Programming Assignments #4 and 5
CS 302 Programming Methodologies and Software Imp.

With the last two programming assignments, you will be implementing your solution using Python using VS Code. Your goal must be to develop a solution using abstractions and hierarchies, but this time using Python. If you did not implement a balanced tree in Program #3, it will be important to implement a balanced tree in Python as instructed in this requirements document. If you didn't implement a BST in Program #3, then it is time to do that now! Make sure that your Design is not centered around data structures – your data structures support the design but shouldn't be the primary emphasis of your design.

Your Python programs must follow these rules:

- **All instance variables should be designated as private or protected using underscore(s).**
    - As a general rule, avoid accessed the instance variables from outside of the class with the exception of your test suite.
    - This means your node must be a class with private data; the node class will need setleft, setright, getleft and getright methods to access the corresponding references.
- **All repetitive data structure algorithms are to be implemented recursively.**
- Make sure to validate all input. Empty input is not allowed.
- Yes, you SHOULD use strings and Python lists.
- Make sure to use a Python list at least once in your assignment and a NumPy array as well (refer to lab).
- Use single inheritance with proper application of super(); there must be a minimum of 5 classes with 3 of them in a hierarchy.
- The application that USES the hierarchy must be in a class on its own. It could be part of a hierarchy, so think about that. Main itself should be minimal.
- Implement at least one constructor with arguments.
- Implement at least two "operator" functions and experiment with how they work in Python (refer back to our lab on operator overloading).
- We expect a thorough implementation of **exception handling**!
- **You may NOT, under any circumstances, use code that is not your own such as from StackOverflow, ChatCPT, or Chegg, or any such website which you can claim as your own.**

Summary of Deliverables for Programs 4 and 5:

1. **Wednesday March 6 -** The first thing to submit is a draft of your specification so we can get an idea of the game that you are intending to build. Fill out this google form with your specifications: https://forms.gle/UatAWwzfMFFutbuk7
2. **Monday March 11th –** Turn in your completed **Specification**, **Black Box** Test Suite using PyTest and the draft **Core Hierarchy** in Python (50 points)
3. **Wednesday March 13th –** Completed Program #4 with the Core Hierarchy finished and progress towards building the game. (50 points)
4. **Wednesday March 20th – Completed Program #5** with the Game and data structures completed. (100 points). Include the revised Specification and PyTest test suite.

Step 1 – Develop a Specification

Treat this programming assignment document as a "requirements documentation". Think of it as what you are "hired" to support. It will be your job to determine what the user interface will be like. This is important to do before you start to code! It will be vital to plan the (1) user interface, (2) the precise rules the program will follow and (3) how the program will conclude. **Your specification should be 600 words** at a minimum and very detailed so that you will be able to build a test plan!

Step 2 – Develop the Black Box Tests

Once the specification has been developed, create your black box tests with PyTest. If it isn't clear what you should test, then it means your specification isn't detailed enough!

For the very first submission, it is expected that you will create a test module to be used with PyTest. The first submission is worth 50% of your program #4 score!

- Use your detailed specification to develop a test suite using "black box testing".
- Then, update your test suite once you have implemented each method, using "glass box testing". With this, it should be clear to us that you have performed unit testing with every path through the software being examined! You will be turning this in!
- **Your PyTest code must be turned in twice – once with the first submission and then again (improved) with your completed assignment #5.**

Final Step – Writeups to Include with the Completed Program

- Your final 400 word writeup should summarize what you have experienced with this assignment. I suggest keeping notes as you develop the test suite from your specification so that you can easily talk about what worked and what needed more detail. Did you create a test plan? Was it helpful? Did each and every path through your software get tested? What type of integration testing did you do?
- Instead of writing 200 words about a debugger, write about how you used VS Code.

REQUIREMENTS document: General background for Programs #4 and 5

For Program #4/5 we will be building a simple adventure game. In adventure games, there is a player that assumes a role of the hero and they take an interactive journey that is story based. It may require the character to explore, answer questions, or even solve puzzles. You may have played such games, where your character encounters a Prince and you are given three choices of what to do (A, B, or C). Based on your answer, you may get a prize, a clue to the next move, or redirected. The key is that your character progresses through dialog but it isn't the same each and every time you play the game.

You can pick the genre of your adventure game and it is expected to be a single player game. It does need to meet the following requirements:

1. Build a narrative so your adventure game makes sense as you plan it. Is it medieval, science fiction, fantasy, etc. Tell the user the story when the game begins.
2. Your character (the Hero) needs a name which needs to be used in any dialog with your character
3. Your character needs an inventory of three types of items of your choice. These items can be collected for use during your hero's adventure. Some ideas include weapons (such as a knife, axe, etc.), food, clothes, tools (such as a hammer, saw, etc.), a shield, rope, and so on. Pick three types!
4. There needs to be three different types of characters that the Hero may encounter during their journey. These could be an Ally or Friend of the Hero, a Villain, a Mentor, a Trickster, a Monster, a Guardian, a Shape shifter, or others. Pick only three types (of your choice).
5. There must be a way to win the game and for the game to end. Consider if you want your character to have a lifetime that can be affected by what they encounter and healed by others.

6. There must be a way for the Hero to select from different choices of what to do.
7. Remember adventure games are all about dialog. A random number generator needs to be used.

Program #4 Requirements –Building the core hierarchy

To begin with, we will start small by focusing on three different types of characters you want to keep track of as part of the adventure game. This needs to be done first, before you create the actual game. Focus on just the underlying data. Plan what you want to keep track of, what functions and operators are important for working with this data, and then build your PyTest test suite for testing out each of these functions.

Build a hierarchy of 3 different types of characters that are derived from a general base class. The common elements of the three should be pushed up and the differences derived. Do not reimplement concepts that another class already implements. Each character must have THREE methods, besides constructors, destructors, and display methods. If one of these classes needs a list of items – use built in options such as a Python List.

The key with this hierarchy will be to plan what functions **and operators** make the most sense. This will require building a specification for this hierarchy first and then building a test suite for just these classes!

Program #5 Requirements - Time to create Collections of Data

Now it is time to create collections of the data. The balanced tree or BST can be used for your hero's inventory and/or the set of characters available to be played within the game (maybe there are 10 monsters with different attributes, 3 guardians and 2 shape shifters all stored in the tree).

Keep track of these in a tree.

- If a balanced tree was not implemented in Program #3, then implement a balanced three (2-3, or red-black tree) now.
- If you did implement a balanced tree in Program #3, then a binary search tree (BST) is required here. **For the BST,** each node has a linear linked list of matching data. Remember to implement a comprehensive set of functions (insert, retrieve, display, and remove an item) and do this recursively.
- For the balanced tree implement insert, retrieve, and display (no remove required) and use the Python List class for any duplicate data.