This problem set is due **Tuesday, November 28 at 11:59pm, KST**. Note that that the due date that you see on Blackboard is not accurate since it shows the time in EST. You should go by the due date in this handout.

Add your name and email address as a comment at the top of each file you submit.

**What to submit**

Hand in your work on Blackboard. Your submission should include the following. Please do **not** submit other files that I did not ask for.

```python
ps6.py
```

Multiple submissions are allowed before the due date and time. Late submissions will **not** be graded.

**Assignment objectives**

For this problem set you will be implementing the a simplified version of the card game Bridge. You do not need to understand the rules to complete the assignment, but the Wikipedia page at [https://en.wikipedia.org/wiki/Contract_bridge](https://en.wikipedia.org/wiki/Contract_bridge) provides the details. Throughout the entire assignment you will be working with objects that represent the playing cards and the players.

**Incomplete work**

Please read what I said about this in PS 2.

**Naming conventions in Python and programming style in general**

Please read what I said about this in PS 2.

**Game Overview**

The game starts off with 52 cards distributed among 4 players. Each player is given 13 cards. When the game begins, the first player plays any card he/she wants. Typically this is the card with the highest rank of any suit. All other players have to follow the suit. If one of the other players doesn’t have a card in that suit, he/she can play a card of any other suit. Typically, this is the lowest rank card the player chooses to play. The winner of a round is determined by the player who placed the highest ranked card of the suit on which the round began. (2 is the lowest rank and Ace is the highest.) The winner of the round scores a point and gets to place his/her card first at the beginning of the next round. This continues until there are no more cards left to play. The winner is determined by the player who has won most number of rounds.
Preliminaries

For this problem set you will be working with the following classes that represent a playing card and a player (They are given in ps6.py):

class Card:
    suit_sym = {0: '♣', 1: '♦', 2: '♥', 3: '♠'}

    def __init__(self, rank, suit):
        self.rank = rank
        self.suit = suit

    def __repr__(self):
        return self.rank_sym[self.rank] + self.suit_sym[self.suit]

    def __eq__(self, other):
        return self.suit == other.suit and self.rank == other.rank

class Player:
    def __init__(self, card_list):
        self.card_list = card_list
        self.score = 0

    def __repr__(self):
        return 'cards: ' + str(self.card_list) + ', score: ' + str(self.score)

Data Encoding for the suit Field in the Card Class is as follows:

- 0 represents Clubs.
- 1 represents Diamonds.
- 2 represents Hearts.
- 3 represents Spades.

Data Encoding for the rank Field in the Card Class is as follows:

- 0 represents 2.
- 1 represents 3.
- ...
- 11 represents K.
- 12 represents A.

Additional Functionality

In the ps6.py file you will find the typical main function and two additional functions: run_tests and simulate_game. Inside the main you will see that the run_tests function is called, but simulate_game is commented out. The run_tests will execute the same test cases given in the handout. The simulate_game function will call the functions that you must write to simulate a full 13-round game of Bridge. Although you don’t have to use the simulate_game to test your work, you may find it useful/interesting to watch a game played out from start to finish. Realistically, however, the simulate_game will crash the program until you
have implemented most of the required functions. You will be required to write the following five functions (not methods) to support playing of the game.

**Part 1. Deal the Cards (20 points)**

In `ps6.py` complete the function `deal_cards` that takes the following parameters, in this order:

1. `player_list`: A list of 4 `Player` objects.
2. `card_list`: A list of 52 `Card` objects.

Your function should deal each `Card` object in `card_list` to each `Player` in `player_list` in the following order:

1. The first `Card` object in the `card_list` list is added to first `Player` object in `player_list` list.
2. The second `Card` object in the `card_list` list is added to second `Player` object in `player_list` list.
3. The third `Card` object in the `card_list` list is added to third `Player` object in `player_list` list.
4. The fourth `Card` object in the `card_list` list is added to fourth `Player` object in `player_list` list.
5. The fifth `Card` object in the `card_list` list is added to first `Player` object in `player_list` list and so on, up until the last `Card` object from `card_list` has been dealt.

This function doesn’t return any values.

**Example:**

Suppose `card_list` contained the following cards, in this order:

```
[4♠, 5♣, A♠, 3♣, 9♣, A♦, 3♣, 7♦, Q♠, 2♠, 8♠, 10♠, K♠, 4♠, 8♣, 7♥, K♣, 6♦, 2♣, A♥, K♠, 10♥, 4♠, 3♠, 2♠, 9♠, A♣, 8♠, K♥, 8♥, 10♥, 2♥, Q♠, J♥, 10♣, 4♥, Q♠, 7♥, 9♥, 5♠, 5♥, 9♣, 5♥, 3♥, 4♦, J♦, 10♦, 5♦]
```

The cards would be distributed to the players as follows:

- **Player 1:** `[4♠, 3♣, 7♦, 10♠, 7♥, A♥, 6♦, 2♣, K♥, Q♠, Q♣, 5♠, 4♦]`
- **Player 2:** `[5♣, 9♣, 9♦, K♠, 6♠, 9♠, 8♠, J♥, 7♣, 9♣, J♣]`
- **Player 3:** `[4♠, A♠, 2♣, 4♣, 6♣, 10♣, 7♠, A♣, 10♥, 0♥, 9♥, 6♥, 10♠]`
- **Player 4:** `[3♠, J♠, 8♠, 8♣, 2♣, J♠, 3♠, 8♣, 2♥, 4♥, 5♣, 3♥, 5♥]`

**Part 2. Taking a Turn (20 points)**

In `ps6.py` complete the function `take_turn` that takes the following parameters, in this order:

1. `current_suit`: The suit for the current round of play.
2. `player`: A `Player` object.

The function finds the `Card` object with the highest rank that matches the `current_suit` from the `card_list` that the player has, removes that card from the player’s card list, and returns the `Card` object.

**Special Cases:**

- If `current_suit` is `None`, find and remove the `Card` object with the highest rank and return it.
- If there is no `Card` object in the player’s card list that matches the `current_suit`, find and remove the `Card` object with the lowest rank and return it.
For the two special cases above, if two or more cards in the player’s card list have same (highest or lowest) rank, return whichever Card object with same rank appears earliest in the card list for that player object.

Example #1:

```
player.card_list = [2♥, 2♦, 3♠, 4♣, 5♦, 6♣]

player.card_list = [Card(0, 2), Card(0, 1), Card(1, 1), Card(2, 1), Card(3, 1), Card(4, 1)]
result = take_turn♠, player))
```

Return Value: 2♥

Updated player.card_list: [Card(0, 1), Card(1, 1), Card(2, 1), Card(3, 1), Card(4, 1)]

player.card_list is now: [2♦, 3♠, 4♣, 5♦, 6♣]

Example #2:

```
player.card_list = [2♦, 3♠, 4♣, 5♦, 6♣, 6♥]

player.card_list = [Card(0, 1), Card(1, 1), Card(2, 1), Card(3, 1), Card(4, 1), Card(4, 2)]
result = take_turn(None, player))
```

Return Value: 6♥

Updated player.card_list: [Card(0, 1), Card(1, 1), Card(2, 1), Card(3, 1), Card(4, 2)]

player.card_list is now: [2♦, 3♠, 4♣, 5♦, 6♥]

Example #3:

```
player.card_list = [2♥, 2♦, 3♠, 9♦, 5♣, 6♣]

player.card_list = [Card(0, 2), Card(0, 1), Card(1, 1), Card(7, 1), Card(3, 1), Card(4, 1)]
result = take_turn♠, player))
```

Return Value: 9♥

Updated player.card_list: [Card(0, 2), Card(0, 1), Card(1, 1), Card(3, 1), Card(4, 1)]

player.card_list is now: [2♥, 2♦, 3♠, 5♣, 6♣]

Part 3. Determine the Round Winner (20 points)

In ps6.py complete the function round_winner that takes the following parameters, in this order:

1. current_suit: The suit for the current round of play.
2. cards_on_table: A list of the four Card objects played by each player in the current round. cards_on_table[0] is the card that Player 1 played, cards_on_table[1] is the card that Player 2 played, etc.

Your function should find the Card object in the list cards_on_table that has the highest rank and matches the current_suit and returns the corresponding index. You are guaranteed that at least one card in cards_on_table will be of the suit current_suit.
Example #1:

cards_on_table = [2♥, 3♥, 4♥, 5♥]
cards_on_table = [Card(0, 2), Card(1, 2), Card(2, 2), Card(3, 2)]
result = round_winner(♥, cards_on_table)
Return Value: 4

Example #2:

cards_on_table = [2♦, 3♦, 4♦, 5♥]
cards_on_table = [Card(0, 1), Card(1, 1), Card(2, 1), Card(3, 2)]
result = round_winner(♦, cards_on_table)
Return Value: 3

Example #3:

cards_on_table = [2♣, 3♣, 4♣, 5♥]
cards_on_table = [Card(0, 3), Card(1, 1), Card(2, 1), Card(3, 2)]
result = round_winner(♣, cards_on_table)
Return Value: 1

Part 4. Determine the Game Winner (20 points)

In ps6.py complete the function find_winner that takes the one argument, player_list, which is a list of Player objects. Your function should iterate over all Player objects in the given player_list and return a list of Player objects that have the highest scores.

Note: It's possible that it has more than one winner.

Example #1:

Player scores: 10, 1, 1, 1
Winner(s): Player #1
Return value: [1]

Example #2:

Player scores: 8, 2, 2, 1
Winner(s): Player #1
Return value: [1]

Example #3:

Player scores: 4, 4, 4, 1
Winner(s): Players #1, 2, and 3

Return value: [1, 2, 3]

Part 5. Report on the Game Status (20 points)

In ps6.py complete the function game_status that takes one argument, cards_on_table, which is a list of four Card objects. Your function should return a string representation of the Card objects on the table in the following format: Player #: Card, with each line ending with a \n character (including the fourth line). For your convenience, we have provided you a string representation of Card class via the the Card class’ _repr_ method. All you need to do is concatenate Card objects to strings for the _repr_ method to be automatically called.

Note: In the returned string there is exactly one space after the word Player and exactly one space after the colon. Your string must be formatted exactly like this too.

Example #1:

cards_on_table = [2♥, 2♦, 2♠, 2♣]
cards_on_table = [Card(0, 2), Card(0, 3), Card(0, 0), Card(0, 1)]
result = game_status(cards_on_table)

Return Value:

Player 1: 2♥
Player 2: 2♦
Player 3: 2♠
Player 4: 2♣

Example #2:

cards_on_table = [2♥, 3♥, 5♥, A♥]
cards_on_table = [Card(0, 2), Card(1, 2), Card(3, 2), Card(12, 2)]
result = game_status(cards_on_table)

Return Value:

Player 1: 2♥
Player 2: 3♥
Player 3: 5♥
Player 4: A♥

Example #3:

cards_on_table = [4♣, 3♥, A♥, Q♠]
cards_on_table = [Card(2, 1), Card(1, 2), Card(12, 2), Card(10, 3)]
result = game_status(cards_on_table)

Return Value:

Player 1: 4♣
Player 2: 3♥
Player 3: A♥
Player 4: Q♠