ICS 31, Summer Session 10-wk, 2017

Lab 5

For this lab you will reimplement your text adventure game from lab 4 using classes. You will also add a few features to your text adventure game.

Part 1

Define a class called Room to represent a room in your dungeon. You will reimplement your text adventure game to use your Room class to represent rooms rather than the namedtuple that you used in lab 4.

The Room Class

Your Room class will include instance variables to contain all of the information about a room. Your room class should have the following instance variables:

1. ***Room number***: This is a positive integer which uniquely identifies the room.
2. ***Description***: This is a string which is printed when the player enters the room.
3. ***North room***: This is the room number of the room immediately to the north of this room. If there is no room to the north of this room then this value is -1.
4. ***South room***: This is the room number of the room immediately to the south of this room. If there is no room to the south of this room then this value is -1.
5. ***East room***: This is the room number of the room immediately to the east of this room. If there is no room to the east of this room then this value is -1.
6. ***West room***: This is the room number of the room immediately to the west of this room. If there is no room to the west of this room then this value is -1.

For this part of the project, your Room class will only need one method, its constructor. The constructor should initialize all of the instance variables.

Changes to loaddungeon

Your text adventure from lab 4 created a new Room namedtuple to contain the information about each room, and all of the Room namedtuples were placed in a dictionary. You will change this so that a new instance of your Room class is created to contain the information about each room instead. A dictionary will still be used to store the Room objects.

Part 2

You will enhance your game to allow the player to have an inventory of items that he is carrying, and allow each room to have a **roomInventory** which is an inventory of items in the room. An inventory will be represented as a dictionary where each key is a string containing the name of an item, and the

corresponding value is the count of how many of each item is contained in the inventory. For example, an inventory containing 2 apples and 1 orange would be represented as the following dictionary:

```
{'apple': 2, 'orange': 1}
```

There should be one global variable called **playerInventory** which is the current inventory of the player. In order to assign an inventory to each room you will add a new instance variable to the Room class called **roomInventory** which is the inventory of items in a room. Assume that the player inventory and all room inventories are initially empty.

You will change your program to accept the following 4 new commands from the user.

1. **roomcontents**: This command takes no arguments and prints out the inventory of the current room. For example, if the current room contains 1 orange and 2 apples, the roomcontents command would print the following:
   `This room contains (orange 1)(apple 2)`
2. **take**: This command removes an item from the inventory of the current room and adds that item to the inventory of the player. The command takes one argument, the name of the item to be taken. The take command should only take 1 item from the room, even if the room contains many items of the requested type. So if the room has 10 apples and the player issues the command "`take apple`", then the user's inventory should contain one more apple and the room's inventory should contain 9 apples. If the item is not contained in the room's inventory the following message should be printed, "That item is not in this room".
3. **drop**: This command removes an item from the inventory of the player and adds that item to the inventory of the current room. The command takes one argument, the name of the item to be dropped. If the player has multiple items of the requested type then only 1 item should be moved to the inventory of the current room. If the item is not contained in the player's inventory the following message should be printed, "You don't have that item".
4. **inventory**: This command prints out the contents of the player's inventory. For example, if the player has 2 apples and one orange, the following would be printed: `apple 2, orange 1`

You will also need to change the behavior of your adventure game when a player enters a room. In the original text adventure game, the description of a room is printed when the player enters a room. You will modify that so that the contents of the **roomInventory** are also printed when a room is entered. For example, if a player enters a room which contains 1 orange and 2 apples, and has the description "This is room 1", then the following should be printed:

```
This is room 1
This room contains (orange 1)(apple 2)
```

If the room has no items in it then only the description should be printed when the room is entered.

Part 3

You will add support for a new command called **loaditems** which will take one argument, the name of an items file which describes the items which are added to the player's inventory.

Items File Format

The items file should have only one line. The data on the line is a set of pairs which include the name of an item and the count of that item. All items on a line in the items file are separated by spaces. For example, suppose we want to create an items file which adds 10 apples and 20 oranges to the player's inventory. The items file would look like this:

```
apple 10 orange 20
```

Every time the **loaditems** command is used, the described in the file will be added to the player's inventory. If the player's inventory already contains some items then the new items will be added in addition to the current items in the player's inventory.

Additional Requirements

- Be sure to define a function called StartDungeon() which starts the execution of your code. At the bottom of your file you should call StartDungeon() to begin running your code.
- Be sure to include comments describing every function that you write. For each function you should have a one-line docstring describing the function, and a comment before the function definition describing the inputs and outputs of the function, and their types.