CompSci130 2021, S2 Assignment

In this assignment you will have to implement a variation of the game Sokoban.

```
* * * * * * * *
*             *
* P   #       *
* * * * *   # *
* o           *
*           o *
* * * * * * * *
```

The aim of this game is for the player, uppercase ‘P’, to push crates, ‘#’s, to fill holes in the ground marked by lowercase ‘o’s.

Game Rules

The rules of this version of Sokoban are as follows:

- The game is played on a two-dimensional grid.
- Each cell of the grid can contain:
  - The **Player** represented by an uppercase P character, “P”
  - A **floor tile** represented by a space character, “ ”
  - A **wall** represented by an asterisk character, “*”
  - A **crate** represented by a hash character, “#”
  - A **hole** represented by a lower-case o character, “o”
- Each turn, the player character can move one unit up, down, left, or right on the grid.
- A player character cannot move into a wall or a hole.
- Each turn, a player character can push a crate one unit in the direction they are trying to move if and only if the next grid cell after the crate in the direction the player is trying to move is a floor tile or a hole. If this condition is not met, neither the player nor the crate move.
- If a player pushes a crate into a cell containing a hole, the hole and the crate disappear, leaving a floor tile in their place. The player still moves into the space previously occupied by the crate before the move.
- If the player attempts to move off the edge of the screen or push a crate off the edge of the screen, the player or the crate should appear on the opposite side of the screen as if the two sides of the screen are connected if the other rules allow.

Interface

You will need to implement a class '**Sokoban**' that implements the following methods:

__init__(self, board)	Creates a Sokoban instance with a given board parameter. The board parameter is a nested list containing the characters of the starting board.
find_player(self)	Returns a tuple containing the row and column of the player character's position on the board. Rows and columns start at 0. In the example on page 1, the player is at position (2, 1)
complete(self)	Returns True if the game is over (i.e. there are no remaining holes) and False otherwise.
get_steps(self)	Returns the number of moves the player character has made where the player's position has changed.
restart(self)	Resets the Sokoban instance to the state it was in before the player started the game.
undo(self)	Undoes the last move made by the player so that the game state is as if the move had never occurred. Can be called repeatedly to undo multiple moves. If undo is called more times than the number of moves the player has made, the board should remain in its initial state.
move(self, direction)	Attempts to move the player by one position and push any crates that are in front of the player as according to the game rules described on page 1. The direction parameter is a string with the value 'w', 'a', 's', or 'd' representing the directions up, left, down, and right respectively. Moves are only counted if the player's position is updated. If the player's position has not changed, the game's state should not change in any way.
__str__(self)	Returns a string representation of the board. Rows are separated by a newline character and cells within each row are separated by a space character.

Boards

Boards will be provided to the Sokoban constructor as a two-dimensional list of characters with each character representing a cell.
For example, the following list:

```
board = [
      ['*', '*', '*', '*', '*', '*', '*', '*'],
      ['*', ' ', ' ', ' ', ' ', ' ', ' ', '*'],
      ['*', ' ', 'P', '#', ' ', ' ', ' ', '*'],
      ['*', '*', '*', '*', '*', ' ', '#', '*'],
      ['*', 'o', ' ', ' ', ' ', ' ', ' ', '*'],
      ['*', ' ', ' ', ' ', ' ', ' ', 'o', '*'],
      ['*', '*', '*', '*', '*', '*', '*', '*']
]
```

Represents the following grid:

```
* * * * * * * *
*             *
*   P #       *
* * * * *   # *
* o           *
*           o *
* * * * * * * *
```

main()

A main(board) function has been provided that runs and controls the game. The method takes a board parameter as a two-dimensional list.
You must not edit this function.

Submission

Submit all files to the Assignment Dropbox (https://adb.auckland.ac.nz/). No marks will be awarded if your program does not run.

You are to electronically submit the following file:

- assignment.py

A template file has been provided in the assignment page on Canvas.

Do NOT have any code outside of the Sokoban class and main() function in your submission.
Do NOT modify the provided main(board) methods in the template file.
Do NOT include any other documents in your submission.

You may electronically submit your assignment through the Web Dropbox (https://adb.auckland.ac.nz/) at any time from the first submission date up until the final date. You can make more than one submission; however, every submission that you make replaces your previous submission.
Only your very latest submission will be marked. Please double check that you have submitted the correct file.

Marking

The assignment marks are distributed as follows:

Name, UPI, and program description at the top in a docstring	1 Mark
__str__() works	1 Mark
find_player() works	1 Mark
complete() works	1 Mark
restart() works	1 Mark
get_steps() works	1 Mark
undo() works	5 Marks
move(direction) works	5 Marks
Complete Program	5 Marks
Code follows the style conventions described below	5 Marks
TOTAL	**26 Marks**

Style

The style conventions that will be checked are:

- All instance variables should be private with two leading underscores.
- One blank line between methods.
- Meaningful variable names that follow Python naming conventions.
- A single space character on either side of operators (+, -, *, /, =, ==, etc).
- A single space after commas but not before.
- This program is simple enough that none of your lines of code should be longer than 80 characters.
- None of your methods should be longer than 20 lines — you may need to define helper methods to achieve this.

I have attached the style conventions that were outlined in the first lecture.