Hw6: Due Monday, Nov 16th, by midnight.

General Instructions

As in hw5, all the code in this homework will be contained in a **single Python module**: please make sure that you **rename** it as `hw6_<RUID>.py` (where `<RUID>` should be replaced by your Rutgers ID).

Download the starter module `hw6_stub.py` that contains three function stubs corresponding to the three problems below: **rename** it as above before you complete and submit it! Make sure that you complete the module **docstrings** with your name (as the author) and collaboration acknowledgments (if any) or help that you may have received on the assignment.

As with the last assignment, you will be responsible for creating the docstrings and suitable doctests for all the functions! Please refer back again to the PDF handouts for homeworks 1 & 2 for **Style Instructions** within your code, and make sure that your docstrings follow the guidelines that we have been observing in class.

Starter Code

There may be some code in the `__main__` block: do not modify this code. The functions in the starter code are very minimal stubs with just their names and the names of the formal parameters (arguments). As mentioned above, you are responsible for creating the docstrings.

Developing and Submitting Code

- All the functions to be implemented in this assignment have been set up for you as **incomplete stubs**, i.e., they have **no** docstrings and have basically empty function bodies that do nothing.
- Once you implement everything, running the completed module (with your RUID inserted in the name appropriately) as `% python3 hw6_<RUID>.py` from the Unix command line should result in the a silent return back to the Unix prompt, which is an indication that all the doctests have succeeded.

Again, please make sure that (a) your code has no syntax errors, (b) has **complete** documentation and (c) has been checked for style. Do this before you submit the module `hw6_<RUID>.py`
by uploading it at the Canvas submission link. You can upload as many times as you want before the deadline.

Problem 1

As you probably know, a standard deck of playing cards contains 52 cards divided into 4 **suits**: Hearts (`H`), Spades (`S`), Diamonds (`D`) and Clubs (`C`). There are 13 cards in each suit ranked in descending order from Ace (`A`), King (`K`), Queen (`Q`), Jack (`J`) and then the numbered cards from `10` down to `2` (these numbers are called *pip values*). The first four type of cards are called **face cards**; altogether, the face cards and the numbered cards make up 13 **kinds** of cards. We will represent a card as a string starting with the suit letter followed by the kind, e.g. 'SQ' for the Queen of Spades, 'C4' for the four of clubs etc.

In the game of **5-card draw**, the most basic version of **poker**, players are dealt five cards cards (their **hands**) from a well-shuffled deck of cards. We will assume that each player can see their entire hand but no one else can. The order in which the cards appearing in a hand were initially dealt is not consequential: what matters is the collection of five cards and the various patterns that can be seen among them. There are basically 10 kinds of patterns (the **poker hands**) given below in *descending* order of importance:

- `'Royal Flush'`: Ace through Jack plus the '10' card from the same suit, e.g., 'DA', 'DK', 'DQ', 'DJ', 'D10'.
- `'Straight Flush'`: Five cards in sequence from the same suit, e.g., 'DJ', 'D10', 'D9', 'D8', 'D7'. Note that an ace can be *high* or *low* (e.g., 'H5','H4', 'H3', 'H2' and 'HA' is also a straight flush).
- `'Four of a kind'`: Four cards of the same kind, e.g., 'HK', 'SK', 'DK', 'CK', 'S3'

- `'Full house'`: Three cards of one kind and two of another, e.g., 'HK', 'SK', 'DK', 'C3', 'S3'
- `'Flush'`: Five cards of the same suit but not in sequence, e.g., 'DJ', 'D9', 'D6', 'D3', 'D2'
- `'Straight'`: Five cards in sequence but with two or more suits, e.g., 'DJ', 'D10', 'C9', 'S8', 'S7'; again an ace can be high or low.
- `'Three of a kind'`: Exactly three cards of the same kind, e.g., 'HK', 'SK', 'DK', 'S5', 'D3'
- `'Two pair'`: Two cards each of a kind, e.g., 'DJ', 'CJ', 'H9', 'S9', 'S7'
- `'One pair'`: Exactly two cards of a kind, e.g., 'HK', 'SK', 'D5', 'C9', 'S3'
- `'High Card'`: No pattern present, e.g., 'HK', 'SJ', 'D9', 'C6', 'D3'

We will represent a dealt hand as a **list containing 5 cards**, where each card is represented by a string as above. The first character in the string is the suit and the rest of the string indicates the kind (i.e., face value/pip). The list of all possible pattern labels to be used in this assignment is provided in the starter code as a constant: you must use exactly these labels as keys in any dictionaries needed in your code. Also provided as constants are the suits and the kind (face value/pip) representations.

Problem 1

As a quick warmup, design a function, `sort_hand` that takes as argument a hand, and returns the hand rearranged so that all the aces come before the kings, the kings before the queens etc. Within cards of the same kind, arrange the cards in the suit sequence 'HSDC' (Hearts-Spades-Diamonds-Clubs). For example, if the hand is ['H9', 'CJ', 'S9', 'S7', 'DJ'] then the function should return ['DJ', 'CJ', 'H9', 'S9', 'S7'].

Problem 2

Design a function called `label_hand` that takes as argument a hand, and returns a tuple containing:

- the **string label of the best poker hand** describing the cards, i.e., if the hand is `['D8', 'D10', 'D7', 'DJ', 'D9']`, the function should return 'Flush' and not 'Straight' as the first component in the result tuple.
- the sorted hand as obtained from the function in problem 1

Hint: To determine what kind of poker pattern hand it is, one needs to **count** the cards by suit (e.g. how many Spades etc.) and by kind (how many Queens, how many 3s etc.). Use separate local dictionaries within the function for this purpose. The information in the dictionaries along with the sorted hand will unambiguously determine the kind of poker hand.

Problem 3

For this problem, a utility function `deal_hand` has already been defined for you: when called, it returns a **random hand** (as though it came from a well-shuffled deck of cards). Do not modify this function!

Use this utility function repeatedly along with your implementation of the `label_hand` function above, to conduct the following experiment: count the relative frequency of full houses, three of a kind and two pairs in a total of 10000 random dealt hands. Your function, called `frq_patterns`, should return a dictionary of the three counts above (you should use the labels of the hands as keys, i.e. 'Full house', 'Three of a kind' and 'Two pairs'.