

The deadline for this exercise sheet is **Tuesday, 29.05.2018, 14:00.**

Contents

The Pseudocode	2
General Hints That Can Be More Or Less Helpful	3
Module Separation	4
Suggested Functions In Module 1	5
Suggested Functions In Module 2	6
Suggested Functions In Module 3	7
Suggested Functions In Module 4	8

DISCLAIMER: These are all just suggestions and not necessarily a complete or the best approach to a solution. It just offers hints, general approaches and ideas.

The Pseudocode

The following pseudocode would be a possible (and simplified) way of doing it:

```
Let User Choose A Maze
Load Maze From File
Convert Maze Into List-like Structure
Print Maze Without Path
Find Start Point
Find Goal Point
Start Recursive Search:
    If Position Is Goal Positon:
        Done
    If Position Is Wall:
        Go Back
    If Position Is Already Visited:
        Go Back
    For All Directions:
        Search(NewPos)

If Way Found:
    Print Maze With Path
Else:
    Print Sad Message Since One Is Forever Trapped
If Save:
    Save Solved Maze To New File
```

General Hints That Can Be More Or Less Helpful

- You need to check the 4 neighbouring fields of each cell
 - be careful with the bounds of the grid
- Each cell you visit needs to be checked before continuing to the neighbours
- Make sure to mark visited cells with a marker so you don't visit a cell twice
- The necessary checks are whether it is a wall, path, start, or goal cell
 - make sure to return according results
- Just like last week, part your code into smaller specific functions
 - Like one function for reading mazes from files, one for converting them into a grid, and one for solving the maze then, and so on...

Module Separation

We suggest the modules:

- **solver.py**: This is where the actual recursion and solving happens. Most importantly it implements the **solve** function, which starts the search for a way
- **fileIO.py**: This module is responsible for reading mazes from files and rebuild them into a usable Python structure. It could also offer the possibility to write back to files, to make it easy to save solutions to mazes or save mazes inputted by the user
- **userIO.py**: This module offers functions to display the maze in a nice way on the terminal and interact with the user, like getting input, varifying the input, etc.
- **maze.py**: This is what brings it all together. It is the "main" module of the whole pack. It imports the functionality of the other 3 modules and runs the main program to choose a maze, load the maze, solve the maze and print the solution

Suggested Functions In Module 1

Module `maze.py` could contain:

- `main()`: the function that starts and runs it all. That's it.

Suggested Functions In Module 2

Module `solver.py` could contain:

- `WALL` a constant that tells the solver which character represents a wall (you may default it to `#`)
- `START` a constant that tells the solver which character represents the start cell
- `GOAL` a constant that tells the solver which character represents the goal cell
- `PATH` a constant that tells the solver which character represents the visited cells - our taken path.
- `solve(maze, startpos, goalpos)`: starts the search for the path by calling the initial recursive search (where `startpos`, and `goalpos` are tuples with coordinates)
- `solve_recursive(maze, currentpos, goalpos)`: recursively searches for a path through the maze by backtracking
- `findstart(maze)`: returns the coordinates of the start position
- `findgoal(maze)`: returns the coordinates of the goal position

Suggested Functions In Module 3

Module `fileIO.py` could contain:

- `MAZEFOLDER` a constant which represents the relative path to the folder in which the mazes are located
- `get_maze_list()`: returns a list of filenames for available mazes. This relates to the *Small-ish Bonus Task 1*
- `load_maze(filename)`: returns the maze from the given file name inside the `MAZEFOLDER`
- `build_maze(raw_maze)`: takes a maze read from a file and builds the maze's grid-like structure that we need to work with, then returns the grid
- `stringify_maze(grid)`: converts a grid back into a maze that can be written into a file so it looks like the others
- `save_maze(maze, filename)`: saves a maze into a file with filename in the `MAZEFOLDER`
- `save_solved(maze, filename)`: saves a solved maze into a file with file name in the `MAZEFOLDER`, maybe with a special **solved** suffix This can of course use `save_maze` in the end so you don't duplicate code
- `get_maze_grid(filename)`: consecutively calls `load_maze()` and `build_maze()` to return a grid. This is a convenience function
- `save_maze_grid(grid, filename, solved=True)`: consecutively calls `stringify_maze()` and on of the save functions. Also a convenience function

Suggested Functions In Module 4

Module `userIO.py` could contain:

- `show_welcome()`: shows some introductory words, and maybe a selection of available options to choose from
- `present_mazes(maze_list)`: displays the mazes to choose from
- `get_user_choice(maze_list)`: gets the choice of the user, and checks if it is a valid choice, then returns the corresponding maze name
- `choose_maze(maze_list)`: consecutively calls `present_mazes` and `get_user_choice` to choose a maze A convenience function
- `print_maze(maze)`: prints the maze onto the terminal, however fancy you would like to make it
- `read_maze()`: reads in a maze from the user and returns it. This is related to *Small-ish Bonus Task 2* and might require some more functionality / functions