Assignment 2: Lecture Section 2 (6-7pm) Instructions

Adventure Time

Download all starter files on our course website.

Part 1: Simple Maze Game

Recommended 'Deadline': Thursday February 22, 2018
This section is worth 10% of your total grade for this assignment.

For this part of the assignment, you will be creating a simple maze game (as you may have guessed from the creative title above).

The maze will be stored as a list of lists (called "grid" in your starter code).
Each sublist represents one row of the maze.

In the maze, the string (o) represents the player.
The string (_) represents an empty space which the player can move to.
The string (*) represents the treasure which the player should try to reach.

Once the player reaches the treasure, they win the game.

As an example, a grid that's like this:
`[['(o)', '(_)'], ['(_)','(_)'], ['(_)', '(*)']]`

Will be printed out like this:
```
(o)(_)
(_)(_)
(_)(*)
```
(The print method is already completed for you in the starter code provided.)

The game begins by asking the player for a width, and a height. (e.g. The above grid has a width of 2, and a height of 3.)

From here, the player has the option to move either North, South, East or West.
This is managed using coordinates.

An important note about (x, y) coordinates:
The player always begins at the coordinates (0, 0). The x-y coordinate plane here is represented with (0,0) as the top left corner. From there, x increases to the right, and y ***increases*** to the bottom. So, for the grid above, the player (x) is considered to be at position (0,0), the treasure (o) is considered to be at position (1,2).

Here is the output of a sample play-through of the game.
Note that if a move given by the user is invalid (e.g. when the player tries to move North at the beginning) nothing changes in the grid, because that falls outside of the grid's boundaries. Same thing happens again when they try to move West. There is no empty space to the West, so nothing changes in the grid.

```
Width: 2
Height: 2
(o)(_)
(_)(*)
Which way would you like to move? Choose N, S, E, W. N
(o)(_)
(_)(*)
Which way would you like to move? Choose N, S, E, W. W
(o)(_)
(_)(*)
Which way would you like to move? Choose N, S, E, W. S
(_)(_)
(o)(*)
Which way would you like to move? Choose N, S, E, W. W
(_)(_)
(o)(*)
Which way would you like to move? Choose N, S, E, W. N
(o)(_)
(_)(*)
Which way would you like to move? Choose N, S, E, W. E
(_)(o)
(_)(*)
Which way would you like to move? Choose N, S, E, W. S
Congratulations! You won.
```

Part 2: Adventure game

This section is worth 70% of your total grade for this assignment.

For this part, you'll be writing code to produce a very simple text-based adventure game. You'll do so by finishing the code in **part2_adventure.py,** so that the final game runs like this:
Winning – https://mcs.utm.utoronto.ca/~108s18/assignments/a2/win1.mov
Game over 1 – https://mcs.utm.utoronto.ca/~108s18/assignments/a2/gameover1.mov
Game over 2 – https://mcs.utm.utoronto.ca/~108s18/assignments/a2/gameover2.mov

I. *What is the starter code?*

Your first step would be to look through the starter code you're provided with to figure out what it does, and what pieces are missing that you need to fill in.

Notice that the main program is already written for you (at the bottom of the file). This is where the game begins. The player is asked for a one letter identifier. Their initial location is set to the coordinates (0, 0), and an empty inventory list is initialized.

Next, the game data is gathered using the three text files that you're provided with (more information about the text files below).

Using the game data, we figure out information about the current location (using the player's coordinates, and the map data which associates each (x, y) coordinate in the map with a location ID). While the game has not been won or lost, the main loop which runs the game will keep iterating.

II. *What are all the text files?*

map.txt

The map you're dealing with this time is a little more complex than the one from part 1. Take a look at the given map.txt file. You'll find a grid of numbers.

In your adventure game, each location will have a numerical ID associated with it. The numbers in the map represent these IDs. Some of the special IDs are as follows:

- 1 is where the player begins the game
- -1 is where the player wants to end up to win the game
- 0s represent places where nothing special occurs
- -2s represent places that are blocked off (e.g. maybe by a wall), and are thus inaccessible to the player. This is just meant to make the map a little more interesting, so the player has to work around some inaccessible areas to reach their destination.

In your Python file, one of your functions (load_map) will deal with reading data from a file structured like this, and turning it into a list of lists as described in the load_map docstring.

locations.txt

This file contains all the information about each location that you can visit during the game. Each location has a numerical ID which corresponds to the map. The structure of this file is as follows:

```
Location ID
[BEGIN DESCRIPTION]
Description...
...
[END DESCRIPTION]
```

This structure is repeated for each location. Some locations may also have an extra section for possible actions that can be taken at that location. This section, if it's there, will be structured as follows:

```
[BEGIN ACTIONS]
Action text, The numerical ID of where you'll end up by doing that action
Action text, The numerical ID of where you'll end up by doing that action
...
[END ACTIONS]
```

In your Python file, one of your functions (load_locations) will deal with reading data from a file structured exactly like this, and turning it into data structures as described in Section III below.

Hint: Remember that the file you're dealing with has this exact format for each location (hence, a loop that deals with each location info, one by one, and an if statement within it that deals with the actions section if it occurs, should be used within this function).

items.txt

This file includes information about each of the three items in the game in the following format:

```
Item name, description, location ID of where the item is found, ID of where item can be used
```

Each item's information is stored as one line, so each item has its own line.

In your Python file, one of your functions (load_items) will deal with reading data from this items file and gathering information about each item, as described in Section III below.

Hint: A loop that deals with each line, line by line, and the split method to split the data within each line would be useful here.

III. *What do I have to do?*

Complete all the functions in the **part2_adventure.py** file so the game runs as shown in the videos above.

Here are some more important details about two of the load functions:

load_items should return a dictionary where each key is an item's name, and each value is a list with three elements in this order: the item's string description, the item's float starting location ID, and the item's float ending goal location ID.

For example, if our file had two items like:

```
t-card,ID card for school,3.1,-1
paintbrush,My favourite painting tool,4.2,-1
```

Then the dictionary returned by this function would be:

```
{"t-card": ["ID card for school", 3.1, -1], "paintbrush": ["My favourite painting tool", 4.2, -1]}
```

load_locations should return a dictionary that is structured as follows:

- Each key is a float that is the location ID
- Each value is a list of two elements:
    - The first element is a string location description
    - The second element is a sub list
        This sublist contains any possible actions at this location.
        If no actions are possible, the sublist will be empty.
        If some actions are possible, each action will be stored as
        a tuple inside this sub list.
        - Each action's tuple will have two elements:
            - The string description of the action, and
            - the float location ID of where you end up if you do the action.

For example, for the locations.txt file that you're given, some of the items in the dictionary you should return would be as follows:

```
"{0.0: ['There is nothing interesting here.', []],
```

```
  1.0: ['You wake up lost in some unfamiliar place. All you know is you need to find a way
home. And fast!', []],
  2.0: ["There is some sort of an information desk with a receptionist sitting behind it. But
they look very busy and don't seem to notice you at all.", [("Clear your throat to get the
receptionist's attention", 2.1), ("Bang the table to get the receptionist's attention", 2.1), ('Wait
quietly until the receptionist notices you', 2.2)]]
  ...
```

And so on, for each of the locations in the file.

All other information is included within the provided docstrings in the .py file.

Part 3: Making It Cooler!

This section is worth 20% of your total grade for this assignment.

For this section, you will be working within the folder **part 3** which has a blank **part3_adventure.py** file, and three blank map, location and items text files. You can copy your solution from **part2_adventure.py** into the new file called **part3_adventure.py** file, to get started.

You may also copy the map, location and items data if you like, although you're probably going to be making a lot of changes to these files for this part as described below.

For this section, we want you to add things and change things about your game to make it better! What things should you add? That's completely up to you.

Some recommendations (none of these are requirements; just suggestions of ways you can improve the game):

- You can change the story and setting of the game entirely, including location and item information, when game overs occur, when victory occurs, etc.
- You can limit the number of moves that the player can make before they have to reach the goal location, and if they don't reach it within the allowed num of moves then time is up and game is over.
- You can change the map. Make it more interesting, more maze-like with more areas blocked off. Add in more locations. Or you can hide the map from the player so they only see text, to make them feel even more lost, etc.
- You can also add in more items. Allow the user to use the items within the game in different ways to obtain more items, or make the story progress, etc.

One key recommendation is adding in more **puzzles** to the game. For a good description of the reasoning behind puzzles, and common types of puzzles, see this page all about puzzles:

https://h2g2.com/edited_entry/A22196289

Some puzzles are rather elaborate, relying on performing specialized verbs on multiple items or imposing specific timing requirements. Your puzzles can be simple -- finding a key in a remote location from the door that it opens -- or more complex. Regardless, implementing puzzles will require that you extend the commands you support and/or extend the information you store in your data files. You could also add in **mini-games** within certain locations of your game.

The text files that you use for this part can either follow the same format that the files you used for Part 2 had, or they can be different. The only requirement is that **ALL** game data must be stored inside these text files, and read by your **part3_adventure.py** file. **DO NOT** hard-code anything about the game locations/items/etc. directly into your Python file. Store them in the text files, and read them in.

If you want to change around these text files, add in other extra text files, etc. you are free to do so!

For inspiration, you can play some actual text adventure games online like Zork:
http://www.web-adventures.org/cgi-bin/webfrotz?s=ZorkDungeon

IMPORTANT: Assessment for Part 3

For this section, you will need to complete the text file found in the **part3** folder called **process_log.txt**. This text file is meant to be a report of your process as you worked on this part of the project.

You MUST complete and hand in this file to earn a mark for Part 3. Your grade for this part will mainly be based on the quality and content of your log.

What to include in the log? Detailed descriptions about the steps you took to code your enhancements. Address the following points:

- What enhancements did you want to add? (List any that you tried to add, even if your attempts didn't work out)
- For attempts that didn't work: Why did it not work? What did you struggle with?
- What worked? What did you enjoy?
- If you had more time, what would you have added?
- Any inspiration or online resources you may have used to help you with enhancements
- etc.

Even if you did NOT finish adding in a feature, but made considerable attempts at doing so, MAKE SURE to include this within your log file.

As mentioned before, this log will be the **main source of your grade for this part**, so make sure it's clear, detailed and honest about your process as you worked on this project.

You will be handing in the entire part3 folder as a zip file.

What you will be handing in:

Submit all the following files on MarkUs by Sunday, March 4, 11:59 P.M.:

1. `part1_maze.py`
2. `part2_adventure.py`
3. `part3.zip`