CISC106 Spring 2013 Lab09

- This lab an all subsequent labs will be due Thursday at 11:55 PM EDT on Sakai.
- The preparation problems below are to develop your understanding without creating extra work for you or the TA; these problems will not be graded. Be sure to read and understand them - they will help with the problems you must submit for grading
- Review the code examples from your notes in class.
- You may work in pairs on your lab. If you do, **one** of you should be designated to submit the assignment on Sakai. **Both of your names** should appear on code that you develop together[1].
- Whom do you think deducts more points: a happy TA, or a frustrated TA? Make your work easy to read! It isn't just good software engineering, it is good for your grade!
- EVERY python program/function must include header, doc string that contains a human-readable desciption of what the function does, and must be followed by a good series of tests, as discussed in class. Always test boundaries. Do not test erroneous input (e.g. a factorial function does not need to correctly handle strings).
- EVERY .py file must have a comment line at the very top containing your name(s), lab section, and a brief description of what the file is.
- **Write the tests first!** Real software engineers do this for very good reasons - so should you!

Problems (to be graded)

1. Create a *GiraffePen* type and write a constructor for it. The constructor should take three values: a number of $rows$, a number of $colums$ and a 2 dimensional grid of $squares$. Each square in $squares$ can either contain a giraffe, or it can be empty. Your constructor should build a GiraffePen such that if $squares$ is the empty list, the pen will be a $rows \times colums$ pen containing no giraffes. Otherwise, if the dimensions of $squares$ are $rows \times columns$, then the pen will be a $rows \times columns$ pen using $squares$ as its grid. **NB:** You should think carefully about what the representation of a square should be. Keep in mind that the relevant information on a square is that it either is occupied by a giraffe or it is not.
2. Now implement a function `place_giraffe` for your GiraffePen. Place giraffe should take not only a pen, but also a row and a column. If the row and column specify a valid location in the pen, and a giraffe is not currently occupying that location, then a giraffe should be added to that location. Otherwise, nothing should change.
3. Implement a function `evacuate_giraffe` for your GiraffePen. Again, it should also take a row and a column. Again, it should ensure the row and column specify a valid location in the pen. If so, and there is a giraffe in that location, the giraffe should be removed. Otherwise, nothing should change.

[1] If you would like to work with someone but don't know whom, your TA may be able to help connect you to other students looking for lab partners.

4. Now implement a function `has_giraffe`, which looks at a specific location in a GiraffePen and returns whether or not that location contains a giraffe.

5. Write a `mergeable` function which operates on two GiraffePens. Two GiraffePens are mergeable *if and only if* (iff) there is no location (row, col) such that (row, col) contains a giraffe in *both* pens.

6. Finally, write a `merge` function which takes two GiraffePens. If the pens are `mergeable`, it should return a *new* pen containing *all* of the giraffes on both of the original pens. Otherwise, it should return the *null pen* - that is, a pen of 0 rows and 0 columns.

7. Now we're going to make a *simplifed*[2] model of wizards fighting in *Final Fantasy Tactics - The Zodiac Brave Story*. First you'll want to create a *Wizard* type and give it a constructor which takes as arguments a *name*, *brave* value, *faith* value, number of *hit points* (HP), *mana points* (MP), *physical attack* value (PA), *magical attack* value (MA), and *speed* and returns a Wizard with these attributes. All of these attributes (except for the name, which is of course a string) should be integer values.

8. Now implement a `nuke` function which takes two Wizards - an $attacker$ and a $target$ - and (as the names all imply) causes the attacker to cast an offensive spell upon the target *provided the attacker has enough MP to do so*. The Final Fantasy Tactics Battle Mechanics Guide describes the damage formula for magical attacks as

$$\left\lfloor \frac{AttackerMA \cdot AttackerFaith \cdot TargetFaith \cdot Q}{10000} \right\rfloor \tag{1}$$

   where Q is a number denoting the *power* of the spell being cast.[3] Luckily for us, this is a simplified model - so we're just going to go ahead and assume that the only spell any of our Wizards know is *Bolt2*, which has a Q value of 18 and costs 10 MP to cast. Keep in mind that an individual's HP can't fall below 0 - if the attacker's spell deals more damage than the target has HP, the target's HP is simply reduced to 0.

You should submit your lab09_tests.py, giraffepen.py, wizard.py, and any other docs required by your TA on Sakai.

[2] Ignoring zodiac compatibilities, gear-based elemental boosts, most of the available spells, reaction abilities, etc.
[3] **NB** that $\lfloor val \rfloor$ on a real number val means "the floor of val".