CSSE1001

Semester 2, 2013
Assignment 1
10 marks

Due Thursday 29 August, 2013, 9:30am

A'maze'ing

1 Introduction

For this assignment you will write a program for exploring a maze. The program you will write is a simple text-based interactive program that loads a maze from a text file and provides commands for moving about the maze.

The files `maze1.txt`, `maze2.txt and maze3.txt` are text files containing representations of simple mazes. The contents of `maze1.txt` is given below.

```
#####
#   #
### #
#X  #
#####
```

The `#`'s represent walls and the spaces represent open squares. The single `X` represents the destination (Finish) square (`X` marks the spot).

Maze files always contain a space in row 1, column 1 (counting from 0) - this is the start square. Each maze file also contains a single finish square and there should be at least one path from the start square to the finish square that follows the squares containing spaces. The first and last rows and columns are all `#`'s.

For your assignment, positions of squares are represented as row, column pairs. We say that a square is **legal** if and only if the square contains a space or an `X`. We say a move from one square (position) to an adjacent square is **legal** if and only if the adjacent square is legal. We also say that a direction is legal if a move in that direction is legal.

Here are two examples of what is expected from your program. The input is everything after `Command:` on a line (and the initial `maze1.txt`). Everything else is output. Your output should be exactly the same as below for the given input.

```
Maze File: maze1.txt
#####
#O  #
### #
#X  #
#####
Command: s
You can't go in that direction
#####
#O  #
### #
#X  #
#####
Command: ?
? - Help.
n - move North one square.
s - move South one square.
e - move East one square.
w - move West one square.
r - Reset to the beginning.
b - Back up a move.
p - List all possible legal directions from the current position.
q - Quit.

#####
#O  #
### #
#X  #
#####
Command: a
Invalid Command: a
#####
#O  #
```

```
### #
#X  #
#####
Command: e
#####
# O #
### #
#X  #
#####
Command: e
#####
#  O#
### #
#X  #
#####
Command: b
#####
# O #
### #
#X  #
#####
Command: b
#####
#O  #
### #
#X  #
#####
Command: b
#####
#O  #
### #
#X  #
#####
Command: e
#####
# O #
### #
#X  #
```

```
#####
Command: e
#####
#  O#
### #
#X  #
#####
Command: p
Possible directions: s,w
#####
#  O#
### #
#X  #
#####
Command: s
#####
#   #
###O#
#X  #
#####
Command: s
#####
#   #
### #
#X O#
#####
Command: q
Are you sure you want to quit? [y] or n: n
#####
#   #
### #
#X O#
#####
Command: w
#####
#   #
### #
#XO #
```

```
#####
Command: w
Congratulations - you made it!
```

Another example.

```
Maze File: maze1.txt
#####
#O  #
### #
#X  #
#####
Command: q
Are you sure you want to quit? [y] or n: y
```

The program first asks for a maze file. You can assume the file supplied is in the correct format.

The command `?` is the help command. It lists all the commands. (The help string is supplied in `assign1.py`.)

The commands `n,s,e,w` attempt to move you one square in the direction given. These commands will output an error message if that direction is illegal.

The command `r` resets the maze exploration to the start square with no moves made.

The command `b` backs up one move. If the command is used, for example, 3 times in a row then the last 3 moves will be undone. If you have backed up all the way to the beginning of the exploration then this command does nothing.

The command `p` lists all the possible legal directions from the current position.

The command `q` is used to quit before reaching the destination.

Any other 'command' is treated as invalid and an error message is printed.

2 Assignment Tasks

For each function that you write you need to provide a suitable comment giving a description, the type and any preconditions. You should use the triple-quote commenting style.

2.1 Download files

The first task is to download the example maze files `maze1.txt`, `maze2.txt` and `maze3.txt`.

We suggest you create a folder in which to write your solution and put these files in that folder.

The file `assign1.py` is for your assignment. Add your name and student number in the space provided. **Do not otherwise modify this comment block or the supplied code and add your code after this**. When you have completed your assignment you will submit the file `assign1.py` containing your solution to the assignment (see Section 4 for submission details).

2.2 Write the code

Finally, write your solution to the assignment making sure you have included suitable comments. There are several functions you need to write and these are described below. **Do not use global variables in your code.** Global constants like `HELP` are OK.

2.2.1 Load Maze

`load_maze(filename)` takes the name of a file containing maze information as described in the introduction and returns a list of lists representing the maze. For example (the actual value displayed in IDLE will be on one line rather than over several lines as below):

```
>>> load_maze('maze1.txt')
[['#', '#', '#', '#', '#'],
```

```
 ['#', ' ', ' ', ' ', '#'],
 ['#', '#', '#', ' ', '#'],
 ['#', 'X', ' ', ' ', '#'],
 ['#', '#', '#', '#', '#']]
>>>
```

The result is a list of rows of the maze and each row is a list of characters representing the column elements.

NOTE: In the example above `load_maze` returns the list of lists and this result is printed by the interpreter. All functions, except `print_maze` and `interact`, DO NOT print out anything.

When reading information from files please use `open(filename, 'rU')` in order to get operating system independent processing of newlines.

2.2.2 Getting an Adjacent Position

`get_position_in_direction(position, direction)` takes a row, column pair representing the position of a square, and a direction character (one of `nsew`) and returns the position of the adjacent square in the given direction. For this function you do not check if that would be a legal move.

For example :

```
>>> get_position_in_direction((2,3), 'e')
(2, 4)
>>> get_position_in_direction((2,3), 's')
(3, 3)
>>>
```

2.2.3 Make a Move

`move(maze, position, direction)` takes, as arguments, the maze (as a list of lists), a position of a square and a direction and returns a pair of the form `(pos, kind)` where `pos` is the position after the move and `kind` provides feedback on the result of the move where the value `' '` means the move was OK; `'X'` means the move was OK and the move takes us to the

desitination square; and `'#'` means the move is invalid. When the move is invalid the new position returned is the same as the old position.

For example:

```
>>> maze = load_maze('maze1.txt')
>>> move(maze, (1,1), 's')
((1, 1), '#')
>>> move(maze, (1,1), 'e')
((1, 2), ' ')
>>> move(maze, (3,2), 'w')
((3, 1), 'X')
>>>
```

2.2.4 Print the maze

`print_maze(maze, position)` takes, as arguments, the maze and the position of the player and prints the maze with the player shown as an `'O'`.

For example:

```
>>> maze = load_maze('maze1.txt')
>>> print_maze(maze, (1,1))
#####
#O  #
### #
#X  #
#####
>>>
```

2.2.5 Where can I go

`get_possible_directions(maze, position)` takes, as arguments, the maze (as a list of lists) and a position of a square and returns a list of legal directions for that square. You may assume the given position is legal.

For example:

```
>>> maze = load_maze('maze1.txt')
>>> get_possible_directions(maze, (1,2))
```

```
['e', 'w']
>>> get_possible_directions(maze, (1,3))
['s', 'w']
```

2.2.6 The Top-Level Interface

`interact()` is the top-level function that defines the text-base user interface as described in the introduction. Note that when either the user quits or when the finish square is found the `interact` function should exit.

2.2.7 Hints

For file and string processing: `open, strip, join`
For user interaction: `raw_input`
For supporting the `B` command you need to keep track of the move history. This can be done, for example, with a list using `append` and `pop`.

3 Assessment and Marking Criteria

In addition to providing a working solution to the assignment problem, the assessment will involve discussing your code submission with a tutor. This discussion will take place in the practical session you have signed up to in week 7. You **must** attend that session in order to obtain marks for the assignment.

In preparation for your discussion with a tutor you may wish to consider:

- any parts of the assignment that you found particularly difficult, and how you overcame them to arrive at a solution;
- whether you considered any alternative ways of implementing a given function;
- where you have known errors in your code, their cause and possible solutions (if known).

It is also important that you can explain to the tutor how each of the functions that you have written operates (for example, if you have used a for loop or a while loop in a function, why this was the right choice).

Marks will be awarded based on a combination of the correctness of your code and on your understanding of the code that you have written. A technically correct solution will not elicit a pass mark unless you can demonstrate that you understand its operation.

We will mark your assignment according to the following criteria.

Criteria	**Mark**
Your code is mostly complete, correct, clear, succinct and well commented. You are able to explain your code.	8 - 10
Your code has some problems OR you have some problems explaining your code.	4 - 7
Your code is clearly incomplete, incorrect, too complex or hard to understand OR you have major problems explaining your code.	1 - 3
Your work has little or no academic merit.	0

A partial solution will be marked. If your partial solution causes problems in the Python interpreter please comment out that code and we will mark that.

Please read the section in the course profile about plagiarism.

4 Assignment Submission

You must submit your completed assignment electronically through Blackboard.

Please read
`http://www.library.uq.edu.au/ask-it/blackboard-assessment`
for information on submitting through Blackboard.

You should electronically submit your copy of the file `assign1.py` (use this name - all lower case).

You may submit your assignment multiple times before the deadline - only the last submission will be marked.

Late submission of the assignment will not be accepted. In the event of exceptional personal or medical circumstances that prevent you from handing in the assignment on-time, you should contact the lecturer in charge and be prepared to supply appropriate documentary evidence. You should be prepared to submit whatever work you have completed at the deadline, if required. Requests for extensions should be made as soon as possible, and preferably before the assignment due date.