Worksheet 2

Miscellaneous Exercises

Problem 1:
For the following code

```
class Car:
  def __init__(self, color, model, year):
    self._color = color
    self.__model = model
    self.__year__ = year

c = Car("red", "Toyota", 2020)
```

Which of the following statements will result in an error:

Statement	**Valid? (Yes/No)**
`print(c._color)`	
`print(c.__model)`	
`print(c.__year__ )`	
`c._Car_color = "green"`	
`c._Car__model = "BMW"`	
`c._Car__year__ = 2019`	
`c.__model = "BMW" #THIS ONE IS TRICKY`	

Continue to the next page...

Capstone Project: MaxHandWins

Problem 2:
It is a good practice to make all your attributes private and access them through getter methods. A getter method is a very simple method that returns the value of a specific attribute, and it usually starts with the `get_` prefix.

Go back to all the classes that you defined for the MaxHandWins game and make all your attributes private. Allow access to these attributes through getter methods.

Problem 3:
Let's add a setter method `set_hand` to the `Player` class (a **setter method** is a method that sets an attribute, whereas a **getter method** is one that reads an attribute) that sets the value of the `hand` attribute. This method should just get the hand as an argument and sets it on the instance.

Problem 4:
We need to be able to quickly identify the largest card that a player has in his hand. This will help identify who wins the round.

In the `Player` class, create a method `strongest_hand` that returns the strongest `PlayingCard` that this player has in his `hand`. Remember that a player only has two cards in his hand at one time.

Problem 5:
For the Deck object, *we need to be able to shuffle the cards in the deck*. Create a method `shuffle` that shuffles the cards in the deck. There are many ways to implement this so any shuffling method you can think of can work.

> 👉**PRO TIP** 👈
> You will find `random.randint` exceptionally handy.

```
import random
x = random.randint(0, 10)
# x will be random int from 0 to 10 (inclusive)
```

Problem 6:

Another action that needs to be performed on the `Deck` instance is to draw cards from the deck's `card` attribute. Create a method `draw` that takes an integer argument `n` where n represents the number of cards to be drawn from the deck.
This method should return a list of the `PlayingCards` that were drawn. Assume that we draw from the tail of the list (last element first).

Example:
d = Deck()
d.cards = [1♠, 5♡, 3♢, 10♣, J♢]
d.draw(2) # returns [10♣, J♢]

Hint1: Return None if the count of the remaining cards in the deck is less than n.