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


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

print("--- Sum diagonale---")
print(sum_diagonal(array))

print("--- Sum anti-diagonal ---")
print(sum_antidiagonal(array))

print("--- Sum all ---")
print(sum_all(array))

print("--- Display ---")
print_array(array)
```

Activity 4

Activity 4 lists_II_4.py

```
##############################
# Lists II - Magic squares
##############################

from random import *

##############################
# From activity 3

def print_array(array):
    """
    Print a square on screen
```

```
    Input: an array of size n x n
    Output: nothing (display on terminal)
    """
    n = len(array)

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

    return

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

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

################################
# Activity 4 - Magic squares
################################

## Question 1 ##

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

print("--- Magic square ---")
# square = [ [1,2,3], [4,5,6], [7,8,9] ]
square_3x3 = [ [4,9,2], [3,5,7], [8,1,6] ]
square_4x4 = [ [1,14,15,4], [7,9,6,12], [10,8,11,5], [16,3,2,13] ]
print("--- Square 3x3 ---")
print_array(square_3x3)
print("--- Square 4x4 ---")
print_array(square_4x4)

## Question 2 ##

################################
def is_magic_square(square):
    n = len(square)
    total = n * (n**2 + 1) // 2

    if sum_diagonal(square) != total:
        return False

    if sum_antidiagonal(square) != total:
        return False

    for row in square:
        if sum(row) != total:
            return False

    for j in range(n):
        S = 0
```

```
        for i in range(n):
            S = S + square[i][j]
        if S != total:
            return False

    return True

print("--- Check magic square ---")
print(is_magic_square(square_3x3))
print(is_magic_square(square_4x4))

## Question 3 ##

##############################
def random_square(n):

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

print("--- Square aléatoire ---")
square = random_square(4)
print_array(square)
print(is_magic_square(square))

## Question 4 ##

##############################
def addition_square(square,k):

    n = len(square)
    new_square = [[0 for j in range(n)] for i in range(n)]

    for i in range(n):
        for j in range(n):
            new_square[i][j] = square[i][j] + k

    return new_square


## Question 4 ##

##############################
def multiplication_square(square,k):

    n = len(square)
    new_square = [[0 for j in range(n)] for i in range(n)]

    for i in range(n):
        for j in range(n):
            new_square[i][j] = square[i][j] * k

    return new_square

# Test
print("--- Addition, multiplication of magic square ---")
# square = [ [1,2,3], [4,5,6], [7,8,9] ]
square = [ [4,9,2], [3,5,7], [8,1,6] ]
sum_square = addition_square(square,-1)
product_square = multiplication_square(square,9)
print_array(square)
print_array(sum_square)
print_array(product_square)

## Question 5 ##
```

```
##############################
def homothety_square(square,k):

    n = len(square)
    new_square = [[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):
            new_square[i][j] = square[i//k][j//k]

    return new_square

# Test
print("--- Homothety magic square ---")
# square = [ [1,2,3], [4,5,6], [7,8,9] ]
# square = [ [4,9,2], [3,5,7], [8,1,6] ]
# big_square = homothety_square(square,3)
# print_array(big_square)

big_square = homothety_square(square_3x3,3)
print_array(big_square)

big_square = homothety_square(square_4x4,2)
print_array(big_square)

## Question 6 ##

##############################
def addition_block_square(big_square,small_square):

    N = len(big_square)
    n = len(small_square)
    # m = N//n

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

    for i in range(N):
        for j in range(N):
            new_square[i][j] = big_square[i][j] + small_square[i%n][j%n]

    return new_square

    # Test
print("--- Addition of blocks - Magic square ---")
small_square = [ [1,2], [3,4] ]
square = [ [1,2,3], [4,5,6], [7,8,9] ]
big_square = homothety_square(square,2)
new_big_square = addition_block_square(big_square,small_square)
print_array(small_square)
print("---")
print_array(big_square)
print("---")
print_array(new_big_square)

## Question 7 ##

##############################
def product_squares(square1,square2):
    n = len(square1)
    # m = len(square2)

    square3a = addition_square(square2,-1)
    # print("---")
    # print_array(square3a)
    square3b = homothety_square(square3a,n)
    # print("---")
```

```
    # print_array(square3b)
    square3c = multiplication_square(square3b,n**2)
    # print("---")
    # print_array(square3c)
    square3d = addition_block_square(square3c,square1)
    # print("---")
    # print_array(square3d)

    return square3d

#### Test ####
square1 = [ [4,9,2], [3,5,7], [8,1,6] ]
square2 = [ [4,14,15,1], [9,7,6,12], [5,11,10,8], [16,2,3,13] ]
square3 = product_squares(square1,square2)
print("--- Product of squares ---")
print_array(square1)
print("---")
print_array(square2)
print("---")
print_array(square3)
print(is_magic_square(square3))

#### Product doesn't commute (a*b is not b*a) ####
square4 = product_squares(square2,square1)
print("--- Product of squares ---")
print_array(square1)
print("---")
print_array(square2)
print("---")
print_array(square4)
print(is_magic_square(square4))

#### Square of size 36 x 36 ####
square5 = product_squares(square1,square4)
# print_array(square5)
print(is_magic_square(square5))
```

13. Binary II

Activities binary_II.py

```
################################
# Binary - part II
################################

from binary_I import *

################################
# Activity 1 - Palindrome en binaire
################################

## Question 1 ##

def is_palindrome_1(mylist):

    p = len(mylist)

    flag = True
```

```
    for i in range(p):
        if mylist[i] != mylist[p-1-i]:
            flag =  False

    return flag

# With optimization
def is_palindrome_1_bis(mylist):

    p = len(mylist)

    for i in range(p//2):
        if mylist[i] != mylist[p-1-i]:
            return False

    return True

def is_palindrome_2(mylist):
    mylist_inverse = list(reversed(mylist))
    return mylist == mylist_inverse

# Test
print("--- Test: palindrome ---")
mylist = [1,0,1,0,0,1,0,1]
print(is_palindrome_1(mylist))
print(is_palindrome_1_bis(mylist))
print(is_palindrome_2(mylist))


## Question 2 ##

def find_binary_palindrome(N):
    num = 0
    for n in range(N):
        list_binary = integer_to_binary(n)
        if is_palindrome_1(list_binary) == True:
            num = num + 1
            print(num,":",n,"=",integer_to_binary(n))
    return

# Test
print("--- Binary palindromes ---")
find_binary_palindrome(1000)

# The 1000th palindrome in binary notation is:
#249903 = [1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1]

## Question 3 ##

def find_decimal_palindrome(N):
    num = 0
    for n in range(N):
        list_decimal = integer_to_decimal(n)
        if is_palindrome_1(list_decimal) == True:
            num = num + 1
            print(num,":",n)
    return

# Test
print("--- Decimal palindromes ---")
find_decimal_palindrome(1000)

# The 1000th palindrome in decimal notation is:
# 90009
```

```
## Question 4 ##

def find_bi_palindrome(N):
    num = 0
    for n in range(N):
        list_binary = integer_to_binary(n)
        list_decimal = integer_to_decimal(n)
        if is_palindrome_1(list_binary) == True and is_palindrome_1(list_decimal):
            num = num + 1
            print(num,":",n,"=",integer_to_binary(n))
    return

# Test
print("--- Bi-palindromes ---")
find_bi_palindrome(1000)

# The 20th bi-palindrome is:
#  585585 = [1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1]


##############################
# Activity 2 - Opérations logiques
##############################

## Question 1 ##

def OReq(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 ANDeq(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 NOT(l1):
    l = []
    for b in l1:
        if b==1:
            l = l + [0]
        else:
            l = l + [1]
    return l

# Test
print("--- Logical operations (same length) ---")
l1 = [1,0,1,0,1,0,1]
l2 = [1,0,0,1,0,0,1]
print(l1)
print(l2)
```

```
print(OReq(l1,l2))
print(ANDeq(l1,l2))
print(NOT(l1))

## Question 2 ##

# Zero padding if necessary

def zero_padding(mylist,p):
    while len(mylist)< p:
        mylist = [0] + mylist

    return mylist

# Test
print("--- Zero padding ---")
print(zero_padding([1,0,1,1],8))

## Question 3 ##

# Logical operations, with lists of different sizes

def OR(l1,l2):
    p = len(l1)
    q = len(l2)
    if p>q:
        ll2 = zero_padding(l2,p)
        return OReq(l1,ll2)
    else:
        ll1 = zero_padding(l1,q)
        return OReq(ll1,l2)

def AND(l1,l2):
    p = len(l1)
    q = len(l2)
    if p>q:
        ll2 = zero_padding(l2,p)
        return ANDeq(l1,ll2)
    else:
        ll1 = zero_padding(l1,q)
        return ANDeq(ll1,l2)

# Test
print("--- Logical operation (general case) ---")
l1 = [1,0,1,0,1,0,1]
l2 = [1,0,0,1,0,]
print(l1)
print(l2)
print(OR(l1,l2))
print(AND(l1,l2))

##############################
# Activity 3 - Morgan law
##############################

## Question 1 ##

def every_binary_number_classical(p):

    list_all_p = []
    for n in range(2**p):
        list_all_p = list_all_p + [integer_to_binary(n)]
    return list_all_p
```

```
# Test
print("--- All binary numbers ---")
print(every_binary_number_classical(3))

## Question 2 ##

def every_binary_number(p):
    if p == 0:
      return []
    if p == 1:
      return [[0],[1]]

    list_all_p_1 = every_binary_number(p-1)

    list_all_p = [ [0] + l for l in list_all_p_1] + [ [1] + l for l in list_all_p_1]

    return list_all_p

# Test
print("--- All binary numbers ---")
print(every_binary_number(3))


## Question 3 ##

# Lois de Morgan

def test_morgan_law(p):
    list_all = [zero_padding(l,p) for l in every_binary_number(p)]
    #list_all = every_binary_number(p)
    for l1 in list_all:
        for l2 in list_all:
            not_l1_or_l2 = NOT(OR(l1,l2))
            not_l1_and_not_l2 = AND(NOT(l1),NOT(l2))
            if not_l1_or_l2 == not_l1_and_not_l2:
                print("It's true!")
                # pass
            else:
                print("It's false!",l1,l2)

    return

# Test
print("--- Test De Morgan law ---")
test_morgan_law(3)
```

14. Probabilities – Parrondo's paradox

Activities proba.py

```
##############################
# Probablities - Parrondo's paradox
##############################

# Reference : "Paradoxe de Parrondo", La gazette des mathématiciens, july 2017

from random import *
```

```
##############################
# Activity 1 - Game A : first losing game
##############################

## Question 1 ##
def throw_game_A():
    x = random()
    if x <= 0.49:
        return +1
    else:
        return -1

## Question 2 ##
def gain_game_A(N):
    gain = 0
    for i in range(N):
        gain = gain + throw_game_A()
    return gain

## Question 3 ##
def expected_value_game_A(N):
    expected_value = gain_game_A(N)/N
    return expected_value

# Test
print("--- Game A ---")
N = 1000000
print(expected_value_game_A(N))

##############################
# Activity 2 - Game B : second losing game
##############################

## Question 1 ##
def throw_game_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_game_B(N):
    gain = 0
    for i in range(N):
        gain = gain + throw_game_B(gain)
    return gain

## Question 3 ##
def expected_value_game_B(N):
    expected_value = gain_game_B(N)/N
    return expected_value
```

```
# Test
print("--- Game B ---")
N = 1000000
print(expected_value_game_B(N))

##############################
# Activity 3 -  Paradoxe de Parrondo
##############################

## Question 1 ##
def throw_game_AB(g):
    x = random()
    if x < 0.5:
        return throw_game_A()
    else:
        return throw_game_B(g)

## Question 2 ##
def gain_game_AB(N):
    gain = 0
    for i in range(N):
        gain = gain + throw_game_AB(gain)
    return gain

## Question 3 ##
def expected_value_game_AB(N):
    expected_value = gain_game_AB(N)/N
    return expected_value

# Test
print("--- Game AB ---")
N = 1000000
print(expected_value_game_AB(N))
```

15. Find and replace

Activity 1

Activity 1 find_1.py

```
##############################
# Find and replace
##############################

##############################
# Activity 1 - Find
##############################

## Question 1 ##

def find_in(string,substring):
    return substring in string

# Test
print("--- With 'in' ---")

string = "TO BE OR NOT TO BE"
```

```
substring = "NOT"
print(find_in(string,substring))

## Question 2 ##

def python_find(string,substring):
    position = string.find(substring)
    return position

# Test
print("--- With find() ---")

position = python_find(string,substring)
print(position)

position = python_find(string,"XYZ")
print(position)

## Question 3 ##

def find_index(string,substring):
    position = string.index(substring)
    return position

# Test
print("--- With index() ---")

position = find_index(string,substring)
print(position)

# position = find_index(string,"XYZ")
# print(position)

## Question 4 ##

def myfind(string,substring):
    len_string = len(string)
    len_substring = len(substring)
    for i in range(len_string-len_substring+1):
        found = True
        for j in range(len_substring):
            if string[i+j] != substring[j]:
                found = False
                break
        if found == True:
            return i
    return None

# Test

print("--- Your own function ---")

position = myfind(string,substring)
print(position)

position = myfind(string,"XYZ")
print(position)
```

Activity 2

Activity 2 find_2.py

```
##############################
# Find and replace
##############################

##############################
##############################
# From activity  1

def myfind(string,substring):
    len_string = len(string)
    len_substring = len(substring)
    for i in range(len_string-len_substring+1):
        found = True
        for j in range(len_substring):
            if string[i+j] != substring[j]:
                found = False
                break
        if found == True:
            return i
    return None

##############################
# Activity 2 - Replace
##############################

string = "TO BE OR NOT TO BE"
substring = "OR"
new_substring = "AND"

## Question 1 ##

print("--- With replace() ---")
new_string = string.replace(substring,new_substring)
print(new_string)

## Question 2 ##

# your own function replace using myfind()

def myreplace(string,substring,new_substring):
    pos = myfind(string,substring)

    if pos is not None:  # If found (short version "if not pos:")
        endpos = pos + len(substring)
        string = string[:pos]+new_substring+string[endpos:]

    return string

print("--- Replace: our function ---")
new_string = myreplace(string,substring,new_substring)
print(new_string)

## Question 3 ##

#
# replace_all() : replace all occurences

def replace_all(string,substring,new_substring):

    pos = myfind(string,substring)
```

```
    while pos is not None:  # as long as not found
        endpos = pos+len(substring)
        string = string[:pos]+new_substring+string[endpos:]
        pos = myfind(string,substring)

    return string

# Attention: this function is to simple because A -> AB will loop forever

print("--- Remplace all: our function ---")
string = "TO BE OR NOT TO BE"
substring = "BE"
new_substring = "HAVE"
new_string = replace_all(string,substring,new_substring)
print(new_string)
```

Activity 3

Activity 3 find_3.py

```
################################
# Find and replace
################################

################################
# Activity 3 - Regex - Regular expressions
################################

## Question 1 ##

from re import *

def python_regex_find(string,exp):
    pattern = search(exp,string)
    if pattern:
        return pattern.group(), pattern.start(), pattern.end()
    else:
        return None

# Test
print("--- With regex search() ---")
string = "TO BE OR NOT TO BE"
exp = "N.T"
print(python_regex_find(string,exp))

exp = "B..O"
print(python_regex_find(string,exp))

exp = "[NM]O"
print(python_regex_find(string,exp))

exp = "[BC]..O[RS]"
print(python_regex_find(string,exp))

## Question 2 ##

# The wildcard "."

def regex_find_wildcard(string,exp):
```

```
    len_string = len(string)
    len_exp = len(exp)

    for i in range(len_string-len_exp+1):
        found = True
        for j in range(len_exp):
            if exp[j] != "." and string[i+j] != exp[j]:
                found = False
                break
        if found == True:
            return string[i:i+len_exp],i,i+len_exp
    return None

# Test
print("--- our regex wildcard ---")
string = "TO BE OR NOT TO BE"
exp = "N.T"
print(regex_find_wildcard(string,exp))

## Question 3 ##

# Choice "[AB]", or [ABC]

def all_choice(exp):
    list_exp = [""]
    mode_choice = False
    for c in exp:
        if c == "[":
            mode_choice = True
            old_list_exp = list(list_exp)
            new_list_exp = []
        elif c == "]":
            mode_choice = False
            list_exp = new_list_exp
        else:
            if mode_choice == False:  # Normal mode
                list_exp = [ l + c for l in list_exp]
            else:  # Choice mode
                new_list_exp = new_list_exp + [ l + c for l in old_list_exp]

    return list_exp

# Test
print("--- our regex choices ---")
exp = "[AB]X[CDE]Y"
print(all_choice(exp))

def regex_find_choice(string,exp):
    list_exp = all_choice(exp)
    for mon_exp in list_exp:
        result = python_regex_find(string,mon_exp)
        if result is not None:
            return result

    return None

# Test
print("--- regex choice ---")
string = "TO BE OR NOT TO BE"
exp = "N[OPQ]T"
print(regex_find_choice(string,exp))
```

```
## Question 4 ##

# Negation [^A] (or [^AB])
```

Activity 4

Activity 4 find_4.py

```
##############################
# Find and replace
##############################

##############################
# From activity 1
##############################
def myfind(string,substring):
    len_string = len(string)
    len_substring = len(substring)
    for i in range(len_string-len_substring+1):
        found = True
        for j in range(len_substring):
            if string[i+j] != substring[j]:
                found = False
                break
        if found == True:
            return i
    return None

##############################
# From activity 2 - replace
##############################
def myreplace(string,substring,new_substring):
    pos = myfind(string,substring)

    if pos is not None:  # If found (short version "if not pos:")
        endpos = pos + len(substring)
        string = string[:pos]+new_substring+string[endpos:]

    return string

##############################
# Activity 4 - Itérations
##############################

## Question 1 ##

 # Test

print("--- One iteration ---")

print("-- Ex 1 --")
sentence = "01001110"
pattern = "01"
new_pattern = "10"
new_sentence = myreplace(sentence,pattern,new_pattern)
print(sentence)
print(new_sentence)
```

```
print("-- Ex 2 --")
sentence = "01001110"
pattern = "0011"
new_pattern = "1100"
new_sentence = myreplace(sentence,pattern,new_pattern)
print(sentence)
print(new_sentence)

print("-- Ex 3 --")
sentence = "01001110"
pattern = "0011"
new_pattern = "111000"
new_sentence = myreplace(sentence,pattern,new_pattern)
print(sentence)
print(new_sentence)

print("-- Ex 4 --")
sentence = "0001"
pattern = "01"
new_pattern = "1100"
print(sentence)
sentence = myreplace(sentence,pattern,new_pattern)
print(sentence)
sentence = myreplace(sentence,pattern,new_pattern)
print(sentence)
sentence = myreplace(sentence,pattern,new_pattern)
print(sentence)


## Question 2 ##

# Global constant for the maximum number of iterations
MAX_ITER = 1000

def iterations(sentence,pattern,new_pattern):
    i = 0
    while i <= MAX_ITER:
        new_sentence = myreplace(sentence,pattern,new_pattern)
        if sentence == new_sentence:
            return i, sentence
        else:
            sentence = new_sentence
            i = i+1
    return None

print("--- Iterations ---")
print("-- Ex 1 --")
sentence = "000011011"
pattern = "0011"
new_pattern = "1100"
result = iterations(sentence,pattern,new_pattern)
print(result)

sentence = "000011011"
print(sentence)
sentence = myreplace(sentence,pattern,new_pattern)
print(sentence)
sentence = myreplace(sentence,pattern,new_pattern)
print(sentence)
sentence = myreplace(sentence,pattern,new_pattern)
print(sentence)
```

```
sentence = myreplace(sentence,pattern,new_pattern)
print(sentence)
sentence = myreplace(sentence,pattern,new_pattern)
print(sentence)
sentence = myreplace(sentence,pattern,new_pattern)
print(sentence)

print("-- Ex 2 --")

sentence = "000011011"
pattern = "001"
new_pattern = "11000"
result = iterations(sentence,pattern,new_pattern)
print(result)

## For Question 3 ##

## binary with zero padding ##

def integer_to_binary(n,p):
    string_b = bin(n)  # Binary notation as a string
    string_b = string_b[2:]   # We remove the prefix

    # Padding with zero if necessary
    nb_zeros = p - len(string_b)
    for i in range(nb_zeros):
        string_b = "0" + string_b

    return string_b

# Test
print("--- Binary notation with zero-padding ---")
print(integer_to_binary(33,8))

## Question 3 ##

def max_iterations(p,pattern,new_pattern):

    maxi_iter = 0
    sentence_maxi_iter = ""
    new_sentence_maxi_iter = ""

    for n in range(2**p):
        sentence = integer_to_binary(n,p)
        result = iterations(sentence,pattern,new_pattern)
        #print(result)
        if result is None:
            return None, sentence
        else:
            nb_iter = result[0]
            if nb_iter > maxi_iter:
                maxi_iter = nb_iter
                sentence_maxi_iter = sentence
                new_sentence_maxi_iter = result[1]
    return maxi_iter, sentence_maxi_iter, new_sentence_maxi_iter

print("--- Maximal iterations ---")

# Exemple
pattern = "01"
new_pattern = "100"
print(max_iterations(4,pattern,new_pattern))

## Question 4 ##
```

```
# Lineair
pattern = "0011"
new_pattern = "110"
print("- Lineaire -")
print(max_iterations(10,pattern,new_pattern))

# Quadratic
pattern = "01"
new_pattern = "10"
print("- Quadratic -")
print(max_iterations(10,pattern,new_pattern))

# Exponential
pattern = "01"
new_pattern = "110"
print("- Exponential -")
print(max_iterations(10,pattern,new_pattern))

# Ne termine pas
pattern = "01"
new_pattern = "1100"
print("- No ending -")
print(max_iterations(4,pattern,new_pattern))
```

16. Polish calculator – Stacks

Activity 1

Activity 1 stacks_1.py

```
##############################
# Stacks - Polish calculator
##############################

##############################
# Activity 1 - Operations on the stack
##############################

# "stack" is a global variable

## Question 1 ##

def push_to_stack(element):
    """  Add an element at the top of the stack
    Input: an object
    Output: nothing
    Action: the stack contains one more element """

    global stack     # In order to modify the stack

    stack = stack + [element]

    return None

# Test
print("--- Push ---")
stack = [4,5,6]
print('Stack before:',stack)
```

```
push_to_stack(7)
print('Stack after:',stack)

## Question 2 ##

def pop_from_stack():
    """ Read the element at the top of stack et remove it
    Input: nothing
    Output: the element at the top
    Action:  the stack contains one element less """

    global stack

    top = stack[len(stack)-1]
    stack = stack[0:len(stack)-1]

    return top

# Test
print("--- Pop ---")
stack = [4,5,6]
print('Stack before:',stack)
val = pop_from_stack()
print('Popped value:',val,'\nStack after:',stack)

## Question 3 ##

def is_stack_empty():
    """ Test if the stack is empty or not
    Input: nothing
    Output: true/false
    Action: doesn't modify the stack """

    if len(stack) == 0:
        return True
    else:
        return False

# Tests
print("--- Test if stack empty ---")

# Test 1
stack = [4,5,6]
empty = is_stack_empty()
print(stack,'stack empty?',empty)

# Test 2
stack = []
empty = is_stack_empty()
print(stack,'stack empty?',empty)
```

Activity 2

Activity 2 stacks_2.py

```
##############################
# Stacks - Polish calculator
##############################

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

```
# From activity 1
##############################
def push_to_stack(element):
    global stack
    stack = stack + [element]
    return None

def pop_from_stack():
    global stack
    top = stack[len(stack)-1]
    stack = stack[0:len(stack)-1]
    return top

def is_stack_empty():
    if len(stack) == 0:
        return True
    else:
        return False

##############################
# Activity 2 - Stack manipulation
##############################

## Question 1 ##

print("--- Manipulation ---")
stack = []
push_to_stack(5)
push_to_stack(7)
push_to_stack(2)
push_to_stack(4)
print(stack)
pop_from_stack()
push_to_stack(8)
push_to_stack(1)
push_to_stack(3)
print(stack)
val =  pop_from_stack()
print('Value:',val)

## Question 2 ##

def is_in_stack(element):
    """  Test if the stack contains the element
    Input: nothing
    Output: true/false
    Action: modify the stack """

    while not is_stack_empty():
        el = pop_from_stack()
        if el == element:
            return True    # If found element it's ok

    return False     # End without finding element

# Tests
print("--- Test if stack contains 7 ---")

# Test 1
stack = [4,5,6]
print(stack,'stack contains 7?',is_in_stack(7))

# Test 2
stack = [4,7,12,99]
```

```
print(stack,'stack contains 7?',is_in_stack(7))

## Question 3 ##

def sum_stack():
    """ Compute the sum of the stack
    Input: nothing
    Output: the sum
    Action: the stack is now empty """

    mysum = 0
    while not is_stack_empty():
        element = pop_from_stack()
        mysum = mysum + element

    return mysum

# Test
print("--- Sum of the values of the stack ---")
stack = [4,5,6]
print('The sum of',stack,'is',sum_stack())

## Question 4 ##

def penultimate():
    """ Return the second last element at the bottom of the stack
    Input: nothing
    Output: the penultimate (=second last) element
    Action: the stack is now empty """

    last = None
    penultimate = None

    while not is_stack_empty():
        penultimate = last          # The last become the second last
        last = pop_from_stack()     # New last

    return penultimate

# Tests
stack = [4,5,6,13]
print('The second last element of',stack,'is',penultimate())

stack = [4,6]
print('The second last element of',stack,'is',penultimate())

stack = [6]
print('The second last element of',stack,'is',penultimate())

stack = []
print('The second last element of',stack,'is',penultimate())
```

Activity 3

Activity 3 stacks_3.py

```
################################
# Stacks - Polish calculator
################################
```

```
##############################
# From activity 1
##############################

def push_to_stack(element):
    global stack
    stack = stack + [element]
    return None

def pop_from_stack():
    global stack
    top = stack[len(stack)-1]
    stack = stack[0:len(stack)-1]
    return top

def is_stack_empty():
    if len(stack) == 0:
        return True
    else:
        return False

##############################
# Activity 3 - La gare de triage
##############################

def sort_wagons(train):
    """ Sort red/blue wagons of a train
    Input: a train with blue wagons (numbers) and red ones (letters)
    Output: the wagons sorted with blue before, then red
    Action: use a stack """

    global stack   # Should be global to be modified
    stack = []

    new_train = ""

    for wagon in train.split():
        if wagon.isdigit():   # Blue wagin directly to output station
            new_train = new_train + wagon + " "
        else:                 # Red wagon in waiting area
            push_to_stack(wagon)

    # Now all blue wagons are well placed
    # We deal with the red ones
    while not is_stack_empty():
        wagon = pop_from_stack()
        new_train = new_train + wagon + " "

    return new_train

# Tests
print("--- Red/blue sort ---")

train = "A 4 C 12"
sorted_train = sort_wagons(train)
print(train,' -> ',sorted_train)

train = "9 K 8 P 17 L B R 3 10 2 N"
sorted_train = sort_wagons(train)
print(train,' -> ',sorted_train)
```

Activity 4

Activity 4 stacks_4.py

```
##############################
# Stacks - Polish calculator
##############################

##############################
# From activity 1
##############################
def push_to_stack(element):
    global stack
    stack = stack + [element]
    return None

def pop_from_stack():
    global stack
    top = stack[len(stack)-1]
    stack = stack[0:len(stack)-1]
    return top

def is_stack_empty():
    if len(stack) == 0:
        return True
    else:
        return False

##############################
# Activity 4 - Polish calculator
##############################

## Question 1 ##

def operation(a,b,op):
    """ Compute the operation 'a + b 'ou 'a * b'...
    Input: a,b (numbers) and 'op' a character '+'' ou '*'
    Output: the result of the computation """

    if op == '+':
        return a + b
    if op == '*':
        return a * b

# Tests
print("--- Operations ---")
a=5 ; b=7
print("The sum of ",a,"and",b,"is",operation(a,b,'+'))
print("The product of",a,"and",b,"is",operation(a,b,'*'))

## Question 2 ##

def polish_calculator(expression):
    """ Evaluate an expression given in Polish notation
    Input: a Polish expression
    Output: the result of the computation
    Action: use a stack """

    global stack
    stack = []

    list_expression = expression.split()
```

```
    for charac in list_expression:
        if (charac == '+') or (charac == '*'):
            b = pop_from_stack()
            a = pop_from_stack()
            partial_result = operation(a,b,charac)
            push_to_stack(partial_result)
        else:
            val = int(charac)
            push_to_stack(val)

    return pop_from_stack()

# Tests
print("--- Polish calculator ---")

exp = "2 3 +"
print("The result of the expression",exp,"is",polish_calculator(exp))

exp = "2 3 + 5 *"
print("The result of the expression",exp,"is",polish_calculator(exp))

exp = "8 7 3 + *"
print("The result of the expression",exp,"is",polish_calculator(exp))

exp = "8 7 3 * +"
print("The result of the expression",exp,"is",polish_calculator(exp))
```

Activity 5

Activity 5 stacks_5.py

```
################################
# Stacks - Polish calculator
################################

################################
# From activity 1
################################

def push_to_stack(element):
    global stack
    stack = stack + [element]
    return None

def pop_from_stack():
    global stack
    top = stack[len(stack)-1]
    stack = stack[0:len(stack)-1]
    return top

def is_stack_empty():
    if len(stack) == 0:
        return True
    else:
        return False

################################
# Activity 5 - Brackets well balanced
################################
```

```
## Question 1 ##

def are_parentheses_balanced(expression):
    """ Test if an expression has correct parentheses
    Input: an expression (a string)
    Output: true/false
    Action: use a stack """

    global stack
    stack = []    # Start froman empty stack

    for charac in expression:
        if charac == "(":
            push_to_stack(charac)

        if charac == ")":
            if is_stack_empty():
                return False     # Problem: one "(" is missing
            else:
                pop_from_stack()

    # At the end:
    if is_stack_empty():
        return True
    else:
        return False

# Test
print("--- Well balanced parentheses ---")

expression = "(a+b)^2 = a^2 + (b^2+2(ab))"
print("The expression",expression,"has its parentheses correct?",are_parentheses_balanced(
    ↪ expression))

expression = "((a+b)^3 = (a+b)"
print("The expression",expression,"has its parentheses correct?",are_parentheses_balanced(
    ↪ expression))

expression = "(a+b)^4) = ((a+b)"
print("The expression",expression,"has its parentheses correct?",are_parentheses_balanced(
    ↪ expression))

## Question 2 ##

def are_brackets_balanced(expression):
    """ Test if an expression has correct square brackets and parentheses
    Input: an expression (a string)
    Output: true/false
    Action: use a stack """

    global stack
    stack = []    # Start from an empty stack

    for charac in expression:
        if charac == "(" or charac == "[":
            push_to_stack(charac)

        if charac == ")" or charac == "]":
            if is_stack_empty():
                return False     # Problem: one "(" or "[" is missing
            else:
                element = pop_from_stack()
                if element == "[" and charac == ")":
                    return False     # Problem of type [)
```

```
                if element == "(" and charac == "]":
                    return False     # Problem of type (]

    # Zt the end
    return is_stack_empty()

# Test
print("--- Expression with correct square brackets and parentheses ---")

expression = "(a+b)^2 = (a^2 + [b^2+[2(ab)]])"
print("The expression",expression,"has its parentheses and square brackets correct?",
    ↪ are_brackets_balanced(expression))

expression = "((a+b)]^3 = [a+b]"
print("The expression",expression,"has its parentheses and square brackets correct?",
    ↪ are_brackets_balanced(expression))

expression = "[a+b)^4] = (a+b)"
print("The expression",expression,"has its parentheses and square brackets correct?",
    ↪ are_brackets_balanced(expression))
```

Activity 6

Activity 6 stacks_6.py

```
##############################
# Stacks - Polish calculator
##############################

global stack

##############################
# From activity 1
##############################

def push_to_stack(element):
    global stack
    stack = stack + [element]
    return None

def pop_from_stack():
    global stack
    top = stack[len(stack)-1]
    stack = stack[0:len(stack)-1]
    return top

def is_stack_empty():
    if len(stack) == 0:
        return True
    else:
        return False

##############################
# From activity 4
##############################

def operation(a,b,op):
    if op == '+':
        return a + b
    if op == '*':
        return a * b
```

```
def polish_calculator(expression):
    """ Evaluate an expression given in Polish notation
    Input: a Polish expression
    Output: the result of the computation
    Action: use a stack """

    global stack
    stack = []

    list_expression = expression.split()

    for charac in list_expression:
        if (charac == '+') or (charac == '*'):
            b = pop_from_stack()
            a = pop_from_stack()
            partial_result = operation(a,b,charac)
            push_to_stack(partial_result)
        else:
            val = int(charac)
            push_to_stack(val)

    return pop_from_stack()


##############################
# Activity 6 - Conversion to Polish notation
##############################

def polish_notation(expression):
    """ Convert a classic expression to Polish notation
    Input: a classic expression
    Output: its Polish notation
    Action: use a stack """

    global stack
    stack = []

    list_expression = expression.split()

    polish = ""    # For the Polsih notation

    for charac in list_expression:
        if charac.isdigit():
            polish = polish + charac + " "

        if charac == "(":
            push_to_stack(charac)

        if charac == "*":
            push_to_stack(charac)

        if charac == "+":
            while not is_stack_empty():
                element = pop_from_stack()
                if element == "*":
                    polish = polish + element + " "
                else:
                    push_to_stack(element)    # Put back the element
                    break
            push_to_stack(charac)

        if charac == ")":
            while not is_stack_empty():
                element = pop_from_stack()
```

```
                if element == "(":
                    break
                else:
                    polish = polish + element + " "

    while not is_stack_empty():
        element = pop_from_stack()
        polish = polish + element + " "

    return polish

# Tests
print("--- Conversion to Polish notation ---")

exp = "2 + 3"
print("The expression",exp,"is written",polish_notation(exp))

exp = "2 * 3"
print("The expression",exp,"is written",polish_notation(exp))

exp = "( 2 + 3 ) * 4"
print("The expression",exp,"is written",polish_notation(exp))

exp = "4 * ( 2 + 3 )"
print("The expression",exp,"is written",polish_notation(exp))

exp = "2 + 4 * 5"
print("The expression",exp,"is written",polish_notation(exp))

exp = "2 * 4 * 5"
print("The expression",exp,"is written",polish_notation(exp))

exp = "( 2 + 3 ) * ( 4 + 8 )"
print("The expression",exp,"is written",polish_notation(exp))

##
# Automatic test and verifiacations

def test_polish(expression):
    classic = eval(expression)
    print("---\n",classic)
    conversion = polish_notation(expression)
    print(conversion)
    polish = polish_calculator(conversion)
    print(polish)
    return classic == polish

exp = "2 + 3"
print(exp, "OK ?",test_polish(exp))

exp = "2 * 3 * 7"
print(exp, "OK ?",test_polish(exp))

exp = "( 2 + 3 ) * ( 4 + 8 )"
print(exp, "OK ?",test_polish(exp))

exp = "( ( 2 + 3 ) * 11 ) * ( 4 + ( 8 + 5 ) )"
print(exp, "OK ?",test_polish(exp))

exp = "( 17 * ( 2 + 3 ) ) + ( 4 + ( 8 * 5 ) )"
print(exp, "OK ?",test_polish(exp))
```

17. Text viewer – Markdown

Activity 1

Activity 1 markdown_1.py

```
##############################
# Text viewer - Markdown
##############################

##############################
# Activity 1 - Afficher du text
##############################

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

from tkinter import *
from tkinter.font import Font

# tkinter window
root = Tk()

canvas = Canvas(root, width=800, height=600, background="white")
canvas.pack(fill="both", expand=True)

# Size of the text page
text_width = 700
text_height = 500

# Colors
background_color = "lightgray"
text_color = "black"

# Box for the text
canvas.create_rectangle(10,10,text_width,text_height,width=2,fill=background_color)

# Fonts
font_text = Font(family="Times", size=12)
font_italic = Font(family="Times", slant="italic", size=12)
font_bold = Font(family="Times", weight="bold", size=12)
font_title = Font(family="Times", weight="bold", size=20)
font_subtitle = Font(family="Times", weight="bold", size=16)

# Test
# canvas.create_text(100,100, text="Math is fun.",anchor=SW,font=font_title,fill=text_color)
# canvas.create_text(200,200, text="Python is cool!",anchor=SW,font=font_subtitle,fill="red
    ↪ ")
# root.mainloop()


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

def text_word(word,myfont):
    """ Put a word in a box
    Input: a string and a font
    Output: display the word and its bounding box """

    # Display the text
    word_canvas = canvas.create_text(100,100, text=word,anchor=SW,font=myfont,fill=
    ↪ text_color)
```

```
    # Coordinates of the rectangle (x1,y1,x2,y2)
    x1,y1,x2,y2 = canvas.bbox(word_canvas)

    # Display the bounding box
    canvas.create_rectangle(x1,y1,x2,y2,width=2)

    return

# Test
# text_word("Some text with a bounding box",font_title)
# root.mainloop()


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

def length_word(word,myfont):
    """ Length of a word
    Input: a string and a font
    Output: the length of the text in pixel """

    # 'Display' some invisible text in order to get its length
    word_canvas = canvas.create_text(100,100, text=word,anchor=SW,font=myfont,fill=
    ↪ background_color)

    # Recover extremities
    x1,y1,x2,y2 = canvas.bbox(word_canvas)

    return x2 - x1

# Test
print("Length of the word 'Hello' :",length_word("Hello",font_title),"pixels")
text_word("Hello",font_title)
root.mainloop()


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

def font_choice(mode,in_bold,in_italics):
    """ Return a font depending on the paramters
    Input: a mode (text, list, title, subtilt), bold or not, italic or not
    Output: the font """

    if mode == "title":
        myfont = font_title
    elif mode == "subtitle":
        myfont = font_subtitle
    else:       # Mode text ou liste
        if in_bold:
            myfont = font_bold
        elif in_italics:
            myfont = font_italic
        else:
            myfont = font_text

    return myfont

# Test
# myfont = font_choice("Text",False,True)
# canvas.create_text(100,100, text="This is italic",anchor=SW,font=myfont,fill=text_color)
# root.mainloop()

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

Activity 2

Activity 2 markdown_2.py

```
##############################
# Text viewer - Markdown
##############################

##############################
# Activity 2 - Afficher du markdown
##############################

from tkinter import *
from tkinter.font import Font

##############################
# From activity 1
##############################

# tkinter window
root = Tk()

canvas = Canvas(root, width=800, height=600, background="white")
canvas.pack(fill="both", expand=True)

# Size of the text page
text_width = 700
text_height = 500

# Colors
background_color = "lightgray"
text_color = "black"

# Box for the text
canvas.create_rectangle(10,10,text_width,text_height,width=2,fill=background_color)

# Fonts
font_text = Font(family="Times", size=12)
font_italic = Font(family="Times", slant="italic", size=12)
font_bold = Font(family="Times", weight="bold", size=12)
font_title = Font(family="Times", weight="bold", size=20)
font_subtitle = Font(family="Times", weight="bold", size=16)


def font_choice(mode,in_bold,in_italics):
    """ Return a font depending on the parameters
    Input: a mode (among: text, list, title, subtitle,...), bold or not, italic or not
    Output: the font """

    if mode == "title":
        myfont = font_title
    elif mode == "subtitle":
        myfont = font_subtitle
    else:       # Mode text ou liste
        if in_bold:
            myfont = font_bold
        elif in_italics:
            myfont = font_italic
        else:
            myfont = font_text

    return myfont
```

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

def print_line_v1(par,posy):
    """ Display some text on one line without any formating
    Input: a paragraph (i.e. a long line), the vertical position
    Output: display """

    posx = 20  # Start on the left of the window

    list_words = par.split()
    for word in list_words:

        word = word + " " # Add a space between words

        word_canvas = canvas.create_text(posx,posy, text=word,anchor=SW,font=font_title,fill
    ↪ =text_color)
        canvas.create_rectangle(canvas.bbox(word_canvas),width=2)

        # New x position for the next word
        posx = canvas.bbox(word_canvas)[2]

    return

# Tests
# print_line_v1("Hello, this is my first text!",100)
# root.mainloop()

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

def print_line_v2(par,posy):
    """ Print the text on one line with respect to some mode:title, subtitle, text, list
    Input: a paragraph (a long line, the vertical position
    Output: display """

    # Default mode: text without indent
    mode = "text"
    indentation = 20

    if par[0:2] == "##":       # Subtitle
        mode = "subtitle"
        par = par[2:]          # Delete the ##
    elif par[0] == "#":        # Title
        mode = "title"
        par = par[1:]          # Delete the #
    elif par[0] == "+":        # List
        mode = "list"
        par = u'\u2022' + par[1:]          # Replace the "+" by a bullet
        indentation = 40

    # Start the line (indent if list)
    posx = indentation

    list_words = par.split()
    for word in list_words:

        myfont = font_choice(mode,False,False)

        word = word + " "   # Add a space between words
        word_canvas = canvas.create_text(posx,posy, text=word,anchor=SW,font=myfont,fill=
    ↪ text_color)
        posx = canvas.bbox(word_canvas)[2]

    return
```

```
# Tests
print_line_v2("# Here is a title",80)
print_line_v2("## And here a subtitle",115)
print_line_v2("Normal text and, below, a list:",150)
print_line_v2("+ Apple",175)
print_line_v2("+ Banana",200)
print_line_v2("+ Cherry",225)
root.mainloop()

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

def print_line_v3(par,posy):
    """ Display text that maybe bold
    Input: a paragraph (a long line, the vertical position
    Output: display """

    mode = "text"
    indentation = 20

    if par[0:2] == "##":       # Subtitle
        mode = "subtitle"
        par = par[2:]          # Delete the ##
    elif par[0] == "#":        # Title
        mode = "title"
        par = par[1:]          # Delete the #
    elif par[0] == "+":        # List
        mode = "liste"
        par = u'\u2022' + par[1:]          # Replace the "+" by a bullet
        indentation = 40

    # Bold / not bold (default not bold, not italic)
    in_bold = False
    in_italics = False

    # Start of the line (with more indentation within a list)
    posx = indentation

    list_words = par.split()
    for word in list_words:

        if word == "**":  # Switch bold / not bold
            in_bold = not(in_bold)
            word = ""

        if in_bold:
            myfont = font_bold

        if word == "*":       # Switch italic / not italic
            in_italics = not(in_italics)
            word = ""

        myfont = font_choice(mode,in_bold,in_italics)

        if word != "":
            word = word + " " # Space between words
        word_canvas = canvas.create_text(posx,posy, text=word,anchor=SW,font=myfont,fill=
    ↪ text_color)
        posx = canvas.bbox(word_canvas)[2]

    return

# Tests
```

```
print_line_v3("These ** words are in bold ** but these * ones are in italics *",100)
print_line_v3("+ Apples and also ** bananas ** but * no cherries *",125)
root.mainloop()


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

# Interline
space_between_lines = 18

def print_paragraph(par,posy):
    """ Display text that maybe bold
    Input: a paragraph (a long line, the vertical position
    Output: display """

    mode = "text"
    indentation = 20

    if par[0:2] == "##":       # Subtitle
        mode = "subtitle"
        par = par[2:]          # Delete the ##
    elif par[0] == "#":        # Title
        mode = "title"
        par = par[1:]          # Delete the #
    elif par[0] == "+":        # List
        mode = "liste"
        par = u'\u2022' + par[1:]          # Replace the "+" by a bullet
        indentation = 40

    # Bold / not bold (default not bold, not italic)
    in_bold = False
    in_italics = False

    # Start of the line (with more indentation within a list)
    posx = indentation

    list_words = par.split()
    for word in list_words:

        if word == "**":  # Switch bold / not bold
            in_bold = not(in_bold)
            word = ""

        if in_bold:
            myfont = font_bold

        if word == "*":       # Switch italic / not italic
            in_italics = not(in_italics)
            word = ""

        myfont = font_choice(mode,in_bold,in_italics)

        if word != "":
            word = word + " " # Space between words
        word_canvas = canvas.create_text(posx,posy, text=word,anchor=SW,font=myfont,fill=
    ↪ text_color)
        posx = canvas.bbox(word_canvas)[2]

        if posx > text_width:
            posx = indentation
            posy = posy + space_between_lines

    return posy

# Tests
```

```
# print_paragraph("# The title",80)
# print_paragraph("## and its subtitle",115)
# print_paragraph("Hello world!"*5,150)
# print_paragraph("Text ** bold ** normal * italic * normal again "*2,175)
# print_paragraph("+ Apple",200)
# print_paragraph("+ Banana",225)
# print_paragraph("+ Cherry",250)

# long_text = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam nec dui ac
    ↪ sem molestie viverra quis sit amet felis. Donec felis mi, tempus in laoreet non,
    ↪ pellentesque non sem. Praesent pretium mi at odio congue eleifend. Integer magna
    ↪ neque, feugiat a commodo eget, malesuada in velit. Donec ac orci quis eros molestie
    ↪ lacinia. Sed nisi mi, pretium et tellus eget, dignissim venenatis felis. Mauris sit
    ↪ amet ex in metus ornare cursus non nec sapien."
# print_paragraph(long_text*2,400)

# root.mainloop()


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

def print_file(filename):
    """ Display the paragraphs from a text file
    Input: a filename of a text file with markdown syntax
    Output: display """

    # Open file
    fi = open(filename,"r")
    list_paragraphs = fi.readlines()
    fi.close()

    posy = 50

    # Display each paragraphs
    for par in list_paragraphs:
        newposy = print_paragraph(par,posy)
        posy = newposy + space_between_lines

    root.mainloop()

    return

# Tests
print_file("markdown1.md")
# print_file("markdown2.md")
```

Activity 3

Activity 3 markdown_3.py

```
##############################
# Text viewer - Markdown
##############################

##############################
# Activity 3 - Justify text
##############################
```

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

from random import randint

# list_lengths = [randint(5,15) for i in range(103)]
# list_lengths = [14, 3, 16, 9, 2, 11, 13, 5, 4, 19, 16, 6, 17, 16, 15, 5, 14, 12, 17, 7]
list_lengths = [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]

line_length = 100
space_length = 1

def justification_simple(list_len):
    """ Compute where to end the line for left alignment (without spaces)
    Input: a sequence of lengths (a list of integers)
    Output: the list of ranks where to cut the line """

    hyphen = [0]

    i = 1
    while i < len(list_len):
        mysum = list_len[i-1]

        while (i < len(list_len)) and (mysum <= line_length):
            mysum +=  list_len[i]
            i += 1

        if mysum > line_length:
            hyphen +=  [i-1]

    hyphen += [len(list_len)]

    return hyphen

################################################
def print_justification_simple():
    """ Test: print justification simple """

    print("\n--- Justification without spaces ---")
    print("Length of words :",list_lengths)

    hyphen = justification_simple(list_lengths)
    print("Justification:",hyphen)

    for i in range(len(hyphen)-1):
        line = list_lengths[hyphen[i]:hyphen[i+1]]
        mysum = sum(line)
        print("\nLine",i,":",line,"\nIndex",hyphen[i],"to",hyphen[i+1]-1,"= list_len[",
    ↪ hyphen[i],":",hyphen[i+1],"]","\nSum =",mysum,"Remainder =",line_length-mysum,)

    return

# Test
print_justification_simple()


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

def justification_spaces(list_len):
    """ Compute where to end the line for left alignment (wit spaces)
    Input: a sequence of lengths (a list of integers)
    Output: the list of ranks where to cut the line """

    hyphen = [0]
```

```
    i = 1

    while i < len(list_len):
        mysum = list_len[i-1]

        while (i < len(list_len)) and (mysum <= line_length):
            mysum +=  space_length + list_len[i]
            i += 1

        if mysum > line_length:
            hyphen +=  [i-1]

    hyphen += [len(list_len)]

    return hyphen


################################################
def print_justification_spaces():
    """ Test: print justifications with spaces """

    print("\n--- Justification with spaces ---")
    print("Length of words :",list_lengths)

    hyphen = justification_spaces(list_lengths)
    print("Justification:",hyphen)

    for i in range(len(hyphen)-1):
        line = list_lengths[hyphen[i]:hyphen[i+1]]
        nb_spaces = len(line)-1
        mysum =  sum(line) + nb_spaces*space_length
        print("\nLine",i,":",line,"\nIndex",hyphen[i],"to",hyphen[i+1]-1,"= list_len[",
    ↪ hyphen[i],":",hyphen[i+1],"]","\nSum with spaces =",mysum,"Remainder =",line_length-
    ↪ mysum,)

    return

# Test
print_justification_spaces()


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

def compute_space_lengths(list_len,hyphen):
    """ Compute the length of the spaces in order to justify the 'text'  (i.e. sum = 100)
    Input: a sequence of lengths (a list of integers) and the already computed
    ↪ justifications
    Output: the list of lengths for spaces for each line """

    list_space_lengths = []

    for i in range(len(hyphen)-2):

        line = list_len[hyphen[i]:hyphen[i+1] ]
        nb_spaces = len(line)-1
        mysum =  sum(line) + nb_spaces*space_length
        rest = line_length - mysum

        if nb_spaces > 0:
            new_space = space_length + rest / nb_spaces
        else:
            new_space = space_length

        list_space_lengths += [new_space]

    # Last line is not justified
```

```
    list_space_lengths += [space_length]

    return list_space_lengths


################################################
def print_space_lengths():
    """ Test: print the lengths of the spaces to get justification """

    print("\n--- Justification with spaces ---")
    print("Length of words :",list_lengths)

    hyphen = justification_spaces(list_lengths)
    print("Justification:",hyphen)

    list_space_lengths = compute_space_lengths(list_lengths,hyphen)
    print("Lengths of spaces for each line:",[float("{0:0.2f}".format(l)) for l in
    ↪ list_space_lengths])

    for i in range(len(hyphen)-1):
        line = list_lengths[hyphen[i]:hyphen[i+1]]
        nb_spaces = len(line) - 1
        mysum =  sum(line) + nb_spaces*list_space_lengths[i]

        print("\nLine",i,":",line,"\nIndex",hyphen[i],"to",hyphen[i+1]-1,"= list_len[",
    ↪ hyphen[i],":",hyphen[i+1],"]","\nSum with spaces =",mysum,"Remainder =",line_length-
    ↪ mysum,)
        print("Lengths of the spaces for this line",list_space_lengths[i])

    return

# Test
print_space_lengths()
```

18. L-systems

Activities lsystems.py

```
################################
# L-systems
################################

from turtle import *

################################
# Activity 1 - Draw a L-system
################################

def draw_lsystem(word,angle=90,scale=1):

    speed("fastest")
    width(2)
    color('blue')
    up()
    goto(-150,0)
    down()

    for c in word:
        if c == "A" or c == "B":
            forward(100*scale)
```

```
        if c == "l":
            left(angle)
        if c == "r":
            right(angle)

    exitonclick()

    return

## Test ##
# draw_lsystem("AlArAArArA")

##############################
# Activity 2 - Only one rule: Koch snowflake
##############################

# A L-system
# One starting word
# One rule of find-and-repalce

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

def replace_1(word,letter,pattern):
    new_word = ""
    for c in word:
        if c == letter:
            new_word = new_word + pattern
        else:
            new_word = new_word + c

    return new_word

## Test ##
print("--- Replace one letter ---")
word = "ArAAl"
new_word = replace_1(word,"A","Al")
print(word)
print(new_word)


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

def iterate_lsysteme_1(start,rule,k):
    word = start
    letter = rule[0]
    pattern = rule[1]

    for i in range(k):
        word = replace_1(word,letter,pattern)

    return word

print("--- Lesson 1: Replace one letter and iterat ---")
word = "BlArB"
rule = ("A","ABA")
for k in range(4):
    new_word = iterate_lsysteme_1(word,rule,k)
    print(new_word)

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

## Koch snowflake
```

```
start_Koch = "A"
rule_Koch = ("A","AlArArAlA")

## Test
print("--- Koch's snowflake ---")
for k in range(4):
    print(k,iterate_lsysteme_1(start_Koch,rule_Koch,k))

k = 3
word = iterate_lsysteme_1(start_Koch,rule_Koch,k)
# draw_lsystem(word,scale=5/3**k)

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

######################
## Other examples ##

######################
start = "ArArArA"
rule = ("A","ArAlAlAArArAlA")
k = 3
word = iterate_lsysteme_1(start,rule,k)
# draw_lsystem(word,scale=0.05)

######################
start = "ArArArA"
rule = ("A","AlAArAArArAlAlAArArAlAlAAlAArA")
k = 2
word = iterate_lsysteme_1(start,rule,k)
# draw_lsystem(word,scale=0.07)

######################
start = "ArArArA"
rule = ("A","AArArArArAA")
k = 3
word = iterate_lsysteme_1(start,rule,k)
# draw_lsystem(word,scale=0.1)

######################
start = "ArArArA"
rule = ("A","AArArrArA")
k = 3
word = iterate_lsysteme_1(start,rule,k)
# draw_lsystem(word,scale=0.1)

######################
start = "ArArArA"
rule = ("A","AArArArArArAlA")

k = 3
word = iterate_lsysteme_1(start,rule,k)
# draw_lsystem(word,scale=0.1)
######################
start = "ArArArA"
rule = ("A","AArAlArArAA")
k = 3
word = iterate_lsysteme_1(start,rule,k)
# draw_lsystem(word,scale=0.15)

######################
start = "ArArArA"
rule = ("A","ArAArrArA")
k = 3
```

```
word = iterate_lsysteme_1(start,rule,k)
# draw_lsystem(word,scale=0.15)

####################
start = "ArArArA"
rule = ("A","ArAlArArA")
k = 4
word = iterate_lsysteme_1(start,rule,k)
# draw_lsystem(word,scale=0.15)


##############################
# Activity 3 - Two rules: Sierpinski's triangle
##############################

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

def replace_2(word,letter1,pattern1,letter2,pattern2):
    new_word = ""
    for c in word:
        if c == letter1:
            new_word = new_word + pattern1
        elif c == letter2:
            new_word = new_word + pattern2
        else:
            new_word = new_word + c

    return new_word

## Test ##
print("--- Replace two letters ---")
word = "ArBlA"
new_word = replace_2(word,"A","ABl","B","Br")
print(word)
print("Good:",new_word)

word1 = replace_1(word,"A","ABl")
word2 = replace_1(word1,"B","Br")
print("Bad:",word2)

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

def iterate_lsysteme_2(start,rule1,rule2,k):
    word = start
    letter1 = rule1[0]
    pattern1 = rule1[1]
    letter2 = rule2[0]
    pattern2 = rule2[1]

    for i in range(k):
        word = replace_2(word,letter1,pattern1,letter2,pattern2)
    return word

print("--- Lesson 1: Replace two letter and iterate ---")
word = "A"
rule1 = ("A","BlA")
rule2 = ("B","BB")

for k in range(4):
    new_word = iterate_lsysteme_2(word,rule1,rule2,k)
    print(new_word)

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

```
## Question 3 ##

## Triangle de Sierpinski
start_Sierp = "ArBrB"
rule_Sierp_1 = ("A","ArBlAlBrA")
rule_Sierp_2 = ("B","BB")

## Test
print("--- Sierpinski ---")
for k in range(3):
    print(k,iterate_lsysteme_2(start_Sierp,rule_Sierp_1,rule_Sierp_2,k))

k = 4
word = iterate_lsysteme_2(start_Sierp,rule_Sierp_1,rule_Sierp_2,k)
# draw_lsystem(word,angle=-120,scale=5/2**k)

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

######################
## Other examples ##

######################
## Dragon's curve
start_dragon = "AX"
rule_dragon_1 = ("X","XlYAl")
rule_dragon_2 = ("Y","rAXrY")

k = 9
word = iterate_lsysteme_2(start_dragon,rule_dragon_1,rule_dragon_2,k)
# draw_lsystem(word,scale=2/k)

######################
## Variant Sierpinski (angle = 60)
start = "A"
rule1 = ("A","BrArB")
rule2 = ("B","AlBlA")
# angle = 60

k = 5
word = iterate_lsysteme_2(start,rule1,rule2,k)
# draw_lsystem(word,angle=60,scale=2/k**2)

######################
## Gosper's curve
start = "A"
rule1 = ("A","AlBllBrArrAArBl")
rule2 = ("B","rAlBBllBlArrArB")
k = 3
word = iterate_lsysteme_2(start,rule1,rule2,k)
# draw_lsystem(word,angle=60,scale=2/k**2)

##############################
# Activity 4 - Draw a L-system with a stack
##############################

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

def draw_lsystem_stack(word,angle=90,scale=1):

    speed("fastest")
    width(3)
    color('blue')
```

```
    up()
    goto(0,-300)
    down()

    stack = []

    for c in word:
        if c == "A" or c == "B":
            forward(100*scale)
        if c == "a":
            up()
            forward(100*scale)
            down()
        if c == "l":
            left(angle)
        if c == "r":
            right(angle)
        if c == "[":
            stack = stack + [(position(),heading())]
        if c == "]":
            up()
            pos,direc = stack.pop()
            goto(pos)
            setheading(direc)
            down()

    exitonclick()

    return

## Test
# draw_lsystem_stack("AaAlAA[lAAA][rAA]A",angle=90,scale=1)
# draw_lsystem_stack("AlA[lAAA]A[rAA]A",angle=90,scale=1)

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

# Plant
start_plant = "lllX"
rule_plant_1 = ("X","A[lX][X]A[lX]rAX")
rule_plant_2 = ("A","AA")

k = 5
word = iterate_lsysteme_2(start_plant,rule_plant_1,rule_plant_2,k)
# draw_lsystem_stack(word,angle=30,scale=1/k**(3/2))


######################
## Example with up-down ##
start = "ArArArA"
rule1 = ("A","AlarAAlAlAAlAalAAralAArArAArAarAAA")
rule2 = ("a","aaaaaa")
k = 2
word = iterate_lsysteme_2(start,rule1,rule2,k)
draw_lsystem_stack(word,scale=0.1)

######################
## Other examples of plants ##

# ######################
# angle = 22.5
start = "lllA"
rule = ("A","A[lA]A[rA][A]")
```

```
k = 4
word = iterate_lsysteme_1(start,rule,k)
# draw_lsystem_stack(word,angle=30,scale=0.2)

# ######################
# angle = 30
start = "lllA"
rule = ("A","A[lA]A[rA]A")
k = 4
word = iterate_lsysteme_1(start,rule,k)
# draw_lsystem_stack(word,angle=30,scale=0.075)

# ####################
# angle = 22.5
start = "lllA"
rule = ("A","AAr[rAlAlA]l[lArArA]")
k = 3
word = iterate_lsysteme_1(start,rule,k)
# draw_lsystem_stack(word,angle=30,scale=0.2)

# ######################
# angle = 25.7
start = "lllX"
rule1 = ("X","A[lX]A[rX]AX")
rule2 = ("A","AA")
k = 5
word = iterate_lsysteme_2(start,rule1,rule2,k)
# draw_lsystem_stack(word,angle=30,scale=0.07)

# ######################
# angle = 25.7
start = "lllX"
rule1 = ("X","A[lX][X]A[lX]rAX")
rule2 = ("A","AA")
k = 5
word = iterate_lsysteme_2(start,rule1,rule2,k)
# draw_lsystem_stack(word,angle=30,scale=0.07)

# ######################
# angle = 30
start = "lllA"
rule1 = ("A","A[rB][lB]")
rule2 = ("B","A[rB]A[lArB]")
k = 5
word = iterate_lsysteme_2(start,rule1,rule2,k)
# draw_lsystem_stack(word,angle=30,scale=0.25)

######################
# angle = 30
start = "lllX"
rule1 = ("X","Ar[[X]lX]lA[lAX]rX")
rule2 = ("A","AA")
k = 4
word = iterate_lsysteme_2(start,rule1,rule2,k)
# draw_lsystem_stack(word,angle=30,scale=0.15)


######################
######################
# Hilbert curve
# For the illustration of each part of the book
```

```
  # \rule{L -> +RF-LFL-FR+}
  # \rule{R -> -LF+RFR+FL-}}
# angle = 30
start = "X"
rule1 = ("X","lYArXAXrAYl")
rule2 = ("Y","rXAlYAYlAXr")
k = 4
word = iterate_lsysteme_2(start,rule1,rule2,k)
draw_lsystem_stack(word,angle=90,scale=0.15)
```

19. Dynamic images

Activities images.py

```
################################
# Dynamic images
################################
import os   # for images files

################################
# Activity 1 - Photo booth
################################


################################
## From other chpater ##
def print_array(array):
    n = len(array)
    m = len(array[0])
    for i in range(n):
        for j in range(m):
            print('{:>3d}'.format(array[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("--- Photo booth transformation ---")
print(transformation(1,1,6))

##############################
## Question 2 ##
def photo_booth(array):
    n = len(array)
    new_array = [[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)
            new_array[ii][jj] = array[i][j]

    return new_array

## Test ##
print("--- Transformation photo_booth ---")
array_before = [ [1,2,3,4], [5,6,7,8],[9,10,11,12], [13,14,15,16] ]
array_after = photo_booth(array_before)
print_array(array_before)
print("---")
print_array(array_after)

##############################
## Question 3 ##
def photo_booth_iterate(array,k):
    n = len(array)
    tab = [[array[i][j] for j in range(n)] for i in range(n)]

    for i in range(k):
        tab = photo_booth(tab)

    return tab

## Test ##
print("--- Iterate transformation photo_booth ---")
array = [ [1,2,3,4], [5,6,7,8],[9,10,11,12], [13,14,15,16] ]
print_array(array)
for k in range(1,10):
    array_iter = photo_booth_iterate(array,k)
    # Not very clever because start from the begining each time
    print("--- k =",k,"---")
    print_array(array_iter)


##############################
# Activity 2 - Conversion array/image
##############################

##############################
## Question 1 ##
def array_to_image(array,image_name):

    # New file to write
    filename = "output/" + image_name + ".pgm"
    fi = open(filename,"w")

    # Header
    fi.write("P2\n")  # Grayscale image

    nb_lin = len(array)
```

```
    nb_col = len(array[0])

    fi.write(str(nb_col) + " " + str(nb_lin) + "\n")
    levels = 255
    fi.write(str(levels) + "\n")

    for i in range(nb_lin):
        line = ""
        for j in range(nb_col):
            color = array[i][j]
            line = line + str(color) + " "
        line = line + "\n"

        # WWrite line to file
        fi.write(line)

    # Clos file
    fi.close()

    return

## Test ##
print("--- Array to image ---")
array =  [[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]]
array_to_image(array,"test")

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

def image_to_array(image_name):

    # New file to write
    filename = "input/" + image_name + ".pgm"
    fi = open(filename,"r")

    i = 0    # Line number
    for line in fi:
        if i == 1:     # Keep first 2 lines
            list_line = line.split()
            nb_col = int(list_line[0])
            nb_lin = int(list_line[1])

            array = [[ 0 for j in range(nb_col)] for i in range(nb_lin)]
        elif i > 2:
            mylist = line.split()
            for j in range(nb_col):
                array[i-3][j] = int(mylist[j])

        i = i + 1

    # Close file
    fi.close()

    return array

print("--- Image to array ---")

test_array = image_to_array("test")
print(test_array)
print_array(test_array)

##############################
## From chapter "Files" ##
## Provide examples of file to test with
```

```
def write_file_gray_image():

    # New file to write
    filenmae = "input/image_gray.pgm"
    fi = open(filenmae,"w")

    # Header
    fi.write("P2\n")  # Grayscale image
    nb_col = 256
    nb_lin = 256
    fi.write(str(nb_col) + " " + str(nb_lin) + "\n")
    levels = 255
    fi.write(str(levels) + "\n")

    for i in range(nb_lin):
        line = ""
        for j in range(nb_col):
            color = (i**2 + j**2) % 256   # one gray level, a function of i and j
            line = line + str(color) + " "
        line = line + "\n"

        # Write line to file
        fi.write(line)

    # Close file
    fi.close()

    return

# Test

print("--- File 'image.pgm' ---")
# write_file_gray_image()

################################
# Activity 1bis - Photo booth
################################

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

def photo_booth_images(image_name,kmax):
    array = image_to_array(image_name)
    array_to_image(array,image_name+"_photo_"+str(0)) # initial image

    n = len(array)
    tab = [[array[i][j] for j in range(n)] for i in range(n)]

    for k in range(1,kmax+1):
        tab = photo_booth(tab)
        array_to_image(tab,image_name+"_photo_"+str(k))

    return

## Test ##
photo_booth_images("image_gray",8)
# photo_booth_images("pi_gimp_new",8)
# photo_booth_images("cat_gimp_new",8)


################################
# Activity 3 - Baker's transfomation
################################

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

```
def baker_stretch(array):
    n = len(array)
    new_array = [[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:
                new_array[i][j] = array[2*i][j//2]
            else:
                new_array[i][j] = array[2*i+1][j//2]

    return new_array

print("--- Baker : stretch an array ---")
array = [ [1,2,3,4], [5,6,7,8],[9,10,11,12], [13,14,15,16] ]
array_stretch = baker_stretch(array)
print_array(array)
print("---")
print_array(array_stretch)

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

def baker_fold(array):
    n = 2*len(array)

    new_array = [[0 for j in range(n)] for i in range(n)]

    # top part
    for i in range(n//2):
        for j in range(n):
            new_array[i][j] = array[i][j]

     # bottom part
    for i in range(n//2,n):
        for j in range(n):
            new_array[i][j] = array[n//2 - i - 1][2*n-1-j]

    # for i in range(n//2):
    #     for j in range(n):
    #         new_array[n-i-1][j] = array[i][2*n-1-j]

    return new_array

print("--- Baker : fold array ---")
array_fold = baker_fold(array_stretch)
print_array(array_stretch)
print("---")
print_array(array_fold)

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

def baker_iterate(array,k):
    n = len(array)
    tab = [[array[i][j] for j in range(n)] for i in range(n)]

    for i in range(k):
        tabb = baker_stretch(tab)
        tab = baker_fold(tabb)

    return tab

print("--- Boulanger : iterate tranformation array ---")
```

```
array = [ [1,2,3,4], [5,6,7,8],[9,10,11,12], [13,14,15,16] ]
print_array(array)
for k in range(1,10):
    array_iter = baker_iterate(array,k)
    print("--- k =",k,"---")
    print_array(array_iter)


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

def baker_images(image_name,kmax):

    array = image_to_array(image_name)
    array_to_image(array,image_name+"_baker_"+str(0)) # image init

    n = len(array)
    tab = [[array[i][j] for j in range(n)] for i in range(n)]

    for k in range(1,kmax+1):
        tabb = baker_stretch(tab)
        tab = baker_fold(tabb)
        array_to_image(tab,image_name+"_baker_"+str(k))

    return

## Test ##
baker_images("image_gray",17)
# baker_images("pi_gimp_new",17)
# baker_images("cat_gimp_new",17)
# baker_images("clock_gimp_new",17)
# baker_images("surf_gimp_new",15)
```

20. Game of life

Activity 1

Activity 1 life_1.py

```
################################
# Game of life
################################

################################
# Activity 1 - Array
################################

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

n, p = 5, 8;
array = [[0 for j in range(p)] for i in range(n)]

# Blinker
array[2][2] = 1
array[2][3] = 1
array[2][4] = 1

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

```
## Question 2 ##

def print_array(array):
    """ Print an array on the screen
    Input: a two dimensional arrayle
    Output: nothing (display on screen) """

    for i in range(n):
        for j in range(p):
            print(array[i][j], end="")
        print()

    return

# Test
print_array(array)
```

Activity 2

Activity 2 life_2.py

```
################################
# Game of life
################################

################################
# From Activity 1
################################

n, p = 5, 8;
array = [[0 for j in range(p)] for i in range(n)]

# Blinker
array[2][2] = 1
array[2][3] = 1
array[2][4] = 1

################################
# Activity 2 - Graphic display
################################

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

from tkinter import *

# Window tkinter
root = Tk()

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

# Scale
scale = 100

def draw_grid():
    """ Show the grid """

    for i in range(n+1):
        canvas.create_line(0,i*scale,p*scale,i*scale)
```

```
    for j in range(p+1):
        canvas.create_line(j*scale,0,j*scale,n*scale)

    for i in range(n):
        canvas.create_text(scale//3,i*scale+scale//2,text=str(i))

    for j in range(p):
        canvas.create_text(j*scale+scale//2,scale//3,text=str(j))

    return

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

def draw_array(array):
    """ Display an array on a graphical screen
    Input: an array
    Output: nothing (display on scree) """

    for i in range(n):
        for j in range(p):
            if array[i][j] != 0:
                canvas.create_rectangle(j*scale,i*scale,(j+1)*scale,(i+1)*scale,fill="red")

    return

# Boutons
def action_button_display():
    canvas.delete("all")
    draw_grid()
    draw_array(array)
    return

button_quit = Button(root,text="Quit", width=8, command=root.quit)
button_quit.pack(side=BOTTOM, padx=5, pady=20)

button_display = Button(root,text="View", width=30, command=action_button_display)
button_display.pack(side=BOTTOM, padx=5, pady=20)

# Test
draw_grid()
draw_array(array)
root.mainloop()
```

Activity 3

Activity 3 life_3.py

```
################################
# Game of life
################################

################################
# From Activity 1
################################

from life_1 import *

n, p = 5, 8;
array = [[0 for j in range(p)] for i in range(n)]
```

```
# Blinker
array[2][2] = 1
array[2][3] = 1
array[2][4] = 1

##############################
# Activity 3 - Evolution
##############################

#################################################
## Question 1 ##
def number_neighbors(i,j,array):
    """ Compute the nb of living neighbors of the box (i,j)
    Input: a box position in a array of cells
    Output: the nb of neighbor cells """

    nb = 0
    # Neighbor top left
    if (i>0) and (j>0) and (array[i-1][j-1] != 0):
        nb += 1
    # Neighbor just above
    if (i>0) and (array[i-1][j] != 0):
        nb += 1
    # Neighbor top right
    if (i>0) and (j<p-1) and (array[i-1][j+1] != 0):
        nb += 1
    # Neighbor left
    if (j>0) and (array[i][j-1] != 0):
        nb += 1
    # Neighbor rigth
    if (j<p-1) and (array[i][j+1] != 0):
        nb += 1
   # Neighbor left below
    if (i<n-1) and (j>0) and (array[i+1][j-1] != 0):
        nb += 1
    # Neighbor just below
    if (i<n-1) and (array[i+1][j] != 0):
        nb += 1
    # Neighbor right below
    if (i<n-1) and (j<p-1) and (array[i+1][j+1] != 0):
        nb += 1

    return nb

# Test
print("--- Number of neighbors ---")
print(number_neighbors(1,1,array))
print(number_neighbors(2,1,array))
print(number_neighbors(3,1,array))
print(number_neighbors(2,0,array))
print(number_neighbors(2,2,array))
print(number_neighbors(3,3,array))

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

def print_neighbors(array):
    """ Print the nb of living neighbors
    Input: an array
    Output: nothin (print on the screen) """

    for i in range(n):
```

```
        for j in range(p):
            print(number_neighbors(i,j,array), end='')
        print()

    return

# Test
print("--- Initial configuration ---")
print_array(array)
print("--- Number of neighbors ---")
print_neighbors(array)

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

def evolution(array):
    """ Caompute the evolution to the next day
    Input: an array
    Output: an array """

    new_array = [[0 for j in range(p)] for i in range(n)]

    for j in range(p):
        for i in range(n):
            # Cell alive or not?
            if array[i][j] != 0:
                cellule_alive = True
            else:
                cellule_alive = False

            # Nombres de neighbors
            nb_neighbors = number_neighbors(i,j,array)

            # Règle du jeu de la vie
            if cellule_alive == True and (nb_neighbors == 2 or nb_neighbors == 3):
                new_array[i][j] = 1
            if cellule_alive == False and nb_neighbors == 3:
                new_array[i][j] = 1

    return new_array

# Test
print("--- Initial configuration ---")
print_array(array)
print("--- Number of neighbors ---")
print_neighbors(array)
print("--- After evolution ---")
array = evolution(array)
print_array(array)
```

Activity 4

Activity 4 life_4.py

```
##############################
# Game of life
##############################

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

```
# From previous activities
##############################

from tkinter import *

# Window tkinter
root = Tk()

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

# Default: nothing
n, p = 25, 25
scale = 40
n, p = 30, 40
scale = 20

array = [[0 for j in range(p)] for i in range(n)]

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

def number_neighbors(i,j,array):
    """ Compute the nb of living neighbors of the box (i,j)
    Input: a box position in a array of cells
    Output: the nb of neighbor cells """

    nb = 0
    # Neighbor top left
    if (i>0) and (j>0) and (array[i-1][j-1] != 0):
        nb += 1
    # Neighbor just above
    if (i>0) and (array[i-1][j] != 0):
        nb += 1
    # Neighbor top right
    if (i>0) and (j<p-1) and (array[i-1][j+1] != 0):
        nb += 1
    # Neighbor left
    if (j>0) and (array[i][j-1] != 0):
        nb += 1
    # Neighbor rigth
    if (j<p-1) and (array[i][j+1] != 0):
        nb += 1
   # Neighbor left below
    if (i<n-1) and (j>0) and (array[i+1][j-1] != 0):
        nb += 1
    # Neighbor just below
    if (i<n-1) and (array[i+1][j] != 0):
        nb += 1
    # Neighbor right below
    if (i<n-1) and (j<p-1) and (array[i+1][j+1] != 0):
        nb += 1

    return nb

def evolution(array):
    """ Caompute the evolution to the next day
    Input: an array
    Output: an array """

    new_array = [[0 for j in range(p)] for i in range(n)]

    for j in range(p):
        for i in range(n):
            # Cell alive or not?
```

```
            if array[i][j] != 0:
                cellule_alive = True
            else:
                cellule_alive = False

            # Nombres de neighbors
            nb_neighbors = number_neighbors(i,j,array)

            # Règle du jeu de la vie
            if cellule_alive == True and (nb_neighbors == 2 or nb_neighbors == 3):
                new_array[i][j] = 1
            if cellule_alive == False and nb_neighbors == 3:
                new_array[i][j] = 1

    return new_array

def draw_grid():
    """ Show the grid """

    for i in range(n+1):
        canvas.create_line(0,i*scale,p*scale,i*scale)

    for j in range(p+1):
        canvas.create_line(j*scale,0,j*scale,n*scale)

    for i in range(n):
        canvas.create_text(scale//3,i*scale+scale//2,text=str(i))

    for j in range(p):
        canvas.create_text(j*scale+scale//2,scale//3,text=str(j))

    return

def draw_array(array):
    """ Display an array on a graphical screen
    Input: an array
    Output: nothing (display on scree) """

    for i in range(n):
        for j in range(p):
            if array[i][j] != 0:
                canvas.create_rectangle(j*scale,i*scale,(j+1)*scale,(i+1)*scale,fill="red")

    return

##############################
# Activity 4 - Game of life: full program
##############################

#####################################################
## Question 0 ##

# Blinker
def blinker():
    """ blinker definition """
    global array
    array = [[0 for j in range(p)] for i in range(n)]
    array[4][7] = 1
    array[4][8] = 1
    array[4][9] = 1
    canvas.delete("all")
    draw_grid()
    draw_array(array)
    return

# Spaceship
```

```
def spaceship():
    """ Spaceship definition """
    global array
    array = [[0 for j in range(p)] for i in range(n)]
    array[3][4] = 1
    array[3][5] = 1
    array[3][6] = 1
    array[2][6] = 1
    array[1][5] = 1
    canvas.delete("all")
    draw_grid()
    draw_array(array)
    return

# Pentadecathlon
def pentadecathlon():
    """ pentadecathlon definition """
    global array
    array = [[0 for j in range(p)] for i in range(n)]
    array[6][4] = 1
    array[6][5] = 1
    array[6][7] = 1
    array[6][8] = 1
    array[6][9] = 1
    array[6][10] = 1
    array[6][12] = 1
    array[6][13] = 1
    array[5][6] = 1
    array[7][6] = 1
    array[5][11] = 1
    array[7][11] = 1
    canvas.delete("all")
    draw_grid()
    draw_array(array)
    return

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

# Boutons

def action_button_evolution():
    global array
    array = evolution(array)
    canvas.delete("all")
    draw_grid()
    draw_array(array)
    return

button_quit = Button(root,text="Quit", width=8, command=root.quit)
button_quit.pack(side=BOTTOM, padx=5, pady=20)

button_display = Button(root,text="Evolve", width=20, command=action_button_evolution)
button_display.pack(side=BOTTOM, padx=5, pady=20)

button_blinker = Button(root,text="Blinker", width=20, command=blinker)
button_blinker.pack(side=TOP, padx=5, pady=5)

button_spaceship = Button(root,text="Spaceship", width=20, command=spaceship)
button_spaceship.pack(side=TOP, padx=5, pady=5)

button_pentadecathlon = Button(root,text="Pentadecathlon", width=20, command=pentadecathlon)
```

```
button_pentadecathlon.pack(side=TOP, padx=5, pady=5)

# root.mainloop()

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

def on_off(i,j):
    """ Change the state of one cell """
    global array
    if array[i][j] == 0:
        array[i][j] = 1
    else:
        array[i][j] = 0
    return

def xy_to_ij(x,y):
    """ Coordonnites (x,y) to coordinates (i,j) """
    i = y // scale
    j = x // scale
    return i, j

def action_mouse_click(event):
    canvas.focus_set()
    # print("Clic à", event.x, event.y)
    x = event.x
    y = event.y
    on_off(*xy_to_ij(x,y))
    canvas.delete("all")
    draw_grid()
    draw_array(array)
    return

# Link mouse click/action
canvas.bind("<Button-1>",action_mouse_click)

draw_grid()
draw_array(array)
root.mainloop()
```

21. Ramsey graphs and combinatorics

Activity 1

Activity 1 ramsey_1.py

```
################################
# Ramsey graphs and combinatorics
################################

################################
# Activity 1 - Définition et amis/étrangers
################################

## Question 1 ##

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

```
# Example 1
n = 3
example_graph_1 = [[0 for j in range(n)] for i in range(n)]
example_graph_1[0][1] = 1; example_graph_1[1][0] = 1
example_graph_1[0][2] = 1; example_graph_1[2][0] = 1

# Example 2
n = 4

example_graph_2 = [[0 for j in range(n)] for i in range(n)]

example_graph_2[0][2] = 1; example_graph_2[2][0] = 1
example_graph_2[0][3] = 1; example_graph_2[3][0] = 1
example_graph_2[1][2] = 1; example_graph_2[2][1] = 1


# Example 3
n = 5

example_graph_3 = [[0 for j in range(n)] for i in range(n)]

example_graph_3[0][2] = 1; example_graph_3[2][0] = 1
example_graph_3[0][3] = 1; example_graph_3[3][0] = 1
example_graph_3[1][2] = 1; example_graph_3[2][1] = 1
example_graph_3[1][4] = 1; example_graph_3[4][1] = 1
example_graph_3[3][4] = 1; example_graph_3[4][3] = 1

# Example 4
n = 6

example_graph_4 = [[0 for j in range(n)] for i in range(n)]

example_graph_4[3][2] = 1; example_graph_4[2][3] = 1;
example_graph_4[1][2] = 1; example_graph_4[2][1] = 1
example_graph_4[3][4] = 1; example_graph_4[4][3] = 1
example_graph_4[4][1] = 1; example_graph_4[1][4] = 1
example_graph_4[0][2] = 1; example_graph_4[2][0] = 1
example_graph_4[5][0] = 1; example_graph_4[0][5] = 1
example_graph_4[5][1] = 1; example_graph_4[1][5] = 1
example_graph_4[0][3] = 1; example_graph_4[3][0] = 1


# Example for the lesson
n = 4

example_graph_lesson_1 = [[0 for j in range(n)] for i in range(n)]

example_graph_lesson_1[0][2] = 1; example_graph_lesson_1[2][0] = 1
example_graph_lesson_1[1][3] = 1; example_graph_lesson_1[3][1] = 1

## Question 2 ##

##################################################
def print_graph(array):
    """
    Print an array on the screen
    Input: un array
    Output: nothing (print on screen)
    """

    n = len(array)

    for j in range(n):
        for i in range(n):
            print(array[i][j], end="")
        print()
```

```
    return

# Test
if __name__ == '__main__':
    print("--- Array of the graph ---")
    print("--- Example 1 ---")
    print_graph(example_graph_1)
    print("--- Example 2 ---")
    print_graph(example_graph_2)
    print("--- Example 3 ---")
    print_graph(example_graph_3)
    print("--- Example 4 ---")
    print_graph(example_graph_4)

    print("--- Lesson 1 ---")
    print_graph(example_graph_lesson_1)

##################################################
##################################################
# Test if a graph contains 3 friends/3strangers

##################################################
def has_3_friends_fix(array,i,j,k):
    """ Test if the vertices i, j, k are linked has friends"""
    if array[i][j] == 1 and array[i][k] == 1 and array[j][k] == 1:
        return True
    else:
        return False

##################################################
def has_3_strangers_fix(array,i,j,k):
    """Test if the vertices i, j, k  are strangers together"""
    if array[i][j] == 0 and array[i][k] == 0 and array[j][k] == 0:
        return True
    else:
        return False

# Test
if __name__ == '__main__':
    print("--- Has 3 friends? Has 3 strangers? ---")
    print(has_3_friends_fix(example_graph_4,1,3,4))
    print(has_3_strangers_fix(example_graph_4,1,3,4))
```

Activity 2

Activity 2 ramsey_2.py

```
##############################
# Ramsey graphs and combinatorics
##############################


##############################
# Activity 2 - Graphic display
##############################

from tkinter import *
from math import *
```

```
from tkinter.font import Font

from ramsey_1 import *  # For examples

# tkinter window
root = Tk()

canvas = Canvas(root, width=800, height=500, background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

# Scale
scale = 200

# Basic version (compute many times the same thing)
##################################################
def display_graph_basic(array):
    """
    Display a graph
    Input: a graph
    Output: nothing
    """
    n = len(array)

    # Edges
    for j in range(n):
        for i in range(n):
            xi = 2*scale + cos(2*i*pi/n)*scale
            yi = 1.5*scale + sin(2*i*pi/n)*scale
            xj = 2*scale + cos(2*j*pi/n)*scale
            yj = 1.5*scale + sin(2*j*pi/n)*scale
            if array[i][j] == 0:
                canvas.create_line(xi,yi,xj,yj,width=4,fill="red")
            if array[i][j] == 1:
                canvas.create_line(xi,yi,xj,yj,width=4,fill="green")

    # Vertices
    for i in range(n):
        x = 2*scale + cos(2*i*pi/n)*scale
        y = scale + sin(2*i*pi/n)*scale
        canvas.create_oval(x-5,y-5,x+5,y+5,fill="black")

    return

# Optimal version
##################################################
def display_graph(array):
    """
    Display a graph
    Input: a graph
    Output: nothing
    """
    n = len(array)  # Number of vertices

    # List of the coordinates (x,y) of the vertices
    coord = [(2*scale + cos(2*i*pi/n)*scale,1.2*scale + sin(2*i*pi/n)*scale) for i in range(
    ↪ n)]

    # Edges
    for j in range(n):
        for i in range(j+1,n):  # i>j
            if array[i][j] == 0:
                canvas.create_line(coord[i],coord[j],width=4,fill="red",dash=(6, 2))
            if array[i][j] == 1:
```

```
                canvas.create_line(coord[i],coord[j],width=4,fill="green")

    myfont = Font(family="Courier", weight="bold",size=18)
     # Vertices
    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=myfont,fill="white")

    return

# Launch of the window
if __name__ == '__main__':
    button_quit = Button(root,text="Quit", width=8, command=root.quit)
    button_quit.pack(side=BOTTOM, padx=5, pady=20)

    # Example
    display_graph(example_graph_4)
    root.mainloop()
```

Activity 3

Activity 3 ramsey_3.py

```
##############################
# Ramsey graphs and combinatorics
##############################

##############################
# Activity 3 - Binary
##############################

def integer_to_binary(p,n):
    str_b = bin(p)  # Conversion to binary notation
    str_bb = str_b[2:]  # We cut off the prefix
    # Transformation to a list of **intgers** 0 or 1
    list_binary = []
    for b in str_bb:
        list_binary = list_binary + [int(b)]

    # Add zeros at the begining if necessary
    nb_zeros = n - len(list_binary)
    for i in range(nb_zeros):
        list_binary = [0] + list_binary

    return list_binary

# Short version, using "format()"
def integer_to_binary_bis(p,n):
    model = '{:0'+str(n)+'b}'
    str_binary = model.format(p)
    list_binary = [int(b) for b in list(str_binary)]
    return list_binary

# Test
if __name__ == '__main__':
    n = 8
    p = 37
```

```
    print(integer_to_binary(p,n))
    print(integer_to_binary_bis(p,n))
```

Activity 4

Activity 4 ramsey_4.py

```
##############################
# Ramsey graphs and combinatorics
##############################

##############################
# Activity 4 - Subsets
##############################
from ramsey_3 import integer_to_binary

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


## Question 1 ##

# Genration of all subsets

#####################################################
def subsets(n):
    """Find all subsets of a set  [0,1,2,...n-1] having n elements """

    all_subsets = []

    for p in range(2**n):

        # Binary conversion
        list_binary = integer_to_binary(p,n)
        #print(list_binary)

        sub = []
        for j in range(n):
            # if list_binary[n-j-1] == 1:
            if list_binary[j] == 1:
                sub = sub + [j]

        all_subsets = all_subsets + [sub]

    return all_subsets

# Test
if __name__ == '__main__':
    print("--- Subsets ---")
    n = 3
    SS_ENS = subsets(n)
    print("For n = ",n)
    print("Number of subsets = ",len(SS_ENS))
    print(SS_ENS)

## Question 2 ##

#####################################################
def fix_subsets(n,k):
    all_fix_subsets = []
```

```
    for sub in subsets(n):
        if len(sub) == k:
            all_fix_subsets = all_fix_subsets + [sub]
    return all_fix_subsets

# Test (suite)
if __name__ == '__main__':
    print("--- Sous-ensembles à 3 éléments ---")

    n = 6
    k = 3
    SS_ENS_3 = fix_subsets(n,k)
    print("For n = ",n," k = ",k)
    print("Number of subsets having",k,"elements = ",len(SS_ENS_3))
    print(SS_ENS_3)
```

Activity 5

Activity 5 ramsey_5.py

```
##############################
# Ramsey graphs and combinatorics
##############################

##############################
# Activity 5 - Proof n=6
##############################

from ramsey_1 import *
from ramsey_1 import has_3_friends_fix
from ramsey_1 import has_3_strangers_fix
from ramsey_3 import integer_to_binary
from ramsey_4 import subsets
from ramsey_4 import fix_subsets

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

# Subsets
n = 6
k = 3
SS_ENS_6_3 = fix_subsets(n,k)

## Question 1 ##

#################################################
def has_3(array):
    """Find if grpahe have 3 vertices friends/strangers"""

    n = len(array)

    #for sub in SS_ENS_6_3:  # For n=6, k=3
    for sub in fix_subsets(n,3):  # Fir any n
        #print(sub)
        has_3_friends = has_3_friends_fix(array,*sub)
        has_3_strangers = has_3_strangers_fix(array,*sub)
        found = has_3_friends or has_3_strangers
        if found == True:
            break
```

```
    # Display
    # if found == True:
    #     print("OK for example:",sub)
    # else:
    #     print("Problem")
    # if found == False:
    #     print("Problem")
    #     print_graph(array)

    return found

# Test
# An example

if __name__ == '__main__':
    print("--- Test conjecture un seul graph ---")
    print("--- Example 1 ---")
    print(has_3(example_graph_1))
    print("--- Example 2 ---")
    print(has_3(example_graph_2))
    print("--- Example 3 ---")
    print(has_3(example_graph_3))
    print("--- Example 4 ---")
    print(has_3(example_graph_4))

## Question 2 ##

##################################################
##################################################
# Computation of all possibles graphs having n vertices
# There are 2^( (n-1)*n/2 )
def print_all_graphs(n):

    N = ((n-1) * n)//2
    print("Total number of graphs:",2**N)

    for p in range(2**N):
        # Binary conversion binary
        list_binary = integer_to_binary(p,N)

        print("p =",p,list_binary)

        graph = [[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 = list_binary.pop()
                graph[i][j] = b
                graph[j][i] = b

        print_graph(graph)

    return

 # Test
# n = 4
# print("--- Print all possible graphs ---")
# print("n = ",n)
# print_all_graphs(n)

## Question 3 ##

##################################################
##################################################
# Test all possible graph having n vertices
# Il y a 2^( (n-1)*n/2 )
```

```
def test_all_graphs(n):

    N = ((n-1) * n)//2
    print("Total number of graphs:",2**N)

    for p in range(2**N):
        # Binary conversion
        list_binary = integer_to_binary(p,N)

        # print("p =",p,list_binary)

        graph = [[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 = list_binary.pop()
                graph[i][j] = b
                graph[j][i] = b

        # print_graph(graph)
        test = has_3(graph)
        if test == False:
            print("Problem with",p)

    return

# Test
n = 6
print("\n\n--- Proof if Ramsey theorem for n = 6 ---")
print("n = ",n)
print("--- Looking for a graph that doesn't satisfy the proposition...")
test_all_graphs(n)
print("... end of computation ---")
print("If nothing has been printed, it's checked!")
```

Activity 6

Activity 6 ramsey_6.py

```
################################
# Ramsey graphs and combinatorics
################################

################################
# Activity 6 - To go further
################################

from ramsey_1 import *
from ramsey_1 import has_3_friends_fix
from ramsey_1 import has_3_strangers_fix
from ramsey_3 import integer_to_binary
from ramsey_4 import subsets
from ramsey_4 import fix_subsets

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


## Question 1 ##

# Subsets
```

```
n = 6
k = 3
SS_ENS_3 = fix_subsets(n,k)


################################################
def has_3(graph):
    """Find if graph has 3 vertices friends/strangers"""

    n = len(graph)

    for sub in SS_ENS_3:
        has_3_friends = has_3_friends_fix(graph,*sub)
        has_3_strangers = has_3_strangers_fix(graph,*sub)
        found = has_3_friends or has_3_strangers
        if found == True:
            break

    # Display
    # if found == True:
    #     print("OK for example:",sub)
    # else:
    #     print("Problem")
    # if found == False:
    #     print("Problem")
    #     print_graph(array)

    return found

# Test all possibles graphs having n vertices
# Il y a 2^( (n-1)*n/2 )

def test_all_graphs(n):

    N = ((n-1) * n)//2
    print("Total number of graphs:",2**N)

    for p in range( ((2**N) // 2)):
        # Binary conversion
        list_binary = integer_to_binary(p,N)

        # print("p =",p,list_binary)

        graph = [[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 = list_binary.pop()
                graph[i][j] = b
                graph[j][i] = b

        # print_graph(graph)
        test = has_3(graph)
        if test == False:
            print("Problem with graph p =",p)

    return

# Test
n = 6
print("\n\n--- Proof if Ramsey theorem for n = 6 ---")
print("n = ",n)
print("--- Looking for a graph that doesn't satisfy the proposition...")
test_all_graphs(n)
print("... end of computation ---")
print("If nothing has been printed, it's checked!")
```

```
# n = 6 : 0.5 seconds
# n = 7 : 20 seconds
# n = 8 : 2500 seconds = 40 min (extrapolation simpling of 10^-2 %)
# n = 9 : 800 000 seconds = 9 jours (extrapolation from simplang of 10^-4 %)

## Question 2 ##

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

# Subsets
n = 7
SS_ENS_3 = fix_subsets(n,3)
SS_ENS_4 = fix_subsets(n,4)

#####################################################
def has_4_friends_fix(graph,i,j,k,l):
    """Test if vertices i, j, k,l are all linked"""
    if graph[i][j] == 1 and graph[i][k] == 1 and graph[i][l] == 1 and graph[j][k] == 1 and
    ↪ graph[j][l] == 1 and graph[k][l] == 1:
        return True
    else:
        return False

# Test of all possible graphs having n vertices
# to see if there 4 friends or 3 strangers

def has_3_4(graph):
    """Test if 3 or 4 vertices are links"""

    n = len(graph)

    # Look for 3 strangers
    for sub in SS_ENS_3:
        has_3_strangers = has_3_strangers_fix(graph,*sub)
        if has_3_strangers == True:
            break

    # If not 3 strangers, look for 4 friends
    if has_3_strangers == False:
        for sub in SS_ENS_4:
            found_4_friends = has_4_friends_fix(graph,*sub)
            if found_4_friends == True:
                break
    else:
        found_4_friends = True # Doesn't matter, since 3 strangers

    found = has_3_strangers or found_4_friends

    return found


def ramsey_4_3(n):

    N = ((n-1) * n)//2
    print("Total number of graphs:",2**N)

    # for p in range(  ((2**N)) // 100000 ):
    for p in range( 1000000 ):
        # Binary conversion
        list_binary = integer_to_binary(p,N)

        # print("p =",p,list_binary)
```

```
        graph = [[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 = list_binary.pop()
                graph[i][j] = b
                graph[j][i] = b

        test = has_3_4(graph)
        if test == False:
            print("Problem with the graph p =",p)

    return

# Test
print("\n\n--- Proof of Ramsey theorem with 4 friends or 3 strangers, n =",n,"---")
print("n = ",n)
print("--- Looking for a graph that doesn't satisfy the proposition...")
ramsey_4_3(n)
print("... end of computation ---")
print("If nothing has been printed, it's checked!")

# n = 7, easy and many counter-examples
# n = 8 counter examples for instance p=111121101
# n = 9 is True! But should have 18 days of computations
```

22. Bitcoin

Activities bitcoin.py

```
################################
# Bitcoin
################################


from random import randint
from time import *


################################
# Activity 2 - Tools for lists
################################

# Global constant for length of blockks
N = 6

# Constant for proof of work
Max = [0,0,25]

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

## Question 1 ##

# Addition of terms of two lists of the same size (and modulo 100)
def addition(mylist1,mylist2):
    list_sum = []
    for i in range(len(mylist1)):
        list_sum = list_sum + [ (mylist1[i]+mylist2[i]) % 100 ]
```

```
    return list_sum

# Test
print("--- Test list sum ---")
print(addition([1,2,3,4,5,6],[1,1,1,1,1,1]))

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

## Question 2 ##

# Test if a list is small than max_list
def is_smaller(mylist,max_list):
    i = 0
    n = len(max_list)
    while (i < n) and (mylist[i] <= max_list[i]):
        i = i + 1
    if i == n:
        return True
    else:
        return False

# Test
print("--- Test list small ---")
print(is_smaller([0,0,24,4,5,6],[0,0,50]))

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

## Question 3 ##

def sentence_to_list(sentence):
    # Transform letters to numbers less than 100
    mylist = [ord(c) % 100 for c in sentence]

    # Rajoute des 0 devant si besoin
    while len(mylist) % N > 0:
        mylist = [0] + mylist

    return mylist

# Test
print("--- Sentence to list ---")
sentence = "Be happy!"
print(sentence)
print(sentence_to_list(sentence))


##############################
# Activity 3 - Hasch function
##############################


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

## Question 3 ##
p = [7,11,13,17,19,23] # prime numbers

def one_round(block):
    # Addition
    block[1] = (block[1]+block[0]) % 100
    block[3] = (block[3]+block[2]) % 100
    block[5] = (block[5]+block[4]) % 100

    # m = p*m + 1 (modulo 100)
    for i in range(N):
        block[i] = (p[i]*block[i]+1) %  100
    # permutation
```

```
    block = [block[N-1]] + block[:N-1]
    return block

# Test
print("--- Test one round ---")
block = [0,1,2,3,4,5]
print(block)
print(one_round(block))

block = [1,1,2,3,4,5]
print(block)
print(one_round(block))

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

## Question 3 ##

def ten_rounds(block):
    for i in range(10):
        block = one_round(block)
    return block

# Test
print("--- Test ten rounds ---")
block = [0,1,2,3,4,5]
print(block)
print(ten_rounds(block))

block = [1,1,2,3,4,5]
print(block)
print(ten_rounds(block))

block = [99,96,87,56,67,76]
print(block)
print(ten_rounds(block))

block = [70,92,22,4,16,90]
print(block)
print(ten_rounds(block))

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

## Question 3 ##
def bithash(mylist):

    while len(mylist)>N:
        block1 = mylist[0:N]  # First block
        block2 = mylist[N:2*N] # Second block
        end_list = mylist[2*N:] # Remaining blocks
        # print(block1)
        # print(block2)
        # print(end_list)

        #block1 = one_round(block1)   # One round
        block1 = ten_rounds(block1) # Ten rounds

        #print(block1)

        new_block_begin = addition(block1,block2)

        mylist = new_block_begin + end_list

    # Lasr ten rounds for mylist (that only contain one block)
    # mylist = one_round(mylist) # One round
    mylist = ten_rounds(mylist) # Ten rounds

    return mylist
```

```
# Test
print("--- Hash of a list ---")

mylist = [1,2,3,4,5,6,1,2,3,4,5,6]
thehash = bithash(mylist)
print(mylist)
print(thehash)

mylist = [1,1,3,4,5,6,1,2,3,4,5,6]
thehash = bithash(mylist)
print(mylist)
print(thehash)

mylist = [0,1,2,3,4,5,1,1,1,1,1,1,10,10,10,10,10,10]
thehash = bithash(mylist)
print(mylist)
print(thehash)


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

##############################
# Activity 4 - Proof of work - Minage
##############################

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

## Question 3 ##

def verification_proof_of_work(mylist,proof):

    list_test = mylist + proof
    thehash = bithash(list_test)
    # print(proof,thehash)
    if is_smaller(thehash,Max):
        return True
    else:
        return False

# Test
print("--- Verif Proof of work ---")

mylist = [0,1,2,3,4,5]
proof = [12, 3, 24, 72, 47, 77]
# Max = [0,0,7]

start_time = time()
print(verification_proof_of_work(mylist,proof))
end_time = time()
duration = end_time-start_time

print("Time of computation:",duration)

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

## Question 2 ##

def proof_of_work(mylist):

    thehash = [1,1,1,1,1,1]

    while not(is_smaller(thehash,Max)):
        proof = [randint(0,99) for i in range(N)]
        list_test = mylist + proof
        thehash = bithash(list_test)
    print(proof,thehash)
```

```
    return proof

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

## Question 2 bis ##

from itertools import product

def proof_of_work_bis(mylist):
    for proof in product(range(100),range(100),range(100),range(100),range(100),range(100)):
        proof = list(proof)
        list_test = mylist + proof
        thehash = bithash(list_test)
        if is_smaller(thehash,Max):
            break

    print(proof,thehash)
    return proof

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

## Question 3 ##

# Test
print("--- Proof of work ---")

start_time = time()
mylist = [0,1,2,3,4,5]
# proof = proof_of_work(mylist)
# proof = proof_of_work_bis(mylist)
end_time = time()
duration = end_time-start_time

print("Time of computation:",duration)

##############################
# Activity 5 - Tes bitcoins
##############################

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

## Question 1 ##

proof_init = [0,0,0,0,0,0]   # random values
blockchain = [proof_init]

def add_transaction(transaction):
    global blockchain
    blockchain = blockchain + [transaction]
    return blockchain

# Test
print("--- Initialization of the register book and first transaction ---")
print(blockchain)
add_transaction("Bob +135")
print(blockchain)

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

## Question 2 ##

def mining():
    global blockchain
    transaction = blockchain[-1]
    prev_proof = blockchain[-2]
    # print(transaction)
```

```
    # print(prev_thehash)
    # print(sentence_to_list(transaction))
    mylist = prev_proof + sentence_to_list(transaction)

    proof = proof_of_work(mylist)

    blockchain = blockchain + [proof]

    return blockchain

# Test
print("--- Mining  ---")
print(blockchain)
mining()
print(blockchain)

print("--- Example for chapter  ---")
Max = [0,0,7]
thehash_init = [3,1,4,1,5,9]   # random values
blockchain = [thehash_init]
add_transaction("Abel +35")
print(blockchain)
mining()
print(blockchain)

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

## Question 3 ##

def verification_blockchain():
    prev_proof = blockchain[-3]
    transaction = blockchain[-2]
    proof = blockchain[-1]
    thehash = bithash(prev_proof+sentence_to_list(transaction)+proof)
    if is_smaller(thehash,Max):
        return True
    else:
        return False

# Test
print("--- Verification of the blockchain  ---")
print(blockchain)
print(verification_blockchain())


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

## Question 4 ##
# Full example

# Constant for proof of work
Max = [0,0,7]

start_time = time()  # start chrono

thehash_init = [0,0,0,0,0,0]   # random values
blockchain = [thehash_init]

print(blockchain)
add_transaction("Abel +135")
print(blockchain)
mining()
print(blockchain)
print(verification_blockchain())

add_transaction("Bob -77")
```

```
print(blockchain)
mining()
print(blockchain)
print(verification_blockchain())

add_transaction("Camille -25")
print(blockchain)
mining()
print(blockchain)
print(verification_blockchain())

end_time = time()
duration = end_time-start_time
print("Time of computation:",duration)
```

23. Random blocks

Activities 1 and 2

Activities 1 and 2 blocks_vertical.py

```
##############################
# Aléatoire
##############################

from random import *
from tkinter import *
import time

##############################
# Activity 1 - Blocks falling
##############################

n = 4   # nb of lines
p = 6   # nb of columns

array = [[0 for j in range(p)] for i in range(n)]

array[3][3] = 1
array[3][2] = 1
array[2][2] = 1
array[1][2] = 1
array[0][4] = 1

##################################################
def print_array():

    for i in range(n):
        for j in range(p):
            print(array[i][j], end="")
        print()

    return

print_array()

##################################################
def can_fall(i,j):
    if i == n-1:      # bottom line?
        return False
```

```
    if array[i+1][j]:  # cell just below?
        return False

    if j>0 and array[i][j-1]:   # block on the left?
        return False

    if j<p-1 and array[i][j+1]:  # block on the right?
        return False

    return True


##################################################
def drop_one_block(j):
    i = 0
    while can_fall(i,j):
        i = i + 1

    array[i][j] = 1

    return i,j

##################################################
def drop_blocks(k):
    # print_array()
    # print()
    for __ in range(k):
        j = randint(0,p-1)
        drop_one_block(j)
        # print_array()
        # print()

    return

# drop_blocks(7)
# print()
# print_array()

# exit()

##############################
# Activity 2 - tkinter static display
##############################

n = 25   # nb of lines
p = 50   # nb of columns

array = [[0 for j in range(p)] for i in range(n)]

scale = 20  # scale
nb_blocks = 50

root = Tk()

canvas = Canvas(root, width=p*scale, height=n*scale, background="white")
canvas.pack(fill="both", expand=True)

def display_array():
    canvas.delete("all")   # Clear all

    for i in range(n):
        for j in range(p):
            if array[i][j]:
                canvas.create_rectangle(j*scale,i*scale,j*scale+scale-1,i*scale+scale-1,
    ↪ width=1,fill='green')
```

```
    return

# Test
# display_array()

def action_blockk():
    drop_blocks(nb_blocks)
    display_array()

    return

button_block = Button(root,text="View blocks", width=20, command=action_blockk)
button_block.pack(pady=10)

button_quit = Button(root,text="Quit", width=20, command=root.quit)
button_quit.pack(side=BOTTOM, pady=10)

root.mainloop()
```

Activity 3

Activity 3 — blocks_circular.py

```
##############################
# Random blocks
##############################

from random import *
from tkinter import *
import time

##############################
# Activity 3 - Circular movement
##############################

n = 10   # nb of lines
p = 10   # nb of columns
boundary = min(n,p)//5  # distance to the boundary

array = [[0 for j in range(p)] for i in range(n)]

array[(n-1)//2][(p-1)//2] = 1  # center

#################################################
def print_array():
    for i in range(n):
        for j in range(p):
            print(array[i][j], end="")
        print()

    return

# print_array()

#################################################
def can_move(i,j):
    # if array[i][j]:   # on an existing block
    #     return False

    if i>0 and array[i-1][j]:   # above?
        return False

    if i<n-1 and array[i+1][j]:  # below?
```

```
        return False

    if j>0 and array[i][j-1]:   # left?
        return False

    if j<p-1 and array[i][j+1]:  # right?
        return False

    return True

##################################################
def is_inside(i,j):
    if (0 <= i < n) and (0 <= j < p):
        return True
    else:
        return False

##################################################
def launch_one_block():
    i = randint(0+boundary,n-1-boundary)
    j = randint(0+boundary,p-1-boundary)

    while is_inside(i,j) and can_move(i,j):
        dx = randint(-1,1)
        dy = randint(-1,1)
        i = i + dx
        j = j + dy

    if is_inside(i,j):
        array[i][j] = 1

    return i,j

##################################################
def launch_blocks(k):
    for __ in range(k):
        launch_one_block()

    return

launch_blocks(5)


##############################
# Static tlinter display
##############################

n = 30   # nb of lines
p = 30   # nb of columns

boundary = min(n,p)//10  # distance to boundary for launch
scale = 20

array = [[0 for j in range(p)] for i in range(n)]

array[(n-1)//2][(p-1)//2] = 1  # center

nb_blocks = 10

root = Tk()

canvas = Canvas(root, width=p*scale, height=n*scale, background="white")
canvas.pack(fill="both", expand=True)

def display_array():
    canvas.delete("all")   # clear all

    for i in range(n):
```

```
        for j in range(p):
            if array[i][j]:
                canvas.create_rectangle(j*scale,i*scale,j*scale+scale-1,i*scale+scale-1,
    ↪ width=1,fill='green')
    return


def action_block():
    launch_blocks(nb_blocks)
    display_array()

    return

button_block = Button(root,text="Launch blocks", width=20, command=action_block)
button_block.pack(pady=10)

button_quit = Button(root,text="Quit", width=20, command=root.quit)
button_quit.pack(side=BOTTOM, pady=10)

root.mainloop()
```