CIT 590 - Spring 2018 - HW3: Tower Blaster

General Idea

This assignment is mainly intended to get you to working with Python lists.

We will be implementing the game Tower Blaster, which is a game that involves re-arranging a group of bricks in order to have an increasing sequence.

While Tower Blaster is often played with 2 human players, we will keep this simple and just play the user versus the computer. The user's moves are decided by the user by asking for input; the computer's moves are decided by you: the programmers!

There is NO right answer for the section that asks you to program a strategy for the computer. All we want you to do is come up with some reasonable enough strategy that ensures a human user does not always beat the computer. So, unlike your previous assignments, this one has a creative component to it.

You must use Python lists for this assignment. Some of you might have seen this package called numpy. You are NOT allowed to use numpy at all.

Rules of the Game

Go to https://www.gamefools.com/onlinegames/free/towerblaster.html to get a feel for the game. Our game will be similar but read our specifications (importantly we won't have levels).

A Tower Blaster game starts with a pile of 60 bricks, each numbered 1 to 60. Think of the numbers on the bricks as the width of the bricks. The objective is to be the first player to arrange the 10 bricks in your tower from lowest to highest (because the tower will be unstable otherwise).

The bricks in the pile are shuffled at the start and both the user and the computer are dealt 10 bricks from the main pile. As a player receives each brick, they must place it on top of their current tower in the order it is received. *Yes, initially your tower is likely to be unstable.*

After the first 10 bricks are dealt, there will be 40 bricks remaining in the main pile. The top brick of the main pile is turned over to begin the discarded brick pile.

On each player's turn, the player chooses to pick up the top brick from the discard pile or to pick up the top brick from the main pile. The top brick from the discard pile is known. In other words, the discard file is 'face up' and everyone knows how wide the top brick is. The main pile is 'face down'. Choosing the top brick from the main pile can be risky because the player does not know what the brick is and the brick cannot be put back down onto the main pile.

Once a player chooses a brick, either from the discard pile or the main pile, the player decides where in the tower to put the brick. The tower is always 10 bricks high, so placing a brick means that an existing brick in the tower is removed and replaced with the new brick.

If the player takes a brick from the main pile (the one that is 'face down'), the player can reject it and place it in the discard pile. This means that nothing in that player's tower changes during that turn. If the player takes a brick from the discard pile (the one that is 'face up'), the player MUST place it into the tower.

The first player to get their 10 bricks in order wins.

The Actual Program

Below you will find explanations of the functions that need to be written. We are expecting to see these functions with these names and these method signatures. Do not change the names of these functions.

You will also have to write some functions by yourself in addition to the ones listed below. Feel free to name those functions whatever you want, as long as they happen to reflect what the function does.

Note that we will make heavy use of lists in the assignment. Since a tower looks more like a vertically oriented structure, we need to have some convention. Our convention is that the tower in the picture to the right is going to be represented as [50, 12, 1, 37, 32, 11, 24, 42, 10, 3]

We know this is a somewhat awkward way to represent the data, but we are deliberately asking you to do it this way in order for you to do more list exercises.

The main pile and discard pile are also going to be represented as lists. Note that in both the main pile and the discard pile, you should only have access to the top. You will have to use the pop function or the append function in some manner.

Required Functions:

- shuffle(bricks)
  - Shuffle the bricks (represented as a list) to start the game.
  - This function does not return anything.
  - You are allowed to import the random module and just use random.shuffle.
  - Also shuffle the discard pile if we ever run out of bricks and need to 'restart' the game.
    - In this case, shuffle the discard pile and move those bricks to the main pile.
    - Then, turn over the top card to be the start of the new discard pile.

- check_tower_blaster(tower)
  - Given a tower (the user's or the computer's list), determine if stability has been achieved.
  - Remember that stability means that the bricks are in ascending order.
  - This function returns a boolean value.

➔ get_top_brick(brick_pile)

- ◆ Remove the top brick from any pile of bricks. This can be the main_pile, discard, or your tower or the computer's tower. In short, the first element of any list.
- ◆ So, it is used at the start of game play for dealing bricks. This same function will also be used during each player's turn to take the top brick from either the discarded brick pile or from the main pile.
- ◆ This function must return an integer.

➔ deal_initial_bricks(main_pile)

- ◆ Start the game by dealing two sets of 10 bricks each.
- ◆ This function returns two lists - one representing the user's hand and the other representing the computer's hand.
- ◆ The method of returning 2 things from a function is to make a tuple out of the return values. For an example of this refer to the lecture slides/code where we return both the maximum and minimum in a list.
- ◆ Make sure that you follow normal brick game conventions of dealing. So you have to deal one brick to the computer, one to the user, one to the computer, one to the user and so on.
- ◆ The computer is always the first person that gets dealt to and always plays first.
- ◆ Remember that the rules dictate that you have to place your bricks one on top of the other. In the earlier picture, this would mean that someone was dealt 3, 10, 42, …, in that order.

➔ add_brick_to_discard(brick, discard)

- ◆ Add the brick (represented as an integer) to the top of the discard pile (which is a list).
- ◆ This function does not return anything.

➔ find_and_replace(new_brick, brick_to_be_replaced, tower, discard)

- ◆ Find the brick to be replaced (represented by an integer) in the tower and replace it with new_brick.
- ◆ The replaced brick then gets put on top of the discard pile.
- ◆ Check and make sure that the brick_to_be_replaced is truly a brick in the hand. If the user accidentally typed a brick number incorrectly, just politely tell them to try again and leave the hand unchanged.

- computer_play(tower, main_pile, discard)
  - This function is where you can write the computer's strategy down. It is also the function where we are giving you very little guidance in terms of actual code.
  - You are supposed to be somewhat creative here, but I do want your code to be deterministic. That is, given a particular tower and a particular brick (either from the discarded brick pile or as the top brick of the main pile), you either always take the brick or always reject it.
  - Here are some potential decisions the computer has to make
    - Given the computer's current tower, do you want to take the top brick on the discarded brick pile or do you want to take the top brick from the deck and see if that brick is useful to you.
    - How do you evaluate the usefulness of a brick and which position it should go into.
    - There might be some simple rules you can give the computer. For instance, it is disastrous (provably so) to put something like a 5 at the very bottom of the tower. You want big numbers over there.
  - You are allowed to do pretty much anything in this function except make a random decision or make a decision that is obviously incorrect. For instance, making your bottom brick a 5 is a recipe for disaster.
  - Also, the computer CANNOT CHEAT. What does that mean? The computer cannot peek at the top brick of the main pile and then make a decision of going to the discard. It's decision making should be something that a human should be able to make as well.
  - This function returns the new hand for the computer.

- main()
  - The function that puts it all together.
  - You play until either the user or the computer gets Tower Blaster.
  - Unlike the previous HW, we want you to assemble the functions into the main function by yourself. Think about how the game should proceed and which function should go where.

Evaluation

The primary goal of this assignment is to get you to feel familiar with lists and to have some level of fun while creating a game.

While we want you to spend time on coming up with some kind of strategy for the computer, that is NOT the primary part of the assignment. Any simple, but reasonable strategy will have you doing some fun things with lists. If the user always does absolutely nothing at all, that is, if they reject the top discarded brick and they reject the top brick from the main pile and move it onto the discard, then we want the computer to win Your computer player should be smart enough to beat the 'stupid, lazy' user.

Also, remember the human player has no idea what you are doing internally in your functions and does not want to be shown a large amount of print statements. At any given point in the game, they should know only 3 things: their entire tower (printed in list form), the top brick of the discard pile, and, if they choose to pick from the main pile, they get to know the top brick of the main pile.

You will be evaluated on 4 things:

1. Does your game work? Again, please do not worry about the TAs trying to crash your program. Just ensure that reasonable inputs will allow a user to play a game.
2. Strategy - is your computer's strategy something that makes sense? Please comment your code for that part of the homework very thoroughly.
3. Usability - This is going to be a subjective evaluation. You are making a game. A user who knows the rules of Tower Blaster should be able to play it without too much trouble. In particular, they should not have to read a single line of your code in order to operate the game.
4. Style - The usual stuff here. Variable names, function names, comments. Style will always be part of your evaluation.

Data Structures

In this HW **you are only allowed to use lists and tuples**. Do not use dictionaries, classes or any in built python libraries(other than the random library for shuffling). Remember that the deck, the discard pile and the two towers (user and computer) are just lists.

Submission

Submit a single file called tower_blaster.py. Remember to put the following lines at the end of your file:

```
if __name__ == '__main__':
    main()
```

Optionally, you can submit a text file that explains your strategy for the computer. Note that this is optional. If you feel that comments in your main Python file are adequate enough to describe your strategy, that is totally fine. We leave this decision up to you.