**Spring, 2014 CIT 590**

# Programming Languages and Techniques

# Homework 6

Feb 21, 2013; Due Mar 7, 2013, *before first recitation*

This homework deals with the following topics

* classes

* everything we have covered so far :)

This is your final Python assignment.

The assignment this time is to make a card game using some classes that are provided to you. We provide you with two classes - a card class that represents a single card and a deck class that represents a single deck of cards.

There are no jokers in this deck. So 52 cards in one deck.

Before jumping into the game, one of the things we want to try and do a little bit more is provide more hands on help.

So with that in mind please download the file cards.py and let us first work on implementing all the unimplemented functions.

And we want to do this in a TDD manner. If we do not manage to do everything in the recitation session, that is fine, but at least you will get some more hands on time with the TAs and I.

You also have 2 weeks to do the assignment. We will not have recitation the Friday before spring break. Just ensure that you submit the assignment before you run off and have fun!

**Please do not read this section until you have your card class and your deck class ready**.

We would like to use the classes we have to design a solitaire game.

Most of you have seen this game before, especially if you had an older windows machine at any point, but instead of me trying to provide you with the rules, here is a link to solitaire.

That same website also has the rules of the game if you look under the info menu.

Since this is a two week assignment and the final one, this assignment encourages you to think of your own design. What methods do you need. What should you call your methods. How do you want to represent your data?

We're essentially taking the training wheels off and encouraging creative thinking.

That being said, we do want you think about this within some level of structure. We would like one class called Solitaire to be defined and that class has to have the following

- An initial deck, waste, foundations and tableaux. These four things are the main reason for having a class since at point in the time they represent the state of the game. All of these will be defined in the **__init__** method
- A play() method which calls into other methods to play the game.
- A method called getUserInput() which asks the user for a move. The user is supposed to only respond in a certain manner which is listed below.
- A method called display() which provides a nice display of the game to the user.

The one place where we want to enforce things is the user interaction piece. Since we do not have any fancy mouse drag and drop abilities yet, we have to do all of this in a text based manner.

We are going to call the 7 stacks at the bottom half of the display - **tableau**

We are going to call the 4 stacks at the top half of the display (where you build from ace upwards) - **foundations**

We are going to call the main deck of cards - **mainDeck**

We are going to call the place where you place 3 cards from the deck - **waste**

Since a picture is worth a thousand words, just use the attached powerpoint file. It shows you the conventions and the numbering as well.

So here is the set of legal commands from the user.

Assume all stacks, foundations etc are numbered from 1. CS thinks it is so cool doing 0 indexing but we do not want to annoy the user with our geekiness.

1. m t[i,j] t[k] - Move the $i^{th}$ card in the $j^{th}$ tableau to the $k^{th}$ tableau. for clarification on the numbering within the tableau see the powerpoint.
2. m t[n] f[m] - Move the card from tableau n to foundation m.
3. g - Give the user one card and place them face up on the waste.

4. m w t[n] - Move the card from the waste to tableau n.
5. m w f[m] - Move the card from the waste to foundation m.
6. r - restart the whole game.

For instance in the attached slide 1 here are some legal moves. The parantheses are just comments. These moves are not being done in sequence, they just represent possible things that the user can say at this stage.

- g
- m t[1,5] t[4]
- m t[1,7] t[1]

For one more example, in slide 2 you can do

- g
- m t[2,2] t[5]
- m t[1,7] f[3]
- m w t[2]

Here are some other general guidelines

- Your main function will contain just 4 or 5 lines of code

```
def main():
    printIntroMessage()
    solitaire = Solitaire()
    solitaire.play()
    choice = raw_input('do you want to play again? y/n')
    while choice != 'n':
        solitaire.play()
        choice = raw_input('do you want to play again? y/n')
```

- You are going to have to represent lots of stacks of cards. Some of them have logical groupings like the foundations and the tableaux. Think about how best to represent all of this. Some of the work you did in Racko will help as well.

- You have to validate the user's inputs. This is one of the tougher things to do since this time we actually allow the user to enter any string and we want you to be robust to it. We do guarantee that the user will be entering some string though.

  Strongly suggest you break this check up several small components and write some smaller functions with unit tests.

  The following function may help

```
def isInt(arg):
    try:
        arg = int(arg)
    except ValueError, e:
        return False
    else:
        return True
```

- Even if the user provides a valid input, they may not be able to make the move because of the rules of the game. So you need some kind of method in the solitaire class which checks that. Come up with your own version of an isLegalMove function.

- You need to display the tableaux in the right manner. For this one try and write a function that given a dictionary of lists can display them in the right manner. For instance

  $\{1 : [5], 2 : [4, 6], 3 : [9, 10, 12, 13]\}$

  should get printed out like this

```
5  4  9
   6  10
      12
      13
```

  Now once you have written that function think about how you can use it for your display method in solitaire.

- Deal with the situation when you have gone through the deck and the waste pile has all the cards. Since we allow the game to restart at any point, just let the user keep going through the cards if they want to.

- You do not need to build in any kind of help for the user. If they do not find a move that actually does exist on the board, it's their own loss.

- There is no scoring system that we want you to implement, but it is fairly simple to check and see if the user has won. If so, you would like to inform them and congratulate them. So write some kind of method to do that.

**Evaluation**

Since this assignment has very few specifications the evaluation is slightly different.

- We have no unit tests that we can run.
- You have to still write unit tests! That will be true for the remainder of the course and that is the way it is actually done in the industry so no apologies about that.
- Remember to write unit tests for everything that is unit testable.
- Your function writing skills and your test cases will be critiqued more. So you have to try your best to make reasonable functions. Break things up into smaller pieces, write docStrings, have good variable names, have good function names. Make sure your code is readable.
- At the end of the day, this is a game. We should want to play it. The weird instructions are something we gave you so you do not have much of a choice there. But you do have the ability to make good displays of the game. Hard work in getting a good display will definitely be rewarded.

  That said, please do not go overboard and import some kind of graphics module to do the display. Everything needs to be text based.
- Be nice to the user. For instance if they are trying a move which is not possible, for instance moving a 7 onto a 9, maybe they do not get the game and it is worth telling them what they are doing wrong.

**What to submit?**

You have to submit 4 things in this assignment.
cards.py (obviously the modified version)
cardsTest.py (the unit tests for the file above)
solitaire.py
solitaireTest.py
helperFunctions.py (put the isInt function over here)
Zip up all 4 or 5 of these into a file called Solitaire.zip and upload just that one file into canvas.