A zipped folder is provided, linked next to this handout on Canvas: **h\_9\_starting.zip.** It contains starting **.py** files for each of the following problems. Use them.

At the start of each of these problems, the name of a Python file is given in **blue**: **foo.py**. You should create and save the requested Python program source code in a file with the same name. Also add a comment at the top of each giving your name.

When finished, upload each **.py** file with the specified name to the Canvas **H9 Assignment** link. Also upload your final versions of the **card.py** and **deck.py** files.

**[H9-1]** (**student.py**) Create a new class **Student.** Its instances should track the name and each quiz scores taken by the student, including the number of quiz scores that are dropped and the resulting average quiz score. Use a list \_**scores** of float values as a **Student** attribute (field), with **\_scores[I-1]** stored as the score of Quiz **I**.

Here are the **Student** methods you should define:

* A constructor with a single argument, setting the name of a newly-initialized **Student**.
* Instance method **addScore(self,score)**, which adds another quiz score to the collection within the **Student**
* Instance method **getNumberQuizzes(self)** which should return the total number of quiz scores
* Instance method **calcQuizStats(self,dropNumber)**, which should calculate and set the **Student** attribute (field) **\_average**, after calculating the **float** average (mean) score after dropping the lowest **dropNumber** quiz scores. If **dropNumber** is greater than or equal to the total number of quiz scores, then calculate the average as **0.0**. This method should also set the field **\_dropNumber**
* Instance method **getAverage(self)** which should return the value of **\_average**
* Instance method **\_\_str\_\_(self)**, which should create a string representation of the student, giving the name, quiz average **\_average**, number of dropped quizzes **\_dropNumber**, and the list of each quiz score, with dropped scores marked with a **\***. Example (your output should show the same information, though your formatting may differ):   
    
  **Name: Moxie Berner  
    
  Quiz Average: 47.0**  
  **Number of Quizzes: 4  
  Dropped Quizzes: 2**  
   **Quiz Scores (\* => dropped):  
  1 - 64.0  
  2 - 30.0  
  3 - 10.5 \*  
  4 - 20.0 \***

In the **main()** method of your class, create two **Student** instances, add some quiz scores to each, calculate the quiz stats, and print out the string equivalent of each resulting instance. One of your students should have the same information as shown above, but you can choose your own information for the second.

**[H9-2]** (**bjackodds.py**) Use the **Card** and **Deck** classes of Lab 9, here provided in separate files **card.py** and **deck.py**. Use them to write a program which simulates the repeated shuffling of a full deck, then drawing its top two cards and checking them for blackjack. "Blackjack" occurs when one card is an Ace, and the other is one of 10, Jack, Queen, or King. Your program should calculate the percentage of the time that "blackjack" occurs.

Run your program multiple times. Do your results agree with theoretical probabilities? Add a comment at the top of your submitted file that addresses this question.

**[H9-3]** (**hand.py**) Create a new class **Hand**, whose objects represent a poker hand of **Card** instances drawn from a **Deck**. Its constructor should create an empty hand, its instance method **addCard(self,card)**should add card to the hand, and its **\_\_str\_\_(self)** method should return a string which lists all the cards currently within the **Hand**. Note there may be any number of cards within a **Hand** instance.

Also define a **main()** method that does the following: create a new **Deck**, shuffle it, then create a **Hand** and add the top five cards of the shuffled **Deck** to it. Finally, print out the string equivalent of the **Hand**.

**[H9-4]** (**poker\_odds3.py**) Refactor your H8 program **poker\_odds2.py**, rewriting it to use the **Hand2** class, created as a modified version of the previous problem's **Hand** class. Refactoring means to change the internal structure of a program without changing its external behavior. Thus, when you run your refactored program, it should generate exactly the same results as before.

Part of your refactoring should alter **Hand** into **Hand2** as follows: convert the function **buildDict(hand)** into a **Hand2** method **buildDict()**, which builds and returns the same dictionary as before. You should also move each of the five H8 instance methods **isOnePair(dict)**, ... from H8 into **Hand2**, modifying them so they have no arguments as methods within **Hand2**. When called, each should first call the internal **buildDict()** method to create the necessary dictionary **dict**, then use it to determine what Boolean value to return.

This kind of reworking of an existing class is a common activity in today's agile software development.

**[H9-5]** (**in\_riffle.py**) Same as L10-4 but implement a perfect **in** riffle shuffle by adding to **Deck** the instance method **inRiffle(self)**. This is similar to the out riffle-shuffle of the Lab. Here the top card before shuffling becomes the second card after interleaving the two deck halves, with the top card of the shuffled deck as the second card of the last half of the **Deck** before shuffling.

Then write a **main()** method that experimentally determines the smallest number of perfect in riffle-shuffles needed to bring the deck back to its original order. You'll find it's different than the minimum number of perfect out riffle-shuffles to do the same thing. Hint: modify your **outRiffle()** method and supporting code from L10-4.