PRESTACAP

Programming Challenge

<u>First task (Mario saving the princess):</u>

Mario and the princess are trapped in a square grid (N*N), Mario needs to reach the princess with minimum number of steps (shortest paths), while avoiding any obstacles. Mario can move UP, DOWN, LEFT and RIGHT and can't go outside of the grid. Create a function or a class that given a square grid (N*N), the location of Mario and the princess will return a list of the shortest path**s** possible from Mario to the princess.

Mario denoted by (m)

Princess denoted by (p)

Obstacle denoted by (x)

Free cell denoted by (-)

Your code should also validate that the given grid is a square matrix which contains one Mario, one princess and empty cells or obstacles. Note that all letters are case sensitive. In case one of the constrains are violated your code should identify it and raise an error flag and empty return for the paths.

Input format:

you have two inputs [N] which is the grid size and [Grid] which is the actual map itself. [N] is a scalar integer, while [Grid] should be a list of string where each entry is a row of the map itself.

Example:

N = 3

Grid = ['--m','-x-','-p-']

```
-  -  m

-  x  -        >>> 3*3 matrix, Mario exists at [0,2], princess at [2,1] and an obstacle at [1,1]

-  p  -
```

Output format:

Your output should be [error_flag] and [paths]. The [error_flag] is a binary output where it is TRUE in case, we have an error and FALSE otherwise, the [paths] is a list of all the possible moves (RIGHT-LEFT-UP-DOWN) that can be followed to get to the princess. So, the output [paths] for the example mentioned above should be:

[(DOWN, DOWN, LEFT), (LEFT, DOWN, DOWN), (DOWN, LEFT, DOWN)]

However, since we have an obstacle in [1,1] the list should be:

[(DOWN, DOWN, LEFT)]

In case all the shortest paths are blocked your code is expected to return at least one possible path (if exists).

As for [error_flag] it should equal FALSE because the grid is formatted correctly.

Second task (API and Database):

- Create a REST API for using the class/function from task one. The endpoints are up to you. However, your API should have inputs/outputs in the same style.
- Create a serverless database (SQLite) which has one table to save data coming from the API.
- Your API should return the game result to the user and save the request time along with input sent into the serverless database (SQLite).

Hints and notes:

1. For the API creation you can use flask or flask restful for a quick implementation.
2. Using an ORM such as sqlalchemy for the database part is a plus.
3. Adding another end point to view the log data from the database in the API is a plus.

Third task (Bonus):

Create a web application to play the Mario game created in task one. The web application must be interactive. Being able to view the output (solution) graphically is a bonus, the design of the game is totally up to you. For this task we recommend using Dash by plotly, however in case you prefer other web frameworks feel free to use it.

General Instructions:

1. Make sure your code is modular and easy to read and understand.
2. The API should be documented properly, especially the end points part.
3. Once you are done with the challenge push it to a repository on your GitHub account.
4. Stick to a python version 3.5+, and incase of using external modules (not supported by default in python) make sure to mention the version.
5. Using (Dash by plotly) is preferred over using other web frameworks, however it won't affect the decision.