Programming and algorithms

Practice 2 & 3

0. Prerequisite

The objective of this practice is to code "Conway's game of life". You can find many simulations online. You shall have one file named `input`. It represents the initial state of the grid.

1. Preparing and coding

In this first part, we will create a first draft of the game. The idea is to have a code that works and that is as simple as possible. The requirements are:

- read the given file `input` (appendix-A.1), and use its data as the initial condition (appendix-A.2)
- encode the evolution, using the rule of the game of life. We will suppose that border cells don't evolve (dead stays dead, live stays live)
- print the grid, in the terminal, after `t` evolution steps (with $t = 50$)

The format of the input file is simply a matrix of `0` (dead cell) and `1` (live cell) with each line separated by a break line `"\n"`. It represents the state of each cell before the beginning of the evolution.

Question 1 *How would you go about encoding this game ? What are the different variables, loops, conditions, functions that you would need ? You should write down a very basic draft of the code's structure.*

Task 1 *Once you feel confident, start coding your program, piece by piece. It is very important that you* ***test each piece of code****, mistakes always happen and it is easier to debug something small.*

2. Going further

Task 2 *Improve your code:*

- *Write the output in a file (use the official python documentation).*
- *Give a graphical representation of the output (appendix-B.1).*
- *Print out every step of the evolution, not simply the last, pausing between each step (for instance 1 second).*
- *Start from a random grid (appendx-B.2). You can also ask the user to choose whether the initial condition is random or from the file.*
- *Create an animation that shows the evolution graphically (appendix-B.3).*
- *Store the animation in a file (use the documentation for* `matplotlib.animation`*).*

A. Help !

A.1. Reading a file

```
my_file = open("path_to_file", "r")
data = my_file.read()
my_file.close()
```

1. The first instruction open the file in reading mode.
2. The second command store the data of the file in the variable `data`, as a string.
3. The last command close the file. **It is very important that you always close a file**.

If you want to split a string line by line and then print the first line, the instruction is (`\n` is the character for a line break):

```
data_splitted = data.split("\n")
print(data_splitted[0]) # Prints the first line
```

A.2. Creating a 2D array

To create an array (= a list) with values from N to 1 simply do as follows:

```
array = [N-i for i in range(N)]
```

To create a 2D array of size $N_0 \times N_1$, use twice the previous code. Here is an example, where the value at position (i, j) is $i + j$:

```
array = [[line+column for column in range(N1)] for line in range(N0)]
```

If you need to transform a string `a` into an integer `b`, you can use the instruction `b=int(a)`. The other way around is `a=str(b)`.

B. Help (going further) !

B.1. Instructions to output a graphic

```
# Importing the graphic library
import matplotlib.pyplot as plt

# M is a 2D array (i.e. M = [ [...], ..,  [...] ] )
plt.imshow(M)
plt.show(block=False)
```

The parameter `block=False` allow you to continue executing code afterwards. To use a grayscale, add the parameter `cmap='gray'` to the `imshow` command.

B.2. Getting a random integer

```
# Importing the numpy library
import numpy as np
# random number between 10 and 49:
my_random = np.random.randint(10,50)
```

B.3. Creating an animation

This is a little bit more complicated. The idea is as follows

1. Create the first frame
2. Code a function that will update the frame (takes as input the picture and returns the full next frame).
3. Tell matplotlib what to show and how to update the picture.

Here is a generic code you may use as a basis:

```
# Importing the graphic library
import matplotlib.pyplot as plt
import matplotlib.animation as animation

# 1) create the first frame
N = 10
M = [ [i+j for j in range(N)] for i in range(N)]

# 2) Create the update function, the first two arguments are the number of the current frame,
    the current "img" (image) and then you can give any number of arguments.
def updater(frame_number, img, M, arg2, arg3):
    ... # Change M as you like
    img.set_data(M)
    return img

# 3) Tell matplotlib what to do
fig, ax = plt.subplots()        # Create a figure
img = ax.imshow(M, cmap='gray') # Plot the first frame
# Create the animation (using your updater)
ani = animation.FuncAnimation(fig, updater, fargs=(img, M, arg2, arg3,),
                              frames=100,
                              repeat=False,
                              interval=200)
plt.show()
```

You can find more developed examples online.