IAE 101 – Introduction to Digital Intelligence
Homework 03
Fall, 2019

Assigned: Sunday, 10/20/2019
Due: Sunday, 10/27/2019, at 11:55 PM

Assignment Objectives:
The goal of this assignment is to help you solve problems and implement algorithms in Python using the tools discussed in class.

Getting Started:
This assignment requires you to write Python code to solve computational problems. To help you get started on the assignment, we will give you a basic starter file for each problem. These files contain function stubs ('function stubs' are functions that have no bodies; the function has a name and a list of parameters, but it is up to you to complete the body of the function) and a few tests you can try out to see if your code seems to be correct (note that the test cases we give you in the starter files are just examples; we will use different test inputs to grade your work!). You need to complete (fill in the bodies of) these functions for the assignments. Do not, under any circumstance, change the names of the functions or their parameter lists.

In addition, you will finish the definition of several Python classes. The class names are already specified in the headers. A function stub for the `__init__()` function for each class has also been provided. The specified parameters of `__init__()` are the only ones that should be passed when constructing an object of that class. However, you must define the attributes of the class inside each `__init__()` function, and there may be additional attributes given in the problem description that are not included as parameters to `__init__()`. Do not add extra parameters to `__init__()` (or any of the other functions).

Directions:
There is one problem. That problem has several parts and is worth 50 points. Solve each part of the following problem to the best of your ability.

- Each starter file has a comment block at the top with various important information. Be sure to add your information (name, ID number, and NetID) to the first three comment lines, so that we can easily identify your work.
- Each of your classes and functions must use the names and parameter lists indicated in the starter code file.
- Do NOT use input() anywhere within your functions; your functions should get all their input from their parameters.
- Be sure to submit your final work as directed by the indicated due date and time. Late work will not be accepted for grading.
- Programs that crash will likely lose a lot of credit, so make sure you thoroughly test your work before submitting it.
- Submit all your completed problemX.py files on Blackboard for Programming Homework 02.
- Do NOT change the names of the files.

Problem 1: Simulating Blackjack

In this problem we will use classes and functions to simulate a simplified game of Blackjack (21). The game begins with a standard deck of 52 playing cards (no jokers). Each player is dealt two cards to start with. The winner of a hand of Blackjack is the player whose hand has the highest value without going over 21.

When calculating the value of a hand, we add up the rank of each card in the player's hand, where the numbered cards have ranks 2 through 10. The face cards, Jack, King, and Queen, each add 10 to the value of a player's hand. The Ace card can be treated as adding either 1 or 11 to the value of the player's hand, depending upon which brings the player closer to winning.

Player's may request additional cards (a "hit") in order to bring the value of their hand closer to 21. However, if the value of a player's hand rises above 21, then the player has "busted" and they automatically lose the game.

In our simulation we are ignoring the distinction between player and dealer. We are also ignoring betting, and so all the more sophisticated player actions (such as "splitting") are also being ignored. The behavior of player's is going to be fixed by a simple algorithm. Details for writing the simulation program are given below.

The program will consist of 4 classes: Card, Deck, Player, Blackjack. Each class represents objects that are elements of a simulated game of Blackjack. To implement each class you will have to complete the implementation of the functions inside the body of the class definition. The names of the classes, the functions, and the function parameters have already been given. DO NOT CHANGE THEM.

The Card class is meant to represent a single playing card. It has two attributes, *suit* and *rank*. Playing cards come in one of four suits: Hearts, Diamonds, Spades, and Clubs. In your program, each suit is represented by a string of a single capital letter: "H" for Hearts, "D" for Diamonds, "S" for Spades, and "C" for Clubs. I have included a list of the suits as a global variable in your program. For each suit there is a Card of one of 13 ranks. Each rank is represented by a string: "2" through "10" for the ranks of the numbered cards, and "J", "Q", "K", and "A" for the Jack, Queen, King, and Ace face Cards.

When a card is constructed the values for suit and rank should be passed to the suit and rank attributes of the Card class. You must implement this in the __init__() function. You should check that the values passed to the Card constructor represent correct suit and rank values.

The Deck class represents a standard deck of 52 playing cards. The Deck class should have a single attribute called `cards`. cards will be a list of Card objects, one for each playing card that makes up a standard deck. In the __init__() function of the Deck class you should create and add a Card object for each of the 52 cards that make up a standard playing card deck. That is, for each suit there should be a Card of each rank added to the cards list in the Deck class.

In addition to the __init__() method, you must also implement a deal_card() function. This function takes a single argument, an object of the Player class. Inside the function you should remove the card from the top of the Deck (from the cards list of the Deck), and then add that card to the hand attribute inside the Player object.

I have provided you with a function shuffle_deck() that will randomize the order of the cards inside cards list inside the Deck object. The function takes a single positive integer that represents how many times the deck will be shuffled. The default value is 5. DO NOT CHANGE THIS FUNCTION.

The Player class represents the individual players involved in a game of Blackjack. Each player has three attributes: *name*, *hand*, and *status*. The name attribute is a string that represents the Player's name. The hand attribute is a list that will contain the Card objects that make up the Player's hand. The status attribute will hold a Boolean value and represents whether the Player is still playing. When you construct a Player object, you should set the Player's name to the string passed in as an argument, the hand attribute should be set to an empty list, and the status attribute should be set to True. Once a Deck and Player have been instantiated, then you can deal cards to the Player from the Deck using the Deck's deal_card() function. If the Player decides to stay or busts during the game, then status will be set to False.

In addition to __init__() function of the Player class, you must implement 2 other functions for Player objects. The first function is called value(). It is used to calculate the current value of the Player's hand. Use the rank of each Card object in the Player's hand to calculate the sum, by adding the ranks together. Remember that the face cards, "J", "Q", and "K" each add 10 to the value of the Player's hand. Be very careful about Ace Cards. Remember that the Ace can add either 1 or 11 to the value of the Player's hand depending upon the circumstances. The value() function should return the current total value of the Player's hand as an integer.

The second function you must implement for the Player class is the choose_play() function. This function will determine whether the Player will *hit*—take another card—or *stay*—refuse cards if the Player is satisfied with the current total value of their hand. In our simulation all Players will follow this strategy. If the current total value of their hand is less than 17, then the Player will hit. Otherwise, the Player will stay. If the Player decides to stay, then their status attribute should be set to False. The choose_play() function should return the string, "Hit", if the Player decides to hit, and return the string, "Stay", if the Player decides to stay.

The last class you must implement is the Blackjack class. This class represents a simulated game of Blackjack. Each Blackjack object has two attributes: *players* and *deck*. The players attribute is a list of Player objects. This list of Player objects is passed to the __init__() function of the Blackjack class when constructing a new Blackjack object. The deck attribute should be assigned a new Deck object. You must do two other things in the __init__() function of the Blackjack class. You must shuffle the deck. You must then deal two cards to each of the Players. You can assume that the list of Players passed to the Blackjack constructor will always contain between 1 and 4 Player objects.

In addition to the __init__() function of the Blackjack class, you must also implement a function called play_game(). This is the function where game play will happen. Gameplay should occur inside of a loop. The loop continues until all of the Player's stop taking new cards. During each iteration of the loop you should do the following two things.

First, for each Player, check the status attribute of the Player. If their status is True, then call the choose_play() function to determine what that Player will do. If choose_play() returns "Hit" for that Player, then deal that Player a new card from the Blackjack deck. Then, check the value() of that Player's hand. If they have gone above 21, then report that Player has busted (print a message) and set the Player's status attribute to False. If choose_play() returns "Stay" for the Player, then move on to the next Player.

Second, for each Player check their status attribute. If status attribute for all Players is set to False, then the game is done and you can end the game play loop.

Once the loop is complete, then you need to determine the winner. The winner is the Player whose hand has the highest value less than or equal to 21. If there is a winner, print a message reporting the winner of the game. If there is a tie, print a message reporting the tie. If there is no winner (if all the Players busted), then print a message reporting that there is no winner.

If you implement each of these classes and their functions correctly, then the test code included in the file will run a simulated game of Blackjack with four players.